

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня
магістра

(назва освітньо-кваліфікаційного рівня)

студента *Рябичева Олега Олеговича*
(ПІБ)

академічної групи *122М-19-1*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

на тему: *Проектування хмарного апаратно-програмного комплексу для 3D друку*

О.О. Рябичев

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинг овою	інституці йною	
розділів кваліфікаційної роботи				
спеціальний	Доц. Сироткіна О.І.			
економічний	Доц. Касьяненко Л.В.			

Рецензент				
-----------	--	--	--	--

Нормоконтролер	Доц. Сироткіна О.І.			
----------------	---------------------	--	--	--

Дніпро
2020

**Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»**

ЗАТВЕРДЖЕНО:

Завідувач кафедри
Програмного забезпечення комп'ютерних систем

(повна назва)

І.М. Удовик
(підпис) (прізвище, ініціали)

« » 20 20 Року

ЗАВДАННЯ

на виконання кваліфікаційної роботи магістра

спеціальності 122 Комп'ютерні науки
(код і назва спеціальності)

студенту 122М-19-1 Рябичеву Олегу Олеговичу
(група) (прізвище та ініціали)

Тема кваліфікаційної роботи Проектування хмарного апаратно-програмного
комплексу для 3D друку

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 22.10.2020 р. № 888-с

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – процес роботи з системами для 3D друку.

Предмет досліджень – моделі та методи створення та вибору архітектури програмно-апаратних комплексів для 3D друку.

Мета роботи – підвищення якості роботи з програмно-апаратними комплексами для 3D друку шляхом проектування надійної та ефективної архітектури.

ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна результатів дипломної роботи:

- удосконалено математичну модель процесів обробки команд, що надходять на керуючу плату принтера;
- отримав подальший розвиток метод багатокритеріальних зважених оцінок для вибору найоптимальнішої архітектури системи.

Практична цінність результатів полягає у тому, що запропонована в роботі архітектура дозволяє забезпечити надійність та ефективність роботи комплексної системи для 3D друку.

4 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	Строки виконання робіт (початок – кінець)
Аналіз теми та постановка задачі	01.09.2020-30.09.2020
Проектування архітектури програмної та апаратної частин системи для 3D друку з використанням методу багатокритеріальних зважених оцінок та формулювання математичної моделі процесів обробки команд, що надходять на керуючу плату принтера	01.10.2020-31.10.2020
Розробка програмного забезпечення для хмарної системи для 3D друку	01.11.2020-30.11.2020

Завдання видав

_____ (підпис)

Сироткіна О.І.

_____ (прізвище, ініціали)

Завдання прийняв до виконання

_____ (підпис)

Рябичев О.О.

_____ (прізвище, ініціали)

Дата видачі завдання: 01.09.2020 р.

Термін подання кваліфікаційної роботи до ЕК 04.12.2020

РЕФЕРАТ

Пояснювальна записка: 128 стор., 38 рис., 9 таблиці, 3 додатка, 70 джерел.

Об'єкт дослідження: процес роботи з системами для 3D друку.

Предмет дослідження: моделі та методи створення та вибору архітектури програмно-апаратних комплексів для 3D друку;

Мета магістерської роботи: підвищення якості роботи з програмно-апаратними комплексами для 3D друку шляхом проектування надійної та ефективної архітектури.

Методи дослідження. Для вирішення поставлених задач використані методи аналізу надійності та ефективності архітектури комплексних апаратно-програмних систем, методи об'єктно-орієнтоване програмування.

Наукова новизна результатів дипломної роботи:

- удосконалено математичну модель процесів обробки команд, що надходять на керуючу плату принтера;
- отримав подальший розвиток метод багатокритеріальних зважених оцінок для вибору найоптимальнішої архітектури системи.

Практична цінність результатів полягає у тому, що запропонована в роботі архітектура дозволяє забезпечити надійність та ефективність роботи комплексної системи для 3D друку.

У розділі «Економіка» проведені розрахунки трудомісткості розробки апаратно-програмного комплексу, витрат на створення і тривалості його розробки, а також проведені маркетингові дослідження ринку збуту створеного продукту.

Список ключових слів: 3D друк, 3D моделі, 3D принтер, NodeJS, Docker, Kubernetes, WEB, хмарні обчислення, апаратно-програмний комплекс.

ABSTRACT

Explanatory note: 128 pages, 38 figures, 9 tables, 3 apps, 70 sources.

Object of research: process of working with system for 3D printing.

Subject of research: models and methods of creation and selection of architecture for 3D printing computer appliance.

Purpose of Master's thesis: increasing the quality of work done with 3D printing computer appliance via the design of a reliable and efficient architecture.

Research methods. In order to solve the given tasks, we used such methods as: analysis of reliability and effectiveness of designs for complex computer appliances, including object-oriented programming.

Originality of the research:

- improved a mathematical model for processing commands received on the motherboard;
- obtained further development for the method of multicriteria-weighted estimations for design of the most optimal system architecture.

Practical value of the results consists of the suggested design, which guarantees the stability and effectiveness of the work in a complex 3D printing system.

In the Economics section, labour input and the length of time taken to develop a 3D printing system was defined and the estimation of expenses for creating it was executed.

Keywords: 3D printing, 3D models, 3D printer, NodeJS, Docker, Cubernetes, WEB, cloud computing, computer appliance.

LIST OF ACRONYMS

DB – data base;

CA – computer appliance;

OS – operating system;

API – application programming interface;

SQL – structured query language;

ORM – object-relational mapping;

UI – user interface;

UX – user experience;

GUI – graphical user interface;

IT – information technologies;

IDE – Integrated Development Environment.

CONTENTS

Introduction.....	9
SECTION 1. ANALYSIS OF THE SUBJECT AREA AND PROBLEM STATEMENT.....	12
1.1. Computer appliance systems and principles of their design.....	12
1.1.1. Examples of successful implementations.....	15
1.2. Main terms and concepts of 3D printing.....	18
1.2.1. 3D printing technologies.....	21
1.2.2. Types of 3D printers.....	23
1.2.3. Types of software for 3D printers.....	24
1.3. Application of 3D printers.....	26
1.3.1. Industrial application.....	26
1.3.2. Household and amateur application.....	27
1.4. Analysis of existing 3D printing systems.....	30
1.4.1. Analysis of problems in the existing 3D printing systems.....	32
1.5. Conclusions for section 1.....	33
SECTION 2. HARDWARE DESIGN.....	34
2.1. Control elements of 3D printer.....	34
2.1.1. Controller board.....	34
2.1.2. User-interaction unit.....	38
2.2. Controlled elements of 3D printer.....	42
2.2.1. Extruder.....	43
2.2.2. Motion controllers.....	46
2.2.3. Heating platform.....	48
2.2.4. Touchscreen.....	50
2.3. Defining the parameters of the 3D printer.....	52
2.4. Hardware architecture design using method of multicriteria weighted estimations.....	54
2.5. Conclusions for section 2.....	59

	8
SECTION 3. SOFTWARE DESIGN.....	60
3.1. Critical software elements for 3D printing system.....	60
3.1.1. Microcontroller firmware.....	60
3.1.1.1. Firmware mathematical model.....	60
3.1.2. Local user interface.....	66
3.1.3. Remote user interface.....	78
3.2. Additional software elements for 3D printing system.....	84
3.2.1. Slicer.....	86
3.2.2. Cloud storage.....	89
3.3. Conclusions for section 3.....	91
РОЗДІЛ 4. ЕКОНОМІКА.....	92
4.1. Визначення трудомісткості розробки програмного забезпечення.....	92
4.2. Витрати на створення програмного забезпечення.....	96
4.3. Маркетингові дослідження ринку збуту розробленого програмного продукту.....	97
4.4. Оцінка економічної ефективності впровадження програмного забезпечення.....	99
CONCLUSIONS.....	102
REFERENCES.....	105
APPENDIX A. SOURCE CODE.....	111
ДОДАТОК Б. ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА.....	127
APPENDIX C. LIST OF FILES ON THE DISC.....	128

Introduction

Relevance of the research. For the last decade we can see a strong trend in development of IT companies in direction of building private ecosystems or computer appliance. The first most popular ecosystem, and basically, the only one that really works is the computer appliance, created by American company, called Apple. This CA includes a lot of different devices, such as: smartphones, tablets, laptops, computers, headphones and smart watches. As well as all kinds of software, such as own operating systems for aforementioned devices, cloud systems for data storing and synchronization, different online services and a lot of other things. As real live experience shows, this approach gives a possibility to provide the best user experience to customers. As a result, profits of the company grow as well as customers' "brand loyalty". Thankfully to self-sufficient of the system, companies gain an opportunity to bring unique features and use cases for their customers.

Mostly, because of the commercial success of computer appliance from Apple, we can see creation of similar solutions from other big, as well as not really big, players in IT market. Such companies, as Google, Microsoft, Amazon invest tons of money and efforts in research, design and development of such kind of systems. Even some of Ukrainian companies, start moving in this direction. The most successful examples are AJAX Systems – a company, which main area of interest lays in area of smart home devices and home security, and PrivatBank – the biggest and one of the most advanced and popular banks in Ukraine.

Moreover, for the last couple of years, we can see not only growth in popularity of CA systems, but also in popularity of 3D printing. Year after year it is being embedded in absolutely different areas of live – from creation of different kinds of spare parts and prototyping, right up to printing of prostheses and food. This splash of popularity comes with a need of simplification of how users work with 3D printers - devices for 3D printing. This issue was addressed by many companies and famous researchers in 3D printing area, some of them are Adrian Bowyer (RepRap), Josef Prusa (Prusa Research), Siert Wijnia, Erik de Bruijn, Martijn Elserman (Ultimaker),

Bre Pettis, Adam Mayer, Zach "Hoeken" Smith, Simon Shen (XYZprinting) and many others.

Unfortunately, right now entrance in area of 3D printing requires from the user some really deep knowledge and expertise in designing of models, work with and maintenance of 3D printers, preparation of models for printing and following afterwork, and sometimes even programming. Such complexity of usage, relatively high price of the equipment itself and its maintenance, as well as lack of user-friendly software, has a remarkable impact on the pace of development of 3D printing technologies.

Nowadays, there are no ready-to-use complex solution, which would have fairly low entry level for an average user of 3D printer.

The purpose of the research is increasing the quality of work done with 3D printing computer appliance via the design of a reliable and efficient architecture.

Tasks of the research. In order to achieve the desired results, such tasks were formed and solved:

1. Describe principles of designing and optimization of CA in 3D printing area;
2. Define the most crucial components of the system;
3. Hardware design through analysis of existing works in this area and choosing the most appropriate components.
4. Software design through analysis of existing works in this area and choosing the most appropriate technologies.
5. Design interaction of all parts of the software among themselves, as well as with the hardware part.
6. Obtain conclusions regarding the feasibility of creation of such system.

Object of research: process of working with system for 3D printing.

Subject of research: models and methods of creation and selection of architecture for 3D printing computer appliance.

Research methods. In order to solve the given tasks, we used such methods as: analysis of reliability and effectiveness of designs for complex computer appliances, including object-oriented programming.

Originality of the research:

- improved a mathematical model for processing commands received on the motherboard;
- obtained further development for the method of multicriteria-weighted estimations for design of the most optimal system architecture.

Practical value of the results consists of the suggested design, which guarantees the stability and effectiveness of the work in a complex 3D printing system.

Author's personal contribution:

1. The scientific results were received solely by the author;
2. Choice of the research methods and implementation tools for the CA;
3. Computer appliance design and software development for 3D printing;
4. Definition of the evaluation criteria of CA's work and argumentative selection of the most critical software and hardware parts of the system;
5. Evaluation of the results.

Structure and capacity of the work. The diploma project consists of introduction, three main chapters and conclusion. It includes 128 pages, including 87 pages of the main part with 38 figures, the list of used resources with 70 items on 6 pages, 3 applications on 17 pages.

SECTION 1

ANALYSIS OF THE SUBJECT AREA AND PROBLEM STATEMENT

1.1. Computer appliance systems and principles of their design

Computer appliance is a complex of hardware and software, which work together in order to execute one or more tasks [15]. Usually, deep integration of software and hardware parts allows manufacturers to achieve not only high level of performance and reliability, but also the best UX from usage of the whole CA, as well as of each single part of it. In such kind of systems, manufacturers can provide the most efficient interaction of the parts of the system, due to having the full control over each separate part of the system.

Generally, hardware part is one or a few devices, which are used to perform some specific tasks of the CA. Communication of the hardware components is done via specific software and a set of hardware interfaces, both wired and wireless. The most common wired interfaces are: USB (2.0, 3.0, 3.1, Type-C, Thunderbolt), HDMI, DisplayPort, pins, etc. [4]. Quite often, in such kind of system, we might find some custom standards of wired interfaces, invented by manufacturers in order to achieve the most effective interaction or to provide unique functionality in boundaries of the system. Those interfaces are, usually, called proprietary interfaces. The most common ones are – Lightning and Magsafe from Apple. If we talk about wireless interfaces, it worth mentioning the most common and popular examples, such as Bluetooth, WiFi and NFC. The hardware part might include devices for the end-customers, such as smartphones, printers, air-conditioners, fridges, headphones, etc. It might also include some internal devices, which are hidden from the end-customers, some of them are: servers, data storages, systems for cloud computations, etc.

The software part is a set of different applications, which are used either for communication between the components of the system or performing some particular tasks for the user. The list of tasks, performed by this software can vary. Starting with some global challenges, such as management of the hardware parts of the system,

generation of graphical user interface, support of system's vital parts, up to some really small and local tasks, among them are storing, processing and other kinds of work with data, execution of some specific computations and other functions. The best example of a complicated software is operating systems, which are responsible for distribution of the system resources, work with internal and external hardware and software, processing of user's input and lots of other tasks. The other one is firmware is a piece of software, which is usually responsible for low-level work with different devices. At the same time, there are some examples of more local software: data storages, data processors, graphical editors, remote control systems, etc.

Those systems have to meet extremely high requirements to reliability, performance and convenience of use of the whole system and each separate part of it. Moreover, quite often some unique functionality and interaction between elements of the system is required from the computer appliance systems.

The successful work of CA relies on many things, one of the most important is stability and efficiency of controlling software, which is defined by the organization level inside the system.

Currently, there are a huge number of standards, which define each and every step of the CA's lifecycle [15] and meant to be the basis for creation of reliable and productive computer appliances. Some problems might arise, while creating such complex systems. Some of them are defined by the specificity of a particular system, meanwhile others are defined by peculiarities of the development process. Development of both hardware and software parts of the system is done simultaneously during the whole lifecycle of the system – from sketch design up to the testing phase. This process is followed by the need of constant improvement discussions, usually contradictory requirements, which the system should meet.

When designing CA, architects should consider requirements to each single element of the system, while providing the best flexibility in terms of functionality extending and introducing new elements of the system and means of their interaction [31]. The mistakes, made on early phases of design seem to become the most critical

when trying to add new elements or communication processes to the existing computer appliance system.

In order to minimize risks when extending the system, the flexible modular approaches for system's architecture are usually used [29]. Modular systems, both software and hardware, when build correctly, are usually marked by high level of flexibility and scalability.

Modular system is a system, which is split into small independent blocks or modules, usually, responsible for performing a single specific task. Based on the complexity of the system it might include from dozens, up to hundreds and thousands of such modules. It allows developers to bring changes into the system, replace, remove and add specific elements of the system almost without any changes to any other parts of the system.

Modular systems are featured with a low level of binding between components and high level of scalability [31]. When building such kind of systems, the separation into modules is performed based on the functional principle, which minimizes the number of cross-modular connections and by doing so – reduces the overall complexity of the system. Usually, a specific software is used to provide cross-modular communication. It helps to keep parts of the system as independent and opened to changes, as possible.

Also, rightly designed modular system makes it much easier and cheaper to test, maintain and support both the system itself and each separate part of it.

One of the most common example of a modular computer appliance is a familiar computer and the operating system, which is installed on it [31]. The computer itself consists of many small independent components, as well the OS is built of many hundreds of components. This whole diversity of software and hardware communicates between itself in order to solve specific tasks of the end-user.

1.1.1. Examples of successful implementations

During the last couple of years, it became really popular, among different big companies, to create proprietary computer appliances. It allows the manufacturers to “bind” the user to their ecosystem by providing some unique software and hardware products and ways of interaction with them.

One of the first and most advanced CA, which defined the direction of evolution of the technical world for many years ahead is ecosystem of devices and software, created by an American company, called Apple.

This computer appliance system (Fig. 1.1.) includes lots of different devices, such as: smartphones, tablets, laptops, computers, TV-stations, smart watches, sound bars, periphery (headphones, chargers, etc.), components of the smart home and others. Moreover, it includes all kinds of applications, responsible for both communication between software and hardware parts of the system, such as: operating systems (OS X, iOS, iPadOS, WatchOS) and systems of remote control for different devices (applications – Watch, Home, etc.), and for solving of particular tasks for the user – browser Safari, Apple Music, Apple TV and many other services.



Fig. 1.1. Apple's CA

Each component of this computer appliance is capable of being a highly efficient independent part, as well as a well-fit part of the whole, huge ecosystem. It allows

developers to create unique use-cases, meant to improve the overall user experience. There are some examples of such scenarios: seamless transferring of the content of clipboard between different devices, unlocking of laptops and phones via smart watches, editing of pictures on the computer via smartphone, seamless reconnection of wireless headphones between different devices. Also, one of the biggest Apple's achievement in the area of computer appliance systems is synchronization of the user's data between all their devices, which makes it more comfortable to switch between devices, update and replace them.

Although, Apple is a leader in development of ecosystems, there are many other popular brands, trying to create their own computer appliances. Such foreign companies are Google, Microsoft (Fig. 1.2.), Samsung (Fig. 1.3.), Huawei (Fig. 1.4.) and Xiaomi (Fig. 1.5.). Each one of their own ecosystems, which have different size and the level of progressiveness. They are being constantly extended with different new software and hardware components.

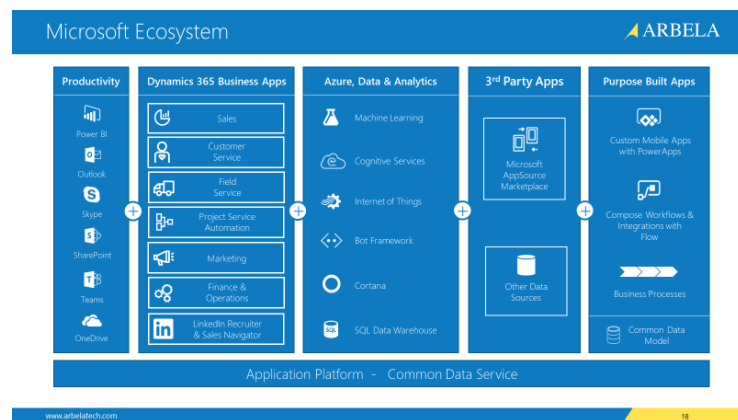


Fig. 1.2. Microsoft's ecosystem

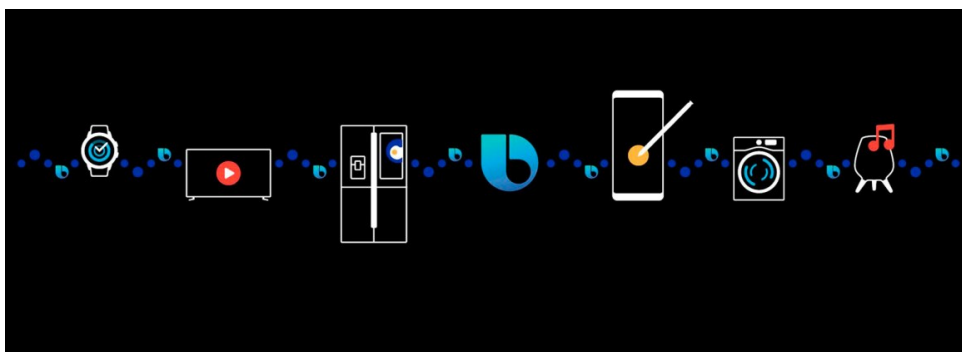


Fig. 1.3. Samsung's ecosystem



Fig. 1.4. Huawei's ecosystem

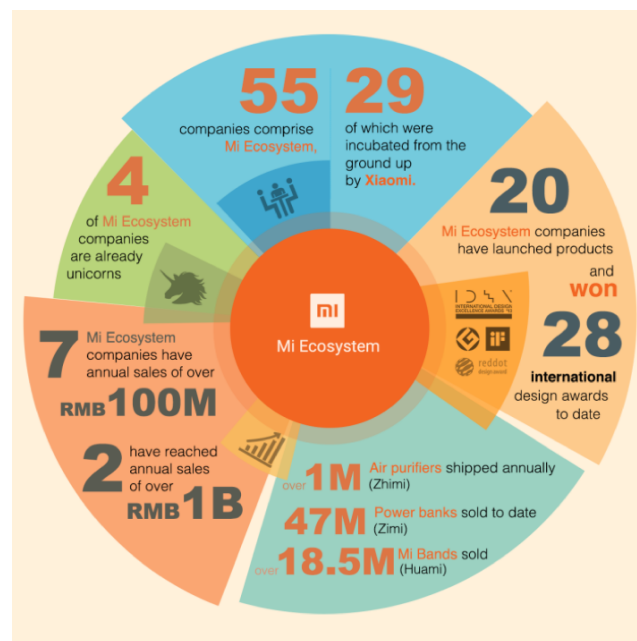


Fig. 1.5. Xiaomi's ecosystem

As well it worth mentioning, that there are some domestic companies, which also work in this direction. Ajax Systems is a manufacturer of hardware and software for smart home and home security, their solutions are ones of the most popular around the globe. Another famous example is one the largest and most advanced banks of Ukraine – PrivatBank. They have one of the largest networks of different terminals, ATMs, as well as different applications for finance management.

So, in general, it is possible to state that there is a stable tendency in direction of creation of proprietary ecosystems.

1.2. Main terms and concepts of 3D printing

Additive technologies (3D printing) – one of the forms of additive technologies manufacturing, where a three-dimensional object is created via sequential imposition (printing, growing) of layers of the material, based on a digital model [1]. Printing is performed by a special device – 3D printer, which assures creation of a physical object by sequential imposition of plastic material, using virtual 3D model. Generally, 3D printers are faster, more accessible and simpler in usage, than other technologies of additive manufacturing. 3D printers give manufactures a possibility to print different kinds of spare parts and mechanisms, using different materials with different mechanical and physical properties in one printing session.

Starting from 2003, we can see a huge growth in amount of sold 3D printers. Moreover, the cost of 3D printers keeps decreasing. 3D printing technology also finds different applications in manufacturing of jewelry, shoes, industrial design, architecture, design and construction as well as in atomic, automobile, aerospace, dental and other areas.

Many of different, concurrent technologies of creating 3D models has become available in the mid-2010s. The main difference between them lay in step of detail layers' construction. Some technologies use melting or softening of the material for creation of layers (SLS, FDM), while others use liquid materials, which become hard, based on different principles [42].

There are 2 ways of 3D models' creation: manual graphical design, using computer and 3D scanning. Manual modeling or preparation of the geometrical data in order to create three-dimensional computer graphic is a complicated and creative process, which kind of similar to sculpture. Usually for creation of 3D models, professionals use specific software. The most famous and popular examples are: Autodesk Maya, Autodesk Mudbox, Autodesk 3Ds Max, Blender, FreeCAD and

OpenSCAD. 3D scanning is a process of automatic gathering and analysis of the real-world object's data, such as form, color and other properties, followed by converting it into a three-dimensional model. Both manual and automated creation of 3D models might cause some troubles for an average user, who does not have any expertise in areas of modeling and 3D printing. That is the main reason, behind growth in popularity of marketplaces for 3D printing.

3D printer is a specific machine with an appropriate software [32], which builds a model, using an additive approach. Such kind of machines can work with different materials, starting with different kinds of plastic: PLA, ABS, PVA, Nylone, HIPS straight to pasty foods, such as cheese, chocolate, pate, etc.

In the process of printing, a 3D printing reads content of a file in special format, usually, GCode, which contains a set of primitive consecutive commands and inflicts consistent layers of special material, constructing a 3D model from a series of cross-sections [32]. Those layers, which correspond to virtual cross-sections of a CAD model are being combined of fused together in order to create an object of the given form. The main advantage of this approach is a possibility to create geometrical forms with almost endless complexity.

“Resolution” of a printer is defined by thickness of layer, which are being accumulated (Z-axis) and accuracy of print head's (extruder's) positioning in horizontal area (X and Y axes) [32]. Resolution is measured in DPI (amount points per inch) or micrometers (microns). Usually standard thickness is 100mkm (250 DPI), however there are some devices, which are capable of printing with higher accuracy, up to 16mkkm (1 600 DPI). Resolution of axes X and Y is similar to usual values of two-dimensional laser printers. The typical particle size is something between 50 and 100mkm (from 510 to 250 DPI) in diameter.

Creation of a model, using modern additive technologies might take from a few hours to a few days and sometimes even weeks, depending on the used methods and model's complexity [48]. Industrial additive systems, in general, are capable of reducing the time, required for a model to be printed to a few hours, but it highly

depends on the type of the machine, size and number of details, which are being produced simultaneously.

When producing huge batches of polymer products, traditional manufacturing methods, such as die-casting might be a lot cheaper. However, additive technologies have some benefits, when producing smaller batches, which allow us to achieve higher production pace and design flexibility, as well as increased economy per unit of manufactured goods.

Production of some models with high accuracy, usually requires some extra processing. Even though, the resolution of modern printers is more than enough for most projects, printing of objects with slightly increased measurements and following subtractive machining with high-precision tools allows users to create models with higher accuracy.

Some methods of additive manufacturing allow usage of a few different materials as well as colors within one production cycle. Lots of 3D printers use “backing” or “supports”, while printing [32]. They are required in order to construct parts of the model, which do not touch neither lower layers nor the working platform. Supports themselves are not a part of the model, so once printing is over, they are being either broke off or melted, using some specific solvent (usually – water or acetone).

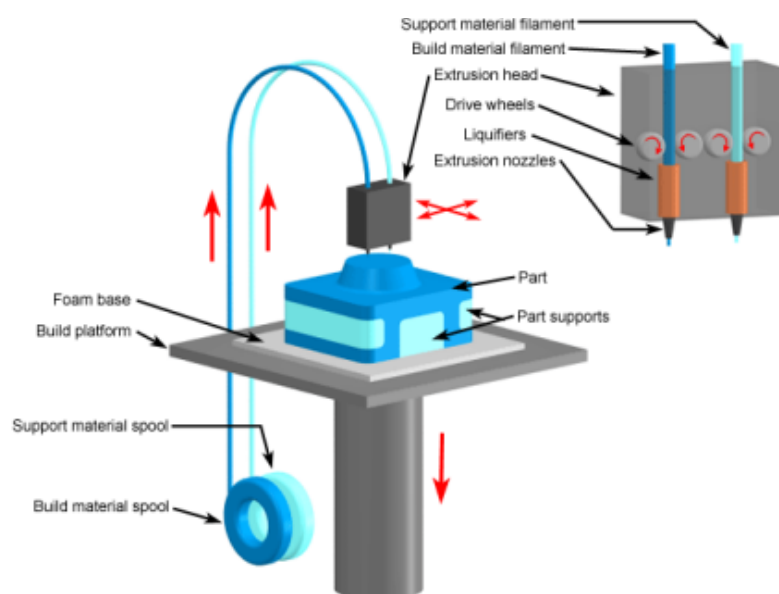


Fig. 1.6. Working scheme for an FDM 3D printer

1.2.1. 3D printing technologies

From the end of the 70s of the 20th century, the world has seen a few methods of 3D printing. The very first printers had huge sizes, high cost and limited functionality [42].

Currently there is a wide variety of different methods of additive manufacturing. The main difference lays in the way of how layers are being created and materials, which can be used for printing. Some methods are based on melting or softening materials to create layers. The most common ones of them are: selective laser sintering (SLS), selective laser melting (SLM), direct metal laser sintering (DMLS), printing, based on layers surfacing (FDM or FFF) [42]. Meanwhile, another direction is production of solid models, using polymerization or liquid materials, well-known as stereolithography (SLA).

In case of laminated object manufacturing (LOM), thin layers of the material are being cut to the required contour, with following junction into a single piece. Paper, polymers and metals can be used as materials for LOM.

Each mentioned method has unique advantages and disadvantages, that is why some companies provide a choice of material, while constructing a model – either polymer or powder. Printers, which are based on LOM technology, usually use common office paper for constructing solid prototypes.

The most important things, while picking the most optimal device are printing speed, price, cost of printed prototypes, variety and pricing of compatible consumables.

Printers, which are capable of producing complete metal models have quite high price, however it is possible to use cheaper devices for producing molds with following casting of metal parts.

It is worth mentioning that some of the printers require specific conditions of use (for example, maintenance of a fixed temperature or humidity of the environment, where they are placed), which implies additional limitations, while selecting the most appropriate device.

Apart, from aforementioned model creation methods there are some other popular ones. The main 3D printing methods are listed in table 1.1.

Table 1.1

The main 3D printing methods

Method	Technology	Materials
Extrusion	Modeling via melting or softening of the material for creation of layers (FDM or FFF)	Thermoplastics (PLA, ABS, etc.)
Wiry	Construction of random forms via electron-beam freeform fabrication (EBF3)	Almost any metal alloys
Powder	Direct metal laser sintering (DMLS)	Almost any metal alloys
	Electron-beam melting (EBM)	Titan alloys
	Selective laser melting (SLM)	Titan, cobalt-chromium alloys, stainless steel, aluminum
	Selective heat sintering (SHS)	Powder thermoplastic
	Selective laser sintering (SLS)	Thermoplastics, metal powders, ceramic powders
Streamed	3D printing (3DP)	Hips, plastics, metal powder, sand mixtures

Continuation of Table 1.1

Lamination	Constructing of objects via lamination	Paper, metal foil, plastic film
Polymerization	Stereolithography (SLA)	Photopolymers
	Digital light processing (DLP)	Photopolymers

1.2.2. Types of 3D printers

In general, all 3D printers can be split in two main categories: industrial machines and household devices [64].

Industrial machines, usually, are being used in huge manufacturing to create prototypes and complex details with high accuracy. Lately, development of industrial additive production has been evolving with tremendous pace. For example, joint American-Israel company Stratasys supplies devices for additive manufacturing, which price goes from 2 000 to 500 000 dollars, while company General Electric uses high-class machines for creation of parts of gas pipes.

Meanwhile, household devices are created for an average user, for solving their daily tasks. Mainly, private companies and enthusiasts are involved in development of such devices. Their count grows every day. The most part of work is done by amateurs for personal or community needs, with help from academic associations and hackers.

Those printers can be used by both, private manufacturers for production of different prototypes, parts of toys and confectionery for creation of unusual and complicated products from chocolate and other pasty mixtures.

Household 3D printers can be separated into two different categories: food and non-food printers. The first ones use different kinds of pasty materials for creation of decorations and deserts, which impress the imagination and attract new customers. While another kind uses differed kinds of plastics and metals. They are the most popular amongst engineers and companies, who have a need for prototyping (for

example, manufacturers of home appliance, automobile spare parts, different machines, etc.).

Typical user of a household 3D printer does not possess any specific knowledge and skills in areas of 3D printing, design, electronics, programming and chemistry. This implies additional requirements to usability and simplicity of usage of those devices. Even though, for the last couple years industry of household 3D printer made a huge step towards an average customer, unfortunately this is still quite complicated piece of equipment, usually without a good user-friendly software and with relatively high price. Those issues scare household customers away from buying, which slows the pace of development, popularization and as a result availability of equipment for 3D printing.

1.2.3. Types of software for 3D printers

3D printing is a complicated process, which consists of many stages with different complexity. Each of those stages requires from the user knowledge and skills in different areas, related to the process of additive models' creation [68]. Some of them require direct physical work from the user, for example, calibration of printers and processing of printed object. While others require some specific software.

The first stage for creation of any model is design. Usually, 3D models are being created, using specific software for 3D modelling. Using it, the user is capable of constructing of geometrical figures with almost any shape and complexity. However, when designing a particular model, complexity of printing it on a specific 3D printer should be taken into consideration. Because, too big or complex figures might simply not fit onto printer's working platform, or take days to be printed on a single device, or it might take tons of material to print it, which will lead to the necessity of replacing coils with material during the printing. All of those problems might have a negative impact on the final product. In order to solve this problem, complicated figures are usually split into smaller parts, which are much easier to print, because they take less space, require less material (in terms of one printing session) and can be printed

simultaneously and independently. Also, this approach reduces the overall complexity of the construction and makes it modular and so more suitable for future usage and maintenance.

After the model was designed, we can export it as an STL file, which contains the information about the object in form of the list of triangle edges, which describe its surface and their normal. However, most of 3D printers do not understand this format, so before the printer will be able to print the desired model it is required to prepare it for printing, using a specific software, called slicer [2].

While preparing the model, the user should correctly set dimensions of the printer (size of working platform), temperature of nozzle and platform (in case if printer supports it), nozzle diameter, material diameter, and much more configurations, which are related to properties of a particular 3D printer. Then they should add one or more models on a virtual platform, move them around the platform and one another, configure their angles of inflection over the platform, their proportions and sizes. In most cases, as a result of the software's work, the customer receives a file in GCode format, which consists of a set of primitive commands, supported by the most of 3D printers.

After the GCode was received, it should be passed onto the 3D printer in some way. Different companies use really different approaches for interaction of customer with the device. An example of the most-primitive interface is a printer with a single socket for a USB flash drive or SD-card. Such kind of printers does not have neither an embedded screen nor a possibility of remote control. Software of this printer has really primitive functionality: read file from the drive and gradually send command from it to the controlling devices (extruder, platform, etc.). This approach allows manufacturers to reduce the cost of the product, but it might not be really convenient for the customer if they work with it on a daily basis.

That is why, for more advanced work with 3D printer, usually, a whole set of software, which is responsible for user's interactions is being developed [5]. The most important and low-level part of it is the firmware. This piece of software is responsible for the low-level interaction with the control elements, such as heating platform,

extruder, drivers for moving platform via axis Z and extruder via axes X and Y, temperature controllers and others. Quite often, user does not have neither a need nor tools to interact with it.

Then, for some 3D printers, a GUI is being developed. It helps user to manipulate control elements, control the printing process and calibrate moving elements of the printer. This piece of software acts as an agent between the user and the device's firmware. Usually, this interface is shown on the display, which is embedded into the printer and gives a possibility to: launch, pause and cancel printing process, control the temperature of the heating elements, position platform and extruder, etc.

The most advanced systems also include a graphical user interface. For. Remote control over the 3D printer. It provides all the features of the local interface, but it does not require the user to be anywhere near the device. This software might be provided in the form of WEB or mobile applications. They connect to the printer, which is connected to the global network via a. specific proxy-server.

By the way, it is possible to see other kinds of software, which acts as different additions and plugins for the core of the ecosystem (firmware, local and remote-control interface). For example, a cloud storage of 3D models, where users can store their models in STL or GCode formats, share them with other users, etc. As well, as a cloud slicer, which allows the customer to prepare their model to be either printed on their particular printer or saved into the aforementioned cloud storage.

1.3. Applications of 3D printers

1.3.1. Industrial application

Industrial 3D printers are often used for quick prototyping and research. Usually those are quite big machines, which use powder metals, sand mixtures, plastic and paper. This type of devices is used by universities and commercial companies.

Achievements in quick prototyping led to creation of materials, which can be used for production of final products, which facilitated evolvement of 3D manufacturing of final products, as alternative to the traditional methods. One of the

biggest advantages of quick manufacturing is relatively low costs of production of small batches.

Some companies provide services for customization of objects, using simplified software with posterior creation of unique 3D models. One of the most popular directions is production of bodies for telephones.

Current printing speed of 3D printing is quite low, which might become a deal-breaker for using them for huge manufacturing. In order to overcome this flaw, some FDM devices use two or more extruders [8], which allow printing using different colors, polymers and even creating a few different models simultaneously. In general, this approach increases productivity, without usage of a few printers. Just one microcontroller is more than enough for reliable work of a few nozzles.

Those devices allow creation of a few identical objects, based on the same digital 3D model, with usage of different materials and colors. The printing speed increases proportionally to the number of nozzles. Moreover, it also results in some kind of economy of electricity, thanks to usage of just one working platform, which quite often requires heating. Together those two factors decrease production costs.

Quick production is still a relatively new method, which is not fully studied yet. However, different experts believe, that quick production is the technology of a new level. Some of the most promising quick prototyping directions, which can be used for quick production are SLS and DMLS [11].

1.3.2. Household and amateur application

Nowadays, household devices for 3D printing attract only enthusiasts and amateurs, while practical usage is still quite limited. However, 3D printers are being used for printing of mechanical watches, gears, decorations, etc. Web markets in the area of household 3D printing often provide design of hooks, doorknobs, massage tools and similar stuff.

3D printing is also being used in amateur veterinary and zoology. In 2013 3D printed prosthesis helped a duckling to stay on its feet. 3D printers are quite popular for household manufacturing of different bijouterie – necklaces, rings, handbags, etc.

The main purpose of a public project, called Fab@Home is development of household printers for general usage. It was organized by a small group of researchers: Evan Malone and Hod Lipson. Over the years many people have joined their cause: Daniel Cohen, Jeffery Lipton, Dan Periard, Max Lobovsky (CEO Formlabs), James Smith, Michael Heinz, Warren Parad, Garrett Bernstein, Tianyou Li, Justin Quartiere, Daniel Sheiner, Kamaal Washington, Abdul-Aziz Umaru, Rian Masanoff, Justin Granstein, Jordan Whitney, Scott Lichtenthal, Karl Gluck. Those devices were tested in laboratories for production of chemical connections. Printer can use any kind of material, which can be extruded from syringe in state of liquid or paste.

3D printing has found application in clothing area. Couturier use printers for experiments, while designing swimwear, shoes and dresses. Commercial usage includes quick prototyping and 3D printing manufacturing of professional sports shoes.

Studies in area of 3D printing are being actively conducted by biotechnological companies and academic institutions [19]. Those studies are targeted at looking for possibilities of usage of inkjet / drip 3D printing in tissue engineering to create artificial organs. Technology is based on infliction of layers of living cells upon gel substrate or sugar matrix, with gradual layered build-up to create three-dimensional structures, including vascular systems. First production system for tissue 3D printing, based on bioprinting technology from NovoGen was introduced in 2009 by Mickael Le Helloco and Thierry Burlot.

Organovo is one of 3D printing pioneers conducts laboratory studies and develop manufacturing of functional three-dimensional samples of human tissue for usage in medical and therapeutic studies. For bioprinting company uses 3D printer NovoGen MMX. Organovo believes, that bioprinting will allow speeding up testing of new medical supplies before clinical studies, which will result in reducing of costs and time, which are required for development of treatments. In long-term perspective

Organovo hops to adapt those technologies for creation of grafts and in surgical applications.

3D printing is used for creation of implants and devices for medical purposes. Successful operations include such examples as implantation of titanium pelvic and jaw implants, as well as plastic tracheal splints. The widest usage of 3D printing is expected in the area of hearing aids and dentistry.

Some companies provide service for online 3D printing, which are available for both private clients and industrial companies. User only has to upload a 3D design to the website, which then will be printed, using industrial machines.

The future appliance of 3D printing might include creation of scientific equipment with opened source code for usage in public laboratories and other scientific usages, like reconstruction of fossils in paleontology, duplicating priceless archaeological artifacts, reconstructing bones and body parts for forensic science, reconstructing badly damaged evidence collected from crime scenes. The technology is also being considered for construction applications.

Usage of 3D scanning technologies allows creation of duplicates of real objects without the need for casting methods, which require high costs, complicated in execution and might have a destructive impact in cases with precious and fragile objects of cultural heritage.

An additional example of appliance of 3D printing technologies is usage of additive manufacturing in construction. It might help to speed up pace of construction, while reducing costs. More specifically, 3D printing technologies are being investigated in terms of usage of those technologies for construction of space colonies.

Additive manufacturing can be used to create waveguides, couplings and bends in terahertz devices. High geometrical complexity of such devices could not be achieved, using traditional production methods.

Additive production requires high flexibility and constant improvements of available technologies from manufacturing companies in order to support competitiveness. Protectors of additive production predict confrontation of 3D printing and globalization, because of potential growth of household manufacturing,

which might eventually replace huge companies. In reality, integration of additive technologies in commercial production serves rather as an addition for traditional subtractive methods than a full replacement.

1.4. Analysis of existing 3D printing systems

Over the last decade the world has seen quite a few commercially successful systems for 3D printing. All of them provide the base features, like printing itself, settings management, etc., as well as more unique features, like models' preparation (slicing), remote control, etc. However, at the time being there is not any system which would not require the user to be a professional engineer. Most commercial and not commercial solutions for 3D printers are based on RepRap project, which was originated by one of the most famous researches in area of 3D printing - Adrian Bowyer.

The main goal of RepRap is to provide means for creation of 3D printers with an open-source code [1], which is provided under the public GNU license. Devices, based on RepRap are capable of printing of plastic component, which could be used to create copies of the original device.

Because of the public access to the blueprints of printers from RepRap, a lot of projects take over analog technical solutions, creating kind of a global ecosystem, which mostly contains free-to-modify devices. Unfortunately, it leads to high diversity of the quality level as well as the complexity of designs and devices, based on them.

One of the most popular 3D printer making companies is Ultimaker. This company was founded by a small group of scientists from Netherlands - 1 by Martijn Elserman, Erik de Bruijn, and Siert Wijnia in 2011. They design and create modern 3D printers as well as software for them. Currently Ultimaker has brought 8 different 3D printers to the market [34]. All of them, primarily use different sorts of plastic as a printing material. Mostly, apart from some low-priced models they provide an embedded screen, with a user interface for local control over the printing process and calibration. But, one of the biggest achievements of Ultimaker company is a specific

software, which helps the user to prepare their models to be printed on their particular 3D printer, called Ultimaker Cura. Cura is a frontend part, of the slicer, which provides a UI, where users have an opportunity to configure lots of different settings, which are specific for a particular 3D printer, like dimensions (X, Y and Z axed), number of extruders, used materials, presence of a heating platform, printing speed, nozzle diameter, printing resolution, dialect of GCode, etc. This piece of software communicates with a general backend, called CuraEngine, which basically does all the heavy lifting in terms of computations and generation of the final result – Gcode file with a set of commands, which can be interpreted by the printer's firmware. Also, it is possible to connect to the printer, using a USB connector and control 3D printing process, using UltimakerCura [34]. But they do not provide any kind of remote-control interface, or cloud storage or any other additional software, which would provide some extra functionality for the core system components.

The most interesting thing about this software, is the fact that it has an open-sourced code base. Which means, that it is free and driven by the community, which grows every day. Moreover, their software can be used as a base for other developers and companies.

Another example of commercially successful 3D printing system is Prusa from Prusa Research. Prusa Research was founded as a one-man startup in 2012 by Josef Prusa, a Czech hobbyist, maker and inventor - and now one of the most famous names in the 3D printing industry. Currently they have delivered 4 different 3D printers which are available as both RepRap kits and fully assembled printers. Their printers support different kinds of plastic, as well as SLA printing. Although, they provide not as much functionality as devices from Ultimaker, but they target the low-priced segment of the market. As well as the system from Ultimaker, Prusa do not provide any kind of secondary software, only the core functionality.

Printers from Prusa do not have neither an embedded colorful touchscreen with a comprehensive UI nor a specific software for slicing. However, their prices start from 299\$, while prices for printers from Ultimaker start from 3 500\$. Most users of their

3D printers are enthusiastic engineers, which now not only how to use the 3D printer, but also how to assemble and maintain it.

However, there are much more other solutions from different companies and independent engineers and scientists, but they are not as famous and popular.

1.4.1. Analysis of problems in the existing 3D printing systems

It is safe to say, that current solutions have at least three major problems:

1. None of the currently available systems provides a full, comprehensive CA, which would meet all modern requirements and cover all phases of computer-related 3D printing, such as: slicing, local UI, remote-control, cloud storage for printed or sliced models, 3D modeling software;
2. Although, we can see a strong tendency of decreasing of the prices on 3D printers, users are still forced to pick between either functionality or reasonable price. As we can see, comparing two of the biggest manufacturers of 3D printers, the price for slightly more advanced devices might be more than 10 times higher, than the price of more simple solutions;
3. However, the biggest problem seems to lay in extremely high entry level for an average customer. It looks like all existing solutions require user to have knowledge and skills in modeling, physics, chemistry, engineering, programming and electronics.

The mentioned issues cause slowing of development and expansion of 3D printing technologies in different areas of life. 3D printing industry might get a huge boost, if someone were to a relatively cheap, reliable, comprehensive and most importantly simple enough system, which would include different devices, such as 3D printers, as well as different kinds of software, like slicers, 3D modelling tools, user-friendly local and remote interface for printing management, cloud storages and a reliable cloud system, which would allow all those parts to communicate between each other.

1.5. Conclusions for section 1

The modern 3D printing industry grows quite rapidly, and made a huge step forward over the last few decades. A lot of different 3D printing methods were introduced and modernized over the years, giving people the opportunity to create all kinds of models, using different materials, like paper, plastic, metals, chocolate, cheese and other kinds of pasty food. 3D printing technologies are being used in all areas of life. Manufacturers of automobiles, toys, household appliances use them for prototyping and even full-size production. Scientists, engineers and designers use them to bring their most interesting and ambitious ideas to life, via prototyping. 3D printers even used for creation of prostheses, which help people with different health problems to get back to the normal life. Recently, even confectioners and bakers had found an appliance of 3D printing in their daily work by printing different sorts of deserts and decorations, which help to attract customers.

The pricing on devices for 3D printing can be really different. From super cheap kits, for 100\$ up to highly advanced complex industrial systems for over 500 000\$. This diversity allows users to select the most appropriate solution, based on their needs, from small household engineering purposes to big industrial manufacturing.

However, it looks like the area of local non-engineering intrapreneurs is not covered well enough. Most of the modern solutions seem to completely ignore this part of market. Almost all of them are either way too complex for an average user or do not provide a complete ecosystem, which would cover all customer's needs with robust user-friendly interface and reliable ways of communications between all parts of the system, both software and hardware.

SECTION 2

HARDWARE DESIGN

2.1. Control elements of 3D printer

A 3D printer is a complicated piece of tech, which consists of hundreds of smaller parts, which are responsible for different functions of the printer. There are lots of different variations of design for 3D printers, each of them has its own cons and pros. However, the one common thing between all of them is that, all of their parts can be split in two main groups [3].

The first ones are controllers, which are responsible for interpreting commands sent by user or other control units and then passing them onto the controlled elements of printer.

Meanwhile the second group is responsible for execution of the received commands. They can be called the controlled elements. Usually, they are responsible for a very limited set of primitive operations.

Quality and efficiency of the both groups are crucial, as the quality of the final product depends on them both, as well as, on their inter-communication. If either of them fails to perform its role, or has any defects it may have a negative impact on the appearance, internal structure and overall integrity of the produced model. In some cases, it might even lead to disability of the printer to create some complex figure or detail.

In this work we will take a look on the biggest and most important hardware parts of a 3D printer, like controller board, user-interaction unit, motion controllers, heating platform and a touch screen.

2.1.1. Controller board

The controller board, also known as motherboard is a main control unit of the 3D printer [3]. It is responsible for maintaining the smooth processing of the machine.

Being responsible for all the fundamental operations, motherboard works as the brain of the 3D printers. It directs the motion components as per the instructions sent from a computer and at the same time, interprets signals from the sensors. Usually, the role of a controller board is given to one of the boards from Arduino family. The first Arduino board was created by famous researches in the computer hardware area: Hernando Barragán, Massimo Banzi and David Cuartielles.

Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE that runs on your computer, used to write and upload computer code to the physical board.

The Arduino platform has become quite popular with people just starting out with electronics, and for good reason. Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board -- you can simply use a USB cable. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program. Finally, Arduino provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package.

The Arduino hardware and software were designed for artists, designers, hobbyists, hackers, newbies, and anyone interested in creating interactive objects or environments. Arduino can interact with buttons, LEDs, motors, speakers, GPS units, cameras, the internet, and even smart-phone or TV. This flexibility combined with the fact that the Arduino software is free, the hardware boards are pretty cheap, and both the software and hardware are easy to learn has led to a large community of users who have contributed code and released instructions for a huge variety of Arduino-based projects.

For everything from robots and a heating pad hand warming blanket to honest fortune-telling machines, and even a Dungeons and Dragons dice-throwing gauntlet, the Arduino can be used as the brains behind almost any electronics project.

There are many varieties of Arduino boards that can be used for different purposes. Some boards look a bit different from the one below, but most Arduinos have the majority of these components in common (Fig. 2.1.):

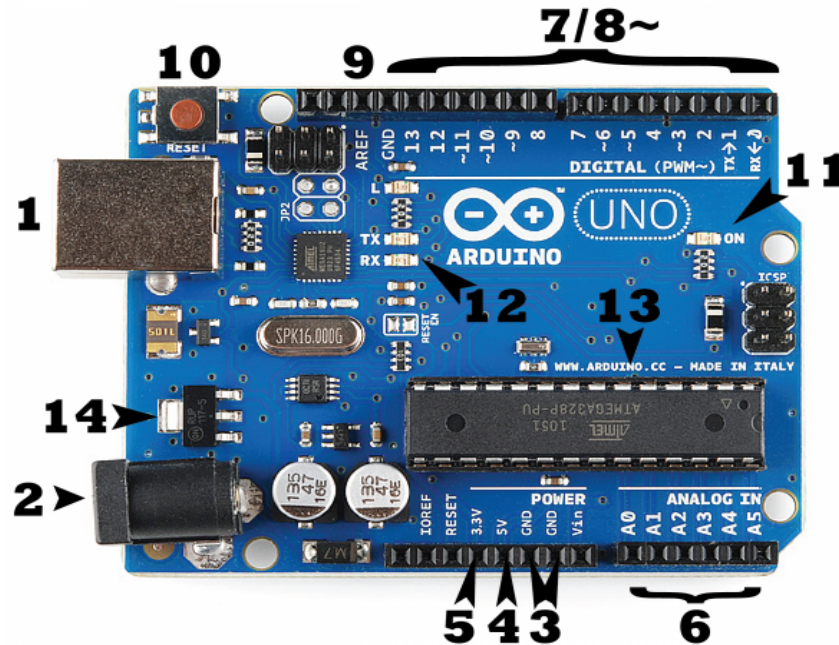


Fig. 2.1 Schematics of Arduino Uno

Every Arduino board needs a way to be connected to a power source. The Arduino UNO can be powered from a USB cable coming from computer or a wall power supply that is terminated in a barrel jack. In the picture above the USB connection is labeled (1) and the barrel jack is labeled (2). The USB connection is also the way of loading the code onto the Arduino board [27].

The pins on the Arduino are the places to connect wires to construct a circuit (probably in conjunction with a breadboard and some wire) [6]. They usually have black plastic ‘headers’ that allow us to just plug a wire right into the board. The Arduino has several different kinds of pins, each of which is labeled on the board and used for different functions.

- GND (3): Short for ‘Ground’. There are several GND pins on the Arduino, any of which can be used to ground the circuit [6].

- 5V (4) & 3.3V (5): As the names imply, the 5V pin supplies 5 volts of power, and the 3.3V pin supplies 3.3 volts of power. Most of the simple components used with the Arduino run happily off of 5 or 3.3 volts [6].
- Analog (6): The area of pins under the ‘Analog In’ label (A0 through A5 on the UNO) are Analog In pins. These pins can read the signal from an analog sensor (like a temperature sensor) and convert it into a digital value that we can read.
- Digital (7): Across from the analog pins are the digital pins (0 through 13 on the UNO). These pins can be used for both digital input (like telling if a button is pushed) and digital output (like powering an LED) [6].
- PWM (8): some of the digital pins have tilde (~) next to them (3, 5, 6, 9, 10, and 11 on the UNO). These pins act as normal digital pins, but can also be used for something called Pulse-Width Modulation (PWM) [6].
- AREF (9): Stands for Analog Reference. Most of the time this pin is not used. It is sometimes used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins [6].

Just like the original Nintendo, the Arduino has a reset button (10). Pushing it will temporarily connect the reset pin to ground and restart any code that is loaded on the Arduino. This can be very useful if the uploaded code doesn't repeat, but user would like to test it multiple times.

Just beneath and to the right of the word “UNO” on the circuit board, there's a tiny LED next to the word ‘ON’ (11). This LED should light up whenever the Arduino is plugged into a power source. If this light doesn't turn on, there's a good chance something is wrong.

TX is short for transmit, RX is short for receive. These markings appear quite a bit in electronics to indicate the pins responsible for serial communication. In our case, there are two places on the Arduino UNO where TX and RX appear - once by digital pins 0 and 1, and a second time next to the TX and RX indicator LEDs (12). These LEDs will give some nice visual indications whenever the Arduino is receiving or transmitting data (like when a new program is being uploaded onto the board).

The black thing with all the metal legs is an IC, or Integrated Circuit (13). It is the brains of the Arduino. The main IC on the Arduino is slightly different from board type to board type, but is usually from the ATmega line of IC's from the ATMEL company.

The voltage regulator (14) does exactly what it says - it controls the amount of voltage that is let into the Arduino board. It is kind of a gatekeeper; it will turn away an extra voltage that might harm the circuit. Unfortunately, it has its limits, so it is not recommended to hook up the Arduino to anything greater than 20 volts.

There are lots of different models and modification of circuits from Arduino family. The most popular and common of them are Arduino Uno, Arduino Mega and Arduino Leonardo. There are a lot of different custom variations of Arduino board, but most of them are proprietary, which means they do not have such a huge community of engineers around them, as the three mentioned boards.

Moreover, some additional supplements and addons for Arduino boards can be found on the market. From different sensors, which can measure light, temperature, degree of flex, pressure, proximity, acceleration, carbon monoxide, radioactivity, humidity, barometric pressure, up to shields, which are basically pre-build circuit boards that fit on top of the Arduino board and provide some additional functions, like controlling motors, connecting to the internet, providing cellular or other wireless communication, controlling an LCD screen.

It is safe to say that the motherboard is one of the most crucial parts of a 3D printer, as it will be responsible for giving orders to different controlled elements, gathering info from different sensors, like extruders' and heating platform's temperature sensors and passing it to the user-interaction unit. So, one should think wisely when choosing a controller board for a 3D printer.

2.1.2. User-interaction unit

Arduino is very powerful tool and in general it can be used for direct user interactions via different control buttons and LCD screens. However, as the examples from the real-world show, it is not possible to provide a simple, efficient and robust

UI, using only the motherboard. So, a good decision here might be to leave the work with controlled elements and sensors to the controller board and pick some other device for interactions with user.

The selected device should be able to: work with external displays with touchscreen, render a complex, modern and robust UI, process input from user via the touchscreen, connect to internet via either Wi-Fi or an Ethernet cable to communicate with remote server for fetching and uploading data and, most importantly, it should support connection to the Arduino board to be able to send user's input in form of commands.

It seems like, some kind of microcomputer would be the most appropriate fit this case. Because, they are usually much cheaper than full-size computers and laptops, they provide all of the requested functionality and moreover, they have really small physical sizes, which makes them a good solution for a relatively small 3D printer. However, it is worth noticing that such kind of devices are much less performant than their desktop alternatives, which comes with much smaller sizes and some of the features, like Wi-Fi or Bluetooth modules, or Ethernet connectors might be missing out-of-the-box, but could be bought as an add-on in form of a dongle.

Currently, ones of the most popular, advanced and well-supported microcomputers on the market are the devices from Raspberry Pi family. The Raspberry Pi Foundation was originated by David Braben, Jack Lang, Pete Lomas, Alan Mycroft, Robert Mullins and Eben Upton.

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse [28]. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It is capable of doing everything you would expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

What is more, the Raspberry Pi has the ability to interact with the outside world, and has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting birdhouses with infra-red cameras.

We want to see the Raspberry Pi being used by kids all over the world to learn to program and understand how computers work.

There have been three generations of Raspberry Pis: Pi 1, Pi 2, Pi 3, and there has generally been a Model A and a Model B of most generations. Model A is a cheaper variant and tends to have reduced RAM and ports like USB and Ethernet. The Pi Zero is a spinoff of the original (Pi 1) generation, made even smaller and cheaper.

The Raspberry Pi operates in the open source ecosystem: it runs Linux (a variety of distributions), and its main supported operating system, Raspbian, is open source and runs a suite of open source software. The Raspberry Pi Foundation contributes to the Linux kernel and various other open source projects as well as releasing much of its own software as open source

The Raspberry Pi's schematics are released, but the board itself is not open hardware. The Raspberry Pi Foundation relies on income from the sale of Raspberry Pis to do its charitable work.

The Raspberry Pi hardware has evolved through several versions that feature variations in the type of the central processing unit, amount of memory capacity, networking support, and peripheral-device support.

The Raspberry Pi Foundation provides Raspberry Pi OS (formerly called Raspbian), a Debian-based (32-bit) Linux distribution for download, as well as third-party Ubuntu, Windows 10 IoT Core, RISC OS, and LibreELEC (specialized media center distribution) [28]. It promotes Python and Scratch as the main programming languages, with support for many other languages. The default firmware is closed source, while unofficial open source is available. Many other operating systems can also run on the Raspberry Pi. Third-party operating systems available via the official website include Ubuntu MATE, Windows 10 IoT Core, RISC OS and specialized distributions for the Kodi media center and classroom management. The formally verified microkernel seL4 is also supported [39].

At the time of the conduction of the research, there are two most popular models of Raspberry Pi on the market, which might fit as a user-interaction unit for 3D printer:

The model with the highest specification is the Raspberry Pi 3 Model B (Fig. 2.2.), so for many general-purpose projects this is the best bet. It is the most powerful Pi, with the fastest clock speed, the most RAM, and best all-round feature set. It provides speed and power as well as some additional benefits in form of built-in Wi-Fi and Bluetooth modules.



Fig. 2.2. Raspberry Pi 3 Model B

The Pi 3 gives a genuinely pleasant desktop PC experience, in no small part thanks to four years of extreme work in optimizing the official Pi operating system, Raspbian. The Pi 3 boots in a matter of seconds, the web browser flies, you can open Minecraft and create a world in no time at all, and intensive applications like LibreOffice and Mathematica respond as they should on a decent PC.

However, as basically, anything in the world it has some cons and pros. The pros are: fast, powerful, excellent value for money, while the main con is a decent amount of consumed power.

Another popular model is Pi Zero (Fig. 2.3.). The Pi Zero is the smallest, lightest, cheapest Pi available.

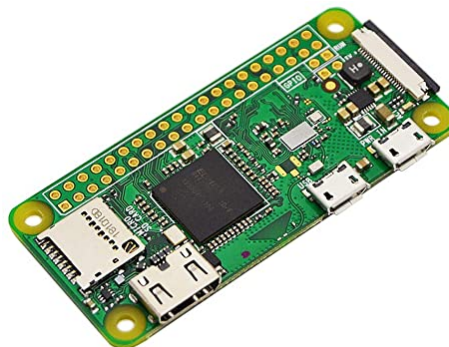


Fig. 2.3. Raspberry Pi Zero

It's not available in bulk, so it's not ideal if lots of them are needed, but it's perfect for embedded projects that don't rely on wireless connectivity, though it is always possible to add a Wi-Fi or Bluetooth dongle.

The Pi Zero's CPU is the Pi 1's BCM2835 overclocked to 1GHz, so it's even faster than a Pi 2 (though only single-core). It also packs 512MB RAM, giving it a surprisingly reasonable desktop experience. In terms of power usage, it's one of the lowest, around the same as a Model A+.

Now that the Pi Zero has a camera interface, it's perfect for projects like high-altitude ballooning, where size and weight really count.

The Pi Zero is not ideal for use as a general-purpose PC, as you need adapters to convert from mini-HDMI and micro-USB, plus a USB hub, although this could still work out cheaper than a full-sized Pi 3 [39].

Its pros are price, size, weight while the cons are limited availability, no wireless connectivity, GPIO header unpopulated.

So, it looks, like, with some additional efforts any of those models might be selected as a user-interaction unit for a 3D printer, because both of them can: be used with an external touchscreen display (via HDMI or mini-HDMI), connect to the Internet via Wi-Fi (though an additional dongle is needed for Pi Zero), connect to Arduino via a USB connection. Most importantly, both of them can run on the full desktop version of a Linux distributive, which would allow us to render a complex UI and provide robust and stable work of the system.

2.2. Controlled elements of 3D printer

Apart from a controlling part of a 3D printer, there are some controlled pieces of hardware, which are responsible for different phases of printing. They should be chosen wisely, as their quality and reliability are crucial, when printing some complex detail or figure.

2.2.1. Extruder

The 3D extruder is the part of the 3D printer that ejects material in liquid or semi-liquid form in order to deposit it in successive layers within the 3D printing volume. In some cases, the extruder serves only to deposit a bonding agent used to solidify a material that is originally in powder form.

Found in 3D Fused Deposition Modeling (FDM) or Fused Filament Fabrication (FFF) printers, the extruder is also required for proper operation of machines using Binder Jetting or Polyjet technologies, and even 3D Systems' CPX machines. These are additive manufacturing machines that need to deposit material before transforming it either by adding a bonding agent to it (Binder Jetting) or by changing the chemical properties (Polyjet and CPX). These technologies are explained in our guide about the different kinds of 3D printing.

The filament extruder on a FDM printer is the part that extrudes the plastic filament in a liquid form and deposits it on a printing platform by adding successive layers. The printing head is made of many distinct parts including a motor to drive the plastic filament and a nozzle (or extruder) to extrude the plastic.

Some 3D FDM / FFF printers are now equipped with two extruders. This enables you, in particular, to print two materials simultaneously in order to obtain 3D prints in two colors. The presence of two extruders also allows support material to be extruded, which can be removed afterward using a solvent.

To regulate the plastic cooling process, some printers are enclosed. This helps maintain a uniform temperature in the manufacturing chamber, ensuring greater consistency in the print result.

The most common kinds of Binder Jetting printers are probably the ProJet printers from 3D Systems. These printers have an extruder that projects a bonding agent (or color) onto a powder material. It's the action of projecting this bonding agent onto successive layers of powder that creates the object.

Polyjet technology, originally developed by Objet (which is now owned by Stratasys), is also based on the projection of resin in the form of droplets onto the printing platform. Once the droplets are projected, UV polymerizes the resin.

The cold end is the cold part in the upper portion of the 3D printer extruder. At this point, there is no heating of the filament. This is just the part with the motor and gearing, pushing the 3D printer filament into the hot end. Different systems actually exist, there is usually a combination of gears and hobbled bolts, dictating the movement of the printing filament

The hot end is the part where the filament is transitioning from solid to liquid, while extruded on the building plate. But how is the filament melting? Indeed, something has to be hot enough to melt materials and as we want to print an accurate part, the temperature between the cold filament, the hot end, and the final cold and solid part has to be perfectly managed. The heat break, in combination with the heat sink, maintains a boundary at which the filament is confronted with high temperatures. There is, in the system, a heater cartridge that is getting hot, transferring heat to the nozzle via the heater block in aluminum.

Most desktop 3D printers ship with 0.4mm nozzles as standard, but there are many other sizes available. Brass is usually used for 3D printer nozzles, but there are also several options. For some materials, stainless steel can be preferred.

There is not a single extruder type, the choice will depend on the kind of 3D printer that you have, on the used materials, and on the required printing speed and accuracy.

There are two different possibilities: Direct or Bowden extruders. The nature of your projects will determine which extruder you need to use. First, all extruders have motors, but there are also geared extruders to control your print speed. It is not essential, but it can help you to customize your setups in order to improve your print quality.

Direct extruders are directly attached to the hot end, while a Bowden extruder (or remote extruder) has a tube to link the hot end and the extruder body. For direct extruders, the gear rotates by a stepper motor driving directly the filament to the

extruder hot end. The filament path is shorter, that is why Direct extruders are better to 3D print flexible materials than Bowden extruders. You can totally 3D print flexible filament with a Direct extruder, but it is not really convenient.

With Bowden extruders you can 3D print your projects faster. It is easier to accelerate or decelerate because there is just the printing head to move, not the whole extruder hot end.

They both can have a direct drive system. Indeed, for both of them the filament drive mechanism can be mounted to the motor shaft, directly.

Moreover, there are extruders for all filament thickness, for 3mm filament, 1.75mm filament, etc.

Differences between Bowden and direct extruders can be seen on Fig. 2.4.

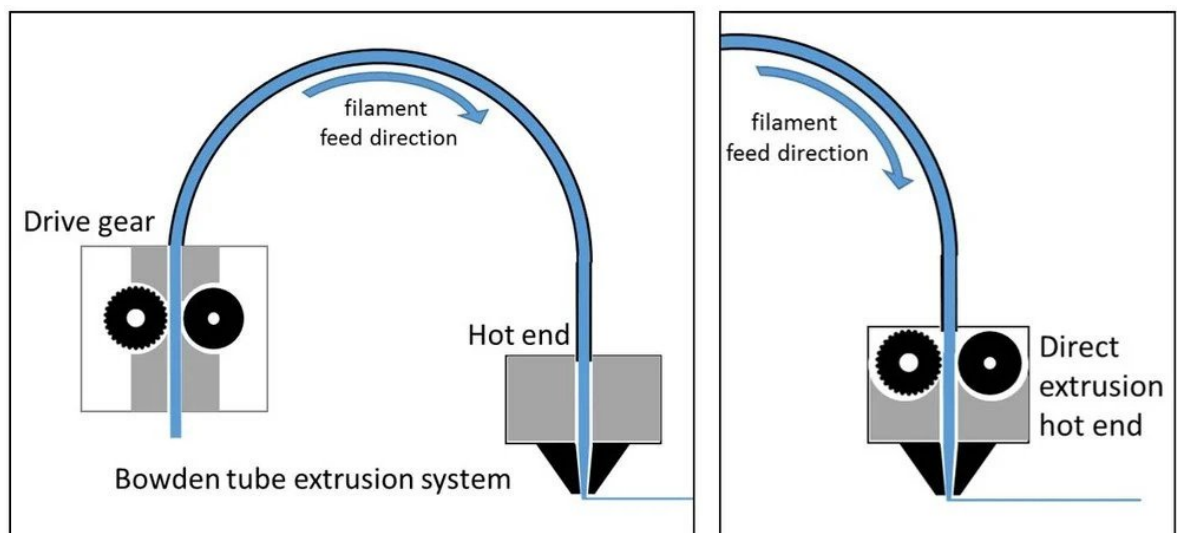


Fig. 2.4. Bowden extruder vs direct extruder

Overall, it looks, like, when selecting a proper extruder for a 3D printer the following moments should be taken into consideration:

- type of the extruder (Bowden or direct);
- material which is expected to be used for this 3D printer;
- the number of extruders

2.2.2. Motion controllers

Motion control systems and motors are invaluable for the growing 3D printing industry. As the industry grows across services and sectors, the precision, accuracy, and speed of the printers are becoming increasingly important.

Motion control systems coordinate and regulate the many moving parts in a 3D printer. They serve to accurately synchronize multiple axes, increase the precision of printing mechanisms, reduce noise, and increase print speed. Motors assist gantries to achieve the necessary positioning and movement.

Motion control systems also provide the precision and accuracy required to make unprecedented progress in 3D printing and other fields. Bioprinting pioneers that are manufacturing living organs, tissue, bones, and cartilage are researching 3D printing as a viable option.

The following motion control systems are used with different end goals in mind:

- The Advanced Motion Controls (AMC) Click & Move motion control system can run several different 3D printers at once. The Click & Move coordinates all the different axes with different motor types, enabling efficient, precise, and effective 3D printing. Furthermore, Click & Move can coordinate systems with multiple print nozzles that are used to make single parts with distinctive materials or colors. The motion control system is modular and scalable, providing ease for 3D printer manufacturers to keep using the system as their product line evolves;
- The PI (Physik Instrumente) ACS-based motion controller with EtherCat connectivity runs the gantry and communicates with the high precision dispenser. The motion controller brings the medical field one step closer to fabricating human organs layer-by-layer. The PI motion controller allows 3D printing systems to be constructed in a few days or less. They are designed to provide a high level of performance and to be compatible with STL files.

A stepper motor (Fig. 2.5.) is a type of electric motor that can be accurately controlled with the controller. Most 3D printer use four or five stepper motors. Three

or four motors control the x/y/z axis movement (sometimes the z axis is controlled by two motors) and one motor is used per extruder.



Fig. 2.5. Stepper motor

A stepper driver (Fig. 2.6.) is a chip that acts as a kind of middle-man between a stepper motor and the controller. It simplifies the signals that need to be sent to the stepper motor in order to get it to move.

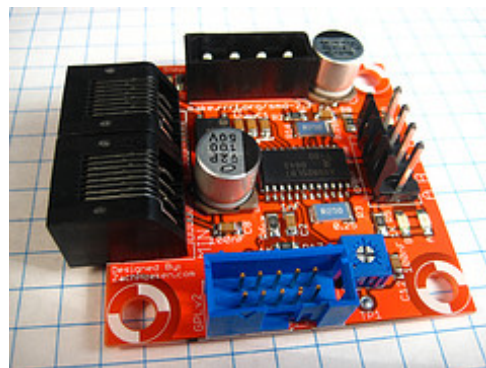


Fig. 2.6. RepRap Stepper Motor Driver v2.x

Sometimes the stepper drivers are on separate circuit boards that are linked to the controller via cables.

Sometimes the stepper drivers are on small circuit boards that plug directly into the controller itself. In this case, the controller will have space for at least 4 of these small circuit boards (one for each stepper motor).

Finally, sometimes the stepper drivers are soldered right onto the controller itself.

Both the horizontal movement of the nozzle gantry and the vertical movement of the build plate need to move and be positioned precisely. The material comes out in small increments, but overlapping or diverging too much within the same layer can make the print very messy. The printer nozzle needs to move very carefully in order to draw out the shape of each layer, which can consist of completely separate parts for some layers of certain shapes. There is very little room for error. And if the nozzle is too close or too far away from the build plate, the filament will not stick correctly.

All of the movement and positioning of the build plate are usually accomplished with some form of linear actuator. The nozzle gantry, on the other hand, usually uses linear motors or stepper motors with belts to achieve the necessary positioning and movement. And where there is a stepper motor or linear motor, there must be some sort of drive, often a servo drive.

Another critical motion control component is in the extruder, the feeding mechanism that pushes and pulls filament through the nozzle. Material isn't always flowing through the nozzle. It needs to stop when the nozzle is moving over a gap in the layer and while the build plate is lowering. The nozzle flow also needs to be regulated when the nozzle moves through different geometries. For example, if the nozzle keeps extruding material at a continuous speed when it makes a sharp turn, you can end up with corner swell. The material flow rate needs to be lowered during the corner to get an even thickness with no swell.

When choosing the most appropriate type of motion controllers, we should consider, if the printer should support multiple nozzles and what precision of the printing is required.

2.2.3. Heating platform

A heated build platform (Fig. 2.7.), also known as a heated bed, on a 3D printer is where the part is printed out, layer by layer. By having the build platform heated, there will be less warping and curling of the plastic by evenly distributing the cooling process for a part

A common type of heated bed uses circuitry boards (PCBs) as heating elements. While such type of printers is cheaper to buy, they are not specifically designed to work as heating elements for the complex process of 3D printing. One disadvantage of using such types of heated beds is that, since the PCBs are made of aluminum and copper strips, they can deform over time after continuously being subjected to heat.



Fig. 2.7. Heating platform

ABS and PLA plastic can sometimes shrink and curl up at the edges as the part cools. This causes a problem as more layers are added, which causes the part to warp. To solve this problem, heated build platforms are used to keep the bottom layers of the part warm as layers are added, which helps the part to cool evenly. The heated bed is insulated so that the parts do not soften or melt. Heated bed material includes glass, metal, or ceramic. Heated build platforms should be about 120° Celsius for ABS plastic and 55° Celsius for PLA plastic.

When extruded plastic is released from the printer nozzle, it begins to cool. During that cooling process, it also shrinks in size, and it is during shrinking that the part may become uneven and warped, since the cooling may occur at different rates at different points on the part's surface.

The heated bed ensures that the printed part stays warm all over during the printing process to allow for more even shrinking once it begins to cool below its melting point. All in all, the heated bed fulfills two tasks:

- It increases the surface energy of the print bed. This improves the bonding strength at the top layer;
- It keeps the bottom part hot enough to eliminate the risk of warping for the rest of the print. The bed carries out a delicate balancing act of cooling the plastic without over-cooling it.

The extruder part of the printer deposits molten plastic into the receiver bed while supplying a certain degree of heat. The temperature of the heat bed needs to be below the glass point to ensure the print cools into a solid. A lot depends on the temperature sensor of the heat bed to get the required heat that needs to be supplied just right.

Regardless of the materials used, the heated bed should be thermally insulated so that it doesn't melt or soften any plastic parts underneath the bed. Commonly used insulator materials are cardboard, wool, and cotton cloth on top of medium density fiberboard MDF. The heated bed can also be mounted directly to a wooden platform without any noticeable negative effects. Wool may be a good option for insulation because its ignition temperature is 600C.

It is safe to conclude, that if we want to achieve the highest quality of printed goods and avoid having a problem with them, being glued to the platform, when using some materials, like ABS and PLA, it is required to use a heating platform.

2.2.4. Touchscreen

The last, but not least is a touch screen. This is the part of the print which will be one of the most used by the customer. So, it should be reliable, responsive, it should be able to properly display the UI and, most importantly, process user's input and send it to the user-interaction unit.

As Raspberry Pi was selected as a user-interaction unit for this project, the touchscreen should be supported by the selected version of Raspberry Pi (either Raspberry Pi 3 Model B or Raspberry Pi Zero).

There are lots of different models of touch screens in the market right now, but still the most reliable and popular solution is the official one from Raspberry Pi, called Raspberry Pi Touch Display (Fig. 2.8.).

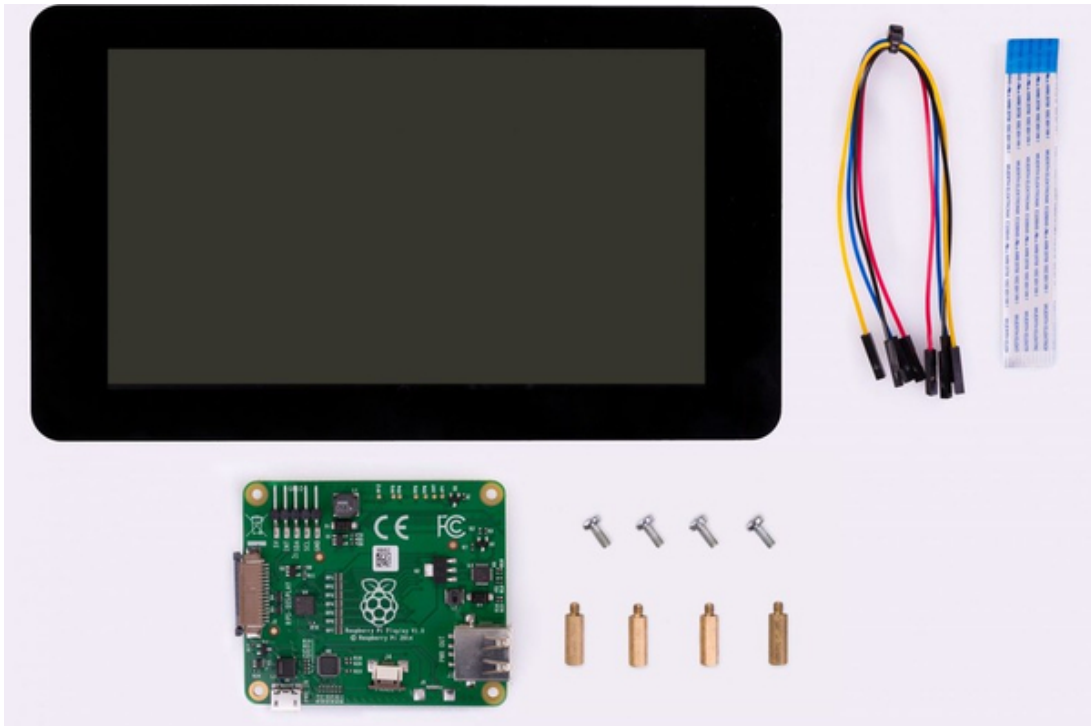


Fig. 2.8. Raspberry Pi Touch Display

The 800 x 480 display connects via an adapter board which handles power and signal conversion. Only two connections to the Pi are required; power from the Pi's GPIO port and a ribbon cable that connects to the DSI port present on all Raspberry Pis (except Raspberry Pi Zero and Zero W). Touchscreen drivers with support for 10-finger touch and an on-screen keyboard will be integrated into the latest Raspberry Pi OS for full functionality without a physical keyboard or mouse.

2.3. Defining the parameters of the 3D printer

In the previous chapters, the most crucial hardware elements of a 3D printer were described. It looks, like for the 3D printer we will need: a controller unit, a user-interaction unit, extruder(s), motion controllers (motion steppers and drivers), heating bed and a touchscreen.

Before choosing the most appropriate hardware, some basic requirements to the 3D printer should be formulated:

- the sizes of the printer;
- number of extruders;
- supported materials;
- accuracy;
- filament thickness;
- speed;
- nozzle diameter;
- presence of a heating bed;
- required sensors.

For this particular project, the following parameters were chosen as optimal (table 2.1.):

Table 2.1

3D printer parameters

Parameter	Requirements
Size of the printer	400x300x400
Number of extruders	1
Supported materials	ABS, ABS+, PLA, HIPS
Accuracy	For a household printer, the recommended level of accuracy is 50microns

Continuation of Table 2.1

Filament thickness	The most common thickness for household printers is 1,75mm
Speed	Recommended speed for a household printer is 50–200mm/s
Nozzle diameter	In general, the nozzle diameter of 0.4mm is enough to provide the chosen accuracy
Presence of a heating bed	As ABS, ABS+ and PLA are listed as supported materials, it is recommended to add a heating platform
Required sensors	As only 1 extruder is used and a heating platform is requested, then we will need at least 2 temperature sensors. Also, as we will have moving steppers for three axes (X, Y and Z), six end-stops are required in order to prevent the moving parts from collapsing into the sides of the printer.

Now, as the strict set of requirements is defined, it is possible to prepare a suggestion on architecture design of hardware part of a 3D printer. In order to design the most optimal, reliable and robust architecture, the most appropriate parts of the system should be picked.

While some parts, like extruder, motion controllers, heating platform and touch screen are pretty standardized, so, basically, any of the currently available offers on the market can be used, there are two crucial parts, which are the “brains” of the printer should be considered and estimated in order to come up with the most optimal solution.

2.4. Hardware architecture design using method of multicriteria weighted estimations

In this work we will use the method of multicriteria weighted estimations to analyze alternatives of controller unit and user-interaction unit and select the most appropriate one.

This method is used for choosing of the most optimal variant in cases, when the choice is made, based on quality criteria. Moreover, it is used for simplification of calculations, when quantity criteria are used.

Quantity criteria give the estimation in quantity indications, like size, square, width, etc.

Meanwhile, quality criteria give the estimation in quality indications, like efficiency, reliability, speed, etc.

Method of multicriteria weighted estimations uses the following algorithm:

1. formulation of choosing criteria;
2. picking weights of criteria;
3. choice of a rating scale;
4. ranking of options according to the selected criteria;
5. calculation of a weighted score for each option for each criterion;
6. calculation of the final weighted score for each option;
7. choosing the best option.

The option with the highest final weighted score is selected, if the calculations use positive criteria, otherwise – the option with the lowest score is selected.

Advantages of this method are:

- the simplicity of calculations;
- it is possible to use this method in cases, when retrieving of quantity criteria is limited or not possible at all;
- the result of calculations might be a choice of the most optimal variant as well ranking according to the degree of attractiveness of all investigated options;

However, this method also has its disadvantages, the biggest one is not the highest accuracy, which is related to the fact, that expert scores, which are used for choosing of criteria as well for defining their weights and estimating by the given criteria are subjective.

In order to select the most appropriate device for both controller board, the following criteria were selected: price, performance, reliability, extensibility.

As follows from the section 2.1.1., we have two most popular options for the motherboard: Arduino Uno and Arduino Mega. They were giving the following weight scores for each criterion:

Table 2.2

Motherboards criterion weight scores

Device / Criterion	Price	Performance	Reliability	Extensibility	Sum $\sum_{i=1}^m r_i$
Arduino Uno	5	2	4	3	14
Arduino Mega	3	5	4	5	17
Scores sum r_i	8	7	8	8	31

The weight of each score can be calculated by the following formula:

$$\sigma_i = \frac{r_i}{\sum_i^m r_i} \quad (2.1)$$

Therefore, the weights for each criterion are following:

- Price: $8 / 31 = 0.26$;
- Performance: $7 / 31 = 0.22$;
- Reliability: $8 / 31 = 0.26$;
- Extensibility: $8 / 31 = 0.26$

Weighted coefficients for each criterion:

Table 2.3

Weighted coefficients for each criterion

Criterion	Criterion weight	Score of the criterion on a 10-point scale	
		Arduino Uno	Arduino mega
Price	0.26	8	5
Performance	0.22	6	9
Reliability	0.26	8	8
Extensibility	0.26	5	9

The weighted score for each option for each criterion can be calculated, using the following formula

$$K_{ij \text{ weighted}} = K_{ij} * \sigma_i \quad (2.2)$$

The calculation of the final score for each option is done, using this formula:

$$S_i = \sum_{j=i}^m K_{ij \text{ weighted}} \quad (2.3)$$

The results of final weighted estimation for each device are listed in table 2.4:

Table 2.4

Result scores for each device

Device / Criterion	Price	Performance	Reliability	Extensibility	Sum
Arduino Uno	2.08	1.32	2.08	1.3	6.78
Arduino Mega	1.3	1.98	2.08	2.34	7.7

Based on the results of the performed calculations, Arduino Mega seems to be a slightly better choice. Even though it has much higher price, than Arduino Uno, it comes with better performance and extensibility, which compensate the price difference. However, the difference of final scores is not very high, which means, it is still possible to select Arduino Uno if the main goal is to create the cheapest possible solution.

The same method can be applied for analysis of available options for user-interaction control unit. As of now, two of the most popular Raspberry Pi versions are: Raspberry Pi 3 Model B and Raspberry Pi Zero. They will be compared, using the same criteria as for motherboard: price, performance, reliability, extensibility.

The user-interaction units criterion weight scores are shown in table 2.5.

Table 2.5

User-interaction units criterion weight scores

Device / Criterion	Price	Performance	Reliability	Extensibility	Sum $\sum_{i=1}^m r_i$
Raspberry Pi 3 Model B	2	4	5	5	16
Raspberry Pi Zero	5	2	3	2	12
Scores sum r_i	7	6	8	7	28

Using formula (2.1), we can get the weights for each criterion are following:

- Price: $7 / 28 = 0,25$;
- Performance: $6 / 28 = 0,21$;
- Reliability: $8 / 28 = 0,29$;
- Extensibility: $7 / 28 = 0,25$.

Weighted coefficients for each criterion:

Table 2.6

Weighted coefficients for each criterion

Criterion	Criterion weight	Score of the criterion on a 10-point scale	
		Raspberry Pi 3 Model B	Raspberry Pi Zero
Price	0.25	5	8
Performance	0.21	7	4
Reliability	0.29	9	5
Extensibility	0.25	9	4

Using formulas (2.2) and (2.3) we can get the results of final weighted estimation for each device:

Table 2.7

Result scores for each device

Device / Criterion	Price	Performance	Reliability	Extensibility	Sum
Raspberry Pi 3 Model B	1.25	1.47	2.61	2.25	7.58
Raspberry Pi Zero	2	0.84	1.45	1	5.29

As we can see from the results table, even though Raspberry Pi 3 Model B is more expensive than Raspberry Pi Zero, it shows higher performance, reliability and extensibility, which completely overweight the price difference. The difference in final scores between models is rather noticeable, so it looks like Raspberry Pi Zero cannot be recommended as a user-interaction unit for a 3D printer.

2.5. Conclusions for section 2

In the second part of the work, the most crucial elements of 3D printer hardware part were described and analyzed. Such elements are: controller board, user-interaction unit, extruder, motion controllers, heating platform and LCD touchscreen.

Then the requirements to the 3D printer were formulated in terms of the sizes of the printer, number of extruders, supported materials, accuracy, filament thickness, speed, nozzle diameter, presence of a heating bed and required sensors.

Some of the hardware components, like extruder, motion controllers, heating platform and LCD touchscreen are quite standard, so almost any available solution on the market would fit the give requirements.

However, in order to select and find some strong arguments in favor of the best, currently available solution on the market for motherboard and user-interaction unit, the method of multicriteria weighted estimations was used.

Using the most important criteria, which are price, performance, reliability and extensibility we were able to conduct, that the most appropriate motherboard for the desired 3D printer is Arduino Mega, while the best choice of a user-interaction unit would be Raspberry Pi 3 Model B.

In general, during this part we were able to formulate the requirements, suggest and prove the most appropriate architecture design for a 3D printer.

SECTION 3

SOFTWARE DESIGN

3.1. Critical software elements for 3D printing system

Now, when the hardware part of the system was defined it is required to come up with the requirements to the software part. The software is just as important as the hardware, because it gives the means for the hardware to work properly as well as it provides different user interfaces for all kinds of interactions and manipulations over the hardware.

The design of the software part should consider all user requirements (such as local and remote UIs), as well as handle all internal system communications between hardware and different parts of software.

It looks like, in order to satisfy the minimal requirements, it is necessary to introduce such pieces of software:

- Microcontroller firmware – a proxy between the printer’s hardware and software, which handles user’s interactions;
- Local UI – to provide the ability to work with the device directly;
- Remote UI – to provide the ability to with the device remotely;
- Slicer – to prepare 3D models to be printed via the printer;
- Cloud Storage – to store original and prepared 3D models and share them with other users;
- Proxy-server – to provide the communication between the 3D printer and Remote UI.

3.1.1. Microcontroller firmware

As stated in 2.1.1, Arduino motherboard was chosen as the primary microcontroller of the printer. It will be one of the most crucial parts of the system,

because it is responsible for passing commands from user to the different hardware parts of the system.

In order to do so, the specific software, called firmware should be installed on the Arduino board.

Firmware, like its name implies, is the bridge between the hardware and software of a computer system. When software sends commands to the hardware of a computer system, the firmware interprets and translates the software commands into a form that is recognizable by the hardware.

When the 3D printer software sends G-code to your 3D printer, the firmware translates the G-code commands into specific electrical signals that are sent to the motors, heaters, fans and other components on the 3D printer.

For example, if the host software sends “G1 X50 Y50” to the 3D printer, the 3D printer firmware determines how far the motors need to turn to move the extruder to $X = 50$ mm and $Y = 50$ mm, then sends the electrical signals to the motors to turn them the appropriate amount.

As this piece of software is highly complex, it seems to be a wise choice to select one of the available open-source solutions, with a few years of production usage and a big, active community.

Nowadays, one of the most popular and well-adopted firmware for 3D printers is Marlin. Marlin is an open-source firmware for the RepRap family of replicating rapid prototypes — popularly known as “3D printers.” It was derived from Sprinter and grbl, and became a standalone open-source project on August 12, 2011. Marlin is licensed under the GPLv3 and is free for all applications.

From the start Marlin was built by and for RepRap enthusiasts to be a straightforward, reliable, and adaptable printer driver that “just works.” As a testament to its quality, Marlin is used by several respected commercial 3D printers. Ultimaker, Printbot, AlephObjects (Lulzbot), and Prusa Research are just a few of the vendors who ship a variant of Marlin. Marlin is also capable of driving CNC’s and laser engravers.

One key to Marlin's popularity is that it runs on inexpensive 8-bit Atmel AVR micro-controllers - Marlin 2.x has added support for 32-bit boards. These chips are at the center of the popular open0source Arduino/Genuino platform. The reference platforms for Marlin are an Arduino Mega2560 with RAMPS 1.4 and Re-Arm with Ramps 1.4.

As a community product, Marlin aims to be adaptable to as many boards and configurations as possible. We want it to be configurable, customizable, extensible, and economical for hobbyists and vendors alike. A Marlin build can be very small, for use on a headless printer with only modest hardware. Features are enabled as-needed to adapt Marlin to added components.

Main features:

- Full-featured G-code with over 150 commands
- Complete G-code movement suite, including lines, arcs, and Bézier curves
- Smart motion system with lookahead, interrupt-based movement, linear acceleration
- Support for Cartesian, Delta, SCARA, and Core/H-Bot kinematics
- Closed-loop PID heater control with auto-tuning, thermal protection, safety cutoff
- Support for up to 5 extruders plus a heated printbed
- LCD Controller UI with more than 30 language translations
- Host-based and SD Card printing with autostart
- Bed Leveling Compensation — with or without a bed probe
- Linear Advance for pressure-based extrusion
- Support for Volumetric extrusion
- Support for mixing and multi-extruders (Cyclops, Chimera, Diamond)
- Support for Filament Runout/Width Sensors
- Print Job Timer and Print Counter

Marlin Firmware runs on the 3D printer's main board, managing all the real-time activities of the machine. It coordinates the heaters, steppers, sensors, lights, LCD display, buttons, and everything else involved in the 3D printing process.

Marlin implements an additive manufacturing process called Fused Deposition Modeling (FDM) – aka Fused Filament Fabrication (FFF). In this process a motor pushes plastic filament through a hot nozzle that melts and extrudes the material while the nozzle is moved under computer control. After several minutes (or many hours) of laying down thin layers of plastic, the result is a physical object.

The control-language for Marlin is a derivative of G-code. G-code commands tell a machine to do simple things like “set heater 1 to 180°,” or “move to XY at speed F.” To print a model with Marlin, it must be converted to G-code using a program called a “slicer.” Since every printer is different, you won't find G-code files for download; you will need to slice them yourself.

As Marlin receives movement commands it adds them to a movement queue to be executed in the order received. The “stepper interrupt” processes the queue, converting linear movements into precisely-timed electronic pulses to the stepper motors. Even at modest speeds Marlin needs to generate thousands of stepper pulses every second.

Heaters and sensors are managed in a second interrupt that executes at much slower speed, while the main loop handles command processing, updating the display, and controller events. For safety reasons, Marlin will actually reboot if the CPU gets too overloaded to read the sensors.

Marlin can be controlled entirely from a host or in standalone mode from an SD Card. Even without an LCD controller, a standalone SD print can still be initiated from a host, so your computer can be untethered from the printer.

Using Marlin as firmware for the 3D printer will give an opportunity to concentrate the efforts on providing the best possible UI and UX, without having to spent lots of time developing a custom firmware.

3.1.1.1. Firmware mathematical model

Firmware for a 3D printer is a complicated and multifunctional piece of software. It is responsible for all the underlying operations, such as communication with hardware parts of the system, as well as for processing commands from the client and sending the appropriate response.

The mathematical model of the commands processing workflow, is represented on Fig. 3.1., where:

X – set of input parameters (commands in form of signals, received from the client);

Y – set of output parameters (commands in form of signals, send to the hardware elements of the 3D printer);

Z – set of disturbing effects of the environment (different signal interferences, connection losses, etc.);

A – operator of a mathematical model that is a set of algorithms and functions.

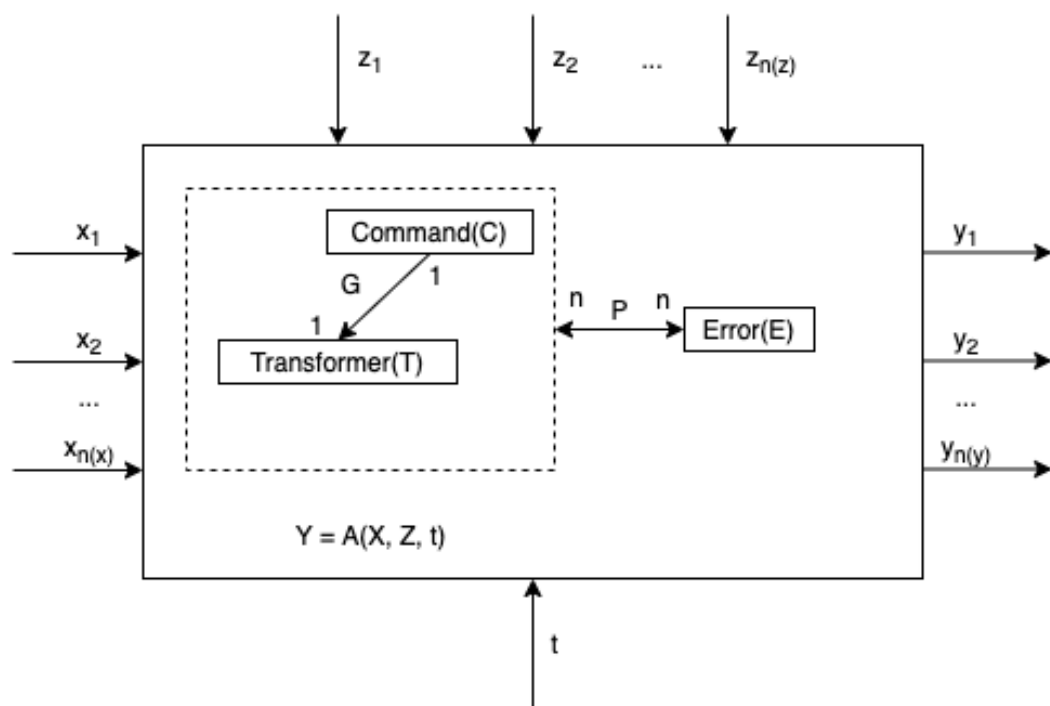


Fig. 3.1. Mathematical model of the command processing workflow

The main terms and definitions from the object-classification model and connections between them, are described further.

Command – a command in form of set of signals, received from the connected client (Raspberry Pi in our case). The set of commands in the firmware is:

$$C = \{c_1, c_2, \dots, c_{n(C)}\}, \quad (3.1)$$

where, $n(C) = |C|$ - the power of the set C , which corresponds to the number of commands, supported by the firmware.

Transformer – the part of the firmware's code, responsible for interpretation of the received commands, to the commands, which can be send to a particular piece of hardware (e.g., extruder, motion gears, etc.). The set of transformers:

$$T = \{t_1, t_2, \dots, t_{n(T)}\}, \quad (3.2)$$

where, $n(T) = |T|$ - the power of the set T , which corresponds to the number of transformations, supported by the firmware

Error – possible errors, which can occur in the firmware during the interpretation of the command. They might be caused by incorrect input parameters from the client, as well as by some disturbing environment events (signal interferences, connection losses, etc.). The set of errors:

$$E = \{e_1, e, \dots, e_{n(E)}\}, \quad (3.3)$$

where, $n(E) = |E|$ - the power of the set E , which corresponds to the number of errors, handled by the firmware.

The connections between classes from the object-classification model are:

G – relation between the received command and transformer, which should process it.

P – relation between the transformation and a set of errors, which might occur in the process.

3.1.2. Local user interface

Even though Marlin will handle all low-level interactions with the hardware, the user can't work with it directly. So, it is necessary to provide some proxy between the user and the firmware.

Usually, a local UI is developed for this purpose. It should consist of at least two parts:

- Backend – which will be responsible for communication between the frontend and the firmware;
- Frontend – which will be responsible for user's interactions and sending the commands to the backend.

The communication between backend, frontend and the firmware is shown in Fig. 3.1.:

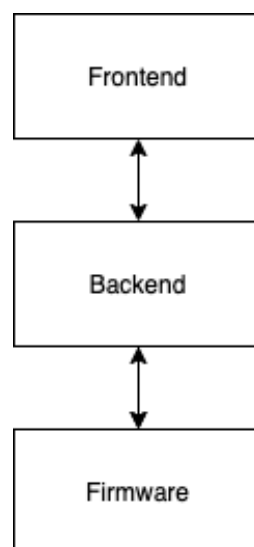


Fig. 3.2. Scheme of communication between Frontend, Backend and Firmware

As was stated in 2.1.2., Raspberry Pi was selected as a user-interaction unit. One of the biggest advantages of the boards from Raspberry foundation is the fact that they

can run a fully-functional distributive of Linux. Which gives unlimited possibilities to the developers.

It seems like, for implementation of the local UI the simplest approach would be to implement a backend, using some server-side technology, like Python, NodeJS, Go or C#, which is capable of communication with the Arduino's firmware. While the frontend part can be implemented as a Web application, which runs in some locally installed browser.

The communication between backend and hardware can be implemented in a straightforward way, as described in 3.1.1. In this way the backend sends different GCode commands to the firmware, and in response it receives some answers, which should be handled properly.

Meanwhile, the Frontend should be very responsive and provide the real-time data about the temperature, position of the extruder, the motion drivers, the platform, about the printing progress, etc.

There are multiple ways to achieve this kind of responsiveness. For example, one of the polling strategies.

The simplest way to get new information from the server is periodic polling. That is, regular requests to the server, which are executed, for example, once every 10 seconds.

In response, the server first takes a notice to itself that the client is online, and second – sends a packet of messages it got till that moment.

That works, but there are downsides:

- Messages are passed with a delay up to 10 seconds (between requests).
- Even if there are no messages, the server is bombed with requests every 10 seconds, even if the user switched somewhere else or is asleep. That is quite a load to handle, speaking performance-wise.

So, this approach seems to be inappropriate for a real-time system, which requires high performance, efficiency and reliability.

Long polling is essentially a more efficient form of the original polling technique. Making repeated requests to a server wastes resources, as each new incoming

connection must be established, the HTTP headers must be parsed, a query for new data must be performed, and a response (usually with no new data to offer) must be generated and delivered. The connection must then be closed, and any resources cleaned up. Rather than having to repeat this process multiple times for every client until new data for a given client becomes available, long polling is a technique where the server elects to hold a client's connection open for as long as possible, delivering a response only after data becomes available or a timeout threshold has been reached.

Implementation is mostly a server-side concern. On the client side, only a single request to the server needs to be managed. When the response is received, the client can initiate a new request, repeating this process as many times as is necessary. The only difference to basic polling, as far as the client is concerned, is that a client performing basic polling may deliberately leave a small-time window between each request so as to reduce its load on the server, and it may respond to timeouts with different assumptions than it would for a server that does not support long polling. With long polling, the client may be configured to allow for a longer timeout period (via a Keep-Alive header) when listening for a response – something that would usually be avoided seeing as the timeout period is generally used to indicate problems communicating with the server.

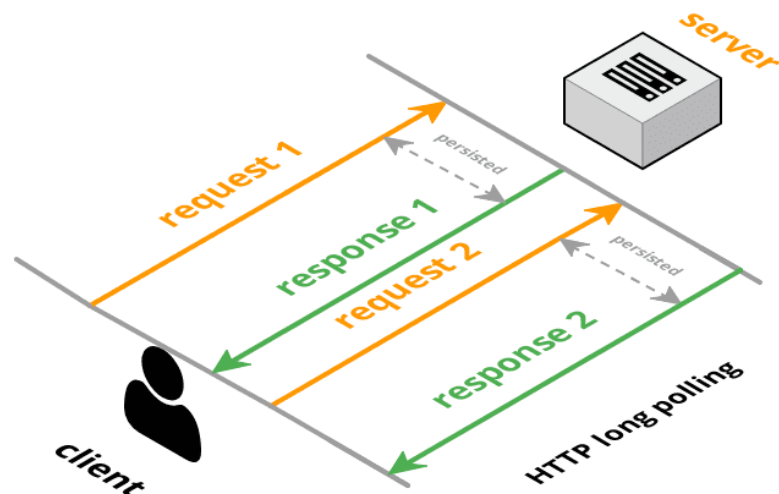


Fig. 3.3. Long polling strategy

Apart from these concerns, there is little else that a client needs to do that would be different than if it were engaging in basic polling. By contrast, the server needs to manage the unresolved state of multiple connections, and it may need to implement strategies for preserving session state when multiple servers and load balancers are in use (commonly referred to as session “stickiness”). It also needs to gracefully handle connection timeout issues.

As long polling is really just an improvisation applied to an underlying request-response mechanism, it comes with an additional degree of complexity in its implementation.

Reliable message ordering can be an issue with long polling because it is possible for multiple HTTP requests from the same client to be in flight simultaneously. For example, if a client has two browser tabs open consuming the same server resource, and the client-side application is persisting data to a local store such as `localStorage` or `IndexedDb`, there is no in-built guarantee that duplicate data won't be written more than once. This could also happen if the client implementation uses more than one connection at a time, whether deliberately or as a result of a bug in the code.

Another issue is that a server may send a response, but network or browser issues may prevent the message from being successfully received. Unless some sort of message receipt confirmation process is implemented, a subsequent call to the server may result in missed messages.

Depending on the server implementation, confirmation of message receipt by one client instance may also cause another client instance to never receive an expected message at all, as the server could mistakenly believe that the client has already received the data it is expecting.

All of these concerns, and more need to be considered when implementing robust support for long polling in any real-time messaging system.

Unfortunately, such complexity is difficult to scale effectively. To maintain the session state for a given client, that state must either be shareable among all servers behind a load balancer – a task with significant architectural complexity – or

subsequent client requests within the same session must be routed to the same server to which their original request was processed. This form of deterministic “sticky” routing is problematic by design, especially when routing is performed on the basis of IP address, as it can place undue load on a single server in a cluster while leaving other servers mostly idle instead of spreading the load around efficiently. This can also become a potential denial-of-service attack vector – a problem which then requires further layers of infrastructure to mitigate that might otherwise have been unnecessary.

So, it is safe to say that neither regular nor long polling are a good choice for a complex real-time system because of their overall limitations and complexity.

An alternative solution might be to use a modern way for handling communication between server and client in real-time systems – WebSockets.

A WebSocket is a persistent connection between a client and server. WebSockets provide a bidirectional, full-duplex communications channel that operates over HTTP through a single TCP/IP socket connection. At its core, the WebSocket protocol facilitates message passing between a client and server.

The idea of WebSockets was borne out of the limitations of HTTP-based technology. With HTTP, a client requests a resource, and the server responds with the requested data. HTTP is a strictly unidirectional protocol — any data sent from the server to the client must be first requested by the client. Long-polling has traditionally acted as a workaround for this limitation. With long-polling, a client makes an HTTP request with a long timeout period, and the server uses that long timeout to push data to the client. Long-polling works, but comes with a drawback — resources on the server are tied up throughout the length of the long-poll, even when no data is available to send.

WebSockets, on the other hand, allow for sending message-based data, similar to UDP, but with the reliability of TCP. WebSocket uses HTTP as the initial transport mechanism, but keeps the TCP connection alive after the HTTP response is received so that it can be used for sending messages between client and server. WebSockets allow us to build real-time applications without the use of long-polling.

The protocol consists of an opening handshake followed by basic message framing, layered over TCP. WebSockets begin life as a standard HTTP request and response. Within that request response chain, the client asks to open a WebSocket connection, and the server responds (if it is able to). If this initial handshake is successful, the client and server have agreed to use the existing TCP/IP connection that was established for the HTTP request as a WebSocket connection. Data can now flow over this connection using a basic framed message protocol. Once both parties acknowledge that the WebSocket connection should be closed, the TCP connection is torn down.

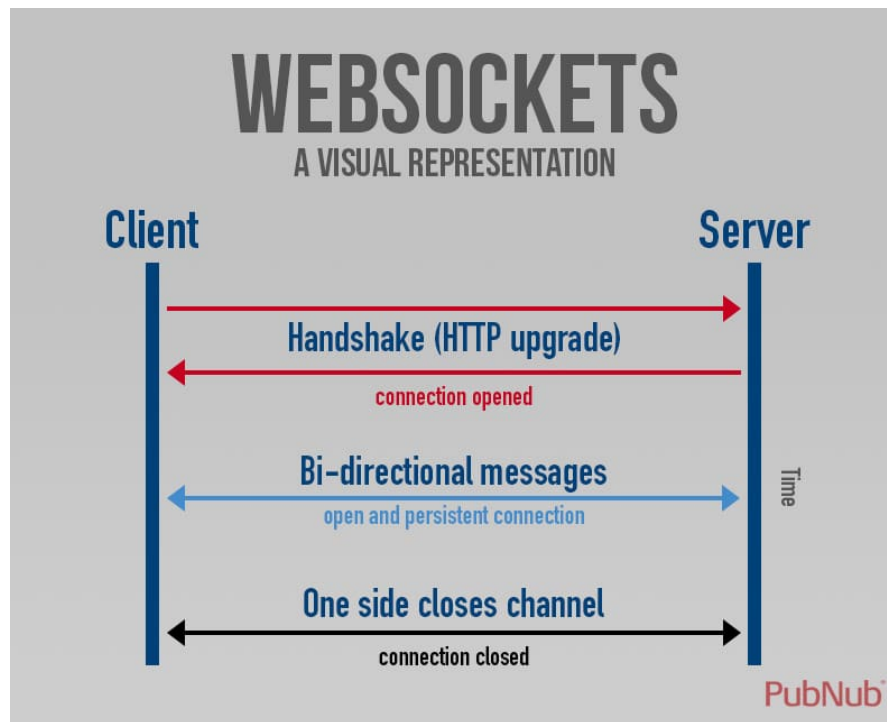


Fig. 3.4. A visual representation of WebSockets

It looks like, in order to provide a robust real-time system for a 3D printer, the WebSocket technology is the best choice.

Now, as the decision on architecture of the Local UI was made, it is necessary to choose a list of technologies, which should be used for the development.

As the Local UI will consist of two parts – Frontend and Backend, we should choose the technologies for both of them.

Frontend is going to be a Web application, so there is not too much of options if we are talking about the programming language. There are a few different languages for Web development around, however the most spread, popular and well-known one is JavaScript. JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.JS, Apache CouchDB and Adobe Acrobat. JavaScript is a prototype-based, multi-paradigm, single-threaded, dynamic language, supporting object-oriented, imperative, and declarative (e.g., functional programming) styles.

JavaScript runs on the client side of the web, which can be used to design / program how the web pages behave on the occurrence of an event. JavaScript is an easy to learn and also powerful scripting language, widely used for controlling web page behavior.

JavaScript can function as both a procedural and an object-oriented language. Objects are created programmatically in JavaScript, by attaching methods and properties to otherwise empty objects at run time, as opposed to the syntactic class definitions common in compiled languages like C++ and Java. Once an object has been constructed it can be used as a blueprint (or prototype) for creating similar objects.

JavaScript's dynamic capabilities include runtime object construction, variable parameter lists, function variables, dynamic script creation, object introspection, and source code recovery (JavaScript programs can decompile function bodies back into their source text).

Although, JS is a really powerful programming language, currently, one of its biggest benefits and struggles at the same time is the lack of a strict types system. This is good for beginners and small solution, like MVPs or quick prototypes, however this approach does not scale really well for big, complex, sometimes even multipart solutions.

In such cases the most appropriate solution would be to select one of the available supersets of JavaScript, which provide a strict compile-time types system. Nowadays, the most popular one is Typescript. TypeScript is an open-source language

which builds on JavaScript, one of the world's most used tools, by adding static type definitions.

Types provide a way to describe the shape of an object, providing better documentation, and allowing TypeScript to validate that the code is working correctly.

The next step would be to select a frontend framework for the development. There are lots of available options, number of which grows daily. However, over the last few years, there have been three main leaders – Angular, React and Vue.

All three of them have many years of production usage, open-source code, huge communities and big companies, like Google and Facebook behind them.

Choose of the framework is usually based on the project specific requirements, however, right now all three main frameworks cover most of the cases, just in different ways, So, it seems to be a choice of preference for a particular developer or team.

Angular is the oldest and the most advanced framework out of the big trinity.

Angular is a platform and framework for building single-page client applications using HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your apps.

The architecture of an Angular application relies on certain fundamental concepts. The basic building blocks of the Angular framework are Angular components that are organized into NgModules. NgModules collect related code into functional sets; an Angular app is defined by a set of NgModules. An app always has at least a root module that enables bootstrapping, and typically has many more feature modules.

Components define views, which are sets of screen elements that Angular can choose among and modify according to your program logic and data

Components use services, which provide specific functionality not directly related to views. Service providers can be injected into components as dependencies, making your code modular, reusable, and efficient.

Modules, components and services are classes that use decorators. These decorators mark their type and provide metadata that tells Angular how to use them.

The metadata for a component class associates it with a template that defines a view. A template combines ordinary HTML with Angular directives and binding markup that allow Angular to modify the HTML before rendering it for display.

The metadata for a service class provides the information Angular needs to make it available to components through dependency injection (DI).

An app's components typically define many views, arranged hierarchically. Angular provides the Router service to help you define navigation paths among views. The router provides sophisticated in-browser navigational capabilities.

The last step would be to select the technologies for the server side of the Local UI. As Typescript was chosen as the programming language for the Frontend part, it might be a wise choice to use the similar technology stack on the backend.

It can be done, via the server-side JavaScript runtime – NodeJS. NodeJS is an open-source, cross-platform, JavaScript runtime environment. It executes JavaScript code outside of a browser.

Nevertheless, the Backend will require some means to be able to connect to a Wi-Fi network, as well as to the Arduino motherboard. Gladly, it has almost full access to the underlying APIs of the Linux distributive, which will give us a possibility to implement those connections.

So, it looks like in order to implement the Local UI, the following technology stack can be used: TypeScript, Angular, NodeJS.

The basic interaction algorithm would look like:

- User uploads a model's file in GCode format to the 3D printer and starts the printing process;
- Backend retrieves the file and starts reading it line-by-line, validating the commands and forwarding them to the Arduino motherboard;
- Arduino handles the commands appropriately and sends response;
- As Backend sends the commands, it can send some progress to the Frontend, as well as different statistic, like current temperature, position of the moving units, etc.;

- Once the printing is over, all moving parts are returned to the initial position and Frontend shows some kind of notification for the user;
- Then the user can verify if the model was printed correctly and get it off the heating platform.

Some of the UI screens are listed below:

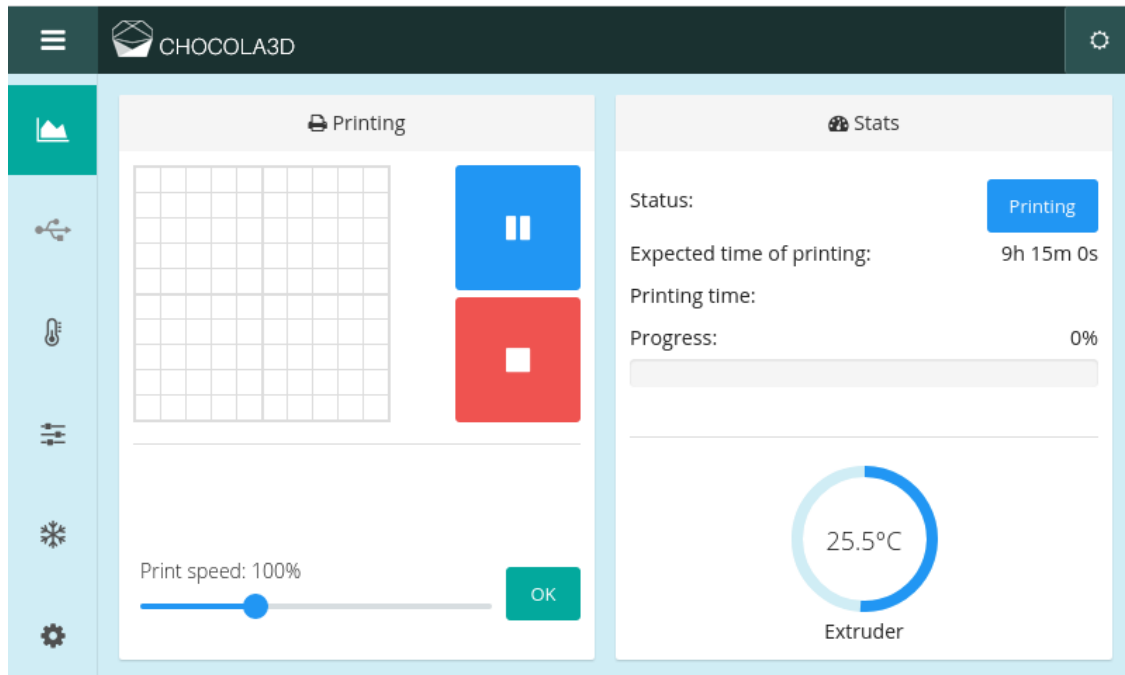


Fig. 3.5. Printing process

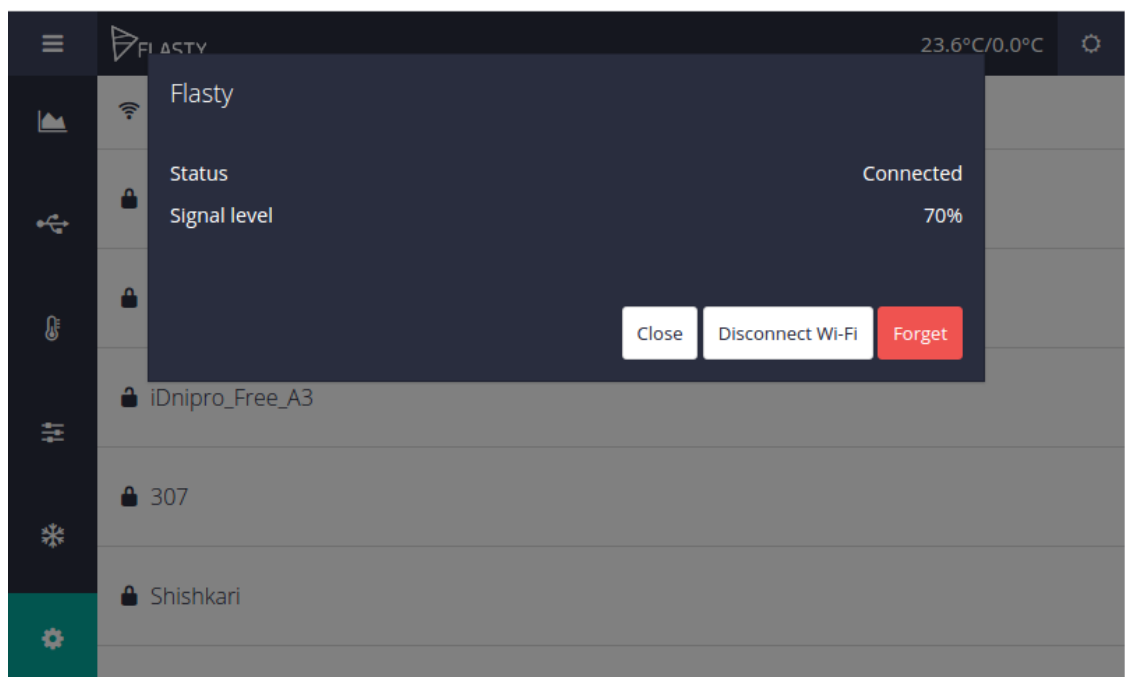


Fig. 3.6. Connection to Wi-Fi network

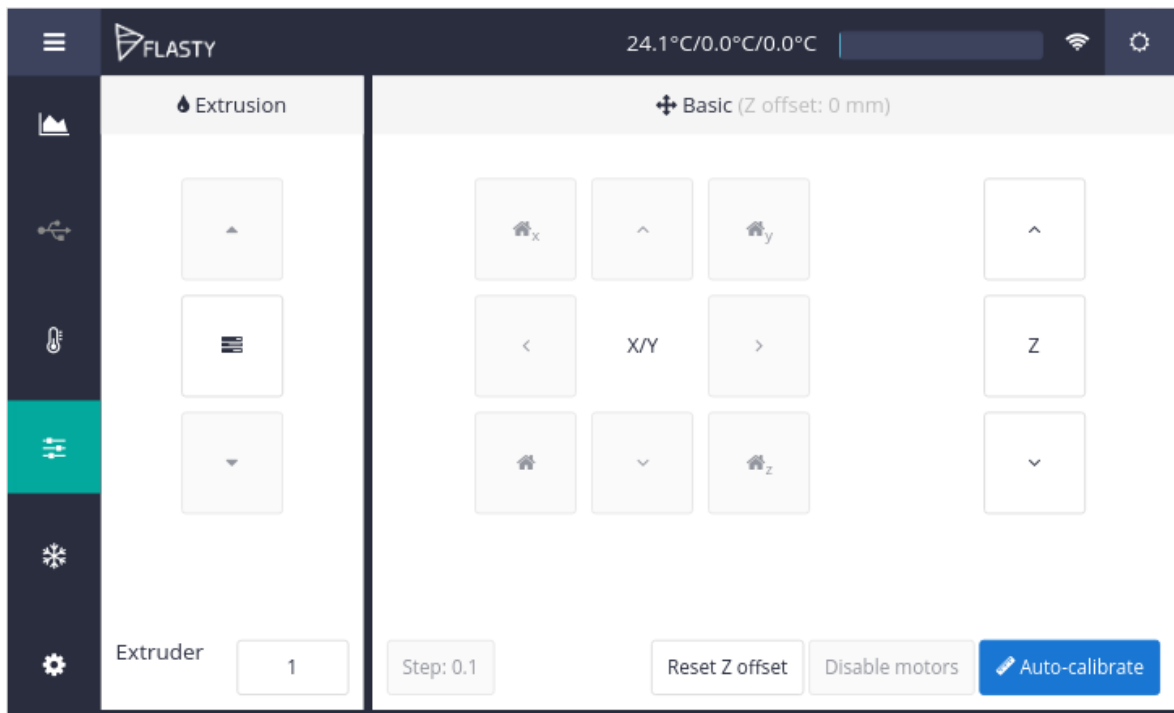


Fig. 3.7. Manual control over the extrusion and X, Y, Z motion-drivers

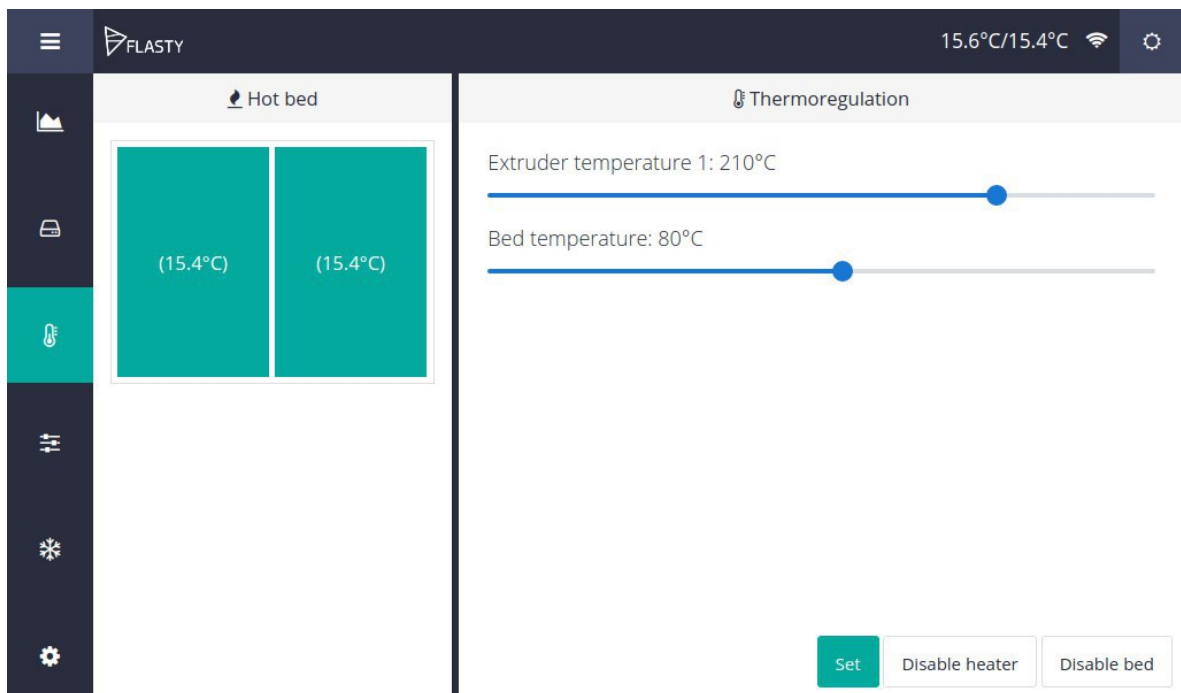


Fig. 3.8. Thermoregulation section

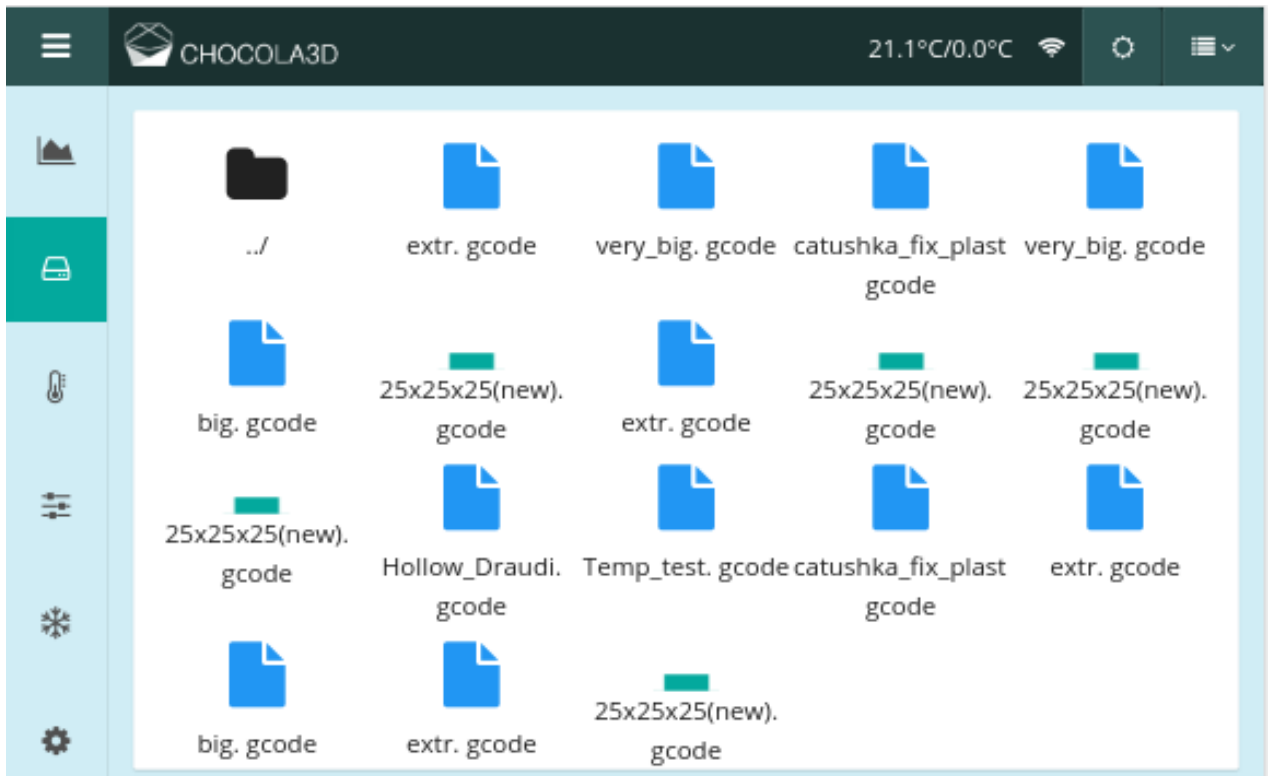


Fig. 3.9. Local file storage

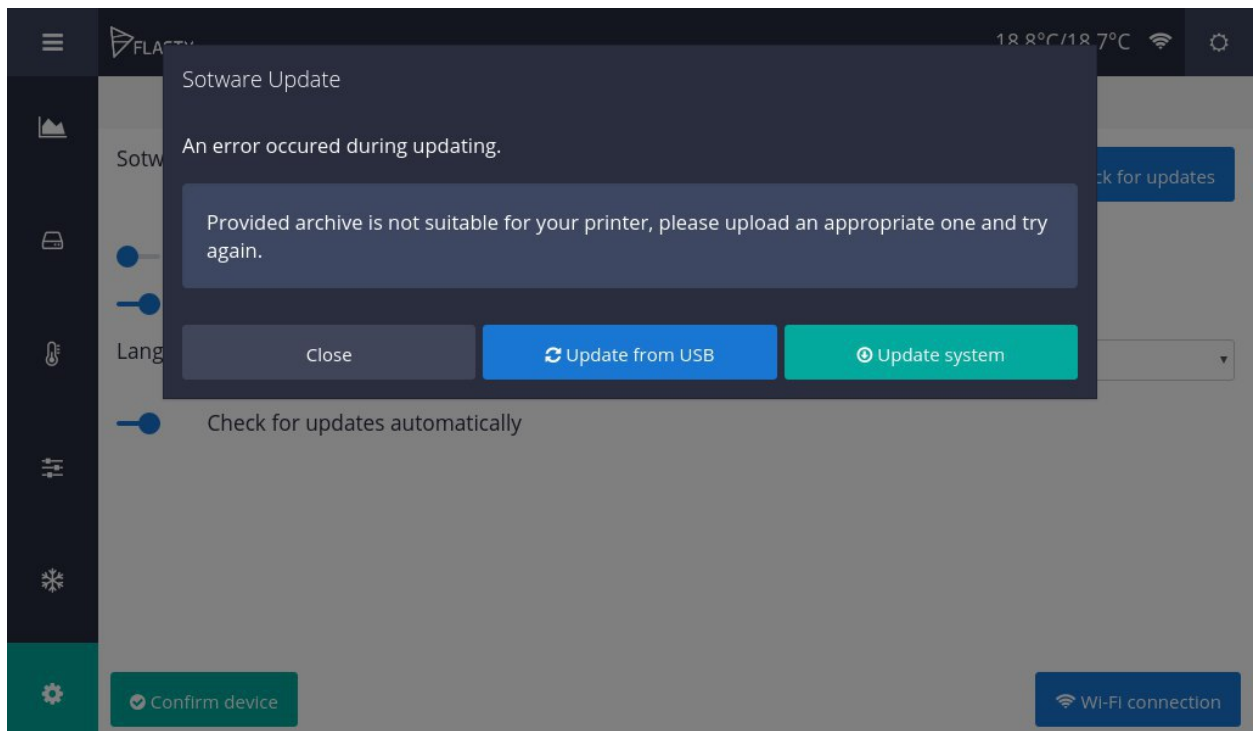


Fig. 3.10. System update UI

3.1.3. Remote user interface

Apart from being able to interact with a 3D printer directly, it should also be possible to be able to access it remotely. This feature might be useful for remote control over the printing process. For example, it is possible to provide an embedded camera as an additional option, which would send the printing live-stream to the Remote UI. Also, sometimes the printing process might take hours or even days. In such cases it would be extremely useful to be able to see the progress to properly plan the next steps or, even, to adjust the temperature of an extruder or the heating if it is necessary.

It seems, like the best choice for a Remote UI, would be a Web-application, because it can give us the best coverage for users on different platforms, mobile, desktop, etc.

A consistent design choice would be to use the same technology stack, as was used for the Local UI – Typescript, Angular, NodeJS, WebSockets.

However, the Remote UI a bit more complicated, than the Local one, because multiple users can access the same device from different devices at the time. This requirement brings a need to support multiple connections to the same device. As 3D printers are usually connect to some local Wi-Fi or Ethernet networks, they don't have a public IP-address, which can be accessed from any part of the planet.

In order to solve this issue, the proposed solution would be to implement a proxy server. This server would be responsible for establishing the communication between a device and client.

In general, the communication algorithm might look like:

- Once the device is connected to a Wi-Fi network, it connects to the proxy-server via the WebSocket connection;
- Proxy-server marks device as “online”;
- When client wants to connect to a particular device, they should pick the one from “online” list;
- Proxy-server registers a connection between the device and the client and starts forwarding messages between them;

- Once the connection is established, the Remote UI can send different commands directly to the device, which means, that now it has the same capabilities as the Local UI.

A simplified connection schema is shown in Fig. 3.11.:

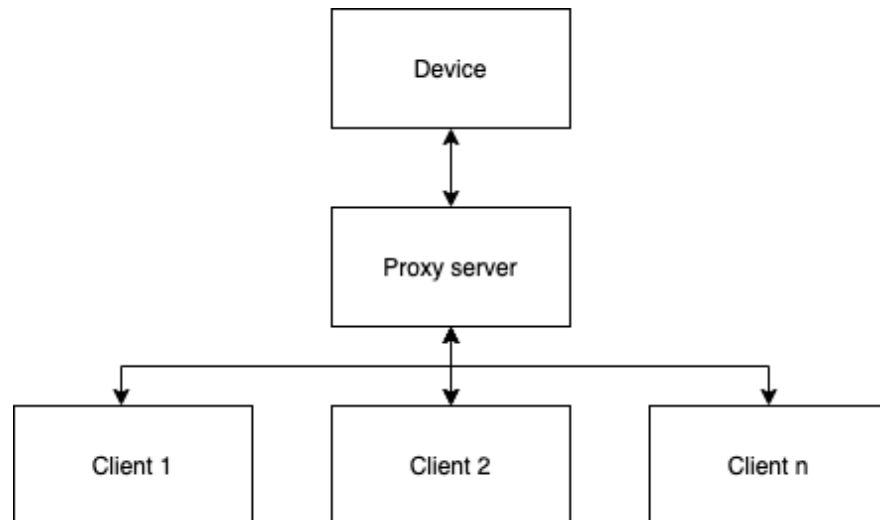


Fig. 3.11. Proxy-server connection schema

Another difference from the Local UI is that, for local UI we always have one user and we do not really care about their authorization and authentication. However, for the Remote UI it is critical to provide some security measures to protect the devices from unsanctioned access.

It seems, like a huge amount of work to implement a custom authorization solution. The better alternative would be to use one of the existing platforms instead. Currently, there are a lot of different offers in the Web ecosystem in area of security. One of the most popular solutions is Auth0.

Auth0 is an identity management platform for application builders and developers. It provides a web-scale cloud solution that includes APIs and tools that enable developers to eliminate the friction of authentication and authorization of their applications and APIs.

Auth0 enables users to single sign-on for applications running on various platforms with various identity providers; add few lines of JavaScript to power their applications; customize various stages of the authentication and authorization pipeline,

and connect their applications and APIs to their database of users and passwords. Its platform also allows users to authenticate to active directory, LDAP, SAML, Integrated Windows Authentication, Google Apps, Salesforce, and other IdPs without having to configure firewall; add and remove users, modify profiles and authorization attributes, and identify root cause user login issues; see a stream of recent logins and their locations; and enable various SaaS and SAML-enabled applications.

Having a reliable authorization system, it is now possible to implement a well-secured, robust and highly accessible Web-Application for a remote access.

So, just to summarize, the Remote UI should meet the following requirements:

- Adaptiveness (supporting desktop, mobile and tablets);
- User-friendly UI;
- Connection to the proxy-server for interactions with the 3D printer;
- All the features of Local UI.

Some screens of the Remote UI can be seen next:

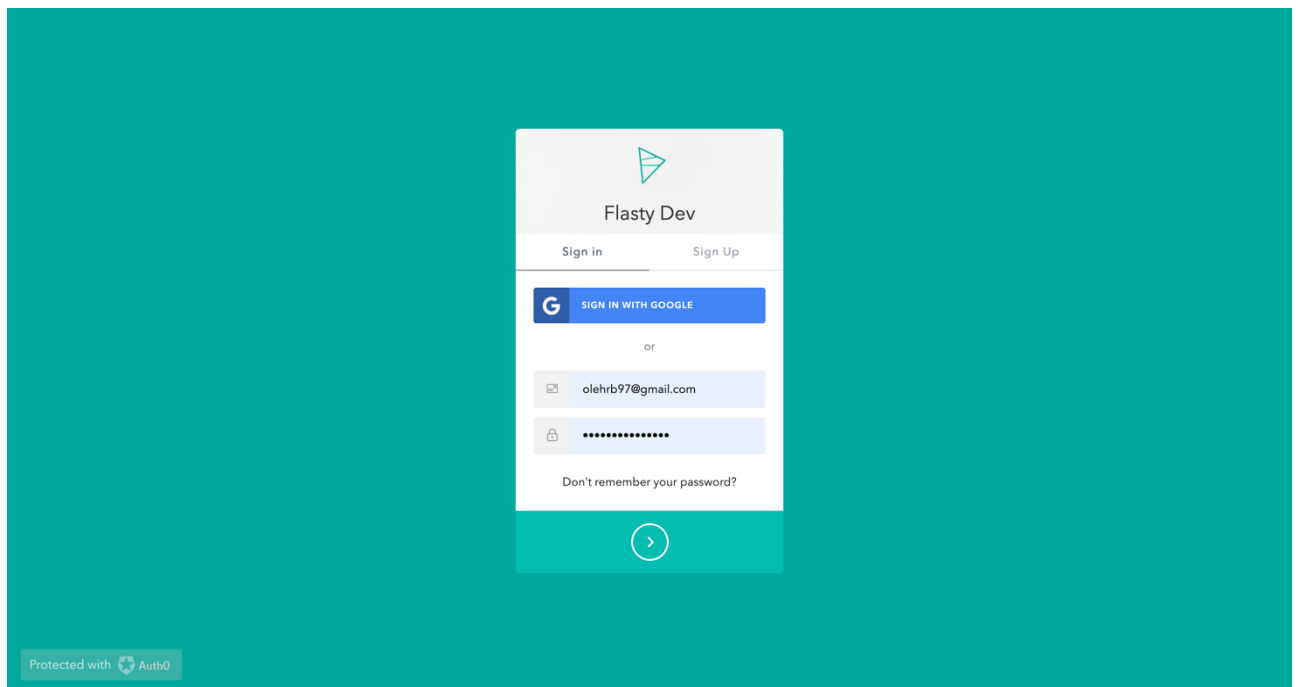


Fig. 3.12. Authorization page

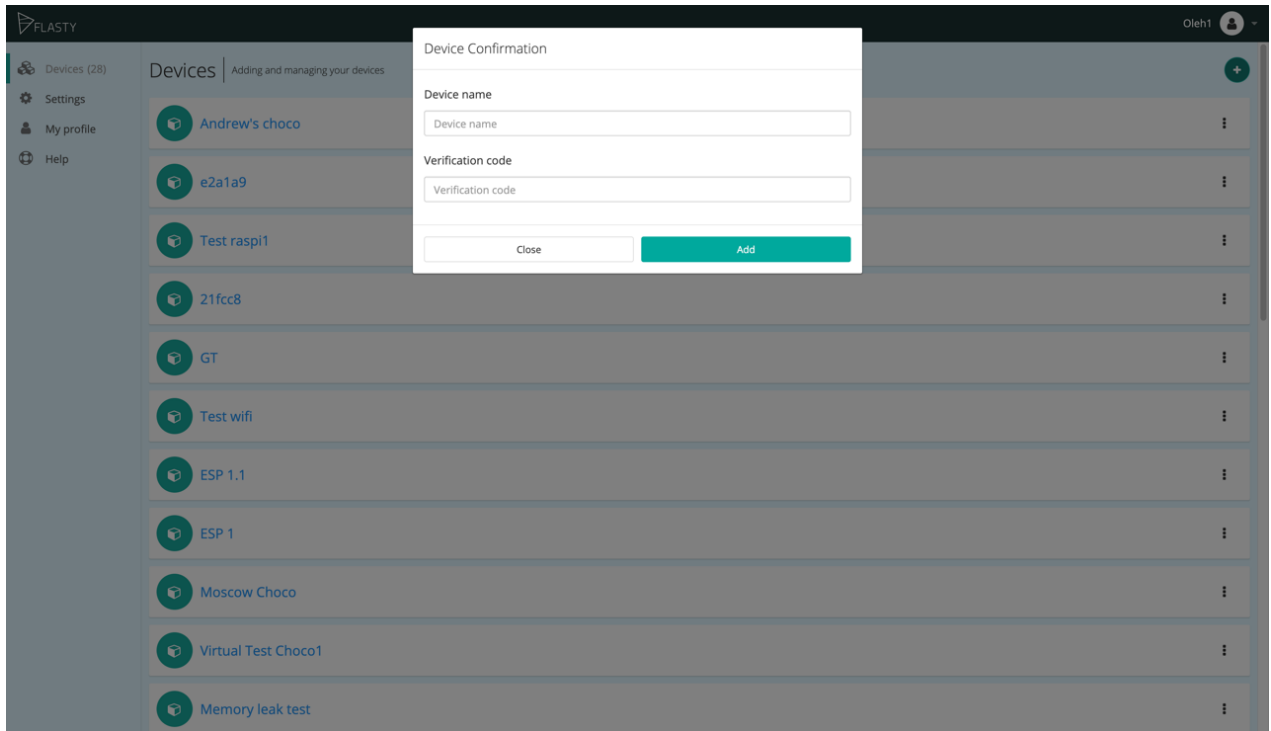


Fig. 3.13. Device Confirmation

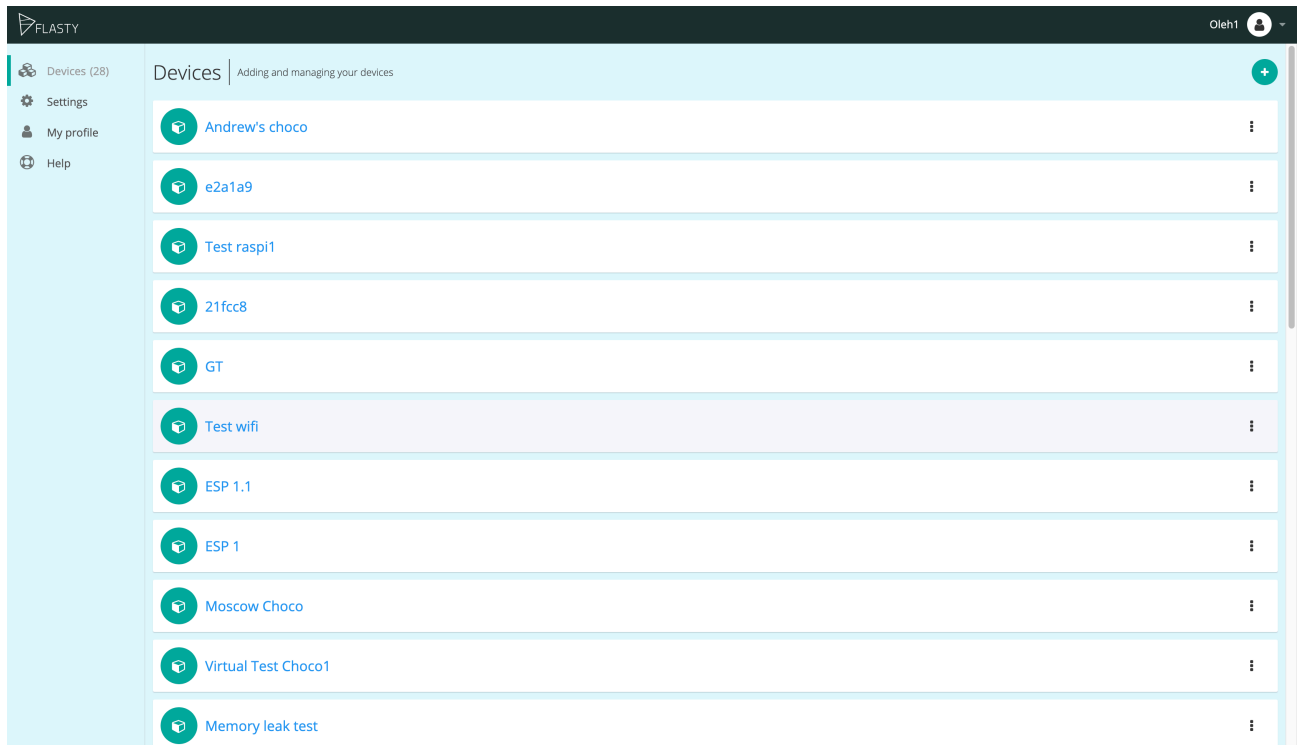


Fig. 3.14. List of devices

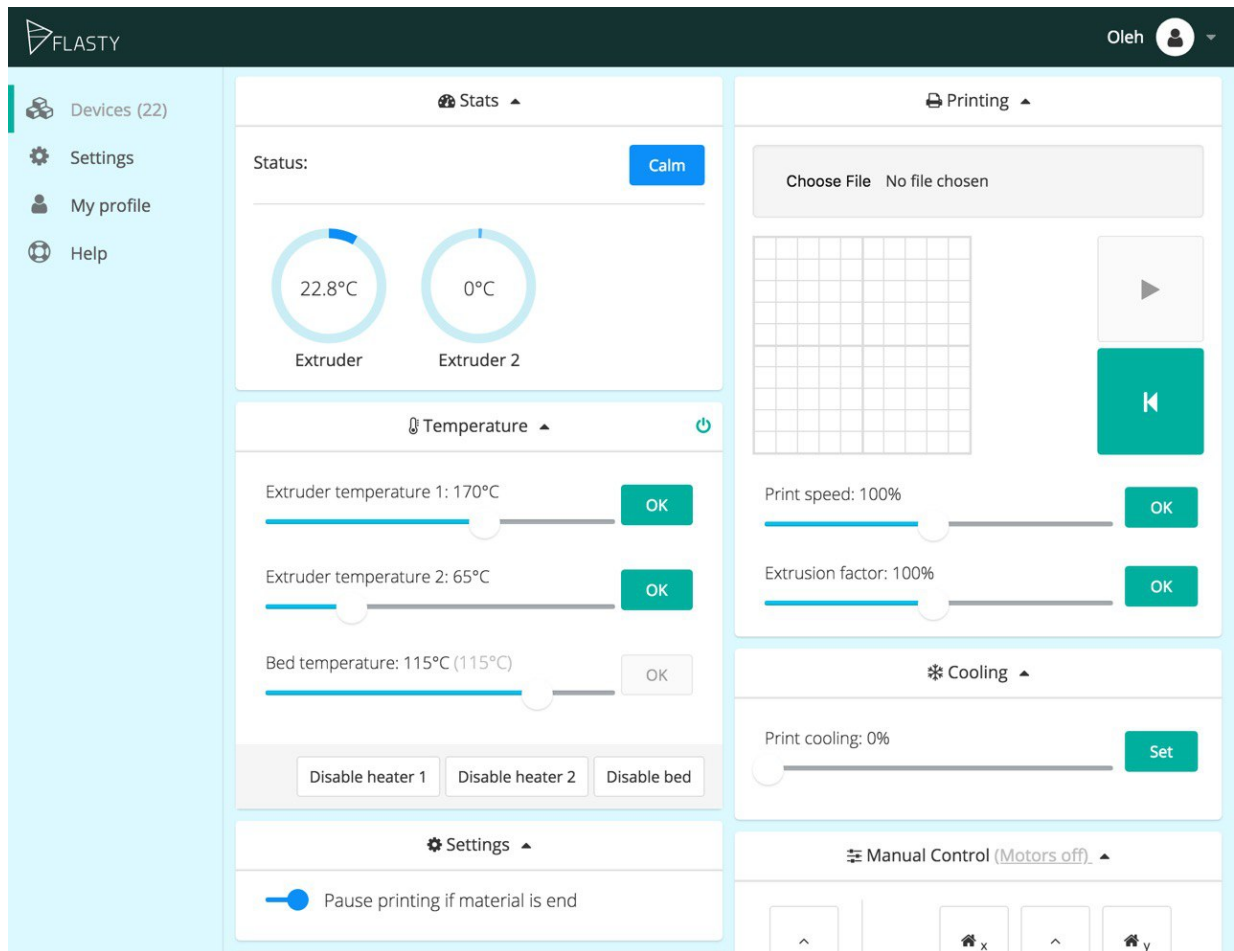


Fig. 3.15. Status, printing, temperature and cooling controls

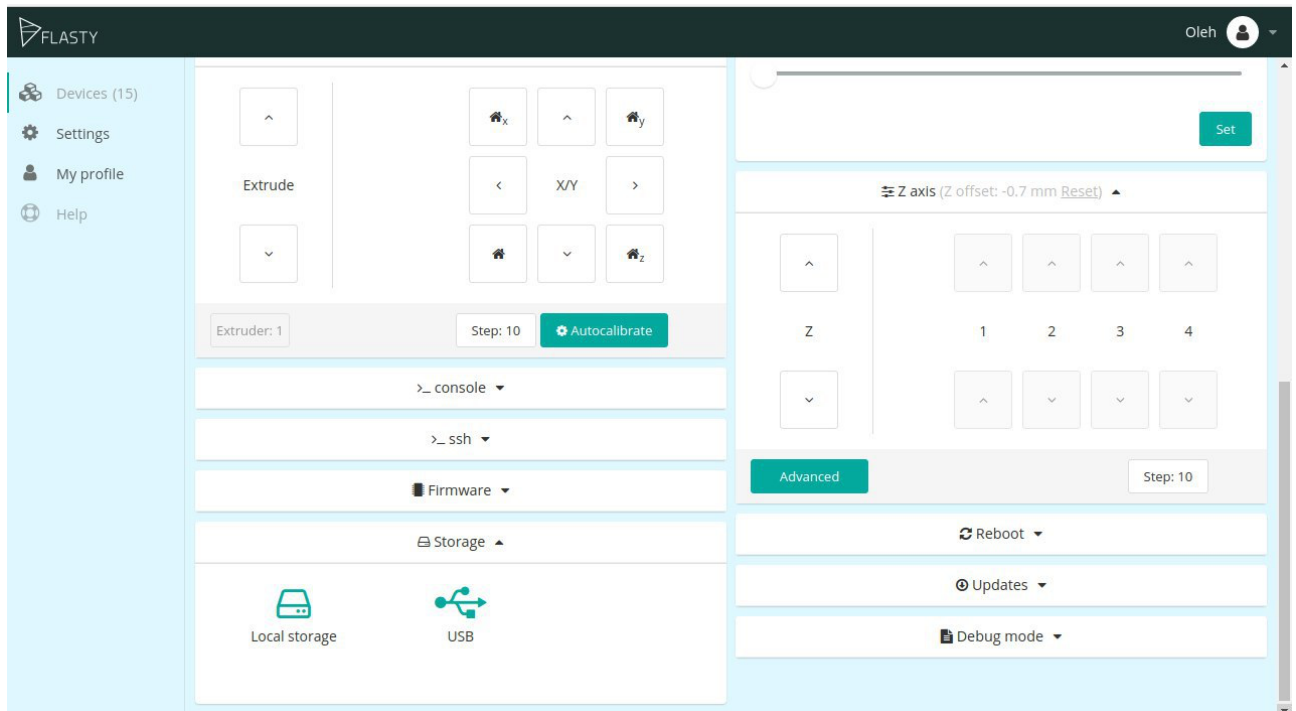


Fig. 3.16. Manual, z-axis, console, storage, firmware, reboot controls

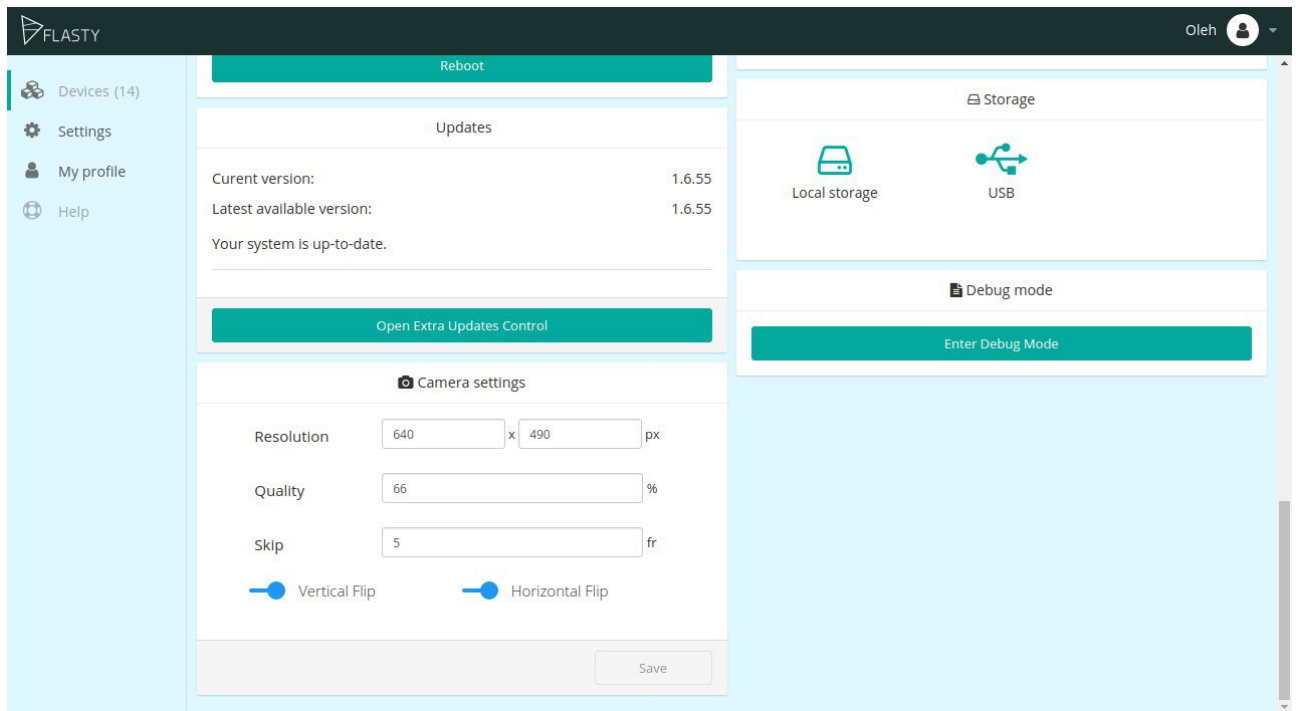


Fig. 3.17. Update, camera, debug mode controls

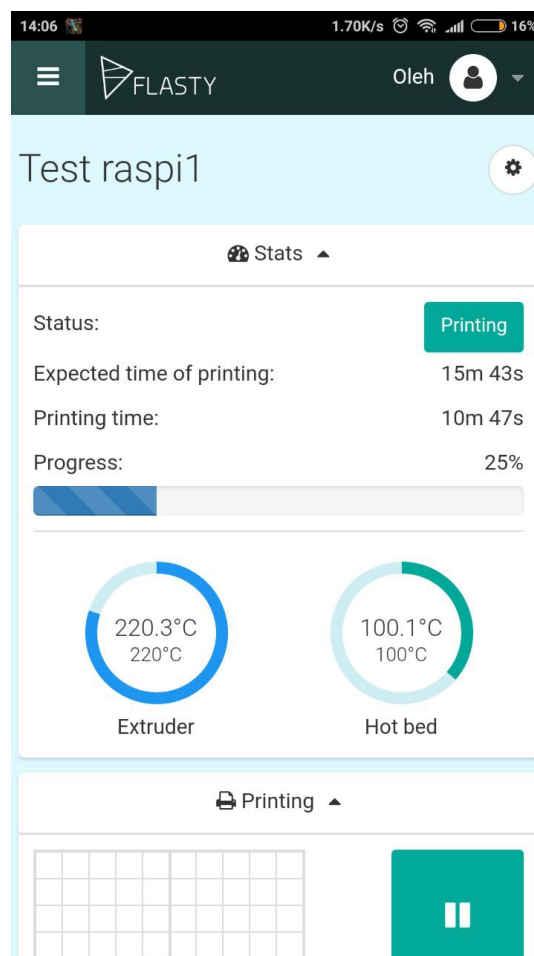


Fig 3.18. Printing process (Mobile layout)

3.2. Additional software elements for 3D printing system

Once all crucial software parts of the system are in place, it is time to finish the initial plans for the ecosystem. The system cannot work without the firmware, Local UI, Remote UI and the proxy server, because they are the core of the ecosystem. Even though, they are the minimal required pieces of software for such kind of system, there is still a place for improving UX.

There might be various ways of bringing additional features and interfaces to simplify the user's daily work with the system. However, this work will focus on two most usable applications:

- Slicer – for preparing 3D models to be printed via a particular 3D printer;
- Cloud storage – for storing the original and prepared 3D models and share them with other users.

Also, just having those two additional interfaces in a vacuum what be really helpful to the user. It is necessary to come up with some kind of communication between those parts of the system, to provide a real ecosystem.

As the main thing between Slicer, Cloud Storage, Local and Remote UI is 3D models (either original in STL format, or converted to GCode), it seems reasonable to build the ecosystem around work with those files.

One of the common user scenarios might be:

1. User uploads the model to Slicer, configures all required settings and receives the GCode;
2. Then they either save the GCode to the Cloud Storage to be printed later, or send it to be printed by a particular 3D printer via the Remote UI, using a specific command in the Slicer's interface.

Another scenario might be as follows:

1. User opens previously prepared GCode file in the Cloud Storage;
2. Then they send it to the Remote UI to be printed by a specific 3D printer via the dedicated command in the Cloud Storage's interface.

Yet another use case might be next:

1. User opens previously uploaded STL file in the Cloud Storage;
2. Then they send it to the Slicer, to prepare it for printing;
3. Once the slicing is done, they send the received GCode to the Remote UI and select, which printer should print it;
4. As an addition step, they might want to save the prepared GCode file to the Cloud Storage, in order to repeat the printing later.

As you can see there is a lot of different ways for user to work with just those 2 addons. In future, as the ecosystem evolves, it is possible to come up with much more different user scenarios and addons, which might improve the ecosystem. To be able to do so, the architecture of the system should be flexible and allow us to add new parts of the system without the need to change or notify in any way any already existing parts of the system.

It looks like, one of the easiest and most flexible approach to this would be come up with a number of commands, supported by each of applications, which could be triggered via the specific URL parameters. For example, in order to print the prepared model from the Cloud Storage, the Remote UI might support a “modelURL” query parameter, which when received, would start a chain of processes which should be done to fetch the model from the Cloud Storage and then send it to a particular printer.

As we are going to use Auth0 as main identity provide, it is crucial to make sure that only owner of the model or 3D printer can interact with them.

In order to do so we would need to provide a seamless authorization between our apps, so the user is not required to login each time they are being redirect from one app to another. Usually, this feature is implemented via the SSO (Single Sign One) approach. Auth0 allows developers to enable this feature out-of-the-box, with just a few additional settings.

At the end it looks like this particular set up will be enough to cover the most basic use-cases and improve the UX, for the users.

So final schema of the 3D printer ecosystem might look like:

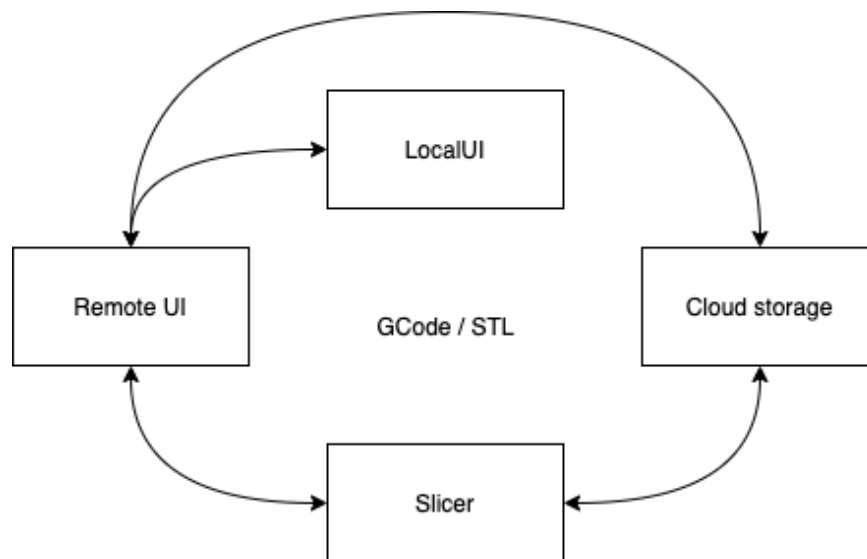


Fig. 3.19. 3D printing ecosystem

3.2.1. Slicer

Slicing is a process, of preparation of 3D models in STL format to be printed by a specific printer, using a tool from the “Slicers” class. Usually, such kind of software provides means and tools to configure the printing parameters for a particular 3D printer, such as platform dimensions, nozzle diameter, temperature mode (based on the type of filament), number of extruders (if the printer supports more than 1 extruder), speed limitations, printing quality, supports, filament skirts and many others. Moreover, it is quite common to see the virtual platform in Slicer. It allows user to place one or more models in one scene and configure their sizes, position and rotation.

The final product of the slicing process is a GCode file, which contains a set of commands, which can be executed by a printer. In order to create such file, based on initial models, their placement on the platform and the printing parameters, a lot of different manipulations should be performed.

To avoid the necessity for creation of a custom transformation engine it seems to be a good idea to use previously mentioned CuraEngine. It takes a STL as input alongside with multiple configuration settings and produces a high quality optimized GCode, which can be then passed onto printer.

Providing user with a possibility to preview the model(s) on the virtual platform requires some kind of an engine for work with graphic. As we selected Web as the target platform for our ecosystem, it is necessary to select an engine which is optimized to be used in Web.

There are various of different 3D engines, which are based on WebGL nowadays. WebGL (Web Graphics Library) is a JavaScript API for rendering high-performance interactive 3D and 2D graphics within any compatible web browser without the use of plug-ins. WebGL does so by introducing an API that closely conforms to OpenGL ES 2.0 that can be used in HTML5 <canvas> elements. This conformance makes it possible for the API to take advantage of hardware graphics acceleration provided by the user's device.

Although, WebGL can be used on its own it requires a lot of work, development wise, to use it. So, we still need to select an engine. One of most popular tools in this area is Three.js. Three.js is a cross-browser JavaScript library and application programming interface (API) used to create and display animated 3D computer graphics in a web browser using WebGL. The source code is hosted in a repository on GitHub. Three.js allows the creation of graphical processing unit (GPU)-accelerated 3D animations using the JavaScript language as part of a website without relying on proprietary browser plugins. This is possible due to the advent of WebGL.

High-level libraries such as Three.js or GLGE, SceneJS, PhiloGL, or a number of other libraries make it possible to author complex 3D computer animations that display in the browser without the effort required for a traditional standalone application or a plugin

So, it seems, like all requirements are covered and the following technology stack can be used to implement the Slicer:

- Typescript;
- NodeJS;
- Angular;
- Auth0;
- Three.js;

But there is a lot of other additional things to consider for such type of system, but they are not too important for this work and so, can be skipped.

Some screens of the applications are listed below:

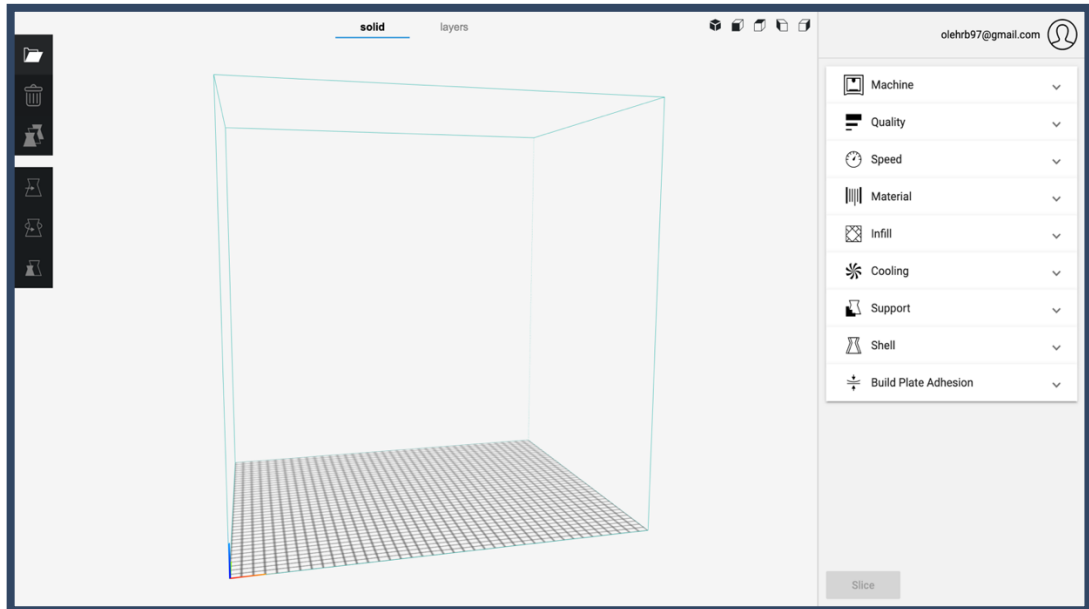


Fig. 3.20. Main interface

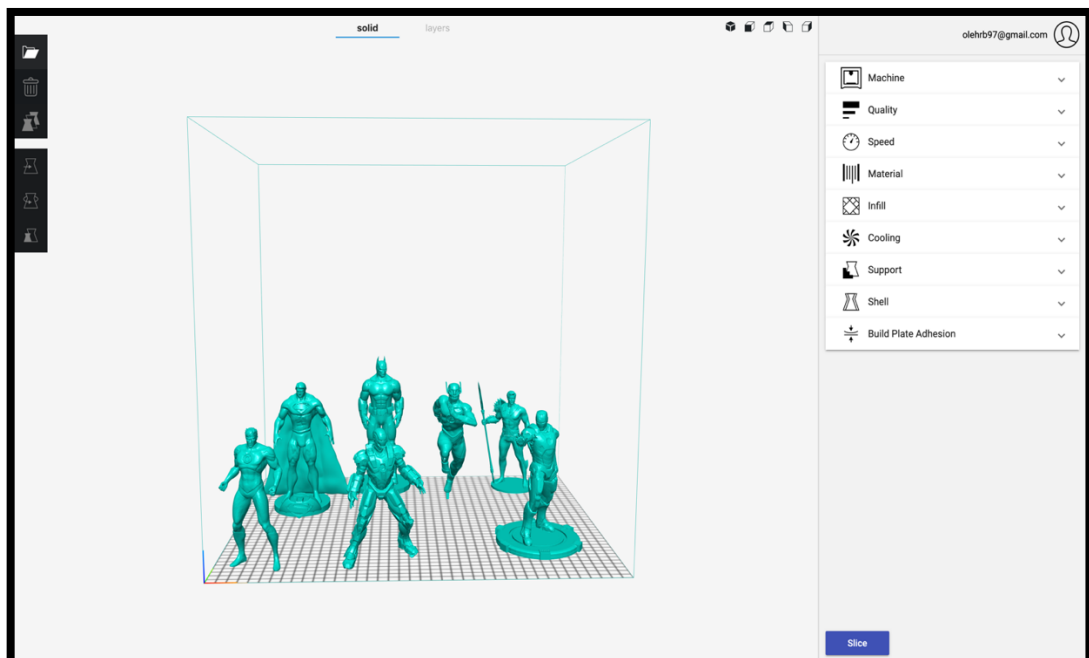


Fig. 3.21. Complex scene

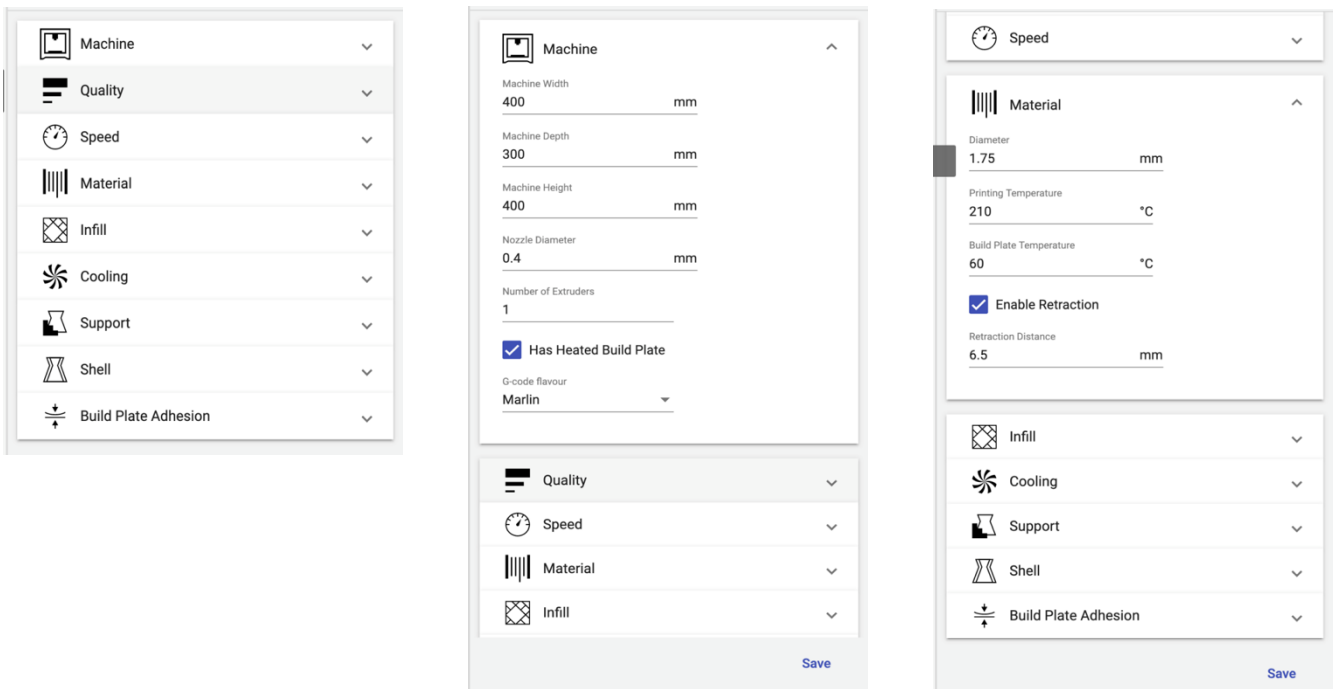


Fig. 3.22. Settings

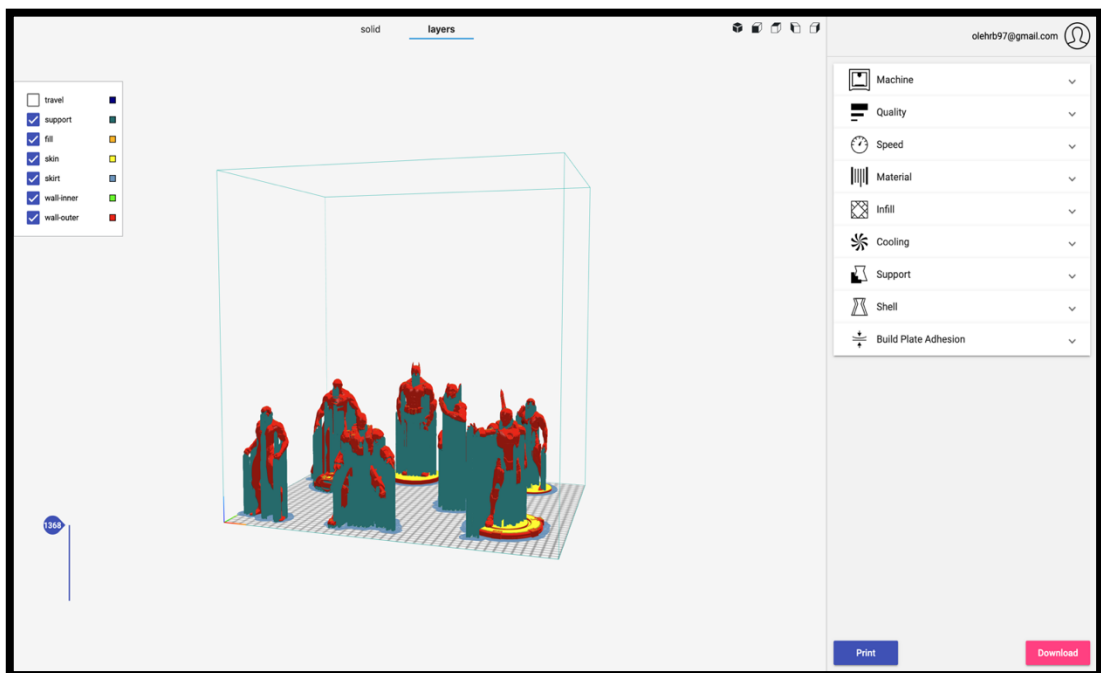


Fig. 3.23. Complex sliced scene

3.2.2. Cloud storage

The last part of the ecosystem is represented by the Cloud Storage, which allows users to store, preview, share and select for printing different models in STL and GCode formats. This is the simplest part of the system, as it serves a very simple and straightforward purpose – keeping user’s data safe and highly available.

However, this application has one single quirk, which should be considered, when building such kind of applications. It is the storage itself. As it seems rather redundant to develop custom implementation of the storage, because it should be fast, reliable, with high availability rate. More appropriate solution might be select something, already available on the market. For example, Google, Microsoft and Amazon provide their own robust data storage centers, which met all the requirements and in addition, they are not really expensive.

Otherwise, the technology stack for this piece of software is rather the same as for the previous parts:

- Typescript, as the programming language;
- NodeJS as the environment for the backend;
- Angular as the frontend framework;
- Auth0 as the identity provider;

Some screens of the application are listed below:

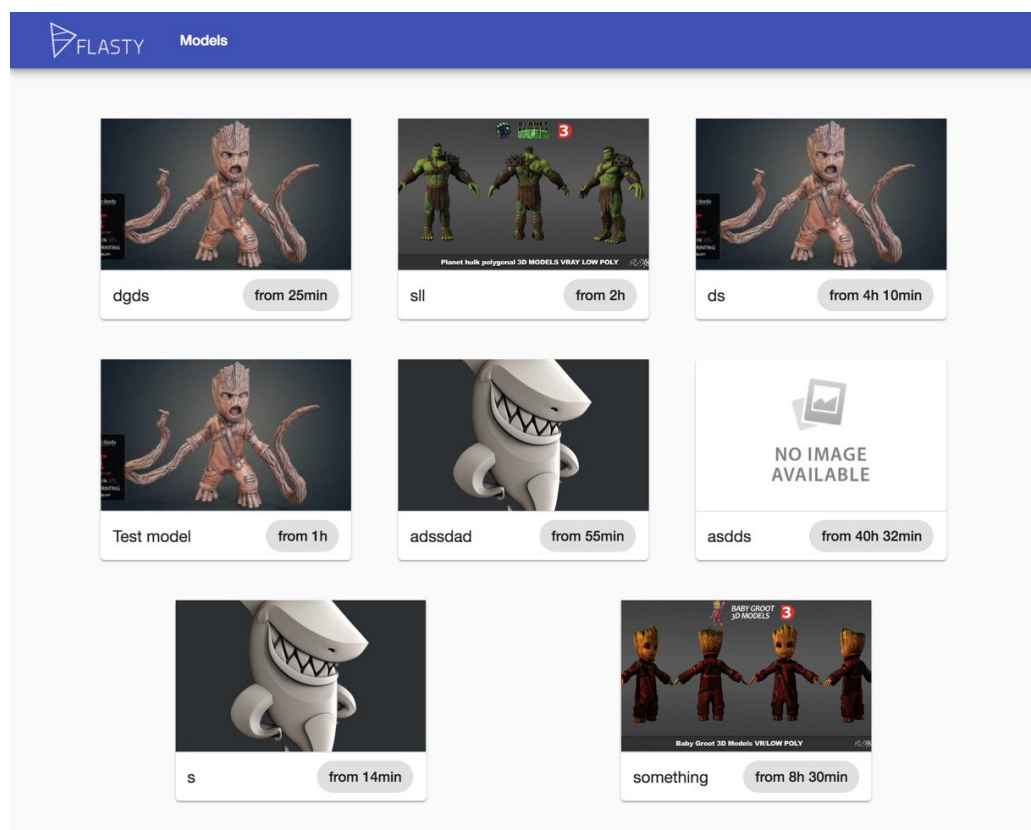


Fig. 3.24. List of uploaded files

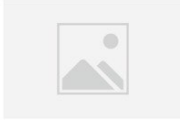



ID	Image	Title	Description	Printing Time ↓
3		Some model	Some model's description	01:15
4		Some model	Lorem ipsum dolor sit amet consectetur adipiscing elit. Molestiae est laudantium nesciunt ipsum facere inodunt blanditiis accusamus! Exercitationem, mollitia rutrum nostrum quis aliquid voluptate ab, tempora, aliquam quis aperiam adipiscit?	01:07
1		Some model	Some model's description	00:06
2		Some model	Some model's description	00:06

Fig 3.25. Files management

3.3. Conclusions for section 3

This part provides a set of suggestions on what should be implemented, software wise, for a 3D printing computer appliance. Those pieces of software create a core of the system, extended by some useful addons, like the Slicer and the Cloud Storage, which improve overall user experience. The proposed design covers all basic user scenarios, when working with a 3D printing system on the daily basis. As always, there is still a room for different improvements and extensions for the platform, but the provided architecture allows us to easily bring new parts to the system from both software and hardware perspective.

РОЗДІЛ 4 ЕКОНОМІКА

4.1. Визначення трудомісткості розробки програмного забезпечення

Початкові дані:

1. передбачуване число операторів програми – 1800;
2. коефіцієнт складності програми – 1,7;
3. коефіцієнт корекції програми в ході її розробки – 0,07;
4. годинна заробітна плата програміста – 70грн/год;
5. коефіцієнт збільшення витрат праці в наслідок недостатнього опису задачі – 1,2;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,0;
7. вартість машино-години ЕОМ – 17 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_{\delta}, \text{ людино-годин} \quad (4.1)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 70 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n -витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ -витрати праці на налагодження програми на ЕОМ;

t_{δ} - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p),$$

де q - передбачуване число операторів (1800);

C - коефіцієнт складності програми (1,7);

p - коефіцієнт корекції програми в ході її розробки (0,07).

Звідси умовне число операторів в програмі:

$$Q = 1,7 \cdot 1800 \cdot (1 + 0,07) = 3274,2 \text{ людино-годин},$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}, \text{ людино-годин},$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи від 2 до 3 років він складає 1,0.

Приймемо збільшення витрат праці внаслідок недостатнього опису завдання не більше 50% ($B = 1,2$). З урахуванням коефіцієнта кваліфікації $k = 1,0$, отримуємо витрати праці на вивчення опису завдання:

$$t_u = (3274,2 \cdot 1,2) / (80 \cdot 1,0) = 49,11 \text{ людино-годин}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20 \dots 25) \cdot k}, \text{ людино-годин} \quad (4.2)$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (4.2), отримаємо:

$$t_a = 3274,2 / (22 \cdot 1,0) = 148,8 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20 \dots 25) \cdot k}, \text{ людино-годин.}$$

$$t_n = 4002 / (22 \cdot 1,0) = 181,9 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{oml} = \frac{Q}{(4..5) \cdot k}, \text{ людино-годин.}$$

$$t_{oml} = 3274,2 / (4 \cdot 1,0) = 818,55 \text{ людино-годин.}$$

- за умови комплексного налагодження завдання:

$$t_{oml}^k = 1,5 \cdot t_{oml}, \text{ людино-годин.}$$

$$t_{oml}^k = 1,5 \cdot 818,55 = 1227,8 \text{ людино-годин.}$$

Витрати праці на підготовку документації визначаються за формулою:

$$t_{\partial} = t_{\partial p} + t_{\partial o} , \text{ люДИНО-ГОДИН,}$$

де $t_{\partial p}$ -трудомісткість підготовки матеріалів і рукопису:

$$t_{\partial p} = \frac{Q}{(15..20) \cdot k} , \text{ люДИНО-ГОДИН,}$$

$t_{\partial o}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{\partial o} = 0,75 \cdot t_{\partial p} , \text{ люДИНО-ГОДИН.}$$

Підставляючи відповідні значення, отримаємо:

$$t_{\partial p} = 3274,2 / (17 \cdot 1,0) = 192,6 \text{ люДИНО-ГОДИН.}$$

$$t_{\partial o} = 0,75 \cdot 192,6 = 144,45 \text{ люДИНО-ГОДИН.}$$

$$t_{\partial} = 192,6 + 144,45 = 337,05 \text{ люДИНО-ГОДИН.}$$

Повертаючись до формули (4.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 70 + 49,11 + 148,8 + 148,8 + 818,55 + 337,05 = 1572,31 \text{ люДИНО-ГОДИН.}$$

4.2. Витрати на створення програмного забезпечення

Витрати на створення ПЗ $K_{ПО}$ включають витрати на заробітну плату виконавця програми $Z_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$K_{ПО} = Z_{ЗП} + Z_{МВ}, \text{ грн.}$$

Заробітна плата виконавців визначається за формулою:

$$Z_{ЗП} = t \cdot C_{ПР}, \text{ грн,}$$

де: t - загальна трудомісткість, людино-годин;

$C_{ПР}$ - середня годинна заробітна плата програміста, грн/година

З урахуванням того, що середня годинна зарплата програміста становить 60 грн / год, отримуємо:

$$Z_{ЗП} = 1572,31 \cdot 70 = 110061,7 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$Z_{МВ} = t_{отл} \cdot C_{мч}, \text{ грн,} \quad (4.3)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год;

$C_{мч}$ - вартість машино-години ЕОМ, грн/год (13 грн/год).

Підставивши в формулу (4.3) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$Z_{мв} = 818,55 \cdot 17 = 13915,35 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$K_{ПО} = 110061,7 + 13915,35 = 123977,05 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс.}$$

де B_k - число виконавців (дорівнює 1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні $F_p=176$ годин).

Звідси витрати на створення програмного продукту:

$$T = 1572,31 / (1 \cdot 176) \approx 8,9 \text{ міс.}$$

4.3. Маркетингові дослідження

Розроблювана система ставить перед собою завдання зменшення порогу входження до сфери адитивного технологій для середньостатистичного користувача. Цей апаратно-програмний комплекс включає в себе як апаратне забезпечення у вигляді приладу для 3D друку – 3D принтеру, так й програмну складову, що забезпечує швидку, стабільну та надійну роботу, як комплексу в цілому так і його окремих елементів.

Така система ставить за мету спрощення щоденної взаємодії користувача з 3D принтерами. Проблема взаємодії є релевантною через те, що більшість сучасних систем для 3D друку розроблюється інженерами для інженерам, а тому

середньостатистичний користувач повинен витратити багато власного часу та коштів, щоб налаштувати робочі процеси для взаємодії з 3D принтером.

Адже, 3D друк є багаторівневим складним процесом, що включає у себе багато комплексних кроків, таких, як: створення 3D моделі, підготовка та налаштування моделі до друку на конкретному 3D принтері, контроль за процесом друку, подальша обробка готового виробу та інші.

Розроблювана система ставить за мету вирішення цієї проблеми шляхом надання користувачу можливості проходження всіх етапів 3D друку в рамках однієї системи, всі компоненти якої пов'язані між собою та надають зручний та простий інтерфейс.

Програмна частина апаратно-програмного комплексу складається з:

- системи для локального керування та взаємодії з 3D принтером;
- системи для віддаленого керування та взаємодії з 3D принтером;
- слайсеру – ПЗ, що дозволяє налаштувати та підготувати 3D моделі до друку на конкретному 3D принтері;
- хмарного сховища – що дозволить користувачам зберігати оригінальні та підготовлені 3D моделі та ділитися ними з іншими.

Унікальність цього продукту полягає у тому, що завдяки контролю повного циклу розробки програмної та апаратної частини системи є можливість забезпечити найкращу інтеграцію елементів системи між собою, що позитивно впливає на загальний досвід користування, спрощує та прискорює роботу користувача в рамках системи.

Наразі на ринку немає жодної системи, що надавала б повноцінну інтеграцію програмного та апаратного забезпечення. Адже, як було виявлено раніше, подібні системи розроблюються з прицілом на інженерів, що здатні самостійно скомпонувати систему, використовуючи окремі, не пов'язані між собою елементи, що унеможлиблює повноцінну та безшовну інтеграцію.

Через відсутність аналогів на ринку не є можливим порівняння розроблюваного продукту.

Розповсюдження програмної частини системи вбачається доцільним забезпечити шляхом публікації її компонентів у вигляді WEB-додатків, що надає можливість покрити найбільший спектр потенційних користувачів, адже WEB відкриває для нас усі популярні платформи, такі, як Windows, OS X, Linux, iOS та Android. При цьому розроблення одного WEB-додатку є значно дешевшим у порівнянні з розробкою спеціального нативного додатку для кожної з платформ. Це дозволить значно прискорити початковий запуск продукту на ринку.

На початку розвитку подібної платформи, гарною ідеєю є надання доступу до неї обмеженій кількості користувачів, наприклад, тільки власників апаратних приладів з екосистеми.

4.4. Економічна ефективність

Економічний ефект від впровадження системи для 3D друку на виробництві, що потребує регулярного друку 3D прототипів різного роду деталей може дозволити підприємству:

- заощадити на досить кошовному навчанні працівників для роботи на 3D принтерах;
- спростити, поліпшити а пришвидшити рутинну роботу працівників, що взаємодіють з 3D принтерами;
- підвищити швидкість друку прототипів, що у свою чергу допоможе прискорити налагодження серійного виробництва продукції.

Таблиця 4.1

Розрахунок чистих грошових надходжень від розробки ПЗ

Показники, тис грн	За роками						Усього за 5 років	Середнє за 5 років
	0	1	2	3	4	5		
1. Інвестиції на ПЗ	124	-	-	-	-	-	124	24,8

Продовження таблиці 4.1

2. Витрати до впровадження ПЗ	-	320	170	320	170	320	1300	260
- на навчання працівників	-	150	60	150	60	150	570	114
- на обслуговування 3D принтерів	-	90	30	90	30	90	330	66
- на електроенергію	-	15	15	15	15	15	75	15
- на щорічну перевірку Держстандарту	-	15	15	15	15	15	75	15
- на оплату праці оператора	-	50	50	50	50	50	250	50
3. Витрати після впровадження ПЗ	-	452,5	92,5	132,5	92,5	132,5	902,5	180,5
- на придбання плат керування	-	320	-	-	-	-	320	64
- на обслуговування 3D принтерів	-	70	30	70	30	70	270	54
- на електроенергію	-	12,5	12,5	12,5	12,5	12,5	62,5	12,5
- на щорічну перевірку Держстандарту	-	15	15	15	15	15	75	15
- на оплату праці оператора	-	35	35	35	35	35	175	35
4. Економія	-	-132,5	77,5	187,5	77,5	187,5	397,5	79,5
5. Амортизація	-	60	35,5	-	-	-	95,5	19,1
6. Чисті грошові надходження	-	-72,5	42	187,5	77,5	187,5	422	84,4

Продовження таблиці 4.1

7. Коефіцієнт дисконтування	-	0,907	0,815	0,724	0,68	0,61	-	-
Дисконтові грошові надходження	-	-65,8	32,2	135,8	52,7	114,4	269,3	53,9

Чиста поточна вартість доходів:

$$NPU = 269,3 - 124 = 145,3 \text{ тис. грн} > 0$$

Строк окупності:

$$T = 124 / 53,9 = 2,3 \text{ років}$$

Індекс прибутковості:

$$ІП = 269,3 / 124 = 2,2$$

Показник економічної ефективності NPU - чиста поточна вартість доходів за роки реалізації впровадження (3-5 років) складе 145,3 тис. грн тобто відповідає умовам ефективності, тому що $NPU > 0$.

Середній строк окупності капвкладень складе _ рік.

Індекс прибутковості за 5 років складе 2,2, тобто $ІД > 1$, проєкт варто прийняти.

Таким чином, показник ефективності свідчить про те, що дане впровадження є економічно вигідним.

CONCLUSIONS

The conducted research studied the idea of implementing a full-size ecosystem in the area of 3D printing. The root of the idea lies in the assumption, that this relatively new approach to building 3D printing systems is going to bring much better UX. This improvement might result in further popularization of 3D printing for wider layers of society and so, give a boost in development of additive technologies.

In order to prove this assumption, the following steps were taken:

- we analyzed existing ecosystems outside the 3D printing area;
- we analyzed existing solutions in 3D printing area;
- we defined requirements for both hardware and software parts of the system;
- we defined the most crucial hardware elements of the system;
- we defined the most crucial software elements of the system;
- we analyzed available options on hardware market;
- we selected the most appropriate hardware elements for the system, using method of multicriteria weighted estimations;
- we described a mathematical model for the firmware;
- we developed the software part of the system;
- we designed a 3D printing computer appliance from scratch.

In general, any modern 3D printer consists of the following main parts: controller board, user interaction unit, motion controllers, extruder, heating platform and a touch screen. However, it is worth noticing, that in this study it was decided to concentrate on the biggest parts of a 3D printer. But there is still a lot of smaller pieces, which should be taken into consideration, when designing a 3D printing system.

Motion controllers, extruder, heating platform and a touch screen are pretty much standardized on the market. But it is not that obvious, when it comes to the brains of the 3D printer – motherboard and user-interaction unit. To provide the most robust, reliable and efficient system it was decided to compare the available options for those two units, using method of multicriteria weighted estimations.

For the motherboard we compared two most popular editions of Arduino board – Arduino Uno and Arduino Mega. Although, both of them are really close to each other, the analysis has shown, that Arduino Mega is slightly more preferable, when building a reliable and robust system.

The two most popular options for user-interaction controller are Raspberry Pi 3 and Raspberry Pi Zero. Here the method of multicriteria weighted estimations got further application and provided a strong analytical data. The data helped to decide in favor of Raspberry Pi 3, as Raspberry Pi Zero proved to be much less reliable and efficient, even though it is much more attractive when it comes to the price.

As the initial analysis shows, in order to satisfy the market's needs, the software part of the system should provide not only the most crucial parts, but also some extra addons.

When it comes to the software part, the most crucial pieces are: firmware, local UI, remote UI and a proxy server to connect a particular 3D printer to a particular remote user.

To get a better understanding of how a 3D printing system should, a mathematical model for the firmware was created and applied to the developed system. The other crucial pieces of software were developed relaying on the most advanced technologies and solutions, available right now. Such as:

- Marlin – for the robust and reliable firmware;
- CuraEngine for the fast and precise slicing;
- Auth0 for secure remote access;
- Typescript – as the main programming language;
- Angular – as frontend framework;
- NodeJS – as server-side environment;
- and a lot of other tools and technologies.

Moreover, apart from the most crucial part, some additional elements were also defined and implemented. In order to improve user experience, decrease the entrance level to 3D printing and provide all required tools for daily work with a 3D printing system, some addons were introduced. Those are:

- Slicer – for preparation of the models to be printed on a particular printer;
- Cloud storage – for keeping and sharing prepared and original models;

However, it would not be a true ecosystem without some inner communication between different software and hardware parts. In order to cover this area, we introduced a special communication protocol, which allows all of the software parts to communicate and cover as much of user scenarios as possible.

So, the result of this research is a design for a full-size 3D printing ecosystem, which includes: 3D printers with embedded software and different WEB-applications, which make it much easier to work within the 3D printing system.

Furthermore, the most critical hardware decisions were made, using method of multicriteria weighted estimations.

REFERENCES

1. Bertier Luyt, Samuel N. Bernier, Tatiana Reinhard Design for 3D Printing: Scanning, Creating, Editing, Remixing, and Making in Three Dimensions / Make Community, LLC – New York, 2015. – 162 p.
2. Neil Rosenberg Designing 3D Printers: Essential Knowledge / 3D Hubs – Amsterdam, 2020 – 197 p.
3. Dr. Dan-Andrei Marinescu How to build a 3D Printer: DIY project: “EASY CoreXY 3D Printer Model 350” / LM Publishing House – Volendam, 2019 – 87 p.
4. William Shotts The Linux Command Line, 2nd Edition: A Complete Introduction / No Starch Press – San Francisco, California, 2019 – 502 p.
5. Ben Redwood, Filemon Schöffner, Brian Garret The 3D Printing Handbook: Technologies, design and applications / 3D Hubs – Amsterdam, 2017 – 304 p.
6. J. M. Hughes Arduino: A Technical Reference: A Handbook for Technicians, Engineers, and Makers (In a Nutshell) / O'Reilly Media – Sebastopol, California, 2016 – 738 p.
7. Hod Lipson Fabricated: The New World of 3D Printing / Wiley – New Jersey, 2012 – 280 p.
8. Joan Horvath Mastering 3D Printing (Technology in Action) / Apress – California, 2014 – 354 p.
9. James Shelby PH.D THE RASPBERRY PI GUIDE BOOK: The Master Guide To Mastering The Fundamental Of Raspberry Pi / Independently published – London, 2020 – 28p.
10. James W. Hardin, Joseph M. Hilbe Generalized Estimating Equations / Chapman and Hall/CRC – London, 2012 – 280 p.
11. Kevin Koekkoek 3D Printing Projects: 20 design projects for your 3D printer / 3Dkev – New Orleans, 2014 – 112 p.
12. Jon Duckett Web Design with HTML, CSS, JavaScript and jQuery Set / Wiley – Hoboken, New Jersey, 2014. – 1152 p.

13. Mark Myers *A Smarter Way to Learn JavaScript*. The new tech-assisted approach that requires half the effort / CreateSpace Independent Publishing Platform – Scotts Valley, 2014 – 254 p.

14. Marijn Haverbeke *Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming* / No Starch Press – San Francisco, California, 2018 – 472 p.

15. Darrel Ince *The Computer: A Very Short Introduction (Very Short Introductions)* / OUP Oxford – Oxford, 2011 – 152 p.

16. Johannes Wild *3D Printing 101: The Ultimate Beginner's Guide* / 3D Hubs – Amsterdam, 2019 – 56 p.

17. Daniel J. Barrett *Linux Pocket Guide: Essential Commands* / O'Reilly Media – Sebastopol, California, 2016 – 274 p.

18. Jacob Cabral, Gareth Halfacree *The Official Raspberry Pi Beginner's Guide: How to use your new computer (The Raspberry Pi Beginner's Guide Book 3)* / Manning Publications – Shelter Island, New York 2020 – 309 p.

19. James Floyd Kelly *3D Printing: Build Your Own 3D Printer and Print Your Own 3D Objects* / Que Publishing – London, 2013 – 190 p.

20. Christopher Barnatt *3D Printing: Third Edition* / ExplainingTheFuture.com – Nottingham, 2016 – 320 p.

21. Stephanie Torta, Jonathan Torta *3D Printing: An Introduction* / Mercury Learning and Information – Herndon, 2019 – 517 p.

22. Christopher D. Winnan *Adventures in 3D Printing: Limitless Possibilities and Profit Using 3D Printers (3D Printing for Entrepreneurs)* / Que Publishing – London, 2013 – 394 p.

23. Russell Scott *Computer Networking Beginners Guide: An Easy Approach to Learning Wireless Technology, Social Engineering, Security and Hacking Network, Communications Systems (Including CISCO, CCNA and CCENT)* / Que Publishing – London, 2019 – 183 p

24. Elisabeth Robson, Eric Freeman *Head First JavaScript Programming: A Brain-Friendly Guide* / O'Reilly Media – Sebastopol, California, 2014 – 704 p.

25. Raymond T Reeves 3D Printing: Modern Technology in a Modern World / LM Publishing House – Volendam, 2015 – 63 p.
26. HowExpert, Zachary Hestand How To Use a 3D Printer / HowExpert – California, 2016 – 25 p.
27. Jeremy Blum Exploring Arduino: Tools and Techniques for Engineering Wizardry / Wiley – New Jersey, 2019 – 512 p.
28. Michail Kölling Raspberry PI: A complete guide to start learning RaspberryPi on your own. Learn an easy way to setup and build your projects, avoid common mistakes, and develop solid skills in computer technology / Independently Published – Bremen, 2020 – 95 p.
29. Martin Campbell-Kelly Computer: A History of the Information Machine (The Sloan Technology Series) / Routledge – Abingdon, 2018 – 360 p.
30. Sean Aranda 3D Printing Failures: 2019 Edition: How to Diagnose and Repair ALL Desktop 3D Printing Issues / Make Community, LLC – New York, 2018. – 291 p.
31. William M. Springer II A Programmer's Guide to Computer Science: A virtual degree for the self-taught developer / Jaxson Media – Boston, 2019 – 190 p.
32. Henry Segerman Visualizing Mathematics with 3D Printing / Johns Hopkins University Press – Baltimore, 2016 – 200 p.
33. Dan Phillips Linux: Learn the Ultimate Strategies to Master Operating System and Command Line. Improve Your Computer Programming Skills and Start Coding / Independently Published – New York, 2020 – 146p.
34. Matthew Kent So You Want to Buy a 3d Printer / Peragrine Ebook Press – Bonn, 2015, 61 p.
35. Edward A. Bender An Introduction to Mathematical Modeling (Dover Books on Computer Science) / Dover Publications (Educa Books) – Mineola, 2000 – 272p.
36. Wladston Ferreira Filho Computer Science Distilled: Learn the Art of Solving Computational Problems / Code Energy LLC – Las Vegas, 2017 – 175 p.
37. John Resig, Bear Bibeault, Josip Maras Secrets of the JavaScript Ninja / Manning Publications – Shelter Island, New York 2016 – 464 p.

38. Joel Kilty, Alex McAllister *Mathematical Modeling and Applied Calculus* / OUP Oxford – Oxford, 2018 – 816 p.

39. Craig Berg *Raspberry Pi 4 For Beginners And Intermediates: A Comprehensive Guide for Beginner and Intermediates to Master the New Raspberry Pi 4 and Set up Innovative Projects* / Independently Published – Milwaukee, 2020 – 135 p.

40. Federico Kereki *Mastering JavaScript Functional Programming: In-depth guide for writing robust and maintainable JavaScript code in ES8 and beyond* / Packt Publishing – Birmingham, 2017 – 386 p.

41. Nathan Rozentals *Mastering TypeScript 3: Build enterprise-ready, industrial-strength web applications using TypeScript 3 and modern frameworks* / Packt Publishing – Birmingham, 2014 – 207 p.

42. M Buth *3D Printer: Patents & Innovations* / Make Community, LLC – New York, 2018. – 291 p.

43. Yakov Fain, Anton Moiseev *Angular Development with Typescript* / Manning Publications – Shelter Island, New York 2018 – 560 p.

44. Steve Fenton *Pro TypeScript: Application-Scale JavaScript Development* / Apress – New York City, 2017 – 320 p.

45. Asim Hussain *Angular 5: From Theory To Practice: Build the web applications of tomorrow using the new Angular web framework from Google* / CodeCraft – Sunnyvale, 2017 – 846 p.

46. Steven Bradley *Sass for Beginners: How to Write More Organized and Maintainable Stylesheets* / Steven Bradley – San Francisco, California, 2018 – 275 p.

47. Luke Watts *Mastering Sass* / Packt Publishing – Birmingham, 2016 – 318 p.

48. Robert Davis *Inexpensive 3D Printer Projects: How to build your own 3D printer and accessories* / Que Publishing – London, 2014 – 112 p.

49. C. Sean Bohun, Samantha McCollum, Thea van Roode, Reinhard Illner *Mathematical Modelling: A case studies approach (Student Mathematical Library)* / American Mathematical Society – Providence, 2002 – 196 p.

50. Gerardus Blokdyk *Auth0 A Complete Guide / 5STARCooks* – New York, 2018 – 168 p.

51. Michael Margolis, Brian Jepson, Nicholas Robert Weldin *Arduino Cookbook: Recipes to Begin, Expand, and Enhance Your Projects / O'Reilly Media* – Sebastopol, California, 2020 – 796 p.

52. Richard Petersen *Linux: The Complete Reference / Independently Published* – London, 2020 – 597 p.

53. Matt Timmons-Brown *Learn Robotics with Raspberry Pi: Build and Code Your Own Moving, Sensing, Thinking Robots / No Starch Press* – San Francisco, California, 2019 – 240 p.

54. Robert Sedgewick *Computer Science: An Interdisciplinary Approach / Addison-Wesley Professional* – Boston, 2016 – 1168 p.

55. Jos Dirksen *Learn Three.js: Programming 3D animations and visualizations for the web with HTML5 and WebGL / Packt Publishing* – Birmingham, 2018 – 528 p.

56. Jos Dirksen *Three.js Cookbook / Packt Publishing* – Birmingham, 2015 – 302 p.

57. Tony Parisi *Programming 3D Applications with HTML5 and WebGL: 3D Animation and Visualization for Web Pages / O'Reilly Media* – Sebastopol, California, 2014 – 404 p.

58. David Herron *Node.js Web Development: Server-side development with Node 10 made easy / Packt Publishing* – Birmingham, 2018 – 492 p.

59. Jim Wilson *Node.js 8 the Right Way: Practical, Server-Side JavaScript That Scales / Pragmatic Bookshelf* – Raleigh, 2018 – 336 p.

60. Lydia Sloan Cline *3D Printer Projects for Makerspaces / McGraw-Hill Education TAB* – New York 2017 – 352 p.

61. Andrew Mead *Learning Node.js Development: Learn the fundamentals of Node.js, and deploy and test Node.js applications on the web / Packt Publishing* – Birmingham, 2018 – 658 p.

62. Nicholas C. Zakas *Understanding ECMAScript 6: The Definitive Guide for JavaScript Developers* / No Starch Press – San Francisco, California, 2016 – 352 p.
63. Jeffry Houser *Learn With: Angular 7, Bootstrap, and NodeJS: Enterprise Application Development with Angular 7 and NodeJS* / DotComIt – Connecticut, 2018 – 262 p.
64. Anna Kaziunas France *Make: 3D Printing: The Essential Guide to 3D Printers* / Maker Media, Inc – New York, 2013 – 471 p.
65. Russell Scott *Computer Networking: This Book Includes: Computer Networking for Beginners and Beginners Guide (All in One)* / Independently Published – London, 2019 – 361 p.
66. Mark Meerschaert *Mathematical Modeling* / Academic Press – Cambridge, 2013 – 384 p.
67. Richard Horne, Kalani Kirk Hausman *3D Printing For Dummies (For Dummies (Computers))* / For Dummies – Hoboken, 2017 – 408 p.
68. John L. Hennessy, David A. Patterson *Computer Architecture: A Quantitative Approach (ISSN)* / Morgan Kaufmann – Burlington, 2017 – 936 p.
69. Simon Monk *Raspberry Pi Cookbook: Software and Hardware Problems and Solutions* / O'Reilly Media – Sebastopol, California, 2019 – 610 p.
70. Simon Monk *Programming Arduino Next Steps: Going Further with Sketches, Second Edition* McGraw-Hill Education TAB – New York 2018 – 320 p.

SOURCE CODE**flicer-api**

```
app.js
'use strict';

const port = 5080;

const server = require('./lib/Server');

server.create({
  port
});

server.start();

controllers/auth.js
'use strict';

const auth = require('../lib/Authorization');

class AuthController {

  /**
   * Signs user in
   * @param { IncomingMessage } req
   * @param { ServerResponse } res
   * @param { function } next
   */
  async signIn(req, res, next) {
    try {
      res.json(await auth.signIn(req.body));
    } catch (error) {
      next(error);
    }
  }

  /**
   * Signs user up
   * @param { IncomingMessage } req
   * @param { ServerResponse } res
   * @param { function } next
   */
  async signUp(req, res, next) {
    try {
      res.json(await auth.signUp(req.body));
    } catch (error) {
      next(error);
    }
  }
}

module.exports = exports = new AuthController();

controllers/config-manager
```

```

'use strict';

const printerHelper = require('../helpers/db/PrinterHelper');
const profileHelper = require('../helpers/db/PrintProfileHelper');

const ConfigManager = require('../lib/ConfigManager');

class ConfigManagerController {

  /**
   * Compiles lists of possible configs
   * @param { IncomingMessage } req
   * @param { ServerResponse } res
   * @param { function } next
   */
  async listConfigs(req, res, next) {
    const { user } = req;

    console.log(user);

    try {
      const printers = await user.getPrinters();
      const profiles = await user.getPrintProfiles();

      const printersList = printers.map(printer => ({
        id: printer.id,
        name: printer.name
      }));

      const profilesList = profiles.map(profile => ({
        id: profile.id,
        name: profile.name
      }));

      return res.json({
        printers: printersList,
        profiles: profilesList
      });
    } catch (error) {
      return next(error);
    }
  }

  /**
   * Retrieves printer(s) record(s) from database
   * @param { IncomingMessage } req
   * @param { ServerResponse } res
   * @param { function } next
   */
  async getConfigs(req, res, next) {
    const { printerId, profileId } = req.body;
    const { user } = req;

    try {
      const printer = await printerHelper.getUserPrinter(user,
printerId);
      const profile = await profileHelper.getUserPrintProfile(user,
profileId);

      const manager = new ConfigManager(printer.path);
      const diff = Object.assign({}, printer.config,
profile.config);

      const allowed = manager.getAllowed(diff);

```



```

        printer.config = allowed.machine;
        profile.config = allowed.print;

        return res.json({
            printer,
            profile
        });
    } catch (error) {
        return next(error);
    }
}

/**
 * Updates configs
 * @param { IncomingMessage } req
 * @param { ServerResponse } res
 * @param { function } next
 */
async updateConfigs(req, res, next) {
    req.body;
    const { printerId, profileId, printerConfig, profileConfig } =
    req.body;
    const { user } = req;

    try {
        const printer = await printerHelper.getUserPrinter(user,
printerId);
        const profile = await profileHelper.getUserPrintProfile(user,
profileId);

        profile.config = profileConfig;
        printer.config = printerConfig;

        await printer.save();
        await profile.save();

        const manager = new ConfigManager(printer.path);
        const diff = Object.assign({}, printerConfig, profileConfig);

        const allowed = manager.getAllowed(diff);

        printer.config = allowed.machine;
        profile.config = allowed.print;

        return res.json({
            printer,
            profile
        });
    } catch (error) {
        return next(error);
    }
}

/**
 * Recomputes configs
 * @param { IncomingMessage } req
 * @param { ServerResponse } res
 * @param { function } next
 */
recomputeConfigs(req, res, next) {
    const { printerConfig, profileConfig, path, dependency } = req.body;

    const manager = new ConfigManager(path);
    const diff = Object.assign({}, printerConfig, profileConfig);

```

```

    const allowed = manager.getAllowedDependants(dependency, diff);

    res.json({
      printer: {
        config: allowed.machine
      },
      profile: {
        config: allowed.print
      }
    });
  }
}

module.exports = exports = new ConfigManagerController();

controllers/mail.js
'use strict';

const mailer = require('../lib/Mailer');

class MailController {
  /**
   * Sends message
   * @param { IncomingMessage } req
   * @param { ServerResponse } res
   * @param { function } next
   */
  async send(req, res, next) {
    const { message } = req.body;

    try {
      await mailer.send(message);
      return res.sendStatus(200);
    } catch (error) {
      next(error);
    }
  }
}

module.exports = exports = new MailController();

controllers/slicer-process.js
'use strict';
const fs = require('fs');
const path = require('path');

const { Slice } = require('../models').models;

const printerHelper = require('../helpers/db/PrinterHelper');
const profileHelper = require('../helpers/db/PrintProfileHelper');

const sliceHelper = require('../helpers/db/SliceHelper');
const metricHelper = require('../helpers/db/MetricHelper');
const figureHelper = require('../helpers/db/FigureHelper');
const previewHelper = require('../helpers/db/PreviewHelper');

const pubSub = require('../lib/PubSub');
const storage = require('../lib/Storage');
const ConfigManager = require('../lib/ConfigManager');

```

```

// Retrieve statuses
const statusesPath = path.join(__dirname, '..', 'config', 'statuses.json');
const statuses = JSON.parse(
  fs.readFileSync(statusesPath, 'utf8')
).reduce((map, status) => {
  map[status.name] = status.statusId;
  return map;
}, {});

class SliceProcessController {

  constructor() {
    this.extension = '.gcode';
  }

  /**
   * Retrieves slice's record
   * @param { IncomingMessage } req
   * @param { ServerResponse } res
   * @param { function } next
   */
  async get(req, res, next) {
    const { id } = req.params;
    const { user } = req;

    try {
      const slice = await sliceHelper.getUserSlice(user, id);
      let preview;

      if (slice.statusId === statuses.completed) {
        preview = await previewHelper.getSlicePreview(slice);
      }

      res.json({
        slice,
        downloadURL: preview && await
storage.getReadUrl(preview.name)
      });
    } catch (error) {
      next(error);
    }
  }

  /**
   * Initializes slice process
   * @param { IncomingMessage } req
   * @param { ServerResponse } res
   * @param { function } next
   */
  async init(req, res, next) {
    const { user } = req;
    const { name, figures, figuresRenderingTime } = req.body;

    try {
      const slice = await user.createSlice({
        name,
        statusId: statuses.uploading
      });

      const parsedFigures = await this._parseFigure(figures,
slice.id);
      await this._createFiguresForSlice(slice, parsedFigures);

```

```

        const uploadURLs = parsedFigures.map(urls => urls.uploadURL);

        const parsedName = name.replace(/.{32}/, ''); // Remove added
uuid      const previewName =
` ${slice.id}/${parsedName}`.replace(/\.+$/, this.extension); // Replace
extension

        await slice.createPreview({
            name: previewName
        });

        await slice.createMetric({
            figuresRenderingTime
        });

        return res.json({
            slice,
            uploadURLs
        });
    } catch (error) {
        return next(error);
    }
}

/**
 * Sends request to start specified slice-process
 * @param { IncomingMessage } req
 * @param { ServerResponse } res
 * @param { function } next
 */
async start(req, res, next) {
    const { id } = req.params;
    const { user } = req;
    const { printerId, profileId, figuresUploadingTime } = req.body;

    try {
        const slice = await sliceHelper.getUserSlice(user, id);
        const printer = await printerHelper.getUserPrinter(user,
printerId);
        const profile = await profileHelper.getUserPrintProfile(user,
profileId);

        const figures = await figureHelper.getSliceFigures(slice);
        const preview = await previewHelper.getSlicePreview(slice);

        await slice.update({
            statusId: statuses.pending
        });

        const manager = new ConfigManager(printer.path);
        const diff = Object.assign({}, printer.config,
profile.config);

        const config = manager.computeRealDiff(diff);

        const metric = await metricHelper.getSliceMetric(slice);

        await metric.update({
            figuresUploadingTime,
            slicePendingTimeStart: Date.now()
        });

        pubSub.publish('pendingSlices', {

```

```

        preview,
        figures,
        config,
        sliceId: slice.id,
        configName: printer.path
    });

    res.json(slice);
} catch (error) {
    return next(error);
}
}

// TODO: Discuss error handling while working with pubsub
/**
 * Updates slice's record after slice-process has been started
 * @param { object } process Object, which describes current state of
slice-process
 * @param { number } process.sliceId
 * @param { number } process.figuresDownloadingTime
 */
async started(process) {
    const { sliceId, figuresDownloadingTime } = process;

    const slice = await Slice.findById(sliceId);
    const metric = await metricHelper.getSliceMetric(slice);

    await metric.update({
        figuresDownloadingTime,
        slicePendingTimeEnd: Date.now(),
        slicingTimeStart: Date.now()
    });

    await slice.update({
        statusId: statuses.started
    });
}

/**
 * Updates slice's during slice-process
 * @param { object } process Object, which describes current state of
slice-process
 * @param { number } process.sliceId
 * @param { number } process.progress
 */
async active(process) {
    const slice = await Slice.findById(process.sliceId);
    const { progress } = process;

    await slice.update({
        progress,
        statusId: statuses.active
    });
}

/**
 * Updates slice's record after slice-process has been completed
 * @param { number } process.sliceId
 * @param { number } process.size
 * @param { number } process.previewUploadingTime
 */
async completed(process) {
    const { sliceId, size, previewUploadingTime } = process;

```

```

const slice = await Slice.findById(sliceId);
await slice.update({
  progress: null,
  statusId: statuses.completed
});

const metric = await metricHelper.getSliceMetric(slice);
await metric.update({
  previewUploadingTime,
  slicingTimeEnd: Date.now()
});

const preview = await previewHelper.getSlicePreview(slice);
await preview.update({
  size
});
}

/**
 * Updates slice's record if slice-process has failed
 * @param { object } process Object, which describes current state of
slice-process
 * @param { number } process.sliceId
 * @param { (string | object) } process.error Error, which lead to slice-
process' failure
 */
async failed(process) {
  const slice = await Slice.findById(process.sliceId);
  const { error } = process;

  await slice.update({
    error,
    statusId: statuses.failed
  });
}

/**
 * Manually cancels slice-process
 * @param { IncomingMessage } req
 * @param { ServerResponse } res
 * @param { function } next
 */
async cancel(req, res, next) {
  const { id } = req.params;
  const { user } = req;

  try {
    const slice = await sliceHelper.getUserSlice(user, id);

    if (
      slice.statusId === statuses.started
      || slice.statusId === statuses.active
    ) {
      pubSub.publish('canceledSlices', { id });
    }

    await slice.update({
      statusId: statuses.canceled
    });

    res.json(slice);
  } catch (error) {
    return next(error);
  }
}

```

```

}

/**
 * Marks slice as seen
 * @param { IncomingMessage } req
 * @param { ServerResponse } res
 * @param { function } next
 */
async seen(req, res, next) {
  const { id } = req.params;
  const { user, body } = req;

  try {
    const slice = await sliceHelper.getUserSlice(user, id);
    const metric = await metricHelper.getSliceMetric(slice);

    const { previewDownloadingTime, previewRenderingTime } = body;

    await metric.update({
      previewDownloadingTime,
      previewRenderingTime,
      seen: true
    });

    res.json(slice);
  } catch (error) {
    return next(error);
  }
}

/**
 * Asynchronously parses given array of figures
 * @
 * @param { object[] } figures
 * @property { string } name
 * @property { object } config
 * @param { number } sliceId
 * @returns { Promise<object[]> } Array of parsed figure, which can be
used to create Figure records
 */
_parseFigure(figures, sliceId) {
  return Promise.all(
    figures.map(async figure => {
      const name = `${sliceId}/${figure.name}`;
      const uploadURL = await storage.getWriteUrl(name);
      return {
        name,
        uploadURL,
        config: figure.config,
        size: figure.size
      };
    })
  );
}

/**
 * Creates Figures records from parsed figures, associated with given
slice
 * @param { Slice } slice
 * @param { object[] } parsedFigures
 */
async _createFiguresForSlice(slice, parsedFigures) {
  await Promise.all(

```

```

        parsedFigures.map(async figure => await
slice.createFigure(figure))
    );
}

/**
 * Merges given bunch of configs into single one
 * @param { ...Object } configs
 * @returns { Object }
 */
mergeConfigs(...configs) {
    return Object.assign({}, ...configs);
}
}
module.exports = exports = new SliceProcessController();

controllers/slices.js
'use strict';

const sliceHelper = require('../helpers/db/SliceHelper');

class SlicesController {

    /**
     * Retrieves slice(s) record(s) from database
     * @param { IncomingMessage } req
     * @param { ServerResponse } res
     * @param { function } next
     */
    async get(req, res, next) {
        const { id } = req.params;
        const { user } = req;

        try {
            res.json(await sliceHelper.getUserSlice(user, id));
        } catch (error) {
            return next(error);
        }
    }

    /**
     * Updates specified slice record
     * @param { IncomingMessage } req
     * @param { ServerResponse } res
     * @param { function } next
     */
    async update(req, res, next) {
        const { id } = req.params;
        const { user } = req;

        const name = req.body.name || req.query.name;

        try {
            const slice = await sliceHelper.getUserSlice(user, id);

            await slice.update({
                name
            });
        } catch (error) {
            return next(error);
        }

        res.sendStatus(201);
    }
}

```



```

/**
 * Deletes specified slice record
 * @param { IncomingMessage } req
 * @param { ServerResponse } res
 * @param { function } next
 */
async delete(req, res, next) {
  const { id } = req.params;
  const { user } = req;

  try {
    const slice = await sliceHelper.getUserSlice(user, id);

    await slice.destroy();
  } catch (error) {
    return next(error);
  }

  res.sendStatus(204);
}
}

module.exports = exports = new SlicesController();

controllers/users.js
'use strict';

const userHelper = require('../helpers/db/UserHelper');

class UsersController {

  /**
   * Retrieves user(s) record(s) from database
   * @param { IncomingMessage } req
   * @param { ServerResponse } res
   * @param { function } next
   */
  async get(req, res, next) {
    const { id } = req.params;

    try {
      if (id) {
        return res.json(await userHelper.getUser(id));
      }

      res.json(await userHelper.getUsers());
    } catch (error) {
      return next(error);
    }
  }

  // ToDo implement or remove
  async create(req, res, next) {
    res.sendStatus(500);
  }

  /**
   * Updates specified user record
   * @param { IncomingMessage } req
   * @param { ServerResponse } res
   * @param { function } next
   */
}

```

```

async update(req, res, next) {
  const { id } = req.params;
  const email = req.body.email || req.query.email;

  try {
    const user = await userHelper.getUser(id);

    await user.update({
      email,
    });
  } catch (error) {
    return next(error);
  }

  res.sendStatus(201);
}

/**
 * Deletes specified user record
 * @param { IncomingMessage } req
 * @param { ServerResponse } res
 * @param { function } next
 */
async delete(req, res, next) {
  const { id } = req.params;

  try {
    const user = await userHelper.getUser(id);
    await user.destroy();
  } catch (error) {
    return next(error);
  }

  res.sendStatus(204);
}
}

module.exports = exports = new UsersController();

helpers/ConfigHelper.js
'use strict';

class ConfigHelper {

  /**
   * Recursevily looks up for field in settings
   * @param { string } field
   * @param { Object } config
   */
  findField(field, settings) {
    let result;

    for (const category in settings) {
      result = this.findInCategory(field, settings[category]);

      if (result) {
        return result;
      }
    }
  }

  /**
   * Looks up for field in category

```

```

    * @param { string } field
    * @param { Object } category
    */
    findInCategory(field, category) {
        let result;

        if (category.children && category.children[field]) {
            return category.children[field];
        }

        for (const child in category.children) {
            if (
                typeof category.children[child] === 'object'
                && category.children[child].children
            ) {
                result = this.findInCategory(field,
category.children[child]);

                if (result) {
                    return result;
                }
            }
        }
    }

    /**
     * Looks up for all leafs in settings
     * @param { Object } settings
     */
    findLeafs(settings) {
        const leafs = [];

        const fields = this.getFieldsList(settings);

        for (const fieldName of fields) {
            const field = this.findField(fieldName, settings);
            if (!field.children && field.valueStatement) {
                leafs.push(field);
            }
        }

        return leafs;
    }

    /**
     * Looks up for all leaf dependants of dependency
     * @param { Object } dependency
     * @param { Object } settings
     */
    findLeafDependants(dependency, settings) {
        if (!dependency.dependants) {
            return [];
        }

        const dependantsFields = dependency.dependants.map(
            dependant => this.findField(dependant, settings)
        );

        const leafs = dependantsFields.filter(dependant =>
!dependant.children);
        const dependencies = dependantsFields.filter(dependant =>
dependant.children);

        for (const dependant of dependencies) {

```

```

        leafs.push(...this.findLeafDependants(dependant, settings))
    }

    const filteredLeaves = leafs.filter((leaf, index, array) =>
        index === array.findIndex(l => (
            leaf.label === l.label
        )))
    );

    return filteredLeaves;
}

/**
 * Looks up for all dependants of dependency
 * @param { Object } dependency
 * @param { Object } settings
 */
findAllDependants(dependency, settings) {
    if (!dependency.dependants) {
        return [];
    }

    const dependantsFields = dependency.dependants.map(
        dependant => this.findField(dependant, settings)
    );

    const dependants = [...dependantsFields];
    const dependencies = dependantsFields.filter(
        dependant => dependant && (dependant.children ||
dependant.dependants)
    );

    for (const dependant of dependencies) {
        if (
            dependency.dependencies
            && dependency.dependencies.includes(dependant.name)
        ) {
            continue;
        }
        dependants.push(...this.findAllDependants(dependant,
settings));
    }

    const filteredDependants = dependants.filter((dependant, index,
array) =>
        dependant && index === array.findIndex(l => (
            dependant.label === l.label
        )))
    );

    return filteredDependants;
}

/**
 * Recursevily compiles list of categories' fields
 * @param { Object } settings
 * @param { Object[] } target Target array
 */
categoryFieldsList(settings, target = []) {
    for (const field in settings) {
        target.push(field);

        if (settings[field].children) {

```

```

        this.categoryFieldsList(settings[field].children,
target);
    }
}

/**
 * Recursevily compiles list of settings' fields
 * @param { Object } settings
 */
getFieldsList(settings) {
    const fields = [];

    for (const category in settings) {
        this.categoryFieldsList(settings[category].children, fields);
    }

    return fields;
}

/**
 * Checks if value includes any dependencies, based on fields' list
 * @param { string } value
 * @param { string[] } fieldsList
 */
valueIncludesDependencies(value, fieldsList) {
    if (typeof value !== 'string') {
        return false;
    }

    for (const field of fieldsList) {
        const hasDependenciesRegExp = new
RegExp(`(?: (?<!(?!("|'|))\\b${field}\\b|\\b${field}\\b(?:?!("|'|))))`);
        const hasResolveOrValueDependenciesRegExp =
/resolveOrValue\\(\\w+(\\w+)_\\w+\\)/;

        if (
            hasDependenciesRegExp.test(value)
            || hasResolveOrValueDependenciesRegExp.test(value)
        ) {
            return true;
        }
    }

    return false;
}
}

module.exports = exports = new ConfigHelper();

helpers/db/FigureHelper.js
'use strict';

class FigureHelper {

    /**
    * Retrieves slice's figures
    * @param { Slice }
    */
    async getSliceFigures(slice) {
        try {
            const figures = await slice.getFigures();

```

```

        if (!figures || figures.length === 0) {
            throw new Error('No figures found');
        }

        return figures;
    } catch (error) {
        throw error;
    }
}

module.exports = exports = new FigureHelper();

helpers/MetricHelper.js
'use strict';

class MetricHelper {

    /**
    * Retrieves slice's metric
    * @param { Slice }
    */
    async getSliceMetric(slice) {
        try {
            const metric = await slice.getMetric();

            if (!metric) {
                throw new Error('No metric found');
            }

            return metric;
        } catch (error) {
            throw error;
        }
    }
}

module.exports = exports = new MetricHelper();

helpers/PreviewHelper.js
'use strict';
class PreviewHelper {

    /**
    * Retrieves slice's preview
    * @param { Slice }
    */
    async getSlicePreview(slice) {
        try {
            const preview = await slice.getPreview();

            if (!preview) {
                throw new Error('No preview found');
            }

            return preview;
        } catch (error) {
            throw error;
        }
    }
}

module.exports = exports = new PreviewHelper();

```

...

ВІДГУК
керівника економічного розділу
на кваліфікаційну роботу магістра
на тему: «Проектування хмарного апаратно-програмного комплексу
для 3D друку»
студента групи 122м-19-1 Рябичева Олега Олеговича

Керівник економічного розділу
доцент каф. ПЕП та ПУ, к.е.н.

Л. В. Касьяненко

LIST OF FILES ON THE DISK

File name	Description
Explanatory documents	
Diploma_Riabychev.doc	Explanatory note to the Diploma Project. Word document.
Diploma_Riabychev.pdf	Explanatory note to the Diploma Project in PDF format
Presentation	
Presentation_Riabychev.ppt	Presentation of Diploma Project