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

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Єгорова Владислава Миколайовича*
(ПІБ)

академічної групи *122-18ск-2*
(шифр)

напряму підготовки *122 Комп'ютерні науки*
(код і назва напряму підготовки)

на тему: *Розробка десктопного додатка для створення і редагування циферблатів серії Mi Band на .NET Core 3.1 і WPF*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*Приходченко С.Д.*			
розділів:				
спеціальний	*Приходченко С.Д.*			
економічний	*Вагонова О.Г.*			
Рецензент				
Нормоконтролер	*Гуліна І.Г.*			

Дніпро
2021

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

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

« » _____ 20____ року

ЗАВДАННЯ
на дипломний проект
бакалавра
(назва освітньо-кваліфікаційного рівня)

студенту *122-18ск-2* (група) *Єгорова Владислава Миколайовича* (прізвище та ініціали)

тема дипломного проекту *Розробка десктопного додатка для створення і редагування циферблатів серії Mi Band на .NET Core 3.1 і WPF*

затверджена наказом ректора НТУ «ДП» від . №

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***24.05.2021 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПО й тривалості його розробки***	***27.05.2021 р.***

Завдання видав _____ *доц. Приходченко С.Д.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Єгоров В.М.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *15.02.2021 р.*

Термін подання кваліфікаційної роботи до ЕК: *. .20 р.*

РЕФЕРАТ

Пояснювальна записка: ___ с., ___ рис., ___дод., ___джерел.

Об'єкт розробки: Розробка десктопного додатка для створення і редагування циферблатів серії Mi Band на .NET Core 3.1 і WPF.

Мета дипломного проекту: розробка десктопного додатка для створення і редагування циферблатів, що допоможе споживачу не тільки користуватися розробленим циферблатом, а ще і змінювати різні формати циферблату на свій смак. Дасть можливість проявити фантазію користувачу.

У вступі розглядається сучасний стан та аналіз проблеми, уточнюється мета та галузь застосування кваліфікаційної роботи, наведено аргументування актуальності теми та визначається постановка завдання.

У першому розділі досліджується предметна галузь, визначається актуальність завдання та призначення розробки, розроблюється постановка завдання, зазначаються вимоги до її програмної реалізації.

У другому розділі розглядаються наявні рішення, обирається платформа для розробки, проектування і розробка програми. Наводиться опис роботи програми, її алгоритма та структура функціонування. Визначаються вхідні і вихідні дані програми, вказуються характеристики склад параметрів технічних засобів.

В економічному розділі визначається трудомісткість розробленого програмного продукта, виконується підрахунок вартості роботи по створенню програми додатка та розрахунок часу на його розробку.

Практичне значення полягає у створенні десктопного додатка для створення і редагування циферблатів , що надає можливість споживачу безкоштовно користуватися даним продуктом, та програмісту збільшити свої знання в цій області.

Актуальність даного програмного продукту визначається великим попитом на подібні десктопні додатки до циферблатів, що користуються великою популярністю серед молоді.

Список ключових слів: АЛГОРИТМ, ГОДИННИК, ПРОГРАМА, ЦИФЕРБЛАТ, ДОДАТОК, ФІТНЕС, БРАСЛЕТ, РЕДАКТОР, РЕНДЕР, АКСЕСУАР, ПРИСТРІЙ, КОНВЕРТЕР, XMI WATCH, ПРОЕКТ, WPF, C #, ФІТНЕС-БРАСЛЕТ, КРОКОМІР, КРОКИ, КАЛОРІЇ, ВІДСТАНЬ, ДИСТАНЦІЯ, БУДИЛЬНИК, СЕКУНДОМІР, АКСЕЛЕРОМЕТР , РУХ, ДИЗАЙН, XIAOMI, MI BAND, BAND.

ABSTRACT

Explanatory note: ___ pp., ___ fig., ___ appendix, ___ sources.

Packaging object: Development of an additional application for creating and editing Mi Band dials on .NET Core 3.1 and WPF.

The purpose of the diploma project: to develop a desktop add-on for creating and editing dials, which helps the consumer not a single member of the marked dials, as well as changes the different formats of dials to your liking. Then you can discover the user's imagination.

The introduction considers the current state and analysis of problems, clarifies the purpose and scope of qualification work, provides an argument for the relevance of topics and defines the problem.

In the first section the subject branch is investigated, the urgency of the task and purpose of development is defined, statement of the task is developed, requirements to its program realization are defined.

Another section discusses the available solutions, uses a platform for development, design and development of programs. The description of work of programs, its algorithms and structure of functioning is given. Input and output programs of programs are determined, which indicate the characteristics of the parameters of technical means.

The economic section determines the complexity of the developed software product, calculates the cost of work for the creation of applications and development time for its development.

Practical value should be created in the desktop application for creating and editing dials, which allows you to use the services of a free participant in this product and program to increase their knowledge in this area.

The relevance of this software product is determined by the high demand for such desktop applications for dials, which are very popular among young people.

Keyword list: ALGORITHMS, CLOCK, PROGRAM DIAL, ANNEX, FITNESS, BRACELETS, EDITOR, RENDER, ACCESSORIES, DEVICES, CONVERTERS, XMI WATCH, COMPANY, COMPANY, WPF DISTANCE, ALARM CLOCK, STOPWATCH, ACCELEROMETER, MOTION, DESIGN, XIAOMI, MI BAND, BAND.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

СМС (англ. short message service) – послуга коротких повідомленью;

ЖК (LCD) – плоский монітор на основі рідких кристалів;

GPS – Global Positioning System;

WPF – Windows Presentation Foundation;

API – Application Programming Interface;

XAML – Extensible Application Markup Language;

MVVM – Model–View–ViewModel;

DnD – Drag’n’Drop;

UI – User Interface;

UX – User Experience;

JIT – Just-In-Time compiler;

IL – Intermediate Language;

CLR – Common Language Runtime;

GC – Garbage Collector;

АОТ – аспектно-орієнтоване програмування;

IO – Input-Output;

ЕОМ – електронна обчислювальна машина;

SDK – Software Development Kit;

CLI – Command-Line Interface.

ВСТУП

Завдання дипломного проекту відповідає розкритті узагальненій тематиці кваліфікаційних робіт і переліку зазначених типових задач діяльності, що пов'язані безпосередньо з напрямом підготовки, якими повинні володіти бакалаври напряму 122 «Комп’ютерні науки».

Завдання даного дипломного проекту та об'єкт його діяльності безпосередньо пов'язані з напрямом підготовки та відповідає узагальненій тематиці кваліфікаційних робіт і переліку зазначених виробничих функцій, типових задач діяльності, умінню та компетенціям, якими повинні володіти бакалаври напряму 122 «Комп’ютерні науки». Виконання кваліфікаційної роботи надає можливості отримання автору кваліфікації «фахівець в галузі інформаційних технологій».

Тематикою дипломного проекту є розробка десктопного додатка для створення і редагування циферблатів серії Mi Band на .NET Core 3.1 і WPF.

Метою кваліфікаційної роботи є вивчення засобів роботи з багатьма підсистемами операційної системи. Перш за все це події вводу в системі та взаємодія з ними. Ці навички є дуже актуальними в сфері розробки програмного забезпечення для контролю доступу та моніторингу активності процесів та користувачів. Також вони допоможуть краще зрозуміти механізми роботи з графічними об’єктами та їх взаємодію на досить низькому рівні.

Дана робота дозволяє познайомитись з методами розробок десктопного додатка для створення і редагування циферблатів.

Додаток-циферблат створюється як і звичайні програми з призначеним для користувача інтерфейсом. Відрізняється тільки створення базового вікна для циферблата, тому що воно знаходиться в спеціальному місці в графічній оболонці годин. У програмі можливо використовувати всі ті ж віджети, сервіси та датчики (GPS, датчик освітлення та ін.), викликати вібрацію пристрої, посилати повідомлення.

Отже завданням даної кваліфікаційної роботи є створення десктопного додатку циферблатів, що сприяє зручності використання користувача.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

В наш час прогрес не стоїть на місці. Велику популярність отримали розумні аксесуари. На сьогоднішній день створення різноманітних додатків до циферблату користуються великим попитом на ринку.

Дуже часто на руці багатьох людей можна помітити якийсь "розумний" аксесуар. То може бути розумний годинник, схожій на смартфон, у інших фітнес-браслет, в залежності від переслідуваних цілей.

Фітнес-браслет, він же має різні назви: смарт-браслет, трекер активності фітнес-трекер.Це електронний пристрій, який допомагає контролювати деякі види активності, а саме: ходьба, біг, відстань, частота пульсу. Також він може виконувати деякі корисні і зручні функції, такі як повідомляти про повідомлення та дзвінки, нагадувати про активність і т.д.

Перші моделі фітнес-браслетів призначалися в основному для вимірювання пульсу, пізніше додалися функції витрачених калорій і підрахунку кроків. Прогрес не стоїть на місці, зараз сучасні фітнес-браслети можуть виконувати практично ті ж функції, що і смарт-годинник.

Фітнес-браслети – це пристрої, які допомагають продовж всього дня стежити за собою, контролювати будь-які зміни в роботі організму і змушувати себе більше рухатися. Такий гаджет у деяких випадках навіть може врятувати життя. Він може визначити нехарактерну частоту пульсу або падіння, і відправити тривожний сигнал родичам або лікарям (рис. 1.1).

Рис. 1.1. Приклад фітнес-браслету, контроль пульса

Фітнес-браслети не тільки збирають дані активності та і ведуть їх облік. Ці дані допомогають початківцям спортсменам і людям, які ведуть активний спосіб життя, вибрати свій оптимальний режим дієти, сну, навантажень. Але в першу чергу фітнес-трекери будуть корисні для тих людей, у яких з'являються проблеми зі здоров'ям (гіпотонія, аритмія, гіпертонія).

Функції які виконують фітнес-трекери [25]:

1. Підрахунок калорій, кроків і відстані. У будь-якому фітнес-трекера є акселерометр – датчик автоматично фіксующий рух. На підставі даних якого можливо дізнатися кількість кроків було пройдено за день, відстань і при цьому скільки було витрачено калорій.

2. Моніторинг сну (рис. 1.2). Фактично всі сучасні трекери активності фіксують загальну тривалість сну, розділяють сон на глибокий і швидкий сон, та надають рекомендації щодо покращення сну. Моніторинг сну активується вручную, а деякі вміють це автоматично.

Рис 1.2. Моніторинг сну

3. Розумний будильник. На браслете встановлюється час пробудження і браслет починає вібрувати на руці і будити тільки свого господаря, а не всіх

оточуючих. У деяких моделях пристроїв підтримується функція "розумний" будильник. Тобто будильник спрацьовує не рівно в призначений час, а в той момент, коли сон найменш глибокий.

4. Датчик частоти серцевих скорочень. На ринку вже велика рідкість зустріти фітнес-браслети без датчика пульсу. За його допомогою можна в будь-який момент виміряти частоту пульса або контролювати його на постійній основі. Також встановлюється межі частоти пульсу, та при виході за межі яких, будильник дасть про це знати. Наприклад, при кардионагрузки для схуднення пульс повинен бути не нижче 130 ударів в хвилину. Якщо пульс перевищує ці показники, буде спалюваться м'язова тканина, а якщо нижче, процес спалювання жирів сповільниться. Тоді на допомогу приходить датчик-контролер.

Підрахунок калорій. Браслет з супутнім додатком робить розрахунки спалених за день калорій. Крім спалених калорій гаджет может порахувати спожиті калорії, дізнатися баланс жирів, вуглеводів, і білків. Дуже актуально для тих, що стежать за власною вагою.

1. Повідомлення. Ще одна дуже корисна функція фітнес-браслета. Браслет підключається до смартфону та повідомляє про СМС, вхідні дзвінки, повідомлення з соціальних мереж легкою вібрацією на руці.

Більш "просунуті" моделі смарт браслетів виконують ряд інших корисних функцій:

– вимірювання артеріального тиску;
– вимірювання температури і потовиділення;
– контроль частоти дихання;
– ЕКГ;
– вимірювання рівня кисню в крові;
– визначення рівня стресу;
– віддалене управління камерою смартфона;
– керування музикою смартфона;
– автоматичне відстеження різних спортивних режимів та плавання;
– визначення місця розташування, швидкості і відстані за допомогою вбудованого GPS-модуля;

– безконтактні платежі NFC;

– відображення прогнозу погоди;

– використання браслета в якості музичного плеєра або як Bluetooth-гарнітури (наприклад, Huawei Talkband B5);

– контроль прийому ліків і води;

– відправка даних тренерам і лікарям;

– дисплей робить пристрій функциональней, однак дисплей набагато збільшить витрату енергії батареї з акумулятором та знизить термін його роботи.

Екрани бувають монохромні і кольорові, монохромні екранами споживають менше електроенергії, їх краще видно на сонці.

Екрани поділяються за типом матриці:

– ЖК (LCD) - енерговитратний тип матриці;

– світлодіодні - напівпровідникові LED світлодіоди;

– OLED - тип матриці з високою передачей кольору та чіткістю;

– для поліпшення якості показань використовується різні датчики;

Основні датчики:

– акселерометр (крокомір);

– гироскоп - датчик визначення місця розташування об'єкта;

– альтиметр (висотометр);

– пульсометр;

– п'єзоелементи - функція вимірювання тиску;

– ЕКГ - дозволяє знімати електрокардіограму в одноканальному режимі;

– датчик SpO2 - визначає насичення крові киснем;

– ІК термометр.

В браслети також вбудовані модулі, що поліпшують визначення параметрів та збільшенню функціоналу:

– Bluetooth – синхронізує зі смартфоном та обмінюється з ним даними;

– GPS – навігаційний модуль, для більш точного визначення пройденої відстані та визначення маршруту;

– NFS – модуль безконтактних платежів;

– Wi-Fi – використовується дуже рідко цей модуль, він вивантажує результати в хмарний сервіс;

– вібромотор – подає вібраційний сигнал користувачу.

Фітнес трекер, при наявності в ньому певних функцій, не є медичним приладом. Він має похибки у вимірюваннях. Похибки залежно від датчика, від виробника браслета можуть становити від 2 до 30%.

1.2. Призначення розробки та постановка задачі

Розроблена система, що виконана для кваліфікаційної роботи, має назву «Розробка десктопного додатка для створення і редагування циферблатів серії Mi Band на .NET Core 3.1 і WPF».

Основні терміни та ключові слова:

– Десктопний додаток – це клієнтське програмне забезпечення, він реалізує Windows Presentation Foundation (WPF) інтерфейс. Додаток встановлюєм на робочу станцію користувача і запускається локально. Можливий варіант запуску дескопного додатка з встановленним .NET Core 3.1 Runtime (x86) [1].

– Windows Presentation Foundation (WPF) інтерфейс – це програмування додатків через спеціальний інтерфейс-обгортку над Win API, яка дозволяє розробляти програми набагато швидше і зручніше, використовуючи готовий набір необхідних базових команд для роботи з візуальною і програмною частиною. Візуальна працює на спеціальній мові розмітки XAML, яка дозволяє швидко створювати компоненти програми, а також цілі вікна. У програмній частині закладається необхідний функціонал для цих компонентів. За рахунок цього підходу додаток легко розбивається на маленьки частини і називається такий підхід MVVM патерном [1].

Платформа розробки WPF завжди підтримує широкий різноманітний набір функцій розробки додатків. Таких як: ресурси, модель програми, прив'язку даних, елементи керування, макет, графіку, документи та безпеку. Фреймворк є частиною .NET. Він використовує розширену мову розмітки усіх додатків (XAML) та забезпечиває декларативну модель для програмування додатків [2].

Фреймворк – це програмна платформа, яка визначає структуру програмної системи, що полегшує розробку та об'єднання різних компонентів великого програмного проекту [3].

Розроблений продукт має використовуватися звичайними користувачами як спортсменами так і людям, які ведуть активний спосіб життя, та контролюють свій оптимальний режим навантажень. Але в першу чергу фітнес-трекери будуть корисні для тих людей, у яких з'являються проблеми зі здоров'ям [25].

Призначення розробки – десктопний додаток повинен бути надійним та зручним для кожної людини. Функціональність та простота використання це основне завдання при створенні додатку.

1.3. Підстава для розробки

Відповідно до освітньої програми, в кінці навчання студент повинен виконати кваліфікаційну роботу, згідно графіків навчального процесу та навчального плану.

Тема роботи узгоджується з керівником проекту кафедри «Програмного забезпечення комп’ютерних систем», та затверджується наказом ректора.

Підставами для розробки (виконання кваліфікаційної роботи) є:

– освітня програма 122 «Комп’ютерні науки»;

– навчальний план та графік навчального процесу;

– наказ ректора Національного технічного університету «Дніпровська політехніка» № ____від __.__ 2021 р;

– завдання на кваліфікаційну роботу на тему «Розробка десктопного додатка для створення і редагування циферблатів серії Mi Band на .NET Core 3.1 і WPF».

1.4. Постановка завдання

Метою проекту є розробка програмного забезпечення для створення і редагування циферблатів Mi Band, а також конвертація їх між моделями.

Поточна версія XMI Watch дозволить:

– швидко і комфортно додавати доступні елементи годинника;

– замінювати існуючі елементи іншими зображеннями, обирати можна як один так і дещо не послідовно, через добре пророблений DnD, який відображає при наведенні на компонент додатка, що буде піддаватися зміни або звичайний імпорт файлів;

– копіювати і вставляти зображення між різними відчиненими проектами програми через спеціальний буфер обміну в контексті програми, а також дотримуючись порядок копіювання, наприклад, якщо це були стану 1,2,5, то вони встануть на місце попередніх саме за цими шляхами;

– кастомізувати існуючі або нові елементи через вбудований редактор тексту, який дозволить вибрати оформлення і автоматично зрозуміє, що ви хочете змінити;

– вбудована система конвертації автоматично змінить розширення картинок і їх розміри щодо потрібної моделі, що дозволить заощадити час на портуванні свого циферблата для різних моделей;

– система збереження дозволить зберігати проекти паралельно з візуальною індикацією прогресу, відразу в запечатаному і готовому для вивантаження на годинник вигляді *.bin або розпакованому варіанті *.json + images;

– відкриття аналогічно збереженню і може відкривати файли *.bin / *.json + images відповідно;

– в програмі можливо відкрити до 10 проектів одночасно, таке обмеження встановлене спеціально як захист від зайвого навантаження;

– інтерфейс дає доступ до віртуального макету обраної моделі, де користувач за допомогою маніпулятора миші або клавіатури може управляти розташуванням елементів;

– перемикання між макетами в один клік;

– містить зручну настройку візуального стану для кожного елемента, що дозволяє більш точно опрацьовує кожен компонент.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Кінцевий продукт повинен дотримуватися наступних функціональних вимог:

– зрозумілий і зручний інтерфейс;

– відкривати і обробляти файли циферблатів для підтримуваного модельного ряду Xiaomi Bands, для цього написані окремі продумані модулі конвертації вихідного формату в формат програми і навпаки, а також перетворення з однієї в іншу і зберігати в форматах * .bin / *. json + images, а також в майбутньому з можливістю збереження в власний формат програми *.xmi, де будуть зберігаються ще й додаткова інформація у вигляді стилів, метаданих та іншого;

– авто-оновлення до нових версій XMI через систему контролю версій Git-Hub і спеціальним окремим ПО для цього;

– можливість повноцінно редагувати відкриті або створені ціфферблати, а також модифікувати їх за допомогою допоміжних модулів ПО, таких як зміна вихідних зображень для користувача або через вбудований дизайнер, який рендерить картинки з введеними даними або автоматично визначені за типом обраного елемента;

– зробити доступним відкриття декількох проектів одночасно для обробки і перенесення даних між ними.

1.5.2. Вимоги до інформаційної безпеки.

Оригінальна програма є повністю ізольованим продуктом для будь-якої модифікації, тому що упаковується воно за допомогою засобів .NET Core в single-file light application, яке дозволяє зберігати весь код, ресурси і будь-які інші файли програми прямо в одному exe файлі і при запуску воно використовує їх.

Оновлення відбуваються також безпечно, тому що використовується оригінальний ресурс для перевірки і завантаження нових версій програми

1.5.3. Вимоги до складу та параметрів технічних засобів

Додаток використовує більшість вбудованих можливостей .NET Core 3.1 і тому вимагає установки його runtime версії для відкриття, а також необхідно мати мінімальні характеристики пристрою, на якому буде відбувається запуск:

- обсяг оперативної пам'яті (ОЗУ): 4 ГБ;
- відеоадаптер: 3D адаптер nVidia, Intel, AMD / ATI;
- графічну пам'ять: 512 МБ;
- сховище (HDD / SSD): 25 МБ;
- процесор (CISC x86 / x64): AMD / Intel;
- операційна система: Windows 7/8/10.

1.5.4. Вимоги до інформаційної та програмної сумісності

Основною мовою програмування було використано C#.

Увесь додаток був розробленій у Visual Studio. WPF розмітка компонентів, вікон і в цілому всього інтерфейсу була виконана за допомогою вбудованого редактора XAML з відображенням вихідного результату в реальному часі.

Щоб використати бажано підключення до інтернету для отримання актуальних оновлень.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Результатом даного проекту повинен бути програмний продукт створений на мові програмування C# і платформі WPF на основі .NET Core 3.1, також дотримуватися таких правил по відношенню до циферблатам серії Mi Band:

– відкриття / збереження
– редагування
– конвертація між моделями
– перенесення даних між проектами
– створення нового дизайну всередині програми
– використання сторонніх ресурсів для дизайну
– передперегляд результату в реальному часі
– параметри відображеного передогляду (час, дата, кроки і т.д.)

2.2. Опис застосованих математичних методів.

Додаток використовує прості математичні обчислення у вигляді додавання, віднімання, множення і ділення в алгоритмах рендеру графічного редактора, для генерації зображень з текстом і заданими параметрами для нього, також для реалізації DnD при оновленні ресурсів ззовні програми.

2.3. Опис використаних технологій та мов програмування

В якості фреймворка був обраний .NET Core 3.1 з використанням технології WPF і патерну MVVM, реалізований на XAML розмітки для графічної частини програми і C # для побудови логіки та зв'язку між ui і business logic, що допоможе дотримувати чисту і зрозумілу архітектуру проекту, а також використання патернів проектування більш правильно [4 – 24].

MVVM

Патерн MVVM (Model-View-ViewModel) не забороняє відділяти закономірність додатка від візуальної частини. Даний шаблон програмування насправді є архітектурним, власне кажучи він задає спільну архітектуру програми [5, 6].

Джон Госсман представил даний шаблон програмування в 2005 році як модифікація шаблону Presentation Model. Він з самого початку був націлений на розробку додатків в WPF. Зараз цей патерн вже вийшов далеко за межі WPF. Він застосовується насправді в самих різних технологіях. Також ми його бачимо при розробці під Android, iOS, проте WPF є досить показовою технологією. Ця технологія як можно широко розкриває всі можливості патерну [4].

MVVM складається з трьох компонентів (рис. 2.1): моделі (Model), моделі подання (ViewModel) і уявлення (View).

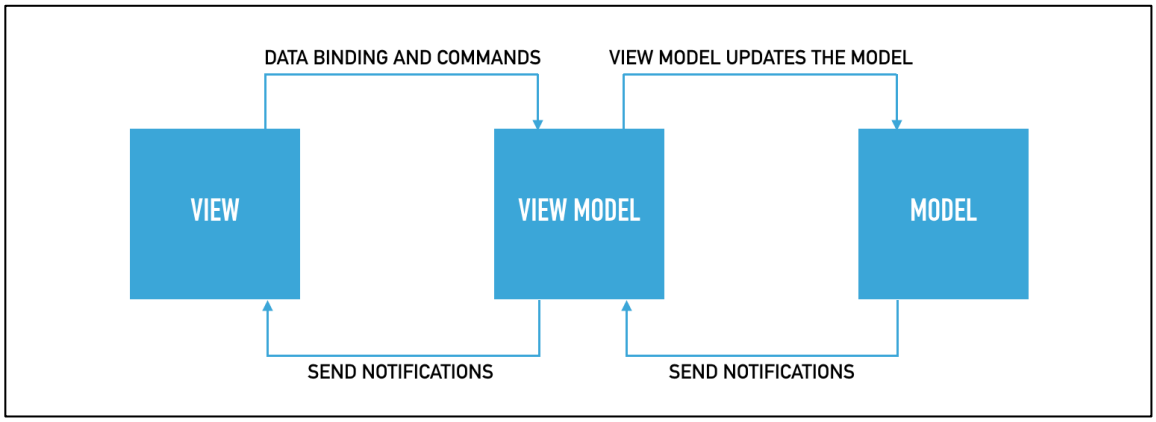

Рис. 2.1. MVVM схема

Model – модель характеризує застосовані в додатку дані. Ці моделі звісно можуть містити логіку, яка пов'язана безпосередньо цими даними. Можемо привести приклад, як логіку валідації властивостей моделі. У той же час вона ні в якому разі не повинна містити ніякої логіки, яка була би пов'язаної з відображуванням даних, та також з взаємодією моделі з візуальними елементами управління.

View – подання визначає візуальний інтерфейс, через який користувач взаємодіє з додатком. Стосовно до WPF уявлення - це код в xaml, який показує інтерфейс у вигляді візуальних елементів, кнопок, а також текстових полів.

ViewModel – модель уявлення пов'язує модель і уявлення через механізм прив'язки даних. Якщо в моделі змінюються значення властивостей, при реалізації моделлю інтерфейсу INotifyPropertyChanged автоматично йде зміна відображуваних даних в поданні, хоча безпосередньо модель і уявлення не пов'язані. Також містить логіку по отриманню даних з моделі, які потім передаються в уявлення. І також VewModel визначає логіку по оновленню даних в моделі.

.NET

За допомогою .NET код і файли проекту виглядають однаково незалежно від типу додатка, який створюється. Є доступ до однакових можливостей виконання, API та мови для кожної програми.

Мови .NET вищого рівня, такі як C #, компілюються до апаратно-агностичного набору інструкцій, який називається Intermediate Language (IL). Коли програма запускається, компілятор JIT переводить IL в машинний код, який розуміє процесор. Компіляція JIT відбувається на тій самій машині, на якій буде працювати код [14].

Оскільки компіляція JIT відбувається під час виконання програми, час компіляції є частиною часу виконання. Тому компілятори JIT повинні збалансувати витрачений час на оптимізацію коду та економію, яку може отримати отриманий код. Але компілятор JIT знає фактичне обладнання та може звільнити розробників від необхідності поставляти різні реалізації для різних платформ.

Компілятор .NET JIT може виконувати багаторівневу компіляцію, що означає, що він може перекомпілювати окремі методи під час виконання. Ця функція дозволяє їй швидко компілювати, зберігаючи при цьому дуже налаштовану версію коду для часто використовуваних методів [16].

Колекціонер сміття (GC) керує розподілом та вивільненням пам'яті для програм. Кожного разу, коли ваш код створює новий об’єкт, CLR виділяє пам’ять для об’єкта з керованої купи. Поки в керованій купі доступний адресний простір, час виконання продовжує виділяти простір для нових об’єктів. Коли залишається

недостатньо вільного адресного простору, GC перевіряє наявність об’єктів в керованій купі, які більше не використовуються додатком. Потім вона відновлює цю пам’ять.

GC – одна із служб CLR, яка допомагає забезпечити безпеку пам'яті. Програма безпечна для пам'яті, якщо вона отримує доступ лише до виділеної пам'яті. Наприклад, час виконання гарантує, що програма не отримує доступ до нерозподіленої пам'яті поза межами масиву [16].

Іноді код потребує посилання на некеровані ресурси. Некеровані ресурси - це ресурси, які не підтримуються автоматично середовищем виконання .NET. Наприклад, дескриптор файлу - це некерований ресурс. Об'єкт FileStream - це керований об'єкт, але він посилається на дескриптор файлу, яким не керують. Коли закінчите використовувати FileStream, потрібно явно відпустити дескриптор файлу [17].

У .NET об'єкти, що посилаються на некеровані ресурси, реалізують інтерфейс IDisposable. Коли закінчите використовувати об'єкт, викликаєте метод Dispose, який відповідає за вивільнення будь-яких некерованих ресурсів. Мови .NET забезпечують зручне використання оператора (C#, F#, VB), що забезпечує виклик методу Dispose [20].

Програми .NET можна публікувати у двох різних режимах [18]:

– публікація програми як автономної (self-contained) створює виконуваний файл, що включає середовище виконання та бібліотеки .NET, а також програму та її залежності. Користувачі програми можуть запускати його на машині, на якій не встановлено середовище виконання .NET. Автономні програми є специфічними для певної платформи, і їх за бажанням можна опублікувати за допомогою форми компіляції AOT.

– публікація програми як залежної від фреймворку (framework-dependent) створює виконуваний файл і двійкові файли (.dll-файли), які включають лише саму програму та її залежності. Користувачі програми повинні окремо інсталювати середовище виконання .NET. Виконавчий файл залежить від платформи, але .dll-файли програм, що залежать від фреймворку, є міжплатформенними. Можна встановити декілька версій середовища виконання

поруч, щоб запускати залежні від фреймворку програми, націлені на різні версії середовища виконання. Для отримання додаткової інформації див. Цільові рамки.

WPF

WPF існує як підмножина типів .NET, які в основному знаходяться в просторі імен System.Windows [5]:

– встановлення властивостей

– методи виклику

– обробка подій

– використання классу

WPF включає більше конструкцій програмування, що покращують властивості та події: властивості залежності та маршрутизовані події.

WPF дозволяє розробляти додатки, використовуючи як розмітку, так і код, що є досвідом, з яким розробники ASP.NET повинні бути знайомі. Як правило, ви використовуєте розмітку XAML для реалізації зовнішнього вигляду програми, використовуючи керовані мови програмування (код позаду) для реалізації її поведінки. Це розділення зовнішності та поведінки має наступні переваги [9]:

– витрати на розробку та обслуговування зменшуються, оскільки розмітка, що залежить від зовнішнього вигляду, не є тісно пов'язаною з кодом, що відповідає поведінці;

– розробка є більш ефективною, оскільки дизайнери можуть реалізовувати зовнішній вигляд програми одночасно з розробниками, які реалізують поведінку програми;

– глобалізація та локалізація програм WPF спрощена.

XAML – це мова розмітки на основі XML, яка декларативно реалізує зовнішній вигляд програми. Зазвичай використання його для визначення вікон, діалогових вікон, сторінок та елементів керування користувача та для наповнення ними елементами керування, фігурами та графікою.

Рис. 2.2. Приклад вікна з кнопкою

Зокрема, цей XAML визначає вікно та кнопку за допомогою елементів Вікно та Кнопка (рис. 2.2). Кожен елемент налаштований за допомогою атрибутів, таких як атрибут Title елемента Window, щоб вказати текст рядка заголовка вікна. Під час виконання WPF перетворює елементи та атрибути, визначені в розмітці, у екземпляри класів WPF. Наприклад, елемент Window перетворюється на екземпляр класу Window, властивість Title якого є значенням атрибута Title.

Оскільки XAML базується на XML, інтерфейс користувача, який створюється з ним, зібраний в ієрархії вкладених елементів, який відомий як дерево елементів. Дерево елементів забезпечує логічний та інтуїтивно зрозумілий спосіб створення та керування інтерфейсами користувача.

Основна поведінка програми - це реалізація функціональних можливостей, що реагують на взаємодію користувачів. Наприклад, натискання меню чи кнопки та виклик бізнес-логіки та логіки доступу до даних у відповідь. У WPF така поведінка реалізована в коді, який пов'язаний з розміткою. Цей тип коду відомий як код позаду.

Розмітка визначає простір імен xmlns:x та відображає його у схемі, яка додає підтримку типів, що містять код. Атрибут x:Class використовується для асоціювання класу, що не відповідає коду, до цієї конкретної розмітки XAML. Враховуючи, що цей атрибут оголошений в елементі <Window>, клас із кодом повинен успадковуватися від класу Window.

InitializeComponent викликається з конструктора класу позаду коду для об’єднання інтерфейсу користувача, який визначений у розмітці, із класом позаду коду. (InitializeComponent створюється для вас під час створення вашої програми, тому вам не потрібно реалізовувати її вручну) Комбінація x:Class та

InitializeComponent гарантує, що ваша реалізація правильно ініціалізується при її створенні.

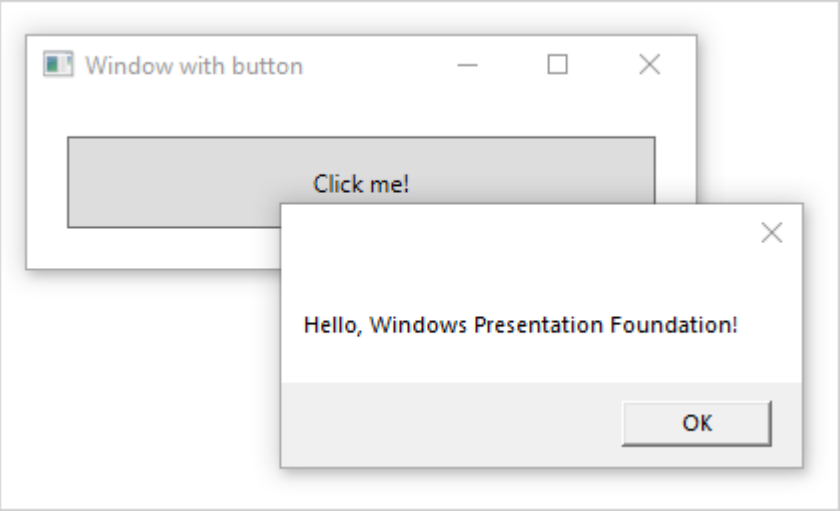

Рис. 2.3. Приклад події

Коли ініціалізація розмітки та коду ініціалізується та працює разом, подія Click для кнопки автоматично відображається у методі button_click. Після натискання кнопки викликається обробник подій і відображається вікно повідомлення, викликаючи метод System.Windows.MessageBox.Show (рис. 2.3).

Елементи керування найчастіше виявляють вхід користувача та реагують на нього. Система введення WPF використовує як прямі, так і маршрутизовані події для підтримки введення тексту, управління фокусом та позиціонування миші.

Програми часто мають складні вимоги до введення. WPF забезпечує командну систему, яка відокремлює дії, введені користувачем, від коду, який відповідає на ці дії. Система команд дозволяє декільком джерелам викликати одну і ту ж логіку команд. Наприклад, візьмемо загальні операції редагування, що використовуються різними програмами: копіювання, вирізання та вставка. Ці операції можна викликати, використовуючи різні дії користувача, якщо вони реалізуються за допомогою команд.

Взаємодія з користувачем, яку надає модель програми, - це побудовані елементи керування. У WPF контроль - це загальний термін, який застосовується до категорії класів WPF, що мають такі характеристики:

– розміщується або у вікні, або на сторінці;

– має користувальницький інтерфейс;

– реалізується якась поведінка.

Створюючи користувальницький інтерфейс, впорядковуєте елементи керування за місцем розташування та розміром, щоб сформувати макет. Ключовою вимогою будь-якого макета є адаптація до змін розміру вікна та налаштувань дисплея. Замість того, щоб змушувати вас писати код для адаптації макета за цих обставин, WPF пропонує першокласну, розширювану систему макетування [6].

Наріжним каменем системи верстки є відносне позиціонування, що збільшує здатність адаптуватися до мінливих умов вікна та відображення. Система компонування також управляє узгодженням між елементами управління для визначення макета. Зв'язок – це двоетапний процес: спочатку елемент управління повідомляє батькові, яке місце і розмір йому потрібні. По-друге, батько повідомляє контролю, який простір він може мати.

Система компонування піддається дочірньому контролю через базові класи WPF. Для загальних макетів, таких як сітки, укладання та стикування, WPF включає кілька елементів керування макетом [22]:

– Canvas: дочірні елементи управління надають свої власні розкладки;

– DockPanel: дочірні елементи керування вирівняні по краях панелі;

– Grid: дочірні елементи керування розташовані за рядками та стовпцями;

– StackPanel: дочірні елементи керування розташовані вертикально або горизонтально;

– VirtualizingStackPanel: дочірні елементи керування віртуалізовані та розташовані в одному рядку, який орієнтований горизонтально або вертикально;

– WrapPanel: дочірні елементи керування розташовані в порядку зліва направо і переносяться на наступний рядок, коли місця недостатньо. на поточному рядку.

Більшість програм створюються для того, щоб забезпечити користувачів засобами перегляду та редагування даних. Для додатків WPF робота зі зберігання та доступу до даних вже передбачена багатьма різними бібліотеками доступу до даних .NET, такими як SQL та Entity Framework Core. Після доступу

до даних та завантаження їх в керовані об'єкти програми починається важка робота для програм WPF. По суті, це включає дві речі:

– копіювання даних з керованих об'єктів у елементи керування, де дані можна відображати та редагувати;

– забезпечення того, що зміни, внесені до даних за допомогою елементів керування, копіюються назад до керованих об’єктів.

Щоб спростити розробку додатків, WPF забезпечує потужний механізм прив'язки даних для автоматичної обробки цих кроків. Основним блоком механізму прив'язки даних є клас Binding, завданням якого є прив'язка елемента керування (цілі прив'язки) до об'єкта даних (джерела прив'язки). Цей взаємозв'язок ілюструється наступним малюнком [23]:

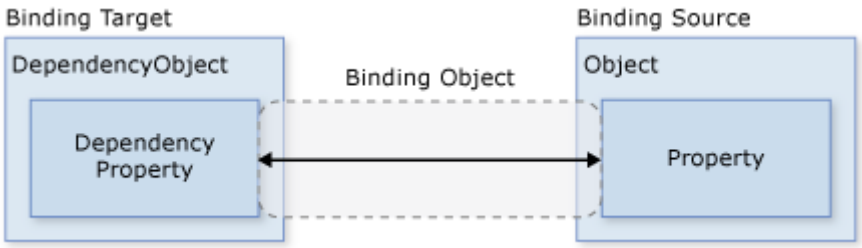

Рис. 2.4. Прив'язок даних у розмітці XAML

WPF підтримує безпосереднє оголошення прив'язок у розмітці XAML. Наприклад, наступний код XAML пов'язує властивість Text TextBox із властивістю Name об'єкта, використовуючи синтаксис XAML "{Binding ...}". Це передбачає наявність об’єкта даних, встановленого для властивості DataContext Вікна з властивістю Name (рис. 2.4).

Механізм прив'язки даних WPF забезпечує більше, ніж просто прив'язку, він забезпечує перевірку, сортування, фільтрацію та групування. Крім того, прив'язка даних підтримує використання шаблонів даних для створення користувацького інтерфейсу для пов'язаних даних.

WPF надає великий та гнучкий набір графічних функцій, які мають наступні переваги:

– графіка, що не залежить від роздільної здатності та від пристрою. Основною одиницею вимірювання в графічній системі WPF є незалежний від

пристрою піксель, який становить 1/96 дюйма, і забезпечує основу для незалежного від роздільної здатності та незалежного від пристрою візуалізації. Кожен незалежний від пристрою піксель автоматично масштабується відповідно до налаштувань точки на дюйм (dpi) системи, на якій він відображається [1].

– покращена точність. Система координат WPF вимірюється числами з плаваючою комою з подвійною точністю, а не одноточністю. Значення перетворень та непрозорості також виражаються як подвійна точність. WPF також підтримує широку кольорову гаму (scRGB) і забезпечує інтегровану підтримку для управління входами з різних колірних просторів.

– розширена підтримка графіки та анімації. WPF спрощує програмування графіки, керуючи для вас сценами анімації; не потрібно турбуватися про обробку сцени, цикли візуалізації та білінійну інтерполяцію. Крім того, WPF надає підтримку перевірки хітів та повну підтримку альфа-композитування.

– апаратне прискорення. Графічна система WPF використовує обладнання для мінімізації використання центрального процесора.

WPF забезпечує бібліотеку загальних векторних 2D-фігур, таких як прямокутники та еліпси. Фігури не просто для демонстрації; фігури реалізують багато функцій, яких ви очікуєте від елементів керування, включаючи введення з клавіатури та миші.

Двовимірні фігури, надані WPF, охоплюють стандартний набір основних фігур. Однак вам може знадобитися створити власні фігури, щоб допомогти в розробці користувацького інтерфейсу. WPF надає геометрії для створення власної фігури, яку можна намалювати безпосередньо, використовувати як пензлик або використовувати для відсікання інших фігур та елементів керування [9].

Підмножина можливостей WPF 2D включає візуальні ефекти, такі як градієнти, растрові зображення, малюнки, малювання за допомогою відео, обертання, масштабування та перекос. Ці ефекти досягаються за допомогою щіток (рис. 2.5).

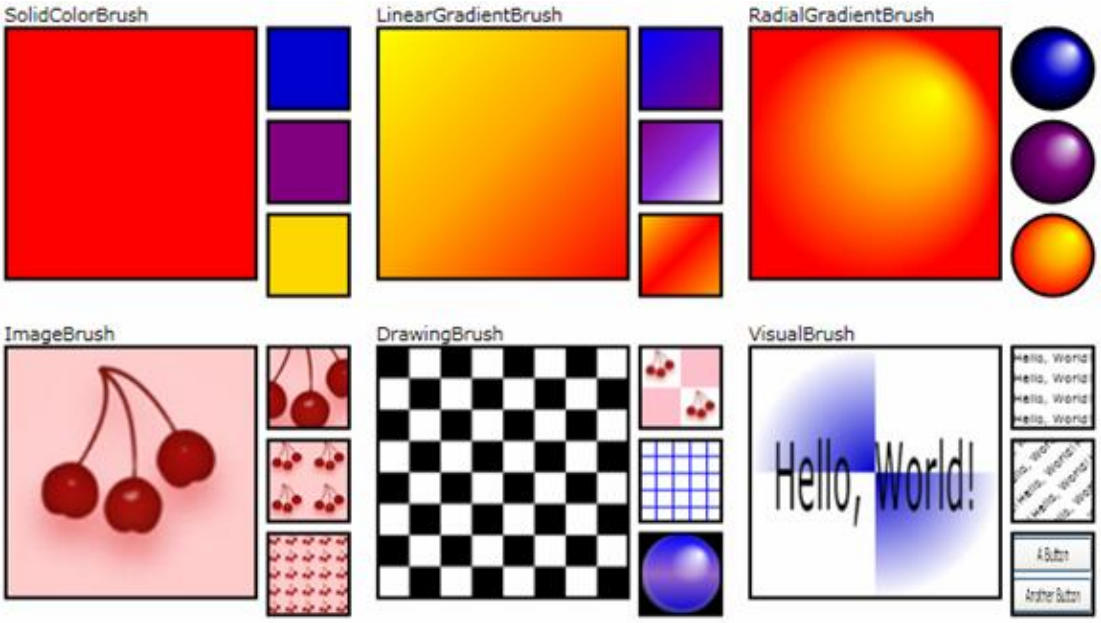

Рис. 2.5. Рендер 2D-зображень

WPF також включає можливості 3D-рендерингу, які інтегруються з 2D-графікою, що дозволяє створювати більш захоплюючі та цікаві користувальницькі інтерфейси. Наприклад, на малюнку показано 2D-зображення, відтворені у 3D-фігури (рис. 2.6).

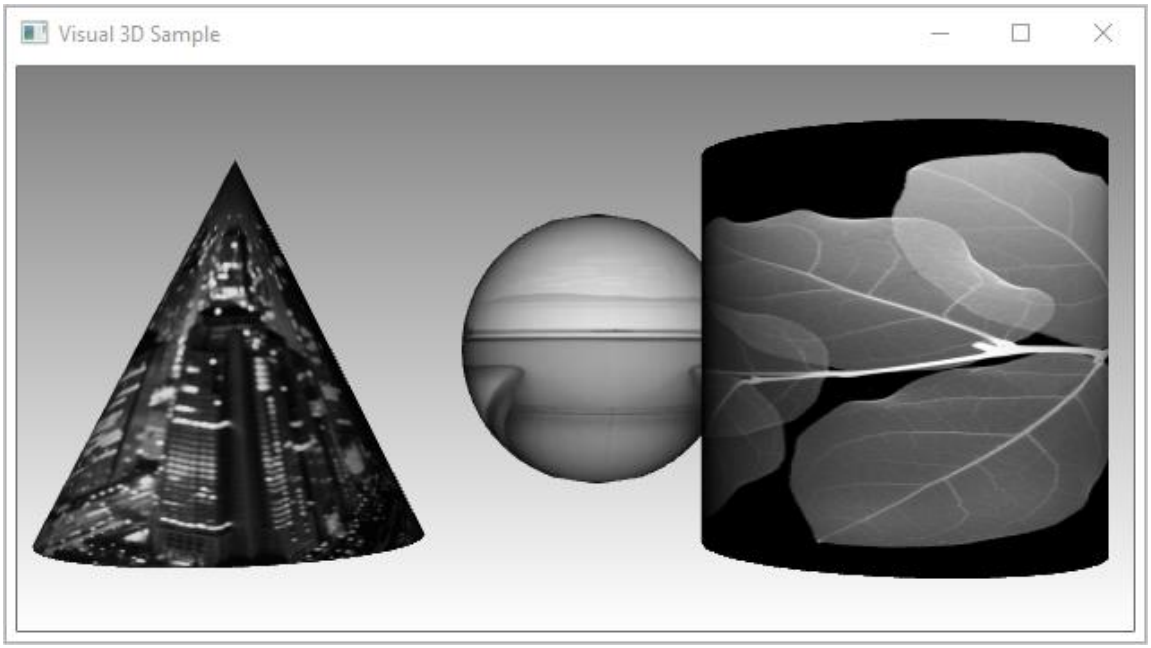

Рис. 2.6. Рендер 2D-зображення, відтворене у 3D-фігури

Підтримка анімації WPF дозволяє змушувати елементи керування рости, струшувати, обертати та зникати, створювати цікаві переходи сторінок тощо. Ви можете анімувати більшість класів WPF, навіть власні класи.

Щоб забезпечити якісний візуалізацію тексту, WPF пропонує такі функції:

– підтримка шрифтів OpenType;

– покращення ClearType;

– висока продуктивність, яка використовує переваги апаратного прискорення;

– інтеграція тексту із засобами масової інформації, графікою та анімацією.

– міжнародна підтримка шрифтів та резервні механізми.

Як демонстрація інтеграції тексту з графікою, на наступному малюнку показано застосування декоративних елементів тексту (рис. 2.7).

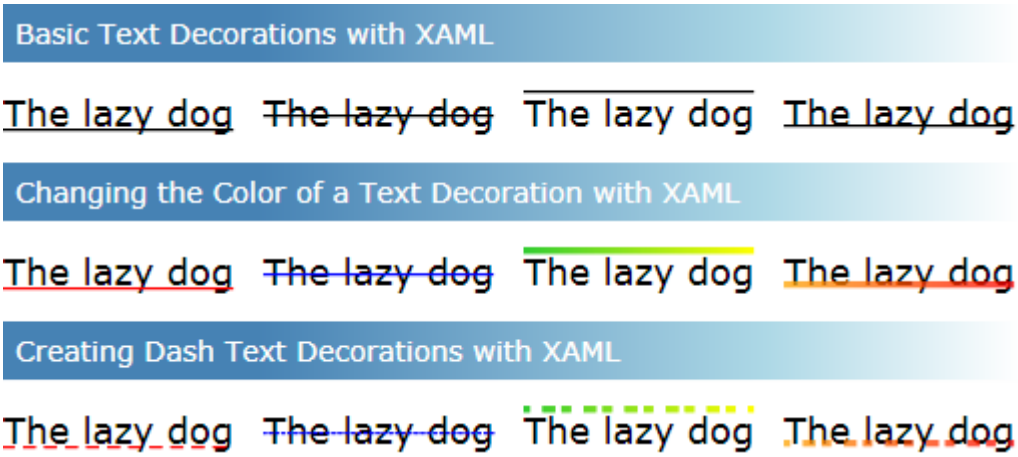

Рис. 2.7. Інтеграція декорацій для тексту

Основною метою більшості елементів керування WPF є відображення вмісту. У WPF тип і кількість елементів, які можуть становити вміст елемента керування, називається моделлю вмісту елемента управління. Деякі елементи керування можуть містити один елемент та тип вмісту. Наприклад, вміст TextBox - це рядкове значення, яке присвоюється властивості Text.

Однак інші елементи керування можуть містити кілька елементів різного типу вмісту, вміст кнопки, зазначений властивістю Content, може містити різні елементи, включаючи елементи керування макетом, текст, зображення та форми.

Хоча основною метою розмітки XAML є реалізація зовнішнього вигляду програми, ви також можете використовувати XAML для реалізації деяких аспектів поведінки програми. Одним із прикладів є використання тригерів для зміни зовнішнього вигляду програми на основі взаємодії користувачів.

Користувальницькі інтерфейси за замовчуванням для елементів керування WPF зазвичай будуються з інших елементів керування та фігур. Наприклад, кнопка складається з елементів керування ButtonChrome і ContentPresenter. ButtonChrome забезпечує стандартний вигляд кнопки, тоді як ContentPresenter відображає вміст кнопки, як зазначено у властивості Content.

Іноді зовнішній вигляд елемента керування може суперечити загальному зовнішньому вигляду програми. У цьому випадку ви можете використовувати ControlTemplate для зміни зовнішнього вигляду користувацького інтерфейсу елемента керування, не змінюючи його вмісту та поведінки.

Наприклад, кнопка викликає подію Click при натисканні. Змінивши шаблон кнопки для відображення фігури Еліпс, змінився візуальний аспект елемента управління, але функціональність - ні. Ви все ще можете натиснути на візуальний аспект елемента керування, і подія Click буде піднята, як очікувалося (рис. 2.8).

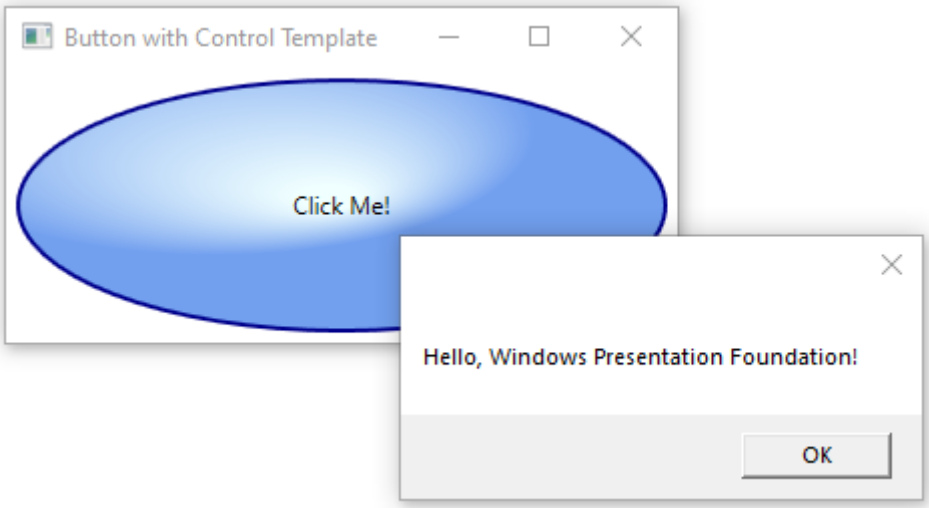

Рис. 2.8. Кастомна кнопка викликає подію Click при натисканні

Якщо шаблон управління дозволяє вказати зовнішній вигляд елемента керування, шаблон даних дозволяє вказати вигляд вмісту елемента керування. Шаблони даних часто використовуються для покращення способу відображення пов'язаних даних. На наступному малюнку показано зовнішній вигляд ListBox, який прив’язаний до колекції об’єктів Task, де кожне завдання має ім’я, опис та пріоритет (рис. 2.9).

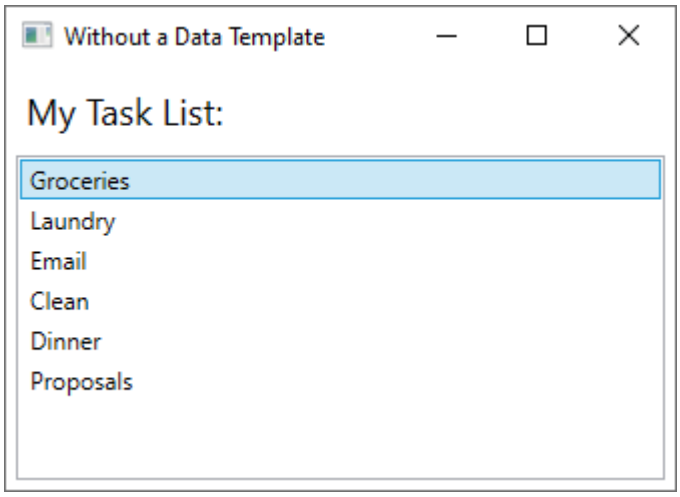

Рис. 2.9. Зовнішній вигляд ListBox, який прив’язаний до колекції

Вигляд за замовчуванням - це те, що ви очікували від ListBox. Однак типовий вигляд кожного завдання містить лише ім'я завдання. Щоб показати назву завдання, опис та пріоритет, зовнішній вигляд зв’язаних елементів списку елементів керування ListBox потрібно змінити за допомогою DataTemplate. Ось приклад застосування шаблону даних, створеного для об’єкта Task (рис. 2.10).

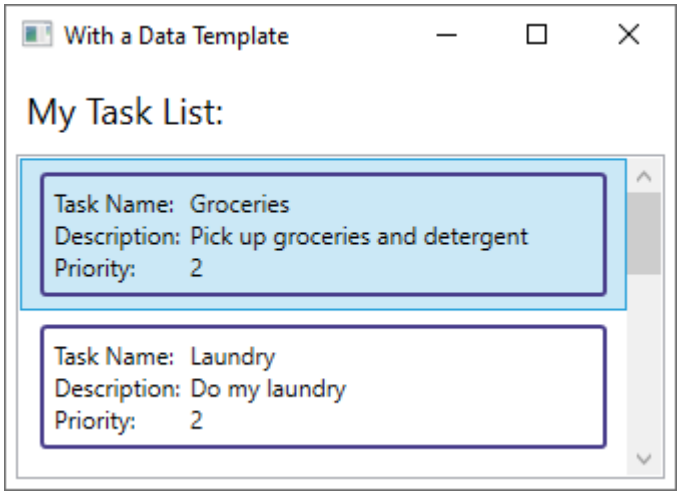

Рис. 2.10. Змінений зовнішній вигляд зв’язаних елементів списку ListBox

Стилі дозволяють розробникам та дизайнерам стандартизувати певний зовнішній вигляд свого товару. WPF забезпечує сильну модель стилю, основою якої є елемент Style. Стилі можуть застосовувати значення властивостей до типів. Вони можуть застосовуватися автоматично до всього відповідно до типу або окремих об'єктів при посиланні. Наступний приклад створює стиль, який задає колір тла для кожної кнопки у вікні оранжевим. Оскільки цей стиль

націлений на всі елементи керування кнопками, стиль автоматично застосовується до всіх кнопок у вікні, як показано на наступному малюнку:

Рис. 2.11. Використання WPF для ресурсів користувальницького інтерфейсу

Елементи керування в програмі повинні мати однаковий зовнішній вигляд, який може включати що завгодно - від шрифтів та кольорів фону до шаблонів управління, шаблонів даних та стилів. Ви можете використовувати підтримку WPF для ресурсів користувальницького інтерфейсу, щоб інкапсулювати ці ресурси в одному місці для повторного використання (рис. 2.11).

Хоча WPF надає безліч засобів налаштування, ви можете зіткнутися з ситуаціями, коли існуючі елементи керування WPF не відповідають потребам ні вашої програми, ні її користувачів. Це може статися, коли:

– користувацький інтерфейс, який вам потрібен, не може бути створений шляхом налаштування зовнішнього вигляду існуючих реалізацій WPF;

– необхідна поведінка не підтримується (або не легко підтримується) існуючими реалізаціями WPF.

На цьому етапі, однак, ви можете скористатися однією з трьох моделей WPF, щоб створити новий елемент керування. Кожна модель націлена на конкретний сценарій і вимагає, щоб ваш власний контроль виходив з певного базового класу WPF. Тут перелічено три моделі [4]:

– модель керування користувачем – спеціальне управління походить від UserControl і складається з одного або декількох інших елементів управління;

– модель управління – спеціальний контроль походить від Control і використовується для побудови реалізацій, які відокремлюють їх поведінку від їх зовнішнього вигляду за допомогою шаблонів, подібно до більшості елементів керування WPF. Виведення з Control дає вам більше свободи для створення

користувацького інтерфейсу користувача, ніж елементи керування користувача, але це може вимагати більше зусиль;

– модель елемента рамки – спеціальний елемент керування походить від FrameworkElement, коли його зовнішній вигляд визначається власною логікою візуалізації (не шаблонами).

C#

C# (вимовляється як "See Sharp") – це сучасна, об'єктно-орієнтована та безпечна для програмування мова програмування. C# дозволяє розробникам створювати багато типів безпечних та надійних додатків, що працюють в екосистемі .NET. C# сягає корінням з сімейства мов C і буде одразу знайомий програмістам на C, C++, Java та JavaScript. Цей тур надає огляд основних компонентів мови в C# 8 та раніше. Якщо ви хочете вивчити мову на інтерактивних прикладах, спробуйте ознайомитись із підручниками C# [10].

C# – це об'єктно-орієнтована, орієнтована на компоненти мова програмування. # забезпечує мовні конструкції для прямої підтримки цих концепцій, роблячи C# природною мовою, на якій можна створювати та використовувати програмні компоненти. З моменту свого зародження C# додав функції для підтримки нових навантажень та нових практик проектування програмного забезпечення.

Кілька функцій C# допомагають створювати надійні та довговічні програми. Збір сміття автоматично відновлює пам’ять, зайняту недосяжними невикористаними об’єктами. Допустимі типи захищають від змінних, які не посилаються на виділені об'єкти. Обробка винятків забезпечує структурований та розширюваний підхід до виявлення та відновлення помилок. Лямбда-вирази підтримують методи функціонального програмування. Синтаксис інтегрованого мовного запиту (LINQ) створює загальний шаблон роботи з даними з будь-якого джерела. Мовна підтримка асинхронних операцій забезпечує синтаксис для побудови розподілених систем. C# має уніфіковану систему типів. Усі типи C#, включаючи примітивні типи, такі як int та double, успадковуються від одного кореневого типу об'єкта. Усі типи мають спільний набір загальних операцій.

Цінності будь-якого типу можна зберігати, транспортувати та використовувати послідовно. Крім того, C# підтримує як визначені користувачем типи посилань, так і типи значень. C# дозволяє динамічно розподіляти об'єкти та зберігати в лінійці легкі конструкції. C# підтримує загальні методи та типи, які забезпечують підвищену безпеку та продуктивність типу. C# надає ітератори, які дозволяють реалізаторам класів колекцій визначати власну поведінку для клієнтського коду.

C# робить акцент на встановленні версій для забезпечення того, щоб програми та бібліотеки могли розвиватися з часом сумісно. Аспекти дизайну C#, на які безпосередньо вплинули міркування щодо версій, включають окремі модифікатори віртуальних та перевизначених, правила дозволу на перевантаження методів та підтримку явних оголошень членів інтерфейсу [16].

У C# є два типи типів: типи значень та типи посилань. Змінні типів значень безпосередньо містять їх дані. Змінні посилальних типів зберігають посилання на свої дані, останні відомі як об'єкти. З посилальними типами можна, щоб дві змінні посилалися на один і той самий об’єкт, а операції над однією змінною могли впливати на об’єкт, на який посилається інша змінна. Для типів значень кожна змінна має власну копію даних, і неможливо, щоб операції над одним впливали на інші (за винятком змінних ref та out параметра).

Ідентифікатор – це ім’я змінної. Ідентифікатор - це послідовність символів Unicode без пробілів. Ідентифікатором може бути слово, зарезервоване на C#, якщо воно має префікс @. Використання зарезервованого слова як ідентифікатора може бути корисним при взаємодії з іншими мовами.

Типи значень C# поділяються далі на прості типи, типи перерахування, типи структур, типи значень, що допускають нульове значення, і типи значень кортежу. Довідкові типи C# поділяються далі на типи класів, типи інтерфейсів, типи масивів та типи делегатів.

Типи значень:

– прості типи: sbyte, short, int, long, byte, ushort, uint, ulong, char, float, double, decimal, bool;

– визначені користувачем типи переліку переліку (enum) E {...}, де це окремий тип з іменованими константами. Кожен тип переліку має базовий тип, який повинен бути одним із восьми цілісних типів. Набір значень типу перечислення є однаковим із набором значень базового типу;

– визначені користувачем типи структури (struct) S {...};

– розширення всіх інших типів значень із нульовим значенням (null);

– визначені користувачем типи (tuple) форми (T1, T2, ...).

Типи посилань:

– класи: кінцевий базовий клас усіх інших типів (об'єкт), рядок, який представляє послідовність кодових одиниць UTF-16, визначені користувачем типи класу форми C {...};

– визначені користувачем типи інтерфейсу форми I {...};

– масиви: одновимірні, багатовимірні та зубчасті. Наприклад: int [], int [,] та int [] [] та інші;

– визначені користувачем типи делегата форми int D (...).

Програми C# використовують декларації типів для створення нових типів. Декларація типу вказує ім'я та члени нового типу. Шість категорій типів C# визначаються користувачем: типи класів, типи структур, типи інтерфейсів, типи перерахувань, типи делегатів та типи значень кортежу:

– тип класу визначає структуру даних, яка містить члени даних (поля) та члени функції (методи, властивості та інші). Типи класів підтримують єдине успадкування та поліморфізм, механізми, завдяки яким похідні класи можуть розширювати та спеціалізувати базові класи;

– тип структури подібний до типу класу тим, що він представляє структуру з членами даних та членами функції. Однак, на відміну від класів, структури є типами значень і зазвичай не вимагають розподілу купи. Типи структур не підтримують задане користувачем успадкування, і всі типи структур неявно успадковують від об’єкта типу;

– тип інтерфейсу визначає контракт як іменований набір публічних членів. Клас або структура, що реалізує інтерфейс, повинен забезпечувати реалізації

членів інтерфейсу. Інтерфейс може успадковуватись від декількох базових інтерфейсів, а клас або структура може реалізовувати кілька інтерфейсів;

– тип делегата представляє посилання на методи з певним списком параметрів і типом повернення. Делегати дають можливість розглядати методи як сутності, які можуть бути присвоєні змінним і передані як параметри. Делегати аналогічні типам функцій, що надаються функціональними мовами. Вони також подібні до концепції покажчиків функцій, що зустрічається в деяких інших мовах. На відміну від покажчиків на функції, представники є об'єктно-орієнтованими та безпечними для типу.

Усі типи класів, структур, інтерфейсів та делегатів підтримують загальні засоби, завдяки чому їх можна параметризувати за допомогою інших типів [18].

C# підтримує одновимірні та багатовимірні масиви будь-якого типу. На відміну від перелічених вище типів, типи масивів не повинні бути оголошені перед тим, як їх можна використовувати. Натомість типи масивів будуються, дотримуючись імені типу у квадратних дужках. Наприклад, int [] - це одновимірний масив int, int [,] - двовимірний масив int, а int [] [] - це одновимірний масив одновимірних масивів, або "зубчастий "масив, з int.

Типи, що допускають відхилення, не потребують окремого визначення. Для кожного ненульованого типу T існує відповідний нульовий тип T ?, який може містити додаткове значення, нуль. Наприклад, int? це тип, який може містити будь-яке 32-бітове ціле число або значення null, і рядок? це тип, який може містити будь-який рядок або значення null.

Система типів C# уніфікована таким чином, що значення будь-якого типу може розглядатися як об'єкт. Кожен тип у C# прямо чи опосередковано походить від типу класу об'єкта, і object є кінцевим базовим класом усіх типів. Значення посилальних типів розглядаються як об'єкти, просто переглядаючи значення як об'єкт типу. Значення типів значень обробляються як об'єкти, виконуючи операції боксу та розпакування. У наступному прикладі значення int перетворюється в об'єкт і назад знову в int.

Коли значення типу значення присвоюється посилання на об'єкт, для розміщення значення виділяється "вікно". Це поле є екземпляром типу

посилання, і значення копіюється в це поле. І навпаки, коли посилання на об’єкт передається типу значень, перевіряється, чи є об’єкт, на який посилається, поле відповідного типу значення. Якщо перевірка успішна, значення в полі копіюється до типу значення.

Уніфікована система типів C# фактично означає, що типи значень розглядаються як посилання на об'єкти "на вимогу". Через уніфікацію бібліотеки загального призначення, які використовують об’єкт типу, можуть використовуватися з усіма типами, що походять від об’єкта, включаючи як посилальні типи, так і типи значень.

У C # існує кілька видів змінних, включаючи поля, елементи масиву, локальні змінні та параметри. Змінні представляють місця зберігання. Кожна змінна має тип, який визначає, які значення можна зберігати у змінній, як показано нижче:

– ненульове значення – значення саме цього типу;

– nullable значення – нульове значення або значення саме цього типу;

– об'єкт – нульове посилання, посилання на об'єкт будь-якого типу посилання або посилання на позначення, яке вкладено в поле, будь-якого типу значення;

– клас – нульове посилання, посилання на екземпляр цього типу класу або посилання на екземпляр класу, похідного від цього типу класу;

– інтерфейс – нульове посилання, посилання на екземпляр типу класу, який реалізує цей тип інтерфейсу, або посилання на позначку в поле типу значення, що реалізує цей тип інтерфейсу;

– масив – нульове посилання, посилання на екземпляр цього типу масиву або посилання на екземпляр сумісного типу масиву;

– делегат – нульове посилання або посилання на екземпляр сумісного типу делегатів.

2.4. Опис структури системи та алгоритмів її функціонування

Для роботи з даними використовується підхід MVVM, описаний раніше. Для кожної сутності є клас, а також набір інтерфейсів до нього, де для інтерфейсів описані загальні методи для роботи з ними, що дозволяє уникнути

великого дублювання коду і оптимізувати роботу програми і його розширення функціоналу (синтаксичні можливості C # дозволяють це) [4].

Інтерфейс (interface) демонструє просто іменований набір абстрактних членів. Ці абстрактні методи не мають ніякої стандартної реалізації. Інтерфейс пропонує поведінку, яку дана структура може обрати для своєї підтримки.

Клас - головний вид даних мови C #, який демонструє собою конструкцію, що збирає до купи поля, методи та властивості. Клас є формулюванням екземплярів класу, а також створення різних об'єктів.

У додатку присутній повноцінний і частковий підхід ООП.

Основні завдання ООП - структурувати код, підвищити його читабельність і прискорити розуміння логіки програми. Побічно виконуються й інші завдання: наприклад, підвищується безпека коду і скорочується його дублювання.

Справа в тому, що набагато зручніше працювати з реальними об'єктами, ніж окремо з набором даних і функціями. Представляючи дані в програмі як властивості об'єкта, а функції по обробці даних - як можливі методи об'єкта, наближається процес програмування до процесу опису методу розв'язання задачі. Це досягається за рахунок додавання знайомої структури абстракцій: адже навіть мова, на якому ми говоримо, дотримується принципів ООП. У кожної букви є вимову і написання, кожне слово включає букви і має свою вимову і написання, то ж вірно і для пропозицій, і для більших конструкцій. Все в цьому світі - об'єкт!

Такий підхід допомагає будувати складні системи більш просто і природно завдяки тому, що вся предметна область розбивається на об'єкти і кожен з них має слабкий зв'язок з іншими об'єктами. Слабка зв'язаність виникає внаслідок дотримання трьох принципів: інкапсуляції, успадкування та поліморфізму.

– Інкапсуляція – приховування поведінки об'єкта всередині нього. Об'єкту «водій» не потрібно знати, що відбувається в об'єкті «машина», щоб вона їхала. Це ключовий принцип ООП;

– Спадкування. Є об'єкти «людина» і «водій». У них є явно щось спільне. Спадкування дозволяє виділити це загальне в один об'єкт (в даному випадку більш загальним буде людина), а водія - визначити як людини, але з додатковими

властивостями і / або поведінкою. Наприклад, у водія є водійські права, а у людини їх може не бути;

– Поліморфізм – це перевизначення поведінки. Можна знову розглянути «людини» і «водія», але тепер додати «пішохода». Людина вміє якось пересуватися, але як саме, залежить від того, водій він або пішохід. Тобто у пішохода і водія подібна поведінка, але реалізоване по-різному: один переміщається ногами, інший - на машині.

ООП дозволяє спростити складні об'єкти, складаючи їх з дрібніших і простих, тому над програмою можуть працювати сотні розробників, кожен з яких зайнятий своїм блоком. Більшість сучасних мов програмування - об'єктно-орієнтовані, і, одного разу зрозумівши суть, ви зможете освоїти відразу кілька мов [17].

2.5. Обґрунтування та організація вхідних та вихідних даних програми

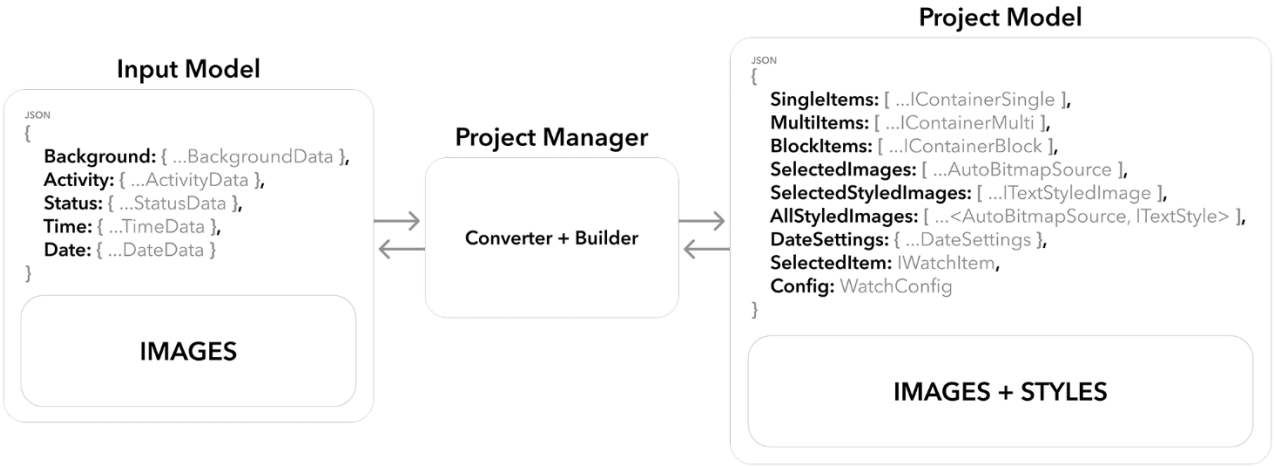

Рис. 2.12. Схема обробки даних

В якості вхідних параметрів використовується * .json дані з распакованого циферблата Mi Band, також разом з файлом * .json в папці знаходяться всі ресурси зображень.

Для кожного типу файлу в проекті є окремий клас із закінченням IO (Input-Output) для введення та виведення даних у відповідному форматі, на даний момент підтримувані файли *.json / *.bin (рис. 2.12). В майбутньому також

планується власний формат програми * .xmi, який буде зберігати дані саме в форматі додатка, зі збереженням стилів і всіх інших даних.

При відкритті моделі відбувається перетворення RAW JSON в підготовлений клас C #, де присутня функція ToModel, яка звертається до іншого статичного класу SeriesBuilder. За підсумком після того, як SeriesBuilder повернув порожню налаштовану модель, починається обробка кожного поля даних з поточних вхідних параметрів "сирої моделі", кожен об'єкт розбитий на різні класифіковані суті, які вже підготовлені для обробки в двох напрямках, для конвертації в модель проекту і навпаки. Кожен метод конвертації окремої сутності має захист в разі, якщо деяких даних не вистачає, тому що циферблат може бути різний і містити не цільну модель даних, а тільки її частину.

Коли закінчилася конвертація всіх полів "сирих" даних, створюється модель проекту і вона потрапляє в загальний контейнер з усіма відкритими проектами. Проект може містити відразу кілька моделей для кожної серії Mi Band, на даний момент Mi Band 4/5. Одночасно відкритих проектів може бути 10.

SeriesBuilder - спеціальний клас, який реалізує патерн будівельник, який дозволяє створювати різні сутності однієї загальної моделі використовуючи деякі вхідні параметри. Він віддає різні готові моделі для заповнення даними з вхідних даних з уже налаштованої конфігурацією обраної серії Mi Band.

2.6. Опис роботи розробленого програмного продукту

2.6.1. Використані технічні засоби

Для користувача є важливим мати систему, спроможну запускати та працювати з сучасною Windows 7/8/10, тож необхідно мати такі мінімальні параметри ЕОМ:

- обсяг оперативної пам'яті (ОЗУ): 4 ГБ;
- відеоадаптер: 3D адаптер nVidia, Intel, AMD / ATI;
- графічну пам'ять: 512 МБ;
- сховище (HDD / SSD): 25 МБ;
- процесор (CISC x86 / x64): AMD / Intel.

2.6.2. Використані програмні засоби

Під час розробки даного застосунку були використані такі програмні засоби:

- Microsoft Visual Studio;
- .NET Core 3.1 SDK;
- Git, GitHub.

Microsoft Visual Studio

Microsoft Visual Studio – є середовищем для кожної програмної розробки (IDE) від Microsoft. Використовується для розробок різноманітних комп'ютерних програм, веб-сервісів, веб-сайтів, веб-програм, а також мобільних додатків. Visual Studio застосовує платформи для разних розробок програмного забезпечення Microsoft. Наприклад: Windows Presentation Foundation, Windows Forms, Windows API, Windows Store та Microsoft Silverlight. Він може складати власний код, а також і керований.

.NET Core 3.1 SDK

.NET SDK - це набір бібліотек та інструментів для розробки та запуску програм .NET.

Завантажуючи .NET, можна вибрати SDK або середовище виконання, наприклад, середовище виконання .NET або середовище виконання ASP.NET Core.

Завантаження SDK включає такі компоненти:

- CLI .NET. Інструменти командного рядка, які можна використовувати для локальної розробки та сценаріїв безперервної інтеграції;
- Драйвер dotnet. Команда CLI, яка запускає залежні від фреймворку програми;
- Компілятори мови програмування Roslyn та F #;
- Двигун збірки MSBuild;

– Час виконання .NET. Забезпечує систему типів, завантаження збірки, збирач сміття, власну взаємодію та інші основні послуги;

– Бібліотеки виконання. Надає примітивні типи даних та основні утиліти;

– Час виконання ASP.NET Core. Надає основні послуги для програм, підключених до Інтернету, таких як веб-програми, програми IoT та мобільні серверні мережі;

– Час роботи робочого столу. Надає основні послуги для настільних програм Windows, включаючи Windows Forms та WPF.

Завантаження під час виконання включає такі компоненти:

– За бажанням, робочий стіл або середовище виконання ASP.NET Core.

– Час виконання .NET. Забезпечує систему типів, завантаження збірки, збирач сміття, власну взаємодію та інші основні послуги.

– Бібліотеки виконання. Надає примітивні типи даних та основні утиліти.

– Драйвер dotnet. Команда CLI, яка запускає залежні від фреймворку програми [18].

Git та GitHub

Git – це приклад системи управління розподіленими версіями (DVCS), яка зазвичай використовується для розробки відкритого комерційного та некомерційного програмного забезпечення. DVCS дозволяють отримати повний доступ до кожного файлу, гілки та ітерації проекту та дозволяє кожному користувачеві отримати доступ до повної та самодостатньої історії всіх змін. На відміну від популярних колись централізованих систем управління версіями, «DVCS типу Git не потребує постійного підключення до центрального сховища. Розробники можуть працювати в будь-якому місці та асинхронно співпрацювати з будь-якого часового поясу».

GitHub – це платформа для розміщення коду для контролю версій та співпраці. Він дозволяє спільно працювати над проектами з будь-якого місця та зберігати версії проекту на віддаленому сервері.

2.6.3. Виклик та завантаження програми

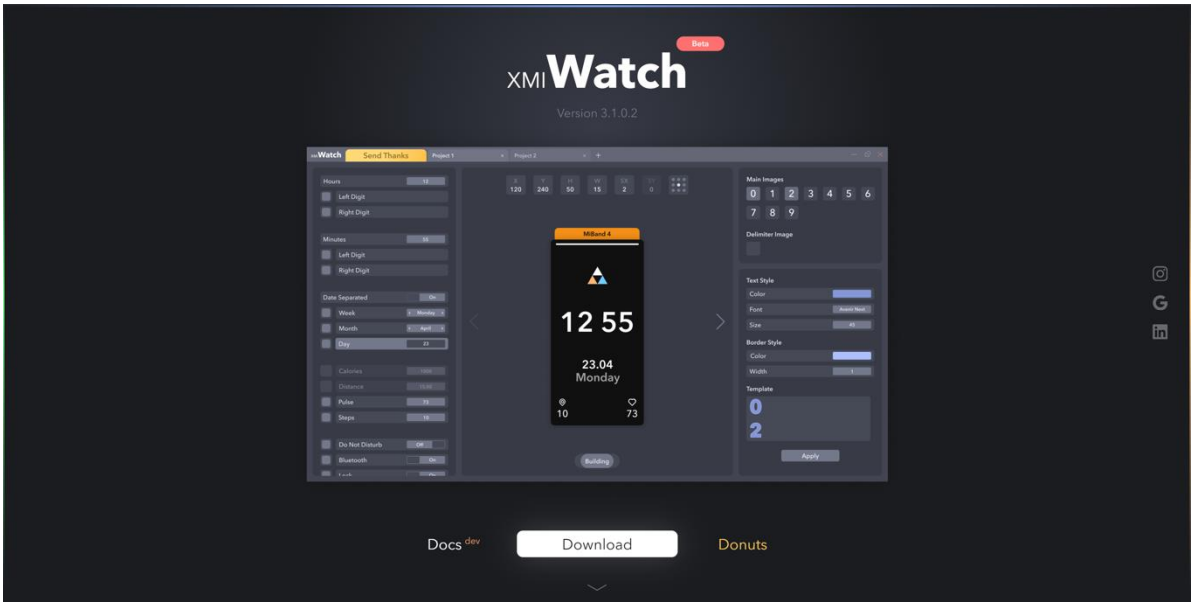

Рис. 2.13. Головна сторінка сайту

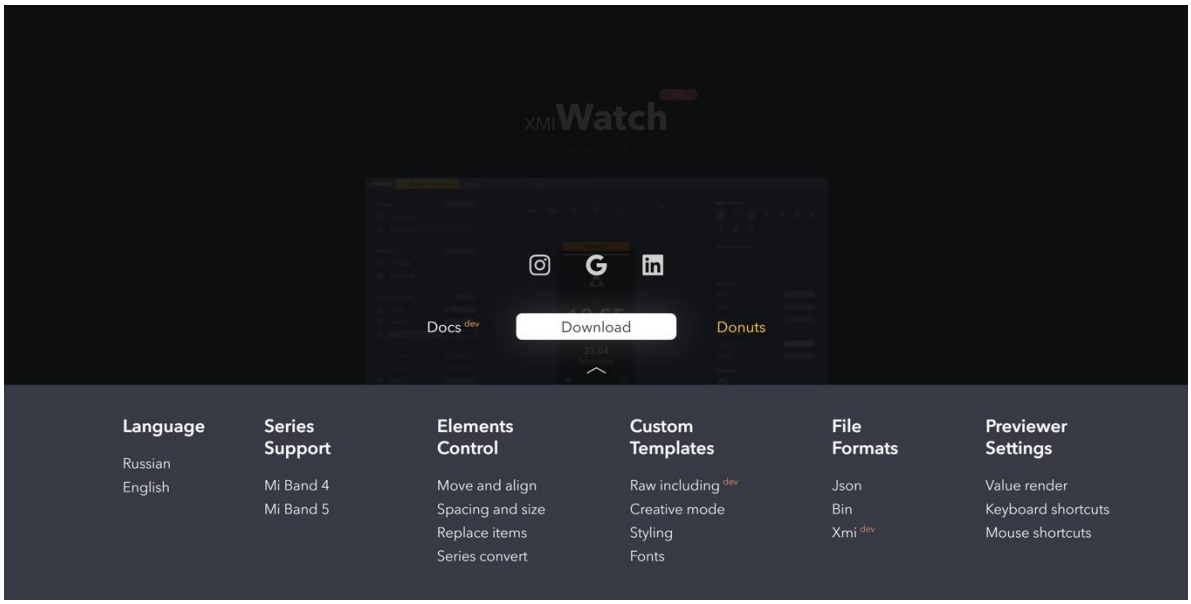

Рис. 2.14. Головна сторінка сайту з футером

Розроблене додаток можна завантажити на офіційній сторінці XMI Watch, де знаходиться beta-версія документації по використанню і можливостям програми. Відразу після скачування додаток буде доступно для запуску на машині з встановленими Windows 7 і вище, а також з встановленої .NET Core runtime x86. Після запуску додаток розпакує необхідні дані в тому ж місці, де знаходиться основний запускає файл (рис. 2.13 і 2.14).

2.6.4. Опис інтерфейсу користувача

За замовчуванням на початку роботи із застосунком користувачу відображається головна сторінка (рис. 2.15). Вона і буде основною, на її базі будуть з’являтися інші компоненти залежно від тієї чи іншої взаємодії користувача з інтерфейсом додатку (рис. 2.16 – 2.39). Такими компонентами взаємодії в більшості своїй є кнопки. Інтерфейс розроблявся з властивістю адаптуватися до екранів пристроїв.

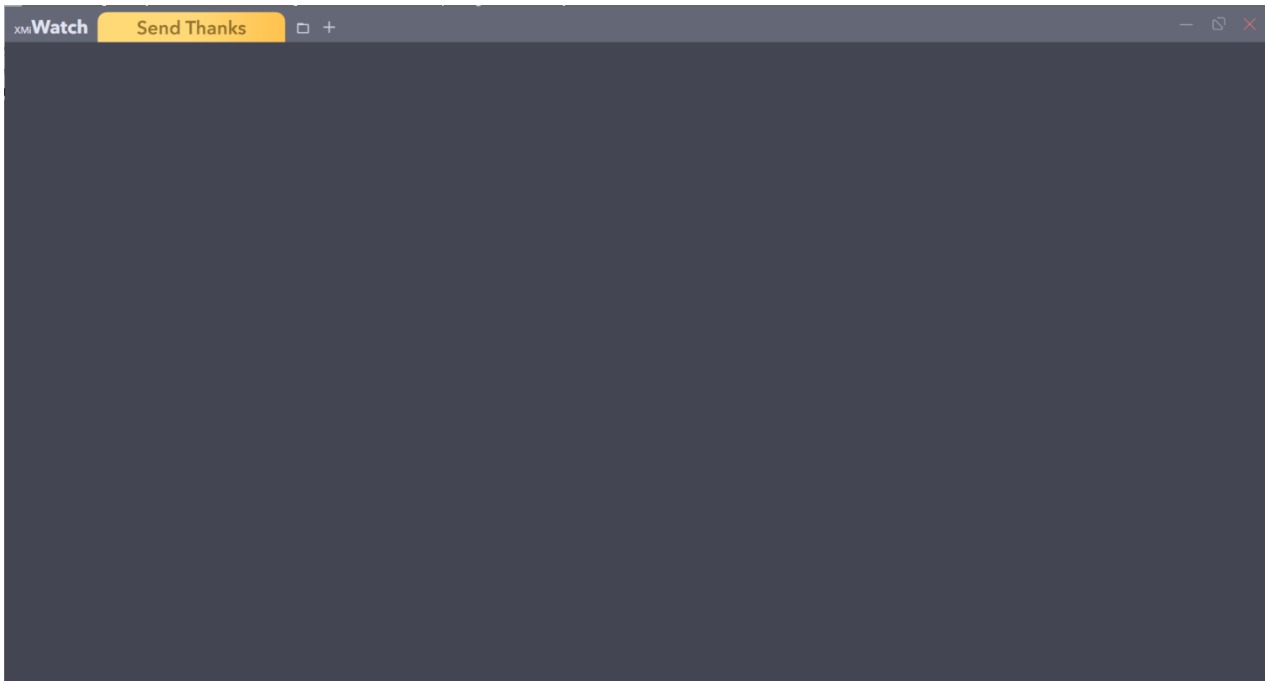

Рис. 2.15 Головне вікно програми

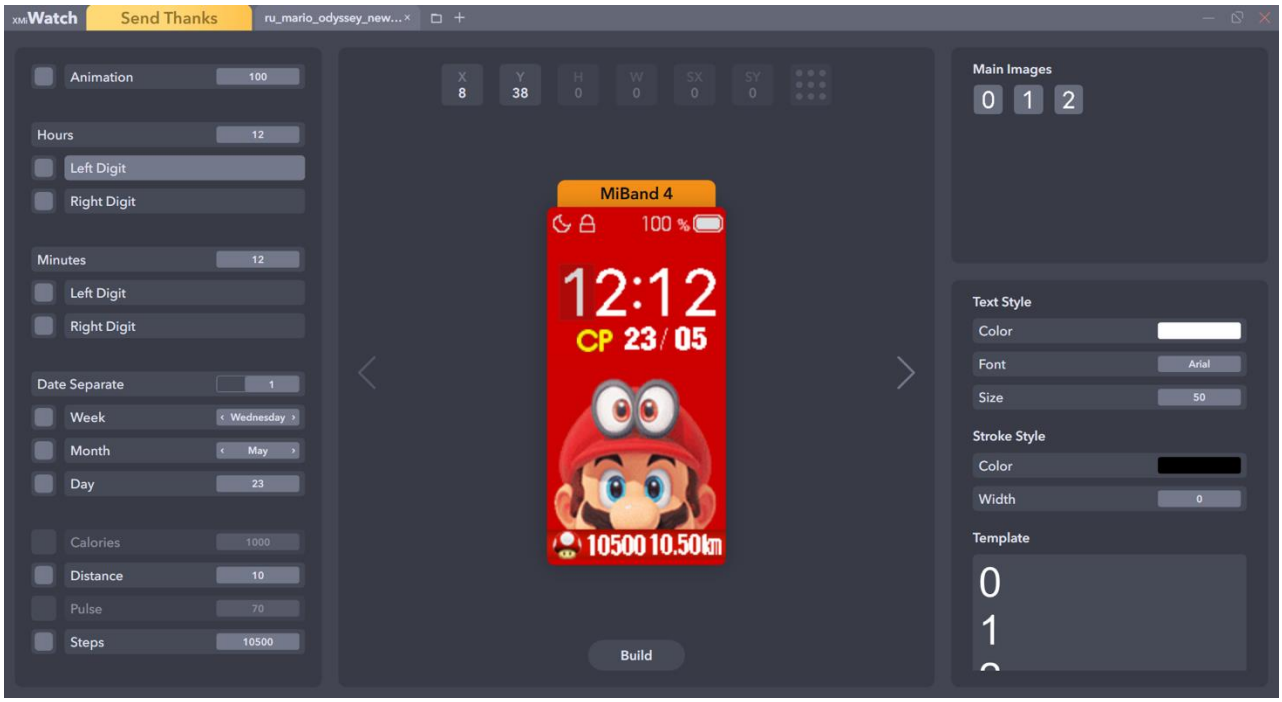

Рис. 2.16. Відкриття циферблату для Mi Band 4

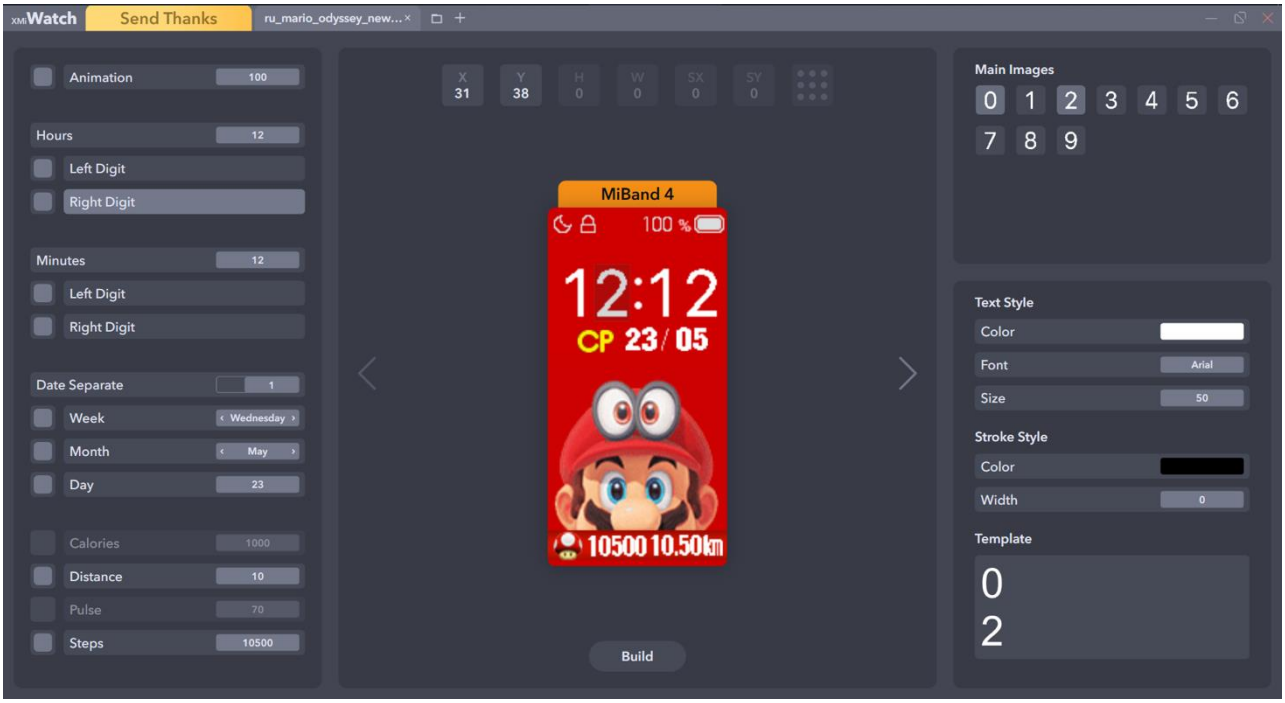

Рис. 2.17. Обирання правої цифри

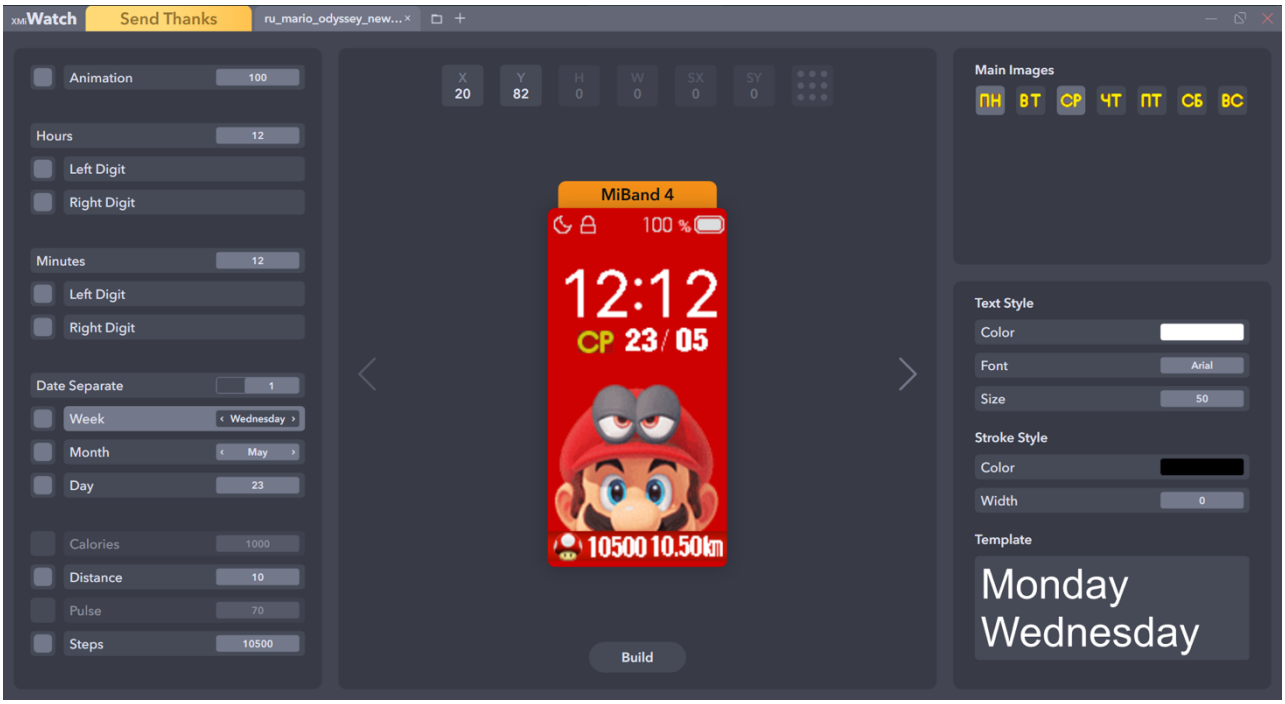

Рис. 2.18. Обирання тижня

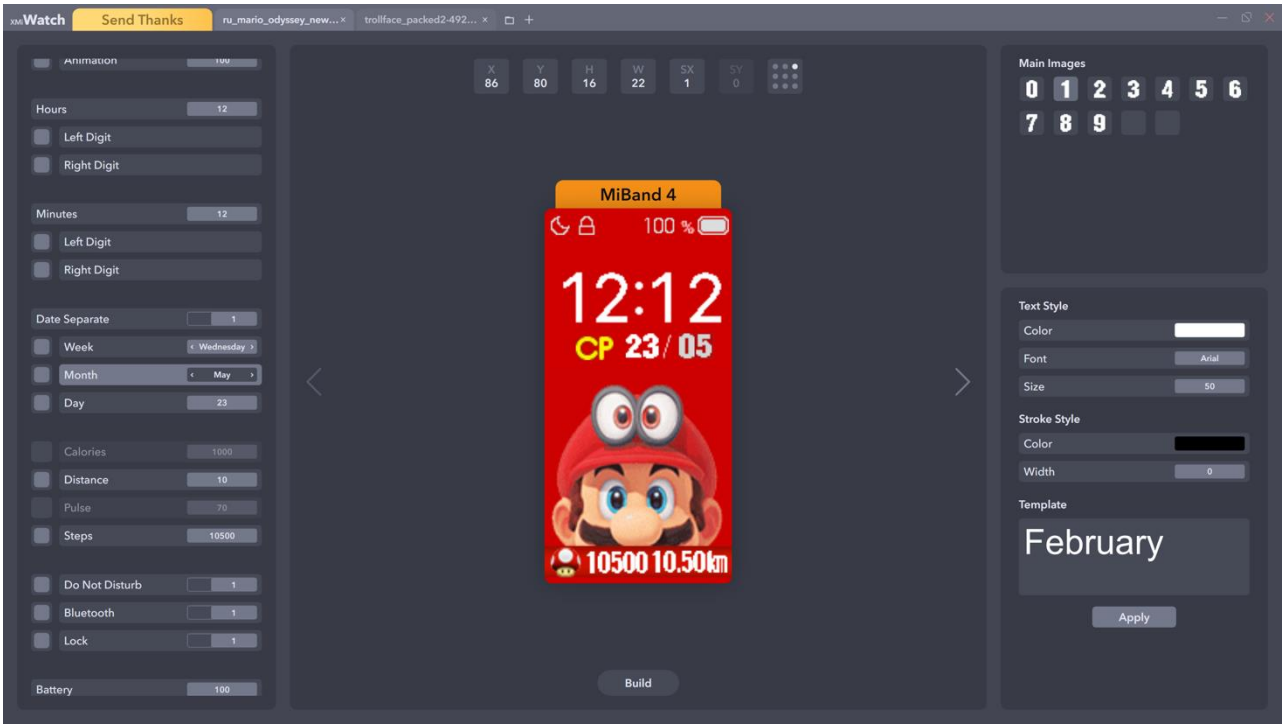

Рис. 2.19. Обирання місяцю

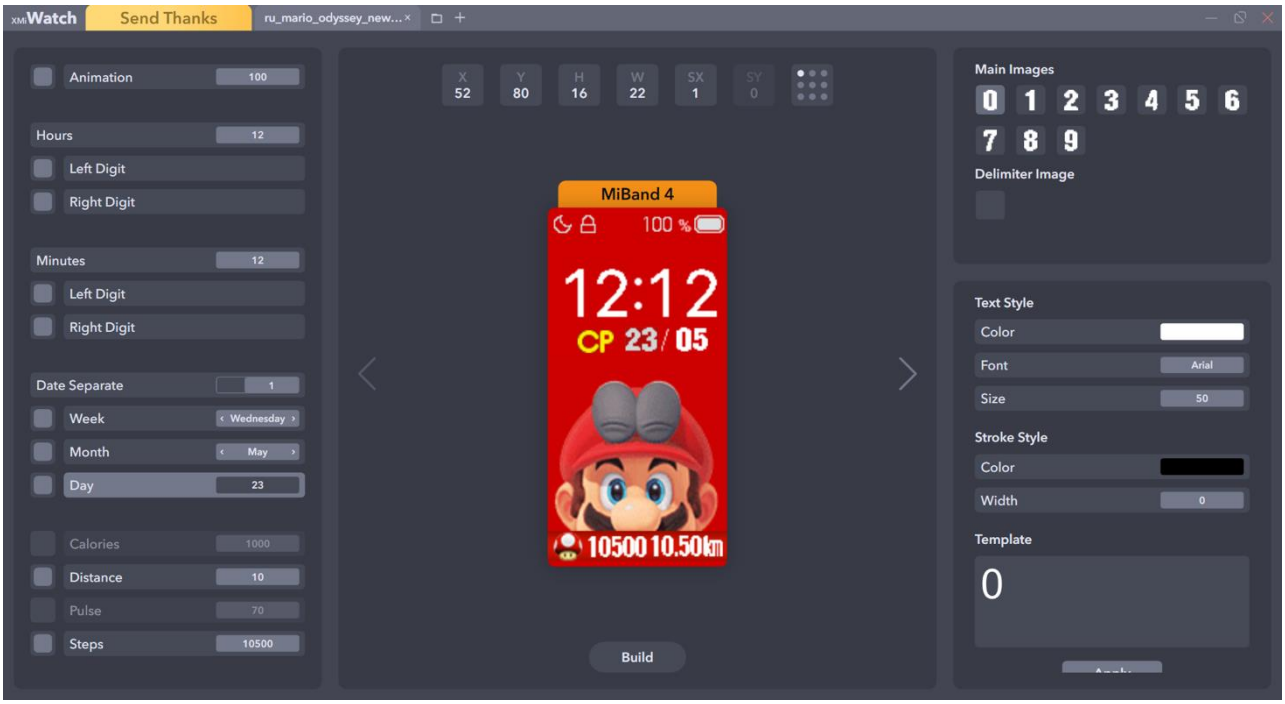

Рис. 2.20. Обирання дню

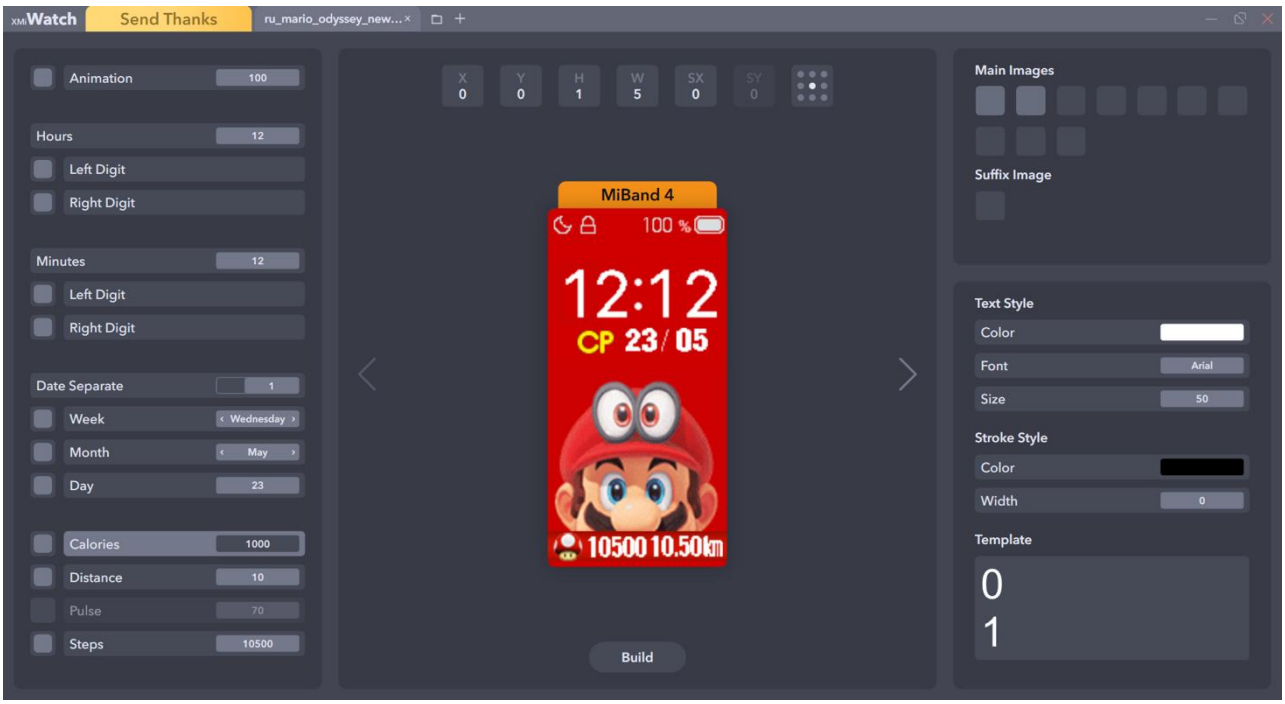

Рис. 2.21. Обирання калорій

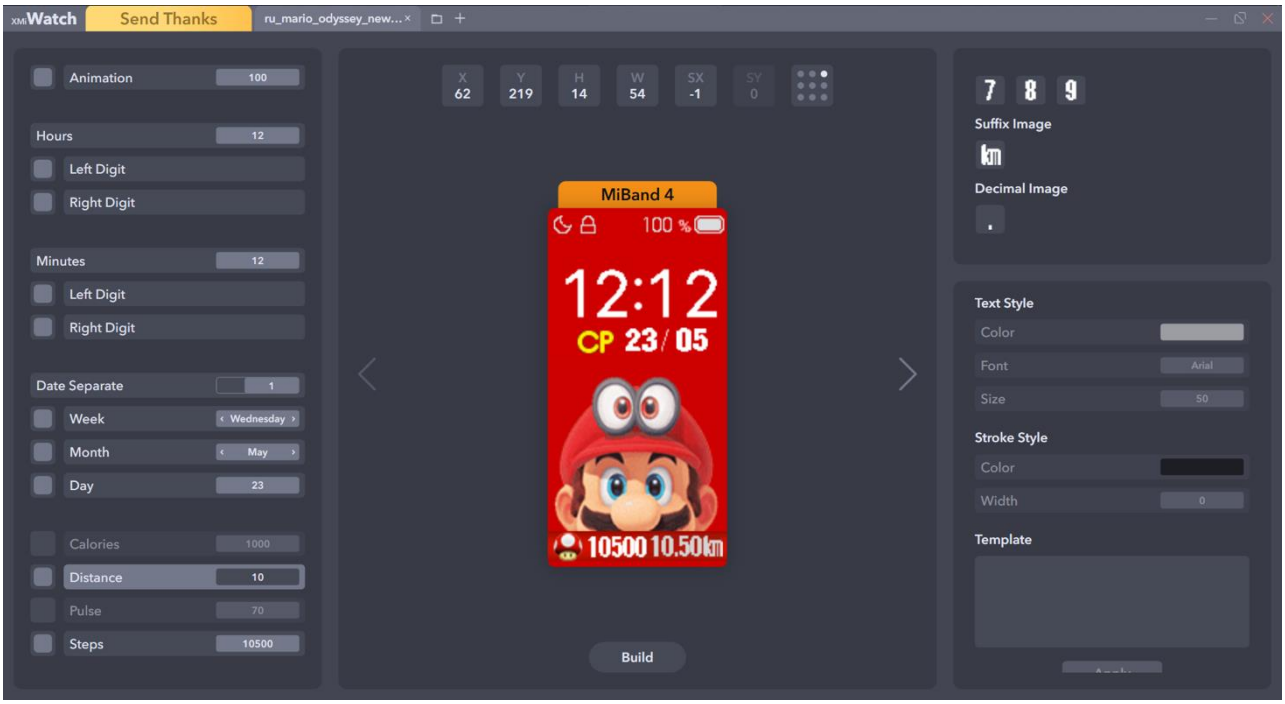

Рис. 2.22. Обирання дистанції

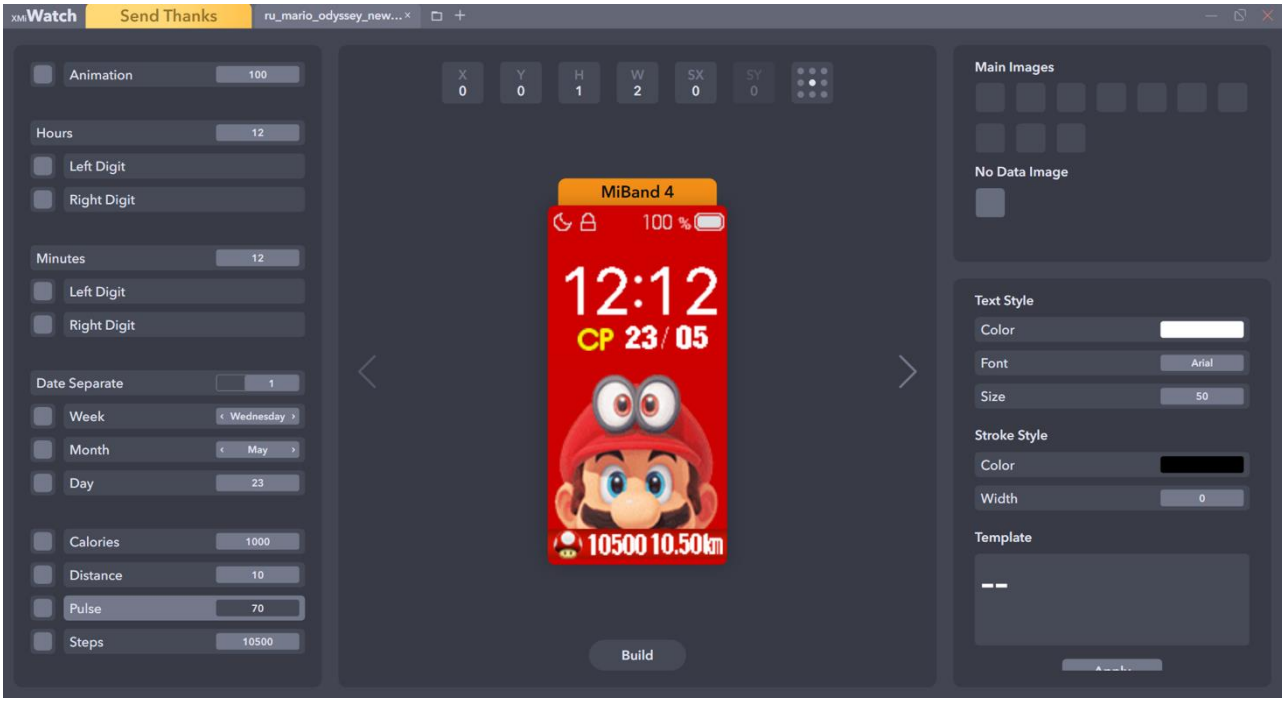

Рис. 2.23. Обирання пульсу

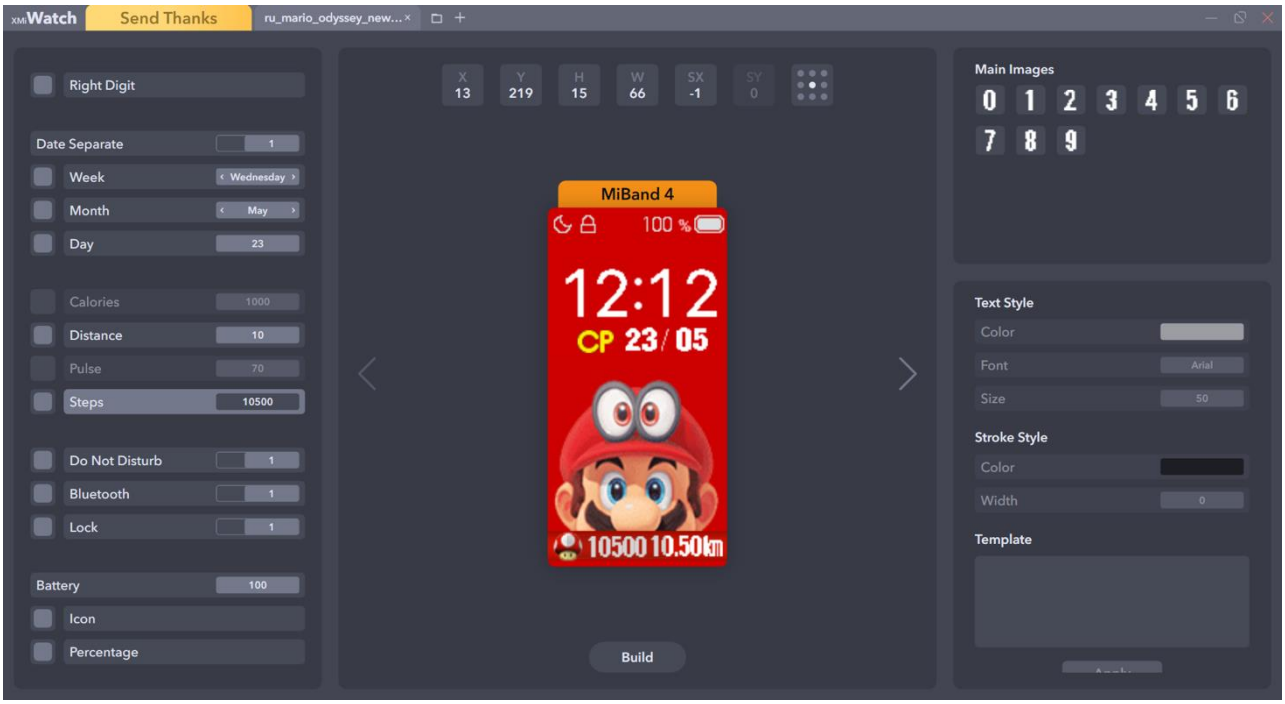

Рис. 2.24. Обирання кроків

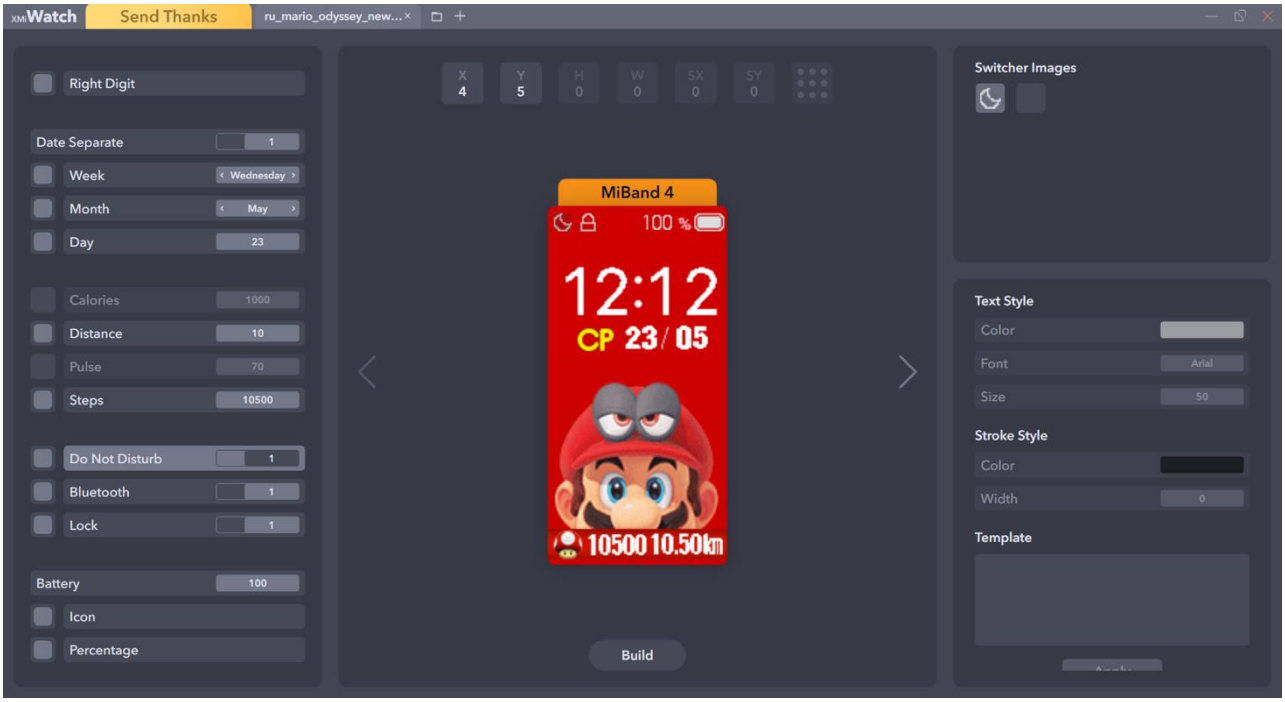

Рис. 2.25. Обирання не турбувати

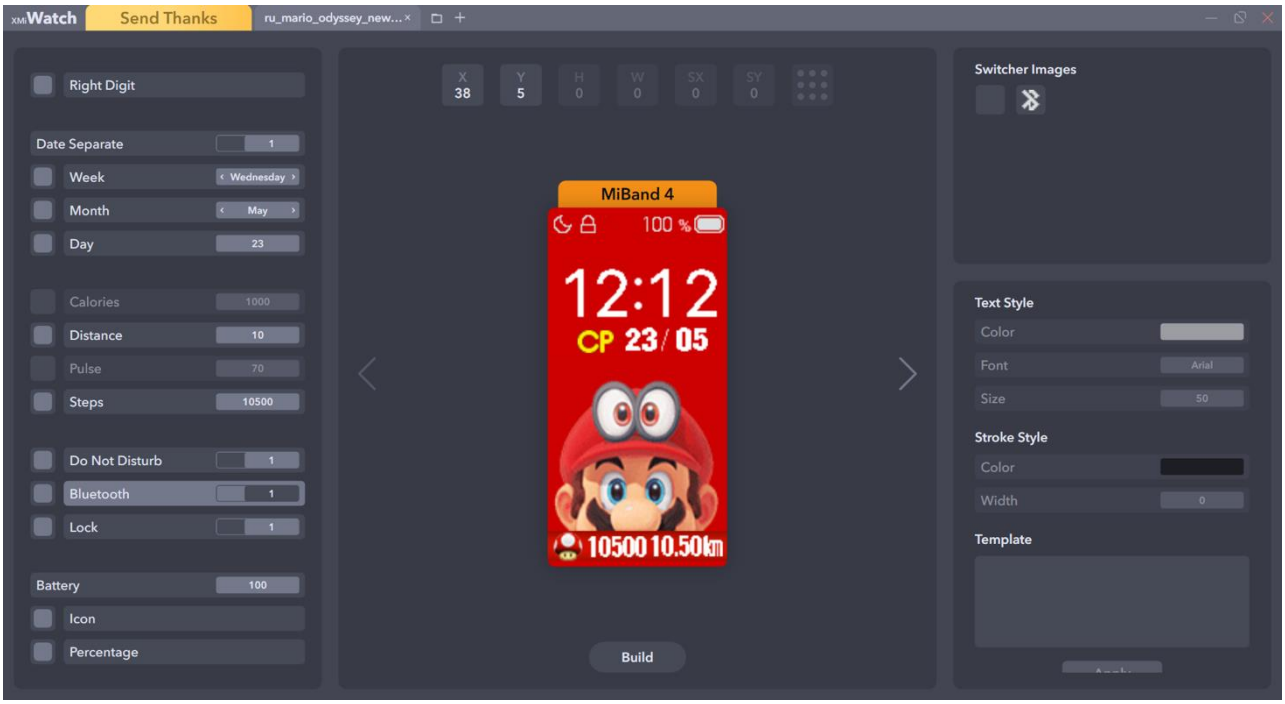

Рис. 2.26. Обирання блютузу

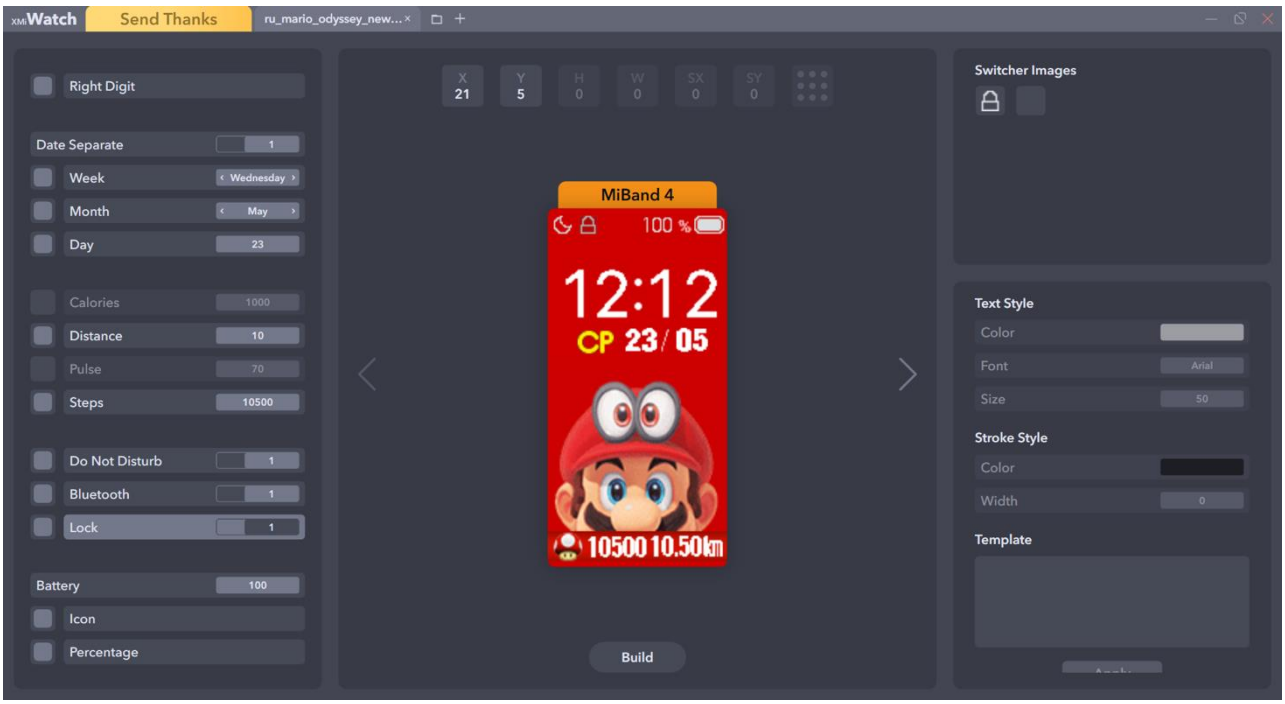

Рис. 2.27. Обирання блокування

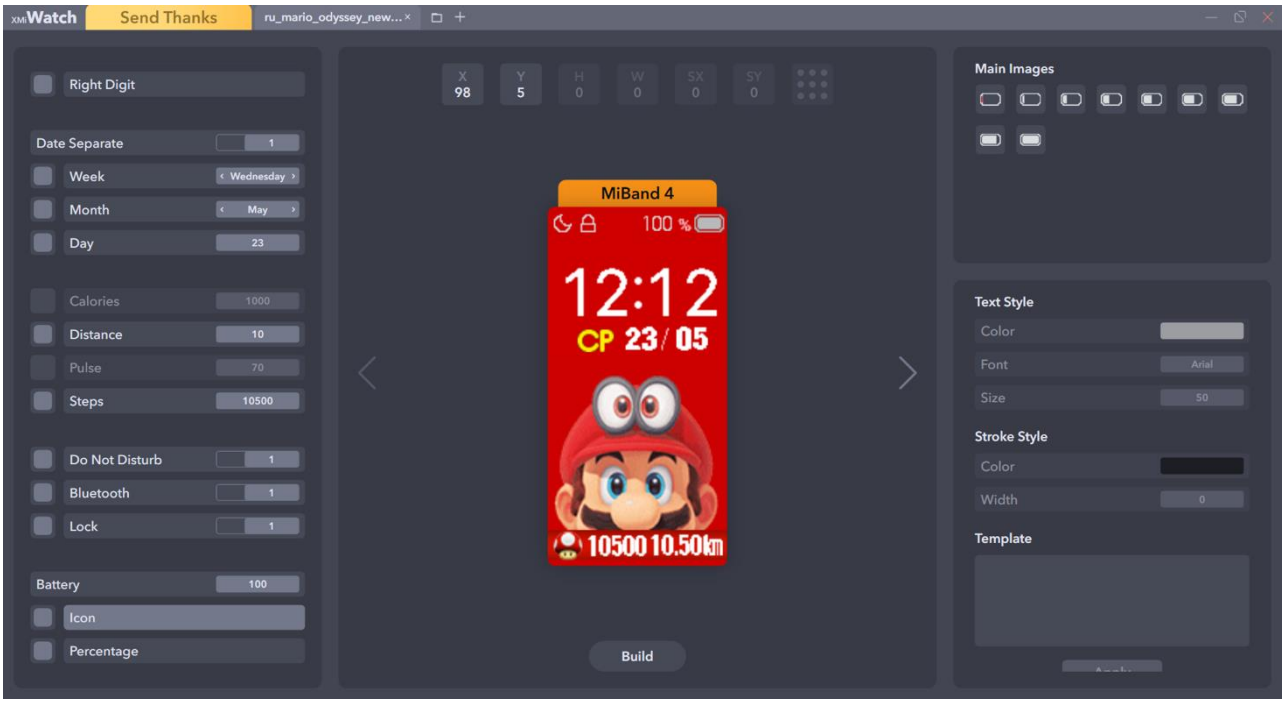

Рис. 2.28. Обирання іконки батареї

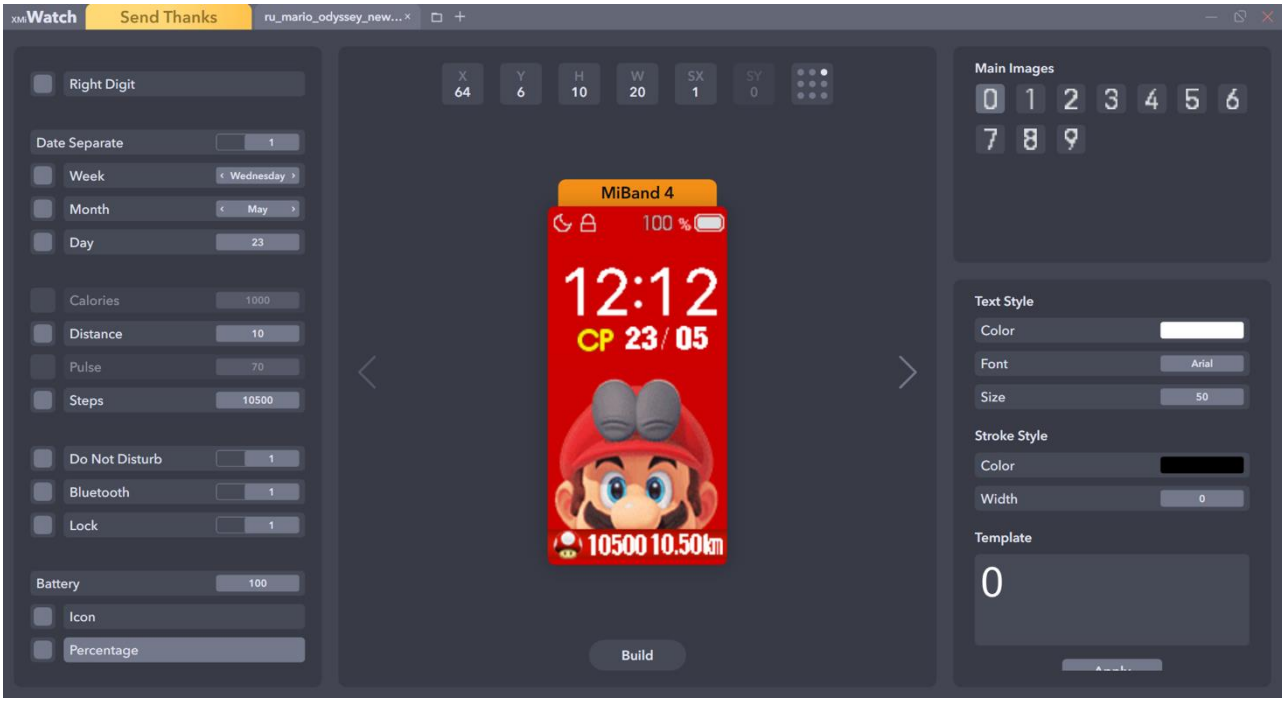

Рис. 2.29. Обирання процентів батареї

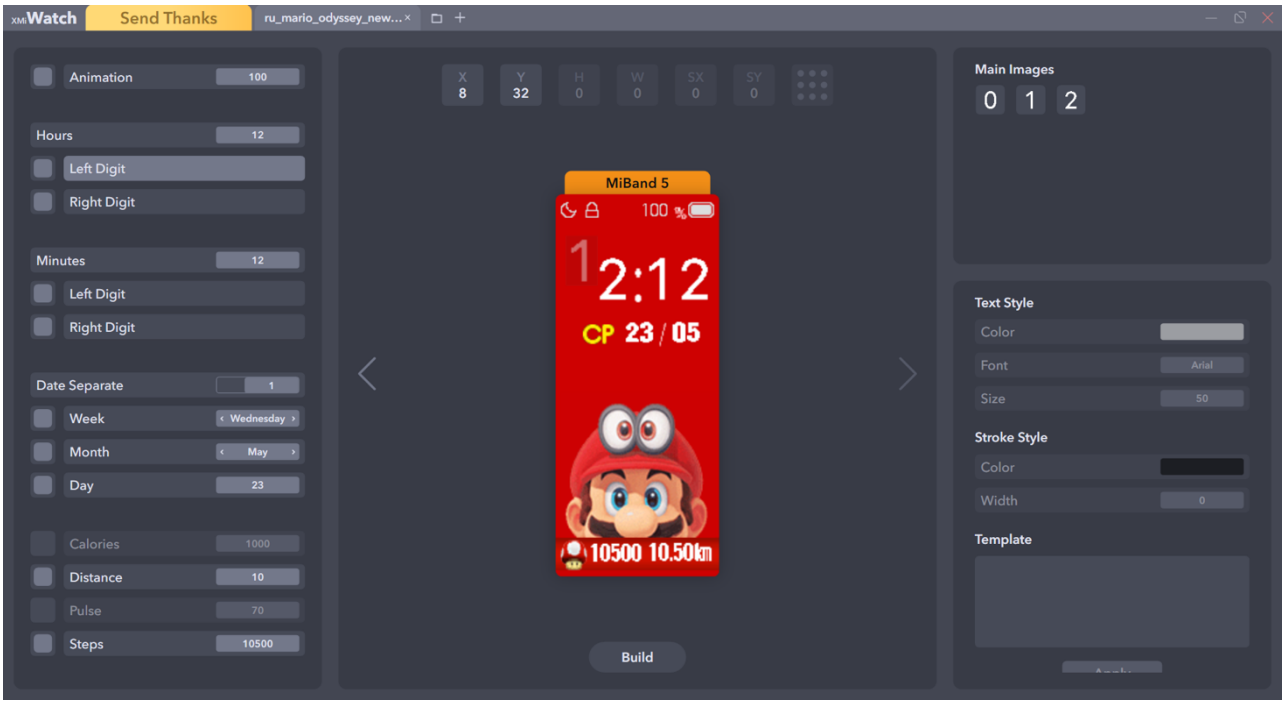

Рис. 2.30. Конвертація та переміщення лівої цифри

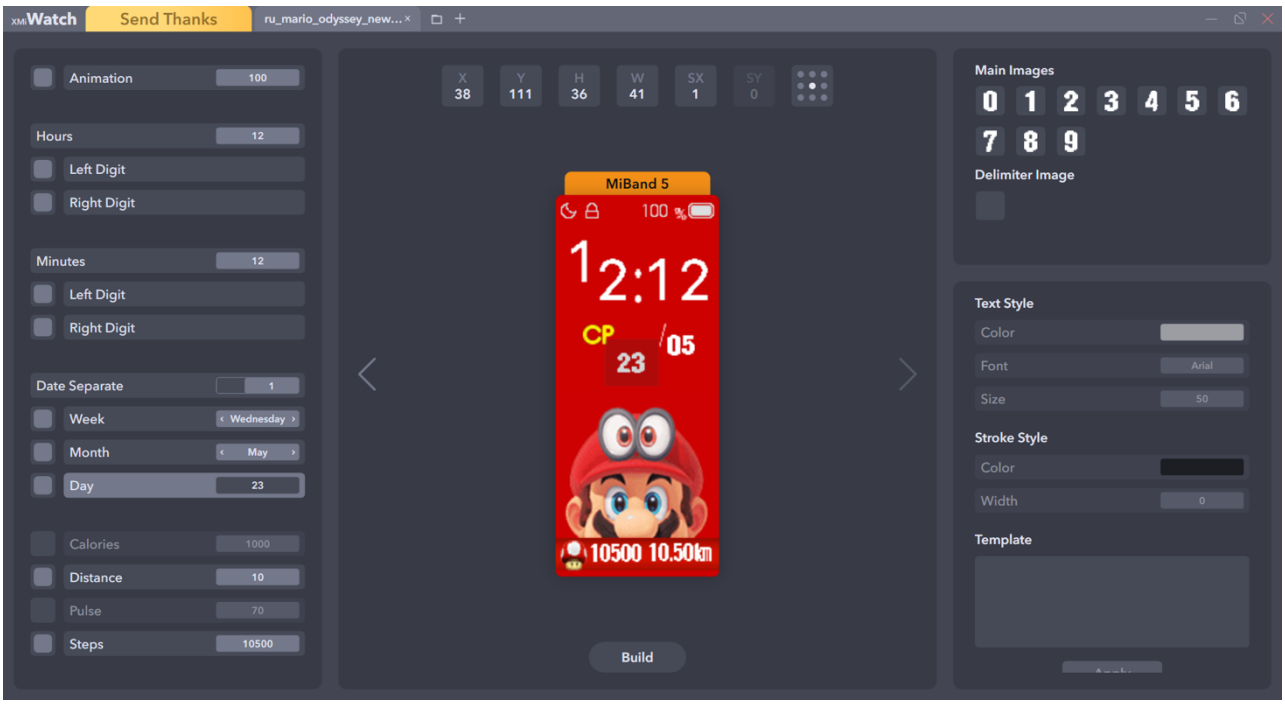

Рис. 2.31. Перміщення та змінення розмірів дню контейнера

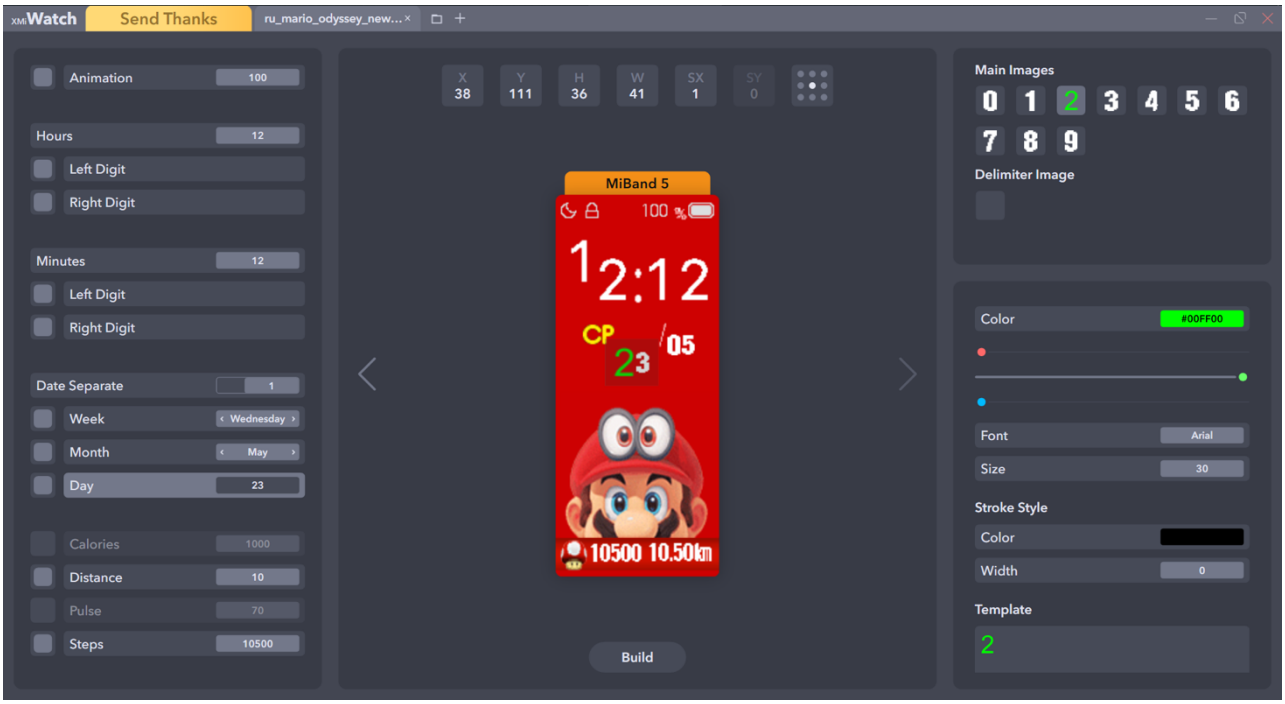

Рис. 2.32. Створення нового стилю для двійки дню

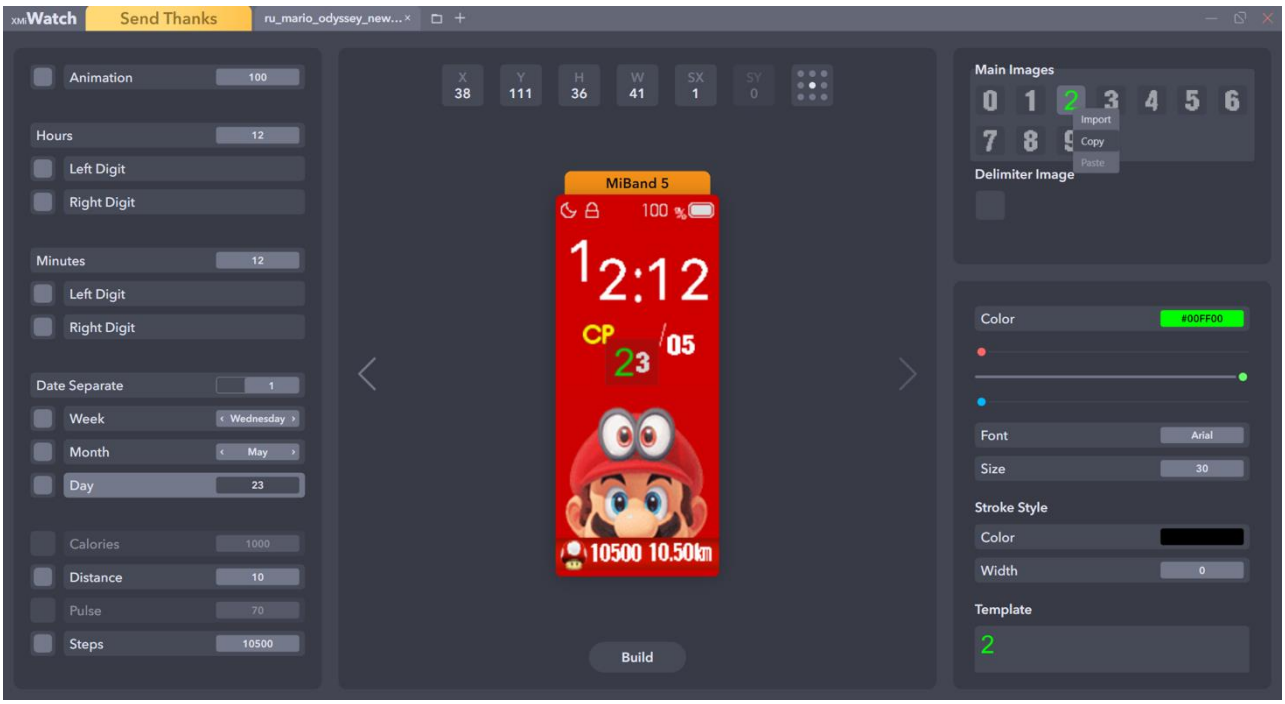

Рис. 2.33. Копіювання двійки дню

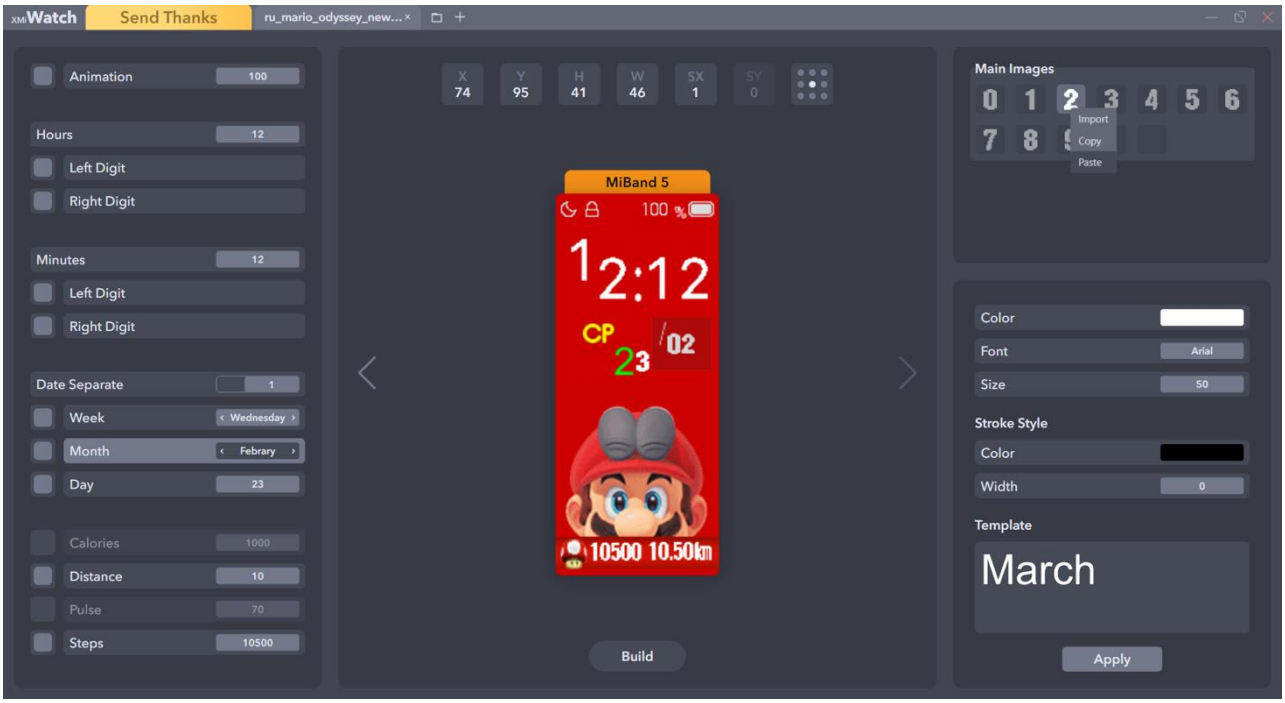

Рис. 2.34. Вставки двійки дню до двійки місяцю

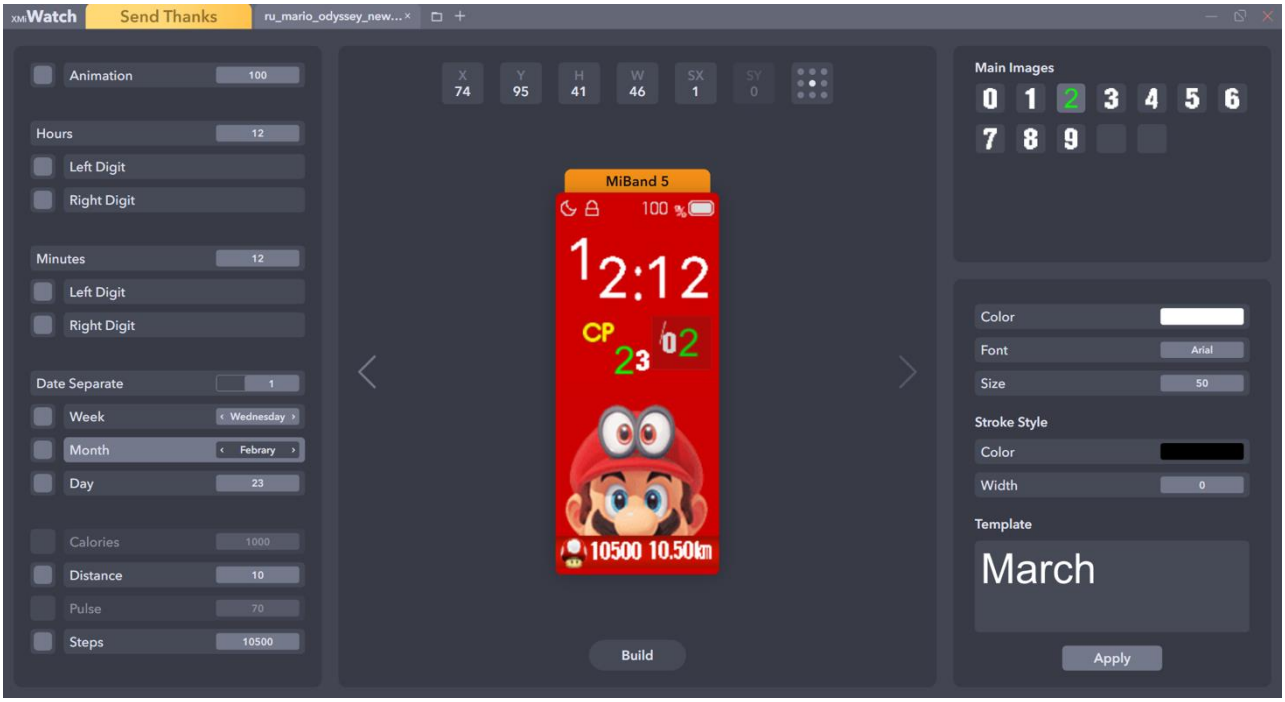

Рис. 2.35. Вставки двійки дню до двійки місяцю

Рис. 2.36. Вставки двійки дню до двійки місяцю

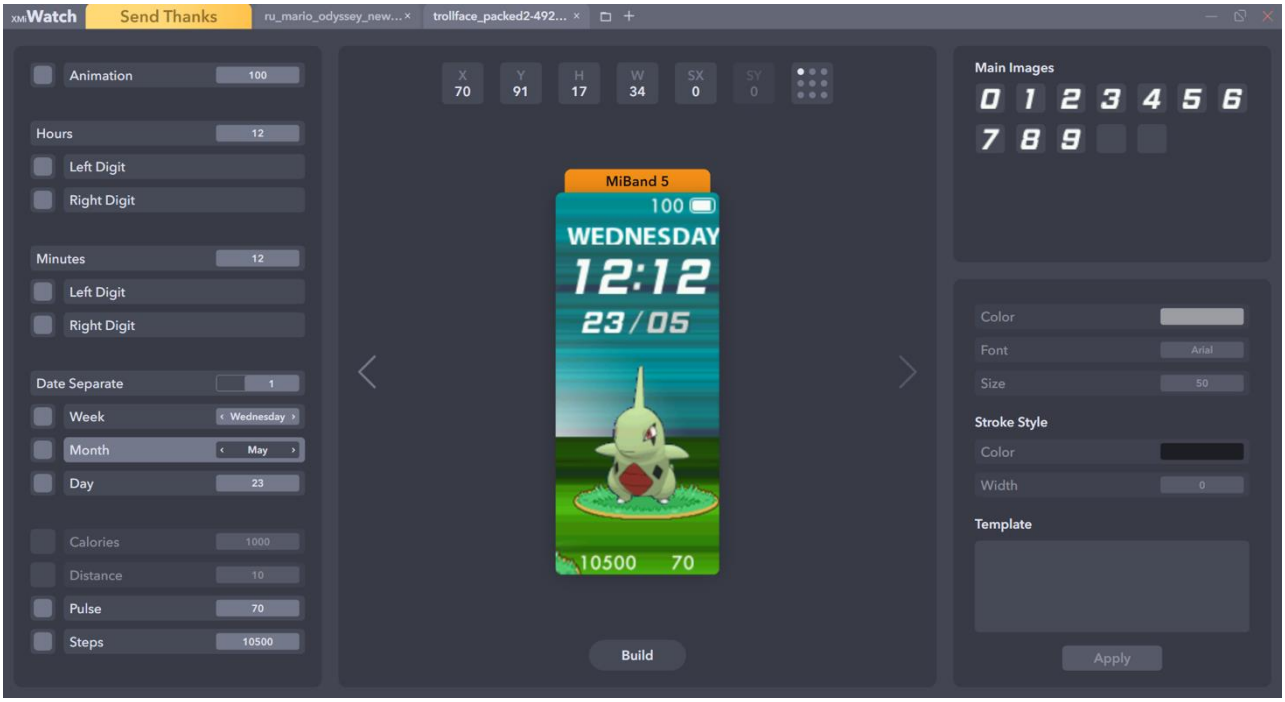

Рис. 2.37. Відкриття циферблату для Mi Band 5

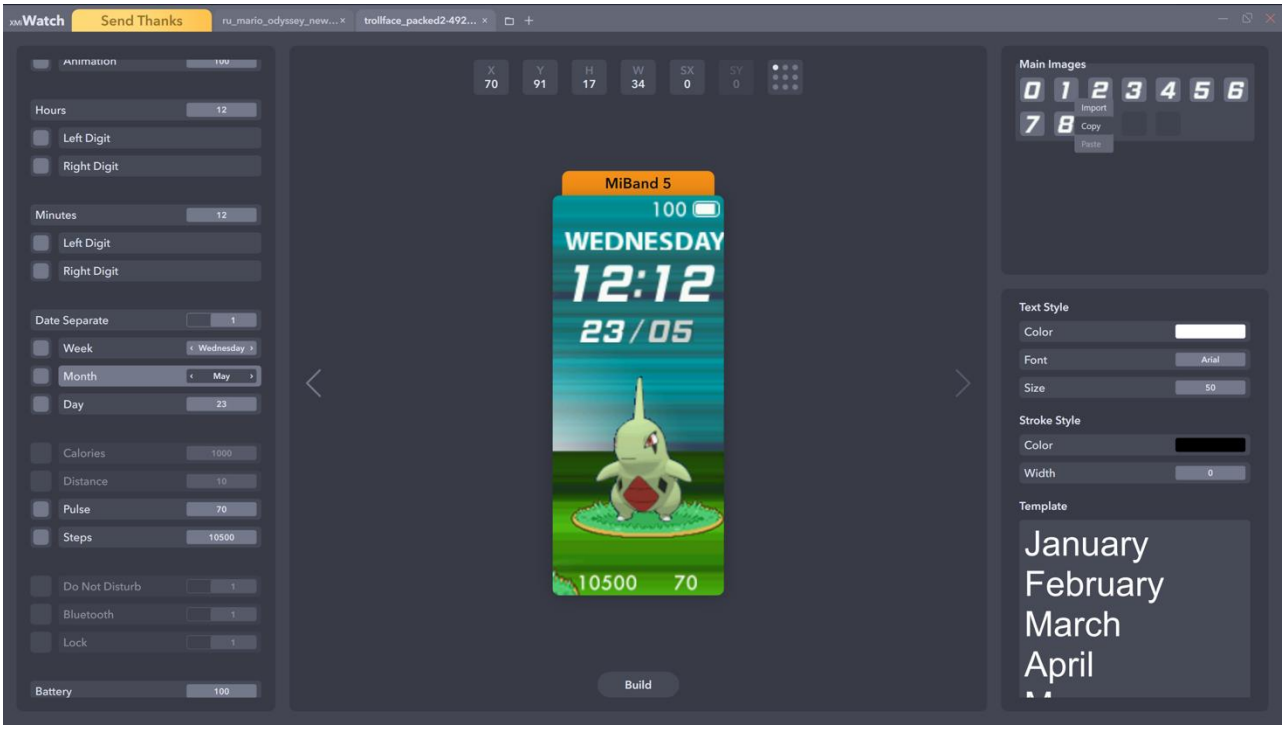

Рис. 2.38. Копіювання цифр місяцю з іншого цифрблата для Mi Band 5

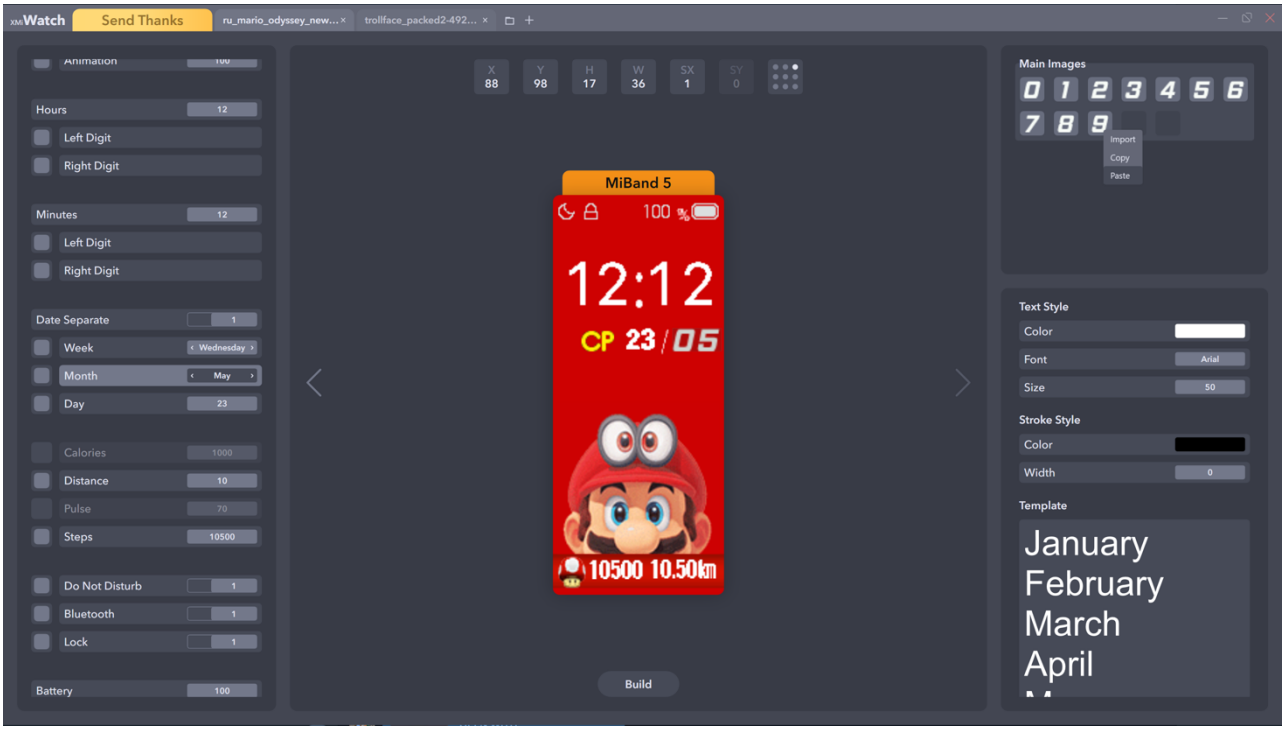

Рис. 2.39. Вставка цифр місяцю з іншого цифрблата для Mi Band 5

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

– передбачуване число операторів програми: 1200;

– коефіцієнт складності програми: 1,6;

– коефіцієнт корекції програми в ході її розробки: 0,07;

– годинна заробітна плата програміста: 70 грн/год;

– коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі: 1,2;

– коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності: 1,2;

– вартість машино-години ЕОМ: 13 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 (людино-годин):

$$t = t_o + t_и + t_п + t_{отл} + t_д \qquad (3.1)$$

де t_o- витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

$t_и$ - витрати праці на дослідження алгоритму рішення задачі;

$t_а$- витрати праці на розробку блок-схеми алгоритму;

$t_п$-витрати праці на програмування по готовій блок-схемі;

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

$t_д$ - витрати праці на підготовку документації.

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

Умовне число операторів (грн):

$$Q = q \cdot C \cdot (1 + p) \tag{3.2}$$

де q – передбачуване число операторів (1200);

C – коефіцієнт складності програми (1,6);

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

Звідси умовне число операторів в програмі:

$$Q = 1{,}6 \cdot 1200 \cdot (1 + 0{,}07) = 2054{,}4$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста (людино-годин):

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k} \tag{3.3}$$

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 3 до 5 років він складає 1,2.

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

$$t_u = \frac{2054{,}4 \cdot 1{,}2}{75 \cdot 1{,}2} = 27{,}39$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою (людино-годин):

$$t_a = \frac{Q}{(20..25) \cdot k} \tag{3.4}$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.4), людино-годин:

$$t_a = \frac{2054{,}4}{20 \cdot 1{,}2} = 85{,}6$$

Витрати на складання програми по готовій схемі (людино-годин):

$$t_{\text{п}} = \frac{Q}{(20..25) \cdot k} \tag{3.5}$$

$$t_{\text{п}} = \frac{2054{,}4}{25 \cdot 1{,}2} = 68{,}48$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання (людино-годин):

$$t_{\text{отл}} = \frac{Q}{(4..5) \cdot k} \tag{3.6}$$

$$t_{\text{отл}} = \frac{2054{,}4}{5 \cdot 1{,}2} = 342{,}4$$

– за умови комплексного налагодження завдання (людино-годин):

$$t_{\text{отл}}^{k} = 1{,}5 \cdot t_{\text{отл}} \tag{3.7}$$

$$t_{\text{отл}}^{k} = 1{,}5 \cdot 342{,}4 = 513{,}6$$

Витрати праці на підготовку документації визначаються за формулою (людино-годин):

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}} \tag{3.8}$$

де $t_{др}$-трудомісткість підготовки матеріалів і рукопису (людино-годин):

$$t_{\text{др}} = \frac{Q}{(15..20) \cdot k} \tag{3.9}$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації (людино-годин):

$$t_{\text{до}} = 0{,}75 \cdot t_{\text{др}} \tag{3.10}$$

Підставляючи відповідні значення, отримаємо (людино-годин):

$$t_{\text{др}} = \frac{1200}{18 \cdot 1{,}2} = 55{,}56$$

$$t_{\text{до}} = 0{,}75 \cdot 55{,}56 = 41{,}67$$

$$t_{\text{д}} = 55{,}56 + 41{,}67 = 97{,}23$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення (людино-годин):

$$t = 50 + 27{,}39 + 68{,}48 + 342{,}4 + 97{,}23 = 585{,}5$$

У результаті ми розрахували, що в загальній складності необхідно 585,5 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення додатку

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ (грн):

$$\text{К}_{\text{ПО}} = \text{З}_{\text{ЗП}} + \text{З}_{\text{МВ}} \tag{3.11}$$

Заробітна плата виконавців визначається за формулою (грн):

$$\text{З}_{\text{ЗП}} = t + \text{С}_{\text{ПР}} \tag{3.12}$$

де: t - загальна трудомісткість, людино-годин;

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

З урахуванням того, що середня годинна зарплата програміста становить 65 грн / год, отримуємо (грн):

$$З_{ЗП} = 585{,}5 \cdot 65 = 38057{,}5$$

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

$$З_{МВ} = t_{отл} \cdot C_{мч} \quad (3.13)$$

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

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

Підставивши в формулу (3.3) відповідні значення, визначимо вартість необхідного для налагодження машинного часу (грн):

$$З_{МВ} = 342{,}4 \cdot 13 = 4451{,}2$$

Звідси витрати на створення програмного продукту (грн):

$$К_{ПО} = 38057{,}5 + 4451{,}2 = 42508{,}7$$

Очікуваний період створення ПЗ (міс):

$$T = \frac{t}{B_k \cdot F_p} \quad (3.14)$$

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 (міс):

$$T = \frac{585{,}5}{1 \cdot 176} \approx 3{,}3$$

Висновок**:** додаток розроблений з метою спростити процес створення циферблатів та їх редагування. Вартість даного додатку становить 42508,7 грн. та не потребує додаткових витрат при розробці проекту. Очікуваний час розробки приблизно 3,3 місяці. Цей термін пов'язаний зі значною кількістю операторів і включає в себе час для дослідження та розробки алгоритму розв'язання задачі, розробку дизайну, створення додатку та підготовку документації.

ВИСНОВКИ

В ході написання дипломного проекту були отримані цінні знання застосування сучасних патернів програмування для побудови якісної архітектури і структури програми на .NET Core, а також створення грамотного і красивого UI / UX в додатку Figma.

Було розроблено додаток для створення, редагування і конвертації циферблатів актуальною лінійки Mi Band 4 і 5, де продумана можливість розширення для майбутнього модельного ряду розумних браслетів від компанії Xiaomi (з функцією авто-оновлення).

Дане рішення дозволяє відкривати і зберігати файли в двох форматах *.bin (упакований вид), а також *.json + дані у вигляді зображень *.png (розпакований вид).

Додаток має вбудований рендер-майстер модуль, який дозволяє емулювати поведінку годинника при різноманітних вхідних даних, в ньому можна вказати:

- час
- дату
- калорії
- дистанцію
- кроки
- пульс
- віджети станів
- батарею

Також можна підключати не доданий в циферблат елементи або відключати вже існуючі (наприклад додати калорії, дистанцію і т.д.)

Існує можливість переносити окремі елементи частково або повністю з інших відкритих циферблатів або імпортувати картинки з дискового простору на комп'ютері (наприклад картинки цифр для дати).

У додатку доданий модуль для створення власних стилізованих картинок з текстом, де можна налаштувати: шрифт, розмір, обведення і колір, після чого застосувати і згенерувати в обрані картинки.

На даний момент програму скачало сумарно понад 4500 осіб і кількість продовжує зростати, для досягнення такої цифри воно було розміщено на різних форумах, в тому числі на популярному закордонному майданчику Reddit.

В «Економічному розділі» визначено трудомісткість розробки програмного забезпечення (585,5 чол-год), підраховані витрати на створення програмного забезпечення (42508,7 грн.) і гаданий період розробки (3 міс.).

СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. Ian Griffiths, Chris Sells, Programming WPF: Building Windows UI with Windows Presentation Foundation. 2007. 873 с.

2. Adam Nathan, WPF 4 Unleashed. 2010. 1134 с.

3. Chris Andrade, Shawn Livermore, Michael Meyers, Scott Van Vliet, Professional WPF Programming: .NET Development with the Windows Presentation Foundation. 2007. 482 с.

4. Ryan Vice, Muhammad Shujaat Siddiqi, MVVM Survival Guide for Enterprise Architectures in Silverlight and WPF. 2012. 913 с.

5. Matthew MacDonald, Pro WPF 4.5 in C#: Windows Presentation Foundation in .NET 4.5. 2012. 1092 с.

6. Pavan Podila, Kevin Hoffman, WPF Control Development Unleashed: Building Advanced User Experiences. 2009. 504 с.

7. Mamta Dalal, Ashish Ghoda, XAML Developer Reference. 2011. 601 с.

8. Philip Japikse, Andrew Troelsen, C# 6.0 and the .NET 4.6 Framework. 2015. 1660 с.

9. Alessandro Del Sole, WPF Debugging and Performance Succinctly. 2017. 79 с.

10. Ian Griffiths, Programming C# 8.0: Build Cloud, Web, and Desktop Applications. 2019. 848 с.

11. Vladimir Khorikov, Unit Testing Principles, Practices, and Patterns. 2020. 304 с.

12. Jon Skeet, C# in Depth. 2008. 420 с.

13. Jennifer Greene, Andrew Stellman, Head First C#. 2007. 778 с.

14. Gary McLean Hall, Adaptive Code via C#. 2014. 448 с.

15. Mark J. Price, C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development: Build Applications with C#, .NET Core, Entity Framework Core, ASP.NET Core, and ML.NET Using Visual Studio Code, 4th Edition. 2019. 819 с.

16. Jeffrey Richter, CLR via C#, 4th Edition. 2012. 896 с.

17. Refactoring and Design Patterns. URL: https://refactoring.guru

18. C# docs - get started. URL: https://docs.microsoft.com/en-us/dotnet/csharp

19. Windows Presentation Foundation for .NET 5 and .NET Core documentation. URL: https://docs.microsoft.com/en-us/dotnet/desktop/wpf

20. C# и .NET. URL: https://metanit.com/sharp/general.php

21. The complete WPF tutorial. URL: https://wpf-tutorial.com

22. Основа WPF. URL: https://professorweb.ru/my/WPF/base_WPF/level1/base_WPF_index.php

23. WPF - Dependency Properties. URL: https://www.tutorialspoint.com/wpf/wpf_dependency_properties.htm

24. A Simple Pattern for Creating Re-useable UserControls in WPF / Silverlight. URL: https://blog.scottlogic.com/2012/02/06/a-simple-pattern-for-creating-re-useable-usercontrols-in-wpf-silverlight.html

25. Xiaomi Mi Band. URL: https://en.wikipedia.org/wiki/Xiaomi_Mi_Band

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

```
// System Libs
using System.Diagnostics;
using System.IO;
using System.Reflection;
using System.Windows;

// Solution Libs
using XMIWatch.Resources;
using XMIWatch.Watch.Services.ProjectIO;

namespace XMIWatch
{
   public partial class App : Application
   {
      protected override void OnStartup(StartupEventArgs e)
      {
         if (!Directory.Exists(ProjectManager.TempFolder))
         {
            var dir = Directory.CreateDirectory(ProjectManager.TempFolder);
            dir.Attributes = FileAttributes.Directory | FileAttributes.Hidden;
         }

         try
         {
            // If start by updater
            if (e.Args.Length > 0 && e.Args?[0] == "upd")
               return;


            var updater = "GitUpdate.exe";
            var updaterActualVer = "1.1.0.1";
            var updaterCurrentVer = updaterActualVer;

            if (File.Exists(updater))
               updaterCurrentVer = FileVersionInfo.GetVersionInfo(updater).FileVersion;

            if (!File.Exists(updater) || updaterCurrentVer != updaterActualVer)
               File.WriteAllBytes(updater, Res.GitUpdate);


            string appName = Process.GetCurrentProcess().MainModule.FileName;
            string version = Assembly
               .GetExecutingAssembly()
               .GetName()
               .Version
               .ToString();


            Process.Start(new ProcessStartInfo()
            {
               FileName = updater,
               Arguments =

               // Git project name
               $"xmi" +

               // App version
               $" {version}" +

               // App name to start after update
```

```
                $" {appName}",
            });
            Process.GetCurrentProcess().Kill();
        }
        catch { }
        finally { base.OnStartup(e); }
    }
}
}
// System Libs
using System.Windows;
using System.Windows.Input;
using System.Windows.Controls;
using System.ComponentModel;
using System.Collections.ObjectModel;
using System.Diagnostics;

// Solution Libs
using XMIWatch.Watch;
using XMIWatch.Watch.Services.ProjectIO;
using XMIWatch.Services;
using XMIWatch.Watch.Services;
using XMIWatch.Resources.Localization;
using XMIWatch.UIControls.Extensions;

namespace XMIWatch
{
   public partial class MainUI : Window
   {
      public MainUI()
      {
         InitializeComponent();

         KeyboardNavigation.SetDirectionalNavigation(this, KeyboardNavigationMode.Once);
         KeyboardNavigation.SetTabNavigation(this, KeyboardNavigationMode.None);

         Width = SystemParameters.PrimaryScreenWidth / 1.2;
         Height = SystemParameters.PrimaryScreenHeight / 1.2;
      }

      private void Donate(object sender, RoutedEventArgs e)
         => Process.Start(new ProcessStartInfo("https://www.donationalerts.com/r/jefoces") { UseShellExecute = true
});

      private void AppMenuSelected(object sender, RoutedEventArgs e)
      {
         var item = (MenuItem)sender;

         switch (item.Tag.ToString())
         {
            case "RU":
               AppSettings.Instance.Localization = AppLanguage.RU;
               break;

            case "EN":
               AppSettings.Instance.Localization = AppLanguage.EN;
               break;
         }
      }

      private void OnClosing(object sender, CancelEventArgs e)
      {
         if (ProjectManager.Opened.Count > 0)
         {
            var (Message, Title) = LocalizationProvider
```

```
                .Localization
                .Value
                .Interface
                .ClosingApp;

            e.Cancel = !Notify.Question(Message, Title);
        }
    }

    private void WheelScroll(object sender, MouseWheelEventArgs e)
        => ((ScrollViewer)sender).VerticalScroll(e);
  }

  public class MainModel
  {
    public MainModel()
    {
      OpenedProjects = ProjectManager.Opened;
      Project = ProjectManager.Selected;

      //ProjectManager.Open(@"P:\Test Watchfaces\mi band 4\mwf\mwf.json");
    }


    public Observe<WatchProject> Project { get; }
    public ReadOnlyObservableCollection<WatchProject> OpenedProjects { get; }
  }
}
using System.Windows;

[assembly: ThemeInfo(
  ResourceDictionaryLocation.None, //where theme specific resource dictionaries are located
                                   //(used if a resource is not found in the page,
                                   // or application resource dictionaries)
  ResourceDictionaryLocation.SourceAssembly //where the generic resource dictionary is located
                                            //(used if a resource is not found in the page,
                                            // app, or any theme specific resource dictionaries)
)]
using System;
using System.Globalization;
using System.Windows.Data;

namespace XMIWatch.Resources.Converters
{
  public class MirrorNumber : IValueConverter
  {
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
      if (value is int)
        return -(int)value;
      else
        return -(double)value;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
      throw new NotImplementedException();
    }
  }
}
using System;
using System.Globalization;
using System.Windows.Data;

namespace XMIWatch.Resources.Converters
```

```
{
    public class IsNotNullBoolean : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            return value != null;
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
using System;
using System.Globalization;
using System.Windows.Data;

namespace XMIWatch.Resources.Converters
{
    public class DevideNumber : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            var devide = double.Parse((string)parameter);

            if (value is int)
                return (int)value / devide;
            else
                return (double)value / devide;
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
using System;
using System.Globalization;
using System.Windows;
using System.Windows.Data;

namespace XMIWatch.Resources.Converters
{
    class BooleanVisibility : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            bool converted;

            if (value is bool) converted = (bool)value;
            else if (value is bool?) converted = (bool?)value == true;
            else converted = value != null;

            return converted == true ? Visibility.Visible : Visibility.Collapsed;
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
//-----------------------------------------------------------------------------
```

```
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace XMIWatch.Resources {
    using System;


    /// <summary>
    ///   A strongly-typed resource class, for looking up localized strings, etc.
    /// </summary>
    // This class was auto-generated by the StronglyTypedResourceBuilder
    // class via a tool like ResGen or Visual Studio.
    // To add or remove a member, edit your .ResX file then rerun ResGen
    // with the /str option, or rebuild your VS project.

[global::System.CodeDom.Compiler.GeneratedCodeAttribute("System.Resources.Tools.StronglyTypedResourceBuild
er", "16.0.0.0")]
    [global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
    [global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]
    internal class Res {

        private static global::System.Resources.ResourceManager resourceMan;

        private static global::System.Globalization.CultureInfo resourceCulture;

        [global::System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1811:AvoidUncalledPrivateCode")]
        internal Res() {
        }

        /// <summary>
        ///   Returns the cached ResourceManager instance used by this class.
        /// </summary>

[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.EditorBrowsableState.
Advanced)]
        internal static global::System.Resources.ResourceManager ResourceManager {
            get {
                if (object.ReferenceEquals(resourceMan, null)) {
                    global::System.Resources.ResourceManager temp = new
global::System.Resources.ResourceManager("XMIWatch.Resources.Res", typeof(Res).Assembly);
                    resourceMan = temp;
                }
                return resourceMan;
            }
        }

        /// <summary>
        ///   Overrides the current thread's CurrentUICulture property for all
        ///   resource lookups using this strongly typed resource class.
        /// </summary>

[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.EditorBrowsableState.
Advanced)]
        internal static global::System.Globalization.CultureInfo Culture {
            get {
                return resourceCulture;
            }
            set {
```

```
            resourceCulture = value;
        }
    }

    /// <summary>
    ///   Looks up a localized resource of type System.Byte[].
    /// </summary>
    internal static byte[] GitUpdate {
        get {
            object obj = ResourceManager.GetObject("GitUpdate", resourceCulture);
            return ((byte[])(obj));
        }
    }
  }
}
using XMIWatch.Resources.Localization.Interfaces;

namespace XMIWatch.Resources.Localization.en_EN
{
  public class LocalizationEn : ILocalization
  {
    private LocalizationEn() { }
    private static LocalizationEn _localization;
    public static LocalizationEn Instance
      => _localization ?? (_localization = new LocalizationEn());

    public IUIWatchElements WatchElements => UIWatchElementsEn.Instance;
    public IUIInterface Interface => UIInterfaceEn.Instance;
  }
}
using System.Collections.Generic;
using XMIWatch.Resources.Localization.Interfaces;

namespace XMIWatch.Resources.Localization.en_EN
{
  public class UIWatchElementsEn : IUIWatchElements
  {
    private readonly List<string> _monthNames;
    private readonly List<string> _weekNames;

    private UIWatchElementsEn()
    {
      _weekNames = new List<string>()
      {
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday",
        "Sunday",
      };

      _monthNames = new List<string>()
      {
        "January",
        "February",
        "March",
        "April",
        "May",
        "June",
        "July",
        "August",
        "September",
        "October",
```

```
        "November",
        "December",
    };
}

private static UIWatchElementsEn _localization;
public static UIWatchElementsEn Instance
    => _localization ??= new UIWatchElementsEn();



public string MainImages => "Main Images";

public string DelimiterImage => "Delimiter Image";

public string SwitcherImages => "Switcher Images";

public string DecimalImage => "Decimal Image";

public string NoDataImage => "No Data Image";

public string SuffixImage => "Suffix Image";



public string Background => "Background";

public string Hours => "Hours";

public string Minutes => "Minutes";

public string LeftDigit => "Left Digit";

public string RightDigit => "Right Digit";

public string DateWeek => "Week";

public string DateDay => "Day";

public string DateMonth => "Month";

public string DateLine => "Date Inline";

public string ActivitySteps => "Steps";

public string ActivityPulse => "Pulse";

public string ActivityCalories => "Calories";

public string ActivityDistance => "Distance";

public string StatusLock => "Lock";

public string StatusBluetooth => "Bluetooth";

public string StatusNotDisturb => "Do Not Disturb";

public string Battery => "Battery";

public string BatteryIcon => "Icon";

public string BatteryPercentage => "Percentage";
```

```
        public List<string> WeekNames => _weekNames;

        public List<string> MonthNames => _monthNames;
    }
}
using XMIWatch.Resources.Localization.Interfaces;

namespace XMIWatch.Resources.Localization.en_EN
{
    public class UIInterfaceEn : IUIInterface
    {
        private UIInterfaceEn() { }
        private static UIInterfaceEn _localization;
        public static UIInterfaceEn Instance
            => _localization ?? (_localization = new UIInterfaceEn());


        public string Copy => "Copy";

        public string Paste => "Paste";

        public string Import => "Import";

        public string Donate => "Send Thanks";

        public string Build => "Build";

        public string Building => "Building";


        // Notify

        public string ProjectsLimit => "You can open 10 project in one time!";

        public string SavingFailed => "Saving failed!";

        public string OpeningFailed => "Opening failed!";

        public string MiBand5Support => "Sorry, MiBand 5 not supported yet :)";

        public string ScreenNotSupport => "Screen size not support!";

        public string FileNotSupport => "File not support!";

        public string MiToolsNotFound => "MiTools not found!";

        public (string Message, string Title) ClosingApp => ("Do you want close without saving?", "App closing..");

        (string Message, string Title) IUIInterface.ClosingProject(string Name)
            => ($"Do you want to save '{Name}' before closing?", "Project closing..");
    }
}
using XMIWatch.Resources.Localization.Interfaces;

namespace XMIWatch.Resources.Localization.ru_RU
{
    public class UIInterfaceRu : IUIInterface
    {
        private UIInterfaceRu() { }
        private static UIInterfaceRu _localization;
        public static UIInterfaceRu Instance
            => _localization ?? (_localization = new UIInterfaceRu());
```

```
        public string Copy => "Копировать";

        public string Paste => "Вставить";

        public string Import => "Заменить";

        public string Donate => "Вкусные Плюшки";

        public string Build => "Собрать";

        public string Building => "Собираю";


        // Notify

        public string ProjectsLimit => "Вы можете открыть не больше 10 проектов одновременно!";

        public string SavingFailed => "Не удалось сохранить!";

        public string OpeningFailed => "Не удалось открыть!";

        public string MiBand5Support => "Извините, поддержка Mi Band 5 появится чуть позже :)";

        public string ScreenNotSupport => "Данный экран не поддерживается!";

        public string FileNotSupport => "Файл не поддерживается!";

        public string MiToolsNotFound => "MiTools не найдено!";

        public (string Message, string Title) ClosingApp => ("Вы хотите закрыть без сохранения?", "Закрытие
приложения..");

        (string Message, string Title) IUIInterface.ClosingProject(string Name)
            => ($"Вы хотите сохранить '{Name}' перед закрытием?", "Закрытие проекта..");
    }
}
using System.Collections.Generic;
using XMIWatch.Resources.Localization.Interfaces;

namespace XMIWatch.Resources.Localization.ru_RU
{
    public class UIWatchElementsRu: IUIWatchElements
    {
        private readonly List<string> _monthNames;
        private readonly List<string> _weekNames;

        private UIWatchElementsRu()
        {
            _weekNames = new List<string>()
            {
                "Понедельн.",
                "Вторник",
                "Среда",
                "Четверг",
                "Пятница",
                "Суббота",
                "Воскресен.",
            };

            _monthNames = new List<string>()
            {
                "Январь",
                "Февраль",
                "Март",
                "Апрель",
```

```
        "Май",
        "Июнь",
        "Июль",
        "Август",
        "Сентябрь",
        "Октябрь",
        "Ноябрь",
        "Декабрь",
    };
}

private static UIWatchElementsRu _localization;
public static UIWatchElementsRu Instance
    => _localization ??= new UIWatchElementsRu();



public string MainImages => "Основные изображения";

public string DelimiterImage => "Разделитель";

public string SwitcherImages => "Переключатель";

public string DecimalImage => "Точка";

public string NoDataImage => "Без данных";

public string SuffixImage => "Суффикс";



public string Background => "Фон";

public string Hours => "Часов";

public string Minutes => "Минут";

public string LeftDigit => "Левая цифра";

public string RightDigit => "Правая цифра";

public string DateWeek => "День недели";

public string DateDay => "Число";

public string DateMonth => "Месяц";

public string DateLine => "Дата блоком";

public string ActivitySteps => "Шаги";

public string ActivityPulse => "Пульс";

public string ActivityCalories => "Калории";

public string ActivityDistance => "Дистанция";

public string StatusLock => "Блокировка";

public string StatusBluetooth => "Bluetooth";

public string StatusNotDisturb => "Не беспокоить";
```

```
        public string Battery => "Батарея";

        public string BatteryIcon => "Иконка";

        public string BatteryPercentage => "Проценты";

        public List<string> WeekNames => _weekNames;

        public List<string> MonthNames => _monthNames;

    }
}
using XMIWatch.Resources.Localization.Interfaces;

namespace XMIWatch.Resources.Localization.ru_RU
{
    public class LocalizationRu : ILocalization
    {
        private LocalizationRu() { }
        private static LocalizationRu _localization;
        public static LocalizationRu Instance
            => _localization ?? (_localization = new LocalizationRu());

        public IUIWatchElements WatchElements => UIWatchElementsRu.Instance;
        public IUIInterface Interface => UIInterfaceRu.Instance;
    }
}
using XMIWatch.Resources.Localization.Interfaces;
using XMIWatch.Resources.Localization.en_EN;
using XMIWatch.Resources.Localization.ru_RU;
using XMIWatch.Services;
using XMIWatch.Watch.Services;

namespace XMIWatch.Resources.Localization
{
    public enum AppLanguage
    {
        RU,
        EN
    }

    public static class LocalizationProvider
    {
        static LocalizationProvider()
        {
            Localization = new Observe<ILocalization>();

            void change() => SetLang(AppSettings.Instance.Localization);

            AppSettings.Instance.PropertyChanged += (s, e) => change();
            change();

            //SetLang(AppLanguage.EN);
        }


        private static void SetLang(AppLanguage lang)
        {
            switch (lang)
            {
                case AppLanguage.EN:
                    Localization.Set(LocalizationEn.Instance);
                    break;
```

```
            case AppLanguage.RU:
                Localization.Set(LocalizationRu.Instance);
                break;
        }
    }


    public static Observe<ILocalization> Localization { get; }
  }
}
namespace XMIWatch.Resources.Localization.Interfaces
{
  public interface IUIInterface
  {
    public string Copy     { get; }
    public string Paste    { get; }
    public string Import   { get; }
    public string Donate   { get; }
    public string Build    { get; }
    public string Building { get; }

    // Notify
    public (string Message, string Title) ClosingProject(string Name);
    public (string Message, string Title) ClosingApp { get; }
    public string ProjectsLimit { get; }
    public string SavingFailed { get; }
    public string OpeningFailed { get; }
    public string MiBand5Support{ get; }
    public string ScreenNotSupport { get; }
    public string FileNotSupport { get; }
    public string MiToolsNotFound { get; }
  }
}
namespace XMIWatch.Resources.Localization.Interfaces
{
  public interface ILocalization
  {
    public IUIWatchElements WatchElements { get; }
    public IUIInterface Interface { get; }
  }
}
using System.Collections.Generic;

namespace XMIWatch.Resources.Localization.Interfaces
{
  public interface IUIWatchElements
  {
    // Images types
    public string MainImages     { get; }
    public string DelimiterImage { get; }
    public string SwitcherImages  { get; }
    public string DecimalImage   { get; }
    public string NoDataImage    { get; }
    public string SuffixImage    { get; }


    // Model
    public string      Background       { get; }
    public string      Hours            { get; }
    public string      Minutes          { get; }
    public string      LeftDigit        { get; }
    public string      RightDigit       { get; }
    public string      DateWeek         { get; }
    public string      DateDay          { get; }
    public string      DateMonth        { get; }
```

```
        public string       DateLine         { get; }
        public string       ActivitySteps    { get; }
        public string       ActivityPulse    { get; }
        public string       ActivityCalories { get; }
        public string       ActivityDistance { get; }
        public string       StatusLock       { get; }
        public string       StatusBluetooth  { get; }
        public string       StatusNotDisturb { get; }
        public string       Battery          { get; }
        public string       BatteryIcon      { get; }
        public string       BatteryPercentage { get; }
        public List<string> WeekNames        { get; }
        public List<string> MonthNames       { get; }
    }
}
// System Libs
using System.ComponentModel;
using System.Reflection;

namespace XMIWatch.Extensions
{
    public static class AttributeExt
    {
        public static string Description<T>(this T source)
        {
            string field = source.ToString();
            FieldInfo info = source.GetType().GetField(field);

            DescriptionAttribute[] attributes = (DescriptionAttribute[])info
                .GetCustomAttributes(typeof(DescriptionAttribute), false);

            if (attributes != null && attributes.Length > 0)
                return attributes[0].Description;
            else return field;
        }
    }
}
// System Libs
using System.IO;

// Microsoft Libs
using Microsoft.VisualBasic.FileIO;

namespace XMIWatch.Extensions
{
    public static class DirectoryExt
    {
        public static void Copy(string from, string to)
        {
            if (Directory.Exists(from))
            {
                if (!Directory.Exists(to))
                    Directory.CreateDirectory(to);

                FileSystem.CopyDirectory(from, to, UIOption.AllDialogs);
            }
        }
    }
}
// System Libs
using System.Drawing;
using System.Drawing.Drawing2D;

// Solution Libs
using XMIWatch.Services;
```

```
using XMIWatch.Painting.Extensions;

// Types
using Point = System.Drawing.Point;

namespace XMIWatch.Painting
{
   public class DrawText
   {
      private int _width;
      private int _height;

      private string _text;
      private Font   _textFont;
      private Color  _textColor;
      private Pen    _textBorder;

      private Bitmap _background;
      private Point  _backgroundPosition;


      public DrawText(string text, Font font)
      {
         ValueService.SafetySet(ref _textFont, ref font);
         ValueService.SafetySet(ref _text, ref text);

         UpdateSize();
      }


      /// <summary>
      /// Set new text.
      /// </summary>
      public DrawText SetText(string text)
      {
         ValueService.SafetySet(ref _text, ref text);
         UpdateSize();
         return this;
      }

      /// <summary>
      /// Set new text font.
      /// </summary>
      public DrawText SetFont(Font font)
      {
         ValueService.SafetySet(ref _textFont, ref font);
         UpdateSize();
         return this;

      }


      /// <summary>
      /// Set text color.
      /// </summary>
      public DrawText SetColor(Color color)
      {
         _textColor = color;
         return this;
      }


      /// <summary>
      /// Set text background.
      /// </summary>
```

```
public DrawText SetBackground(Bitmap image)
{
   ValueService.SecureSet(ref _background, image.Copy());
   return this;
}


/// <summary>
/// Set text background and position.
/// </summary>
public DrawText SetBackground(Bitmap image, int x, int y)
{
   SetBackground(image);
   return SetPosition(x, y);
}


/// <summary>
/// Set text position on background.
/// </summary>
public DrawText SetPosition(int x, int y)
{
   _backgroundPosition = new Point(x, y);
   return this;
}


/// <summary>
/// Set text border.
/// </summary>
public DrawText SetBorder(Pen border)
{
   ValueService.SafetySet(ref _textBorder, ref border);
   return this;
}


/// <summary>
/// Text pixel width.
/// </summary>
public int Width => _width;


/// <summary>
/// Text pixel height.
/// </summary>
public int Height => _height;


/// <summary>
/// Returns image with setted parameters.
/// </summary>
public Bitmap Build()
{
   Bitmap output = new Bitmap(_width, _height);

   using (var mask = new GraphicsPath().AddString(_text, _textFont))
   {
      using (var canvas = Graphics.FromImage(output))
      {
         // Rendering
         canvas.SmoothingMode = SmoothingMode.AntiAlias;

         // Draw background
         if (_background != null)
```

```
                {
                    int border = (int)(_textBorder?.Width ?? 0f)
                            + (_textBorder == null && _textColor.IsEmpty ? 0 : 3);

                    canvas.DrawImage(_background
                            .Crop(_width, _height, _backgroundPosition.X, _backgroundPosition.Y)
                            .CropMask(mask, border), 0, 0);
                }

                // Draw border
                if (_textBorder != null)
                    canvas.DrawPath(_textBorder, mask);

                // Draw text
                if (!_textColor.IsEmpty)
                    canvas.FillPath(new SolidBrush(_textColor), mask);

                // Write image
                canvas.Flush();
            }
        }

        return output;
    }


    /// <summary>
    /// Update image size for build.
    /// </summary>
    private void UpdateSize()
    {
        using (var canvas = Graphics.FromImage(new Bitmap(1, 1)))
        {
            var size = canvas.MeasureString(_text, _textFont);
            _width  = (int)size.Width;
            _height = (int)size.Height;
        }
    }
  }
}// System Libs
using System.Drawing;
using System.Drawing.Drawing2D;


namespace XMIWatch.Painting.Extensions
{
  public static class GraphicsPathExt
  {
    /// <summary>
    /// Adds text string to this path.
    /// </summary>
    public static GraphicsPath AddString(this GraphicsPath path, string text, Font font)
    {
        path.AddString(text, font.FontFamily, (int)font.Style, font.Size, new Point(0, 0), null);
        return path;
    }
  }
}
// System Libs
using System;
using System.Drawing;
using System.Windows.Interop;
using System.Windows.Media.Imaging;
using System.Runtime.InteropServices;
using System.Windows;
```

```
using System.Drawing.Drawing2D;
using System.IO;
using System.Drawing.Imaging;

namespace XMIWatch.Painting.Extensions
{
   public static class BitmapExt
   {
      static BitmapExt()
      {
         EmptyImage.SetResolution(96, 96);
      }


      /// <summary>
      /// Returns bitmap 1x1 with alpha pixel.
      /// </summary>
      public readonly static Bitmap EmptyImage
         = new Bitmap(1, 1);


      /// <summary>
      /// Clear hBitmap cache from unmanaged memory.
      /// </summary>
      [DllImport("gdi32.dll")]
      public static extern bool DeleteObject(IntPtr hObject);


      /// <summary>
      /// Convert to Bitmap to InteropBitmap.
      /// </summary>
      public static BitmapSource ToBitmapSource(this Bitmap bitmap)
      {
         IntPtr hBitmap = bitmap.GetHbitmap();
         BitmapSource converted;

         try
         {
            converted = (BitmapSource)Imaging.CreateBitmapSourceFromHBitmap(
                  hBitmap,
                  IntPtr.Zero,
                  Int32Rect.Empty,
                  BitmapSizeOptions.FromEmptyOptions());
         }
         finally
         {
            DeleteObject(hBitmap);
         }

         return converted;
      }


      /// <summary>
      /// Converts a <see cref="Image"/> into a WPF <see cref="BitmapSource"/>.
      /// </summary>
      public static BitmapSource ToBitmapSource(this Image source)
      {
         Bitmap bitmap = new Bitmap(source);

         var bitSrc = bitmap.ToBitmapSource();

         bitmap.Dispose();
         return bitSrc;
      }
```

```
/// <summary>
/// Returns image from file stream or null if not exists.
/// </summary>
/// <param name="path">Path to image file.</param>
public static Bitmap OpenToRam(string path)
{
   Bitmap image = null;

   if (path != null && File.Exists(path))
      using (var stream = new FileStream(path, FileMode.Open))
         image = new Bitmap(stream).Copy();

   return image;
}


/// <summary>
/// Returns image with mask filter.
/// </summary>
public static Bitmap CropMask(this Bitmap bitmap, GraphicsPath mask, int border = 3, bool isFilled = false)
{
   var masked = bitmap.CreateMask(mask, isFilled, border);

   for (int x = 0; x < masked.Width; ++x)
      for (int y = 0; y < masked.Height; ++y)
         if (masked.GetPixel(x, y).A != 0)
            masked.SetPixel(x, y, bitmap.GetPixel(x, y));


   return masked;
}


/// <summary>
/// Returns cropped bitmap.
/// </summary>
public static Bitmap Crop(this Bitmap bitmap, int width, int height, int x = 0, int y = 0)
{
   static void Normalize(int max, ref int axis, ref int size)
   {
      if (axis < 0) axis = 0;
      if (size > max) size %= max;

      int offset = size + axis;
      if (offset > max) axis = offset % max;
   }

   Normalize(bitmap.Width, ref x, ref width);
   Normalize(bitmap.Height, ref y, ref height);

   return bitmap.Clone(new Rectangle(x, y, width, height), bitmap.PixelFormat);
}


/// <summary>
/// Returns new full copy from original image.
/// </summary>
public static Bitmap Copy(this Bitmap bitmap)
   => bitmap.Clone(new Rectangle(0, 0, bitmap.Width, bitmap.Height), bitmap.PixelFormat);


/// <summary>
/// Returns new filled bitmap by black mask.
```

```
        /// </summary>
        public static Bitmap CreateMask(this Bitmap bitmap, GraphicsPath mask, bool isFilled = true, int border = 0)
        {
            var masked = new Bitmap(bitmap.Width, bitmap.Height);
            using (var canvas = Graphics.FromImage(masked))
            {
                canvas.SmoothingMode = SmoothingMode.AntiAlias;

                if (isFilled)
                    canvas.FillPath(Brushes.Black, mask);
                if (border > 0)
                    canvas.DrawPath(new Pen(Brushes.Black, border), mask);

                canvas.Flush();
            }

            return masked;
        }
    }
}
using System.Windows;
using System.Windows.Input;

namespace XMIWatch.AppContext
{
    public static class PreviewerContext
    {
        public static IInputElement LastFocus { get; set; }
        public static void Focus()
        {
            if (LastFocus != null)
                Keyboard.Focus(LastFocus);
        }
    }
}
using System;
using System.Collections.Generic;

// Solution Libs
using XMIWatch.Services;
using XMIWatch.Watch.Base.ProjectModel.Classes;
using XMIWatch.Watch.Base.ProjectModel.Interfaces;

namespace XMIWatch.AppContext
{
    public static class ClipboardContext
    {
        /// <summary>
        /// Contains copied images for IWatchItem.
        /// </summary>
        public readonly static Observe<List<AutoBitmapSource>> ItemImages
            = new Observe<List<AutoBitmapSource>>();


        /// <summary>
        /// Contains copied IWatchItem.
        /// </summary>
        public static Observe<IWatchItem> Item
            = new Observe<IWatchItem>();
    }
}
// Solution Libs
using XMIWatch.Services;
using XMIWatch.Watch.Series.MiBand4.Project;
using XMIWatch.Watch.Base.InputModel.Interfaces;
```

```
using XMIWatch.Watch.Base.InputModel.Interfaces.Parts;
using XMIWatch.Watch.Series.Services.InputConverting;
using XMIWatch.Watch.Services.Converting;
using XMIWatch.Watch.Base.InputModel.Interfaces.Extensions;
using XMIWatch.Watch.Series.MiBand4.Project.Extentions;

namespace XMIWatch.Watch.Series.MiBand4.Input
{
   public partial class InputMiBand4Model : IModelConvertible<InputMiBand4Model, MiBand4Model>
   {
      public MiBand4Model ToModel()
      {
         var model = new MiBand4Model();

         var indexChecker   = GetIndexChecker();
         var indexConverter = new ImageIndexConverter(indexChecker);
         var itemConverter  = new InputConverter(indexConverter);


         // Background
         itemConverter.ToMovableItem(Background?.Image, model.Background);


         // Time
         itemConverter.ToMovableItems(Time?.Hours?.Tens, model.TimeHourL);
         itemConverter.ToMovableItems(Time?.Hours?.Ones, model.TimeHourR);
         itemConverter.ToMovableItems(Time?.Minutes?.Tens, model.TimeMinuteL);
         itemConverter.ToMovableItems(Time?.Minutes?.Ones, model.TimeMinuteR);


         // Date
         {
            var date     = Date?.MonthAndDay;
            var inline   = date?.OneLine;
            var separate = date?.Separate;

            if (inline != null)
            {

               itemConverter.ToNumber(inline?.Number, model.DateDay);
               itemConverter.ToNumber(inline?.Number, model.DateMonth);
               indexConverter.ToDelimiter(inline, model.DateDay.Value);

               model.DateSettings.Value.TwoDigitsDay.Set(true);
               model.DateSettings.Value.TwoDigitsMonth.Set(true);
            }
            else
            {
               itemConverter.ToNumber(separate?.Day, model.DateDay);
               itemConverter.ToNumber(separate?.Month, model.DateMonth);

               model.DateSettings.Value.IsSeparated.Set(true);
               model.DateSettings.Value.TwoDigitsDay.Set(date?.TwoDigitsDay ?? false);
               model.DateSettings.Value.TwoDigitsMonth.Set(date?.TwoDigitsMonth ?? false);
            }
         }

         itemConverter.ToMovableItems(Date?.WeekDay, model?.DateWeek);


         // Activity
         {
            var from = Activity?.Distance;
            var to = model.ActivityDistance;
```

```
            if (from != null)
            {
                itemConverter.ToNumber(from.Number, to);
                indexConverter.ToSuffix(from, to?.Value);
                indexConverter.ToDecimal(from, to?.Value);
            }
        }
        {
            var from = Activity?.Calories;
            var to = model.ActivityCalories;

            if (from != null)
            {
                itemConverter.ToNumber(from.Number, to);
                indexConverter.ToSuffix(from, to?.Value);
            }
        }
        {
            var from = Activity?.Pulse;
            var to = model.ActivityPulse;

            if (from != null)
            {
                itemConverter.ToNumber(from.Number, to);
                indexConverter.ToNoData(from, to?.Value);
            }
        }

        itemConverter.ToNumber(Activity?.Steps?.Number, model.ActivitySteps);


        // Status
        itemConverter.ToSwitcher(Status?.Bluetooth, model.StatusBluetooth);
        itemConverter.ToSwitcher(Status?.Lock, model.StatusLock);
        itemConverter.ToSwitcher(Status?.DoNotDisturb, model.StatusNotDisturb);
        itemConverter.ToNumber(Status?.Battery?.Text, model.BatteryPercentage);
        itemConverter.ToMovableItems(Status?.Battery?.Icon, model.BatteryIcon);


        ImageLoader.Load(FolderPath, indexChecker.Build(), model.ImagesPool);

        return model.ReLink();
    }


    /// <summary>
    /// Create input model instance from <see cref="MiBand4Model"/>
    /// </summary>
    public InputMiBand4Model(MiBand4Model model)
    {
        // Background
        {
            var background = OutputConverter.CreateImageItem(model.Background.Value);
            if (background != null)
            {
                Background = new Background
                {
                    Image = background
                };
            }
        }


        // Time
        {
```

```
    var hours = new TimeBlock
    {
        Ones = OutputConverter.CreateImageItems(model.TimeHourR.Value),
        Tens = OutputConverter.CreateImageItems(model.TimeHourL.Value)
    };
    if ((hours?.Ones ?? hours?.Tens) == null) hours = null;


    var minutes = new TimeBlock
    {
        Ones = OutputConverter.CreateImageItems(model.TimeMinuteR.Value),
        Tens = OutputConverter.CreateImageItems(model.TimeMinuteL.Value)
    };
    if ((minutes?.Ones ?? minutes?.Tens) == null) minutes = null;


    if ((hours ?? minutes) != null)
    {
        Time = new Time
        {
            Hours = hours,
            Minutes = minutes
        };
    }
}


// Date
{
    var day = OutputConverter.CreateNumber(model.DateDay.Value);
    var week = OutputConverter.CreateImageItems(model.DateWeek.Value);

    var date = new DateMonthDay();
    var isInit = false;

    if (model.DateSettings.Value.IsSeparated.Value)
    {
        var month = OutputConverter.CreateNumber(model.DateMonth.Value);
        if ((day ?? month) != null)
        {
            date.Separate = new DateSeparate
            {
                Day = day,
                Month = month
            };

            isInit = true;
        }
    }
    else
    {
        if (day != null)
        {
            date.OneLine = new DateLine
            {
                Number = day
            };
            date.OneLine.CopyDelimiter(model.DateDay.Value);

            isInit = true;
        }
    }


    if (isInit)
```

```
    {
        date.TwoDigitsDay = model.DateSettings.Value.TwoDigitsDay.Value;
        date.TwoDigitsMonth = model.DateSettings.Value.TwoDigitsMonth.Value;
    }
    else date = null;


    if (date != null || week != null)
    {
        Date = new Date
        {
            MonthAndDay = date,
            WeekDay = week
        };
    }
}


// Activity
{
    var activity = new Activity();
    var isInit = false;

    // Calories
    {
        var item = new ActivityCalories();
        var from = model.ActivityCalories.Value;

        item.Number = OutputConverter.CreateNumber(from);

        if (item?.Number != null)
        {
            item.CopySuffix(from);

            activity.Calories = item;
            isInit = true;
        }
    }

    // Distance
    {
        var item = new ActivityDistance();
        var from = model.ActivityDistance.Value;

        item.Number = OutputConverter.CreateNumber(from);


        if (item?.Number != null)
        {
            item
                .CopySuffix(from)
                .CopyDecimal(from);

            activity.Distance = item;
            isInit = true;
        }
    }

    // Steps
    {
        var item = new ActivitySteps();
        var from = model.ActivitySteps.Value;

        item.Number = OutputConverter.CreateNumber(from);
```

```
            if (item?.Number != null)
            {
                activity.Steps = item;
                isInit = true;
            }
        }

        // Pulse
        {
            var item = new ActivityPulse();
            var from = model.ActivityPulse.Value;

            item.Number = OutputConverter.CreateNumber(from);

            if (item?.Number != null)
            {
                item.CopyNoData(from);

                activity.Pulse = item;
                isInit = true;
            }
        }

        if (isInit) Activity = activity;
    }


    // Status
    {
        var status = new Status();
        var isInit = false;


        status.Bluetooth = OutputConverter.CreateSwitcher(model.StatusBluetooth.Value);
        status.Lock = OutputConverter.CreateSwitcher(model.StatusLock.Value);
        status.DoNotDisturb = OutputConverter.CreateSwitcher(model.StatusNotDisturb.Value);

        if ((status?.Bluetooth ?? status?.Lock ?? status?.DoNotDisturb) != null) isInit = true;

        // Battery
        {
            var item = new Battery
            {
                Text = OutputConverter.CreateNumber(model.BatteryPercentage.Value),
                Icon = OutputConverter.CreateImageItems(model.BatteryIcon.Value)
            };

            if (item?.Text != null || item?.Icon != null)
            {
                status.Battery = item;
                isInit = true;
            }
        }

        if (isInit) Status = status;
    }
}

private ImageIndexChecker GetIndexChecker()
{
    var counter = new ImageIndexChecker(FolderPath);

    // Background
    counter.Add(Background?.Image);
```

```
            // Time
            counter.Add(Time?.Hours?.Tens);
            counter.Add(Time?.Hours?.Ones);
            counter.Add(Time?.Minutes?.Tens);
            counter.Add(Time?.Minutes?.Ones);


            // Date
            {
               var date = Date?.MonthAndDay;
               var inline = date?.OneLine;
               var separate = date?.Separate;

               if (inline != null)
               {
                  counter.Add(inline);
                  counter.Add(inline?.Number);
               }
               else if (separate != null)
               {
                  counter.Add(separate?.Day);
                  counter.Add(separate?.Month);
               }
            }
            counter.Add(Date?.WeekDay);


            // Activity
            counter.Add(Activity?.Steps?.Number);
            {
               var item = Activity?.Calories;

               counter.Add(item);
               counter.Add(item?.Number);
            }
            {
               var item = Activity?.Distance;

               counter.Add(item?.Number);
               counter.Add((IInputSourceIndexDecimal)item);
               counter.Add((IInputSourceIndexSuffix)item);
            }
            {
               var item = Activity?.Pulse;

               counter.Add(item);
               counter.Add(item?.Number);
            }


            // Status
            counter.Add(Status?.Bluetooth);
            counter.Add(Status?.Lock);
            counter.Add(Status?.DoNotDisturb);
            counter.Add(Status?.Battery?.Icon);
            counter.Add(Status?.Battery?.Text);


            return counter;
         }
      }
   }
}
// Solution Libs
using XMIWatch.Watch.Series.MiBand4.Project;
```

```
using XMIWatch.Watch.Base.InputModel.Classes.Containers;
using XMIWatch.Watch.Base.InputModel.Interfaces;
using XMIWatch.Watch.Base.InputModel.Interfaces.Parts;

namespace XMIWatch.Watch.Series.MiBand4.Input
{
   public partial class InputMiBand4Model : IModelConvertible<InputMiBand4Model, MiBand4Model>
   {
      public InputMiBand4Model() { }

      public string FolderPath { get; set; }
      public Background Background { get; set; }
      public Activity Activity { get; set; }
      public Status Status { get; set; }
      public Time Time { get; set; }
      public Date Date { get; set; }
   }

   #region Background Model

   public class Background
   {
      public InputImageItem Image { get; set; }
   }

   #endregion

   #region Time Model

   public class TimeBlock
   {
      public InputImageItems Tens { get; set; }
      public InputImageItems Ones { get; set; }
   }


   public class Time
   {
      public TimeBlock Hours { get; set; }
      public TimeBlock Minutes { get; set; }
   }

   #endregion

   #region Date Model

   public class DateSeparate
   {
      public InputNumber Day { get; set; }
      public InputNumber Month { get; set; }
   }

   public class DateLine : IInputSourceIndexDelimiter
   {
      public InputNumber Number { get; set; }
      public int? DelimiterImageIndex { get; set; }
   }

   public class DateMonthDay
   {
      public DateSeparate Separate { get; set; }
      public DateLine OneLine { get; set; }
      public bool TwoDigitsMonth { get; set; } = true;
      public bool TwoDigitsDay { get; set; } = true;
   }
```

```
public class Date
{
   public DateMonthDay MonthAndDay { get; set; }
   public InputImageItems WeekDay { get; set; }
}

#endregion

#region Activity Models

public class ActivitySteps
{
   public InputNumber Number { get; set; }
}

public class ActivityPulse : ActivitySteps, IInputSourceIndexNoData
{
   public int? NoDataImageIndex { get; set; }
}

public class ActivityCalories : ActivitySteps, IInputSourceIndexSuffix
{
   public int? SuffixImageIndex { get; set; }
}

public class ActivityDistance : ActivityCalories, IInputSourceIndexDecimal
{
   public int? DecimalPointImageIndex { get; set; }
}

public class Activity
{
   public ActivitySteps Steps { get; set; }
   public ActivityPulse Pulse { get; set; }
   public ActivityDistance Distance { get; set; }
   public ActivityCalories Calories { get; set; }
}

#endregion

#region Status Model

public class Coordinates : IInputContainerPartAxis
{
   public int X { get; set; }
   public int Y { get; set; }
}

public class InputSwitcher : IInputSourceIndexSwitcher
{
   public Coordinates Coordinates { get; set; }
   public int? ImageIndexOn { get; set; }
   public int? ImageIndexOff { get; set; }
}

public class Battery
{
   public InputNumber Text { get; set; }
   public InputImageItems Icon { get; set; }
}

public class Status
{
   public InputSwitcher DoNotDisturb { get; set; }
```

```
        public InputSwitcher Lock { get; set; }
        public InputSwitcher Bluetooth { get; set; }
        public Battery Battery { get; set; }
    }

    #endregion
}// System Libs
using System;

// Solution Libs
using XMIWatch.Services;
using XMIWatch.Watch.Base.ProjectModel.Services;
using XMIWatch.Watch.Base.ProjectModel.Services.Containers;
using XMIWatch.Watch.Base.ProjectModel.Interfaces;

namespace XMIWatch.Watch.Series.MiBand4.Project
{
    public class MiBand4Model : WatchBase, ICloneable
    {
        /// <summary>
        /// Init all items to default.
        /// </summary>
        public MiBand4Model() : base(new WatchConfig(WatchSeries.MiBand4))
        {
            // Background
            Background = new Observe<MovableItem>(new MovableItem(WatchItemType.Background));


            // Time
            TimeMinuteL = new Observe<MovableItems>(new MovableItems(WatchItemType.MinuteL));
            TimeMinuteR = new Observe<MovableItems>(new MovableItems(WatchItemType.MinuteR));
            TimeHourL   = new Observe<MovableItems>(new MovableItems(WatchItemType.HourL));
            TimeHourR   = new Observe<MovableItems>(new MovableItems(WatchItemType.HourR));


            // Date
            DateWeek     = new Observe<MovableItems>(new MovableItems(WatchItemType.Week));
            DateDay      = new Observe<NumberDelimiter>(new NumberDelimiter(WatchItemType.Day));
            DateMonth    = new Observe<Number>(new Number(WatchItemType.Month));
            DateSettings = new Observe<DateSettings>().Init();


            // Activity
            ActivitySteps    = new Observe<Number>(new Number(WatchItemType.Steps));
            ActivityPulse    = new Observe<NumberNoData>(new NumberNoData(WatchItemType.Pulse));
            ActivityCalories = new Observe<NumberSuffix>(new NumberSuffix(WatchItemType.Calories));
            ActivityDistance = new Observe<NumberSuffixDecimal>(new
NumberSuffixDecimal(WatchItemType.Distance));


            // Status
            StatusLock       = new Observe<Switcher>(new Switcher(WatchItemType.Lock));
            StatusBluetooth  = new Observe<Switcher>(new Switcher(WatchItemType.Bluetooth));
            StatusNotDisturb = new Observe<Switcher>(new Switcher(WatchItemType.NotDisturb));


            // Battery
            BatteryIcon      = new Observe<MovableItems>(new MovableItems(WatchItemType.BatteryIcon));
            BatteryPercentage = new Observe<Number>(new Number(WatchItemType.BatteryPercentage));
        }


        /// <summary>
        /// Init copy of <see cref="MiBand4Model"/>.
        /// </summary>
```

```
    public MiBand4Model(MiBand4Model model) : base(model)
    {
        // Background
        Background = Background.Clone();


        // Time
        TimeMinuteL = TimeMinuteL.Clone();
        TimeMinuteR = TimeMinuteR.Clone();
        TimeHourL   = TimeHourL.Clone();
        TimeHourR   = TimeHourR.Clone();


        // Date
        DateWeek     = DateWeek.Clone();
        DateDay      = DateDay.Clone();
        DateMonth    = DateMonth.Clone();
        DateSettings = DateSettings.Clone();


        // Activity
        ActivitySteps    = ActivitySteps.Clone();
        ActivityPulse    = ActivityPulse.Clone();
        ActivityCalories = ActivityCalories.Clone();
        ActivityDistance = ActivityDistance.Clone();


        // Status
        StatusLock       = StatusLock.Clone();
        StatusBluetooth  = StatusBluetooth.Clone();
        StatusNotDisturb = StatusNotDisturb.Clone();


        // Battery
        BatteryIcon       = BatteryIcon.Clone();
        BatteryPercentage = BatteryPercentage.Clone();
    }


    public Observe<MovableItem>         Background        { get; private set; }
    public Observe<MovableItems>        TimeMinuteL       { get; private set; }
    public Observe<MovableItems>        TimeMinuteR       { get; private set; }
    public Observe<MovableItems>        TimeHourL         { get; private set; }
    public Observe<MovableItems>        TimeHourR         { get; private set; }
    public Observe<MovableItems>        DateWeek          { get; private set; }
    public Observe<NumberDelimiter>     DateDay           { get; private set; }
    public Observe<Number>              DateMonth         { get; private set; }
    public Observe<DateSettings>        DateSettings      { get; private set; }
    public Observe<Number>              ActivitySteps     { get; private set; }
    public Observe<NumberNoData>        ActivityPulse     { get; private set; }
    public Observe<NumberSuffix>        ActivityCalories  { get; private set; }
    public Observe<NumberSuffixDecimal> ActivityDistance  { get; private set; }
    public Observe<Switcher>            StatusLock        { get; private set; }
    public Observe<Switcher>            StatusBluetooth   { get; private set; }
    public Observe<Switcher>            StatusNotDisturb  { get; private set; }
    public Observe<MovableItems>        BatteryIcon       { get; private set; }
    public Observe<Number>              BatteryPercentage { get; private set; }


    /// <summary>
    /// Returns full copy of <see cref="MiBand4Model"/>.
    /// </summary>
    public override object Clone()
        => new MiBand4Model(this);
}
```

```
    public class DateSettings : ICloneable
    {
        /// <summary>
        /// Init all items to default.
        /// </summary>
        public DateSettings()
        {
            IsSeparated    = new Observe<bool>();
            TwoDigitsDay   = new Observe<bool>();
            TwoDigitsMonth = new Observe<bool>();
        }


        /// <summary>
        /// Init copy of <see cref="DateSettings"/>.
        /// </summary>
        public DateSettings(DateSettings settings)
        {
            IsSeparated    = settings.IsSeparated.Copy();
            TwoDigitsDay   = settings.TwoDigitsDay.Copy();
            TwoDigitsMonth = settings.TwoDigitsMonth.Copy();
        }


        public Observe<bool> IsSeparated    { get; private set; }
        public Observe<bool> TwoDigitsMonth { get; private set; }
        public Observe<bool> TwoDigitsDay   { get; private set; }


        /// <summary>
        /// Returns full copy of <see cref="DateSettings"/>.
        /// </summary>
        public object Clone()
            => new DateSettings(this);
    }
}
// System Libs
using System.Collections.ObjectModel;
using System.Windows.Media.Imaging;

// Solution Libs
using XMIWatch.Services;
using XMIWatch.UIControls.Previewer.Classes;
using XMIWatch.Watch.Base.ProjectModel.Classes.Extentions;
using XMIWatch.Watch.Base.ProjectModel.Services;

namespace XMIWatch.Watch.Series.MiBand4.Project
{
    public class MiBand4Preview
    {
        private readonly PreviewManager _preview;

        /// <summary>
        /// Binding all items for <paramref name="model"/>.
        /// </summary>
        public MiBand4Preview(MiBand4Model model)
        {
            _preview = new PreviewManager(model);

            // Load Background
            Container.Add(new PreviewItem(model.Background.Value,
                new Observe<BitmapSource>(model.GetAutoSource(model.Background.Value).Source)));
```

```
    // Binding Hours
    Hours = new Observe<int>(12);
    _preview.BindingTime(Hours, model.TimeHourL.Value, model.TimeHourR.Value, Limit.Hours);


    // Binding Minutes
    Minutes = new Observe<int>(12);
    _preview.BindingTime(Minutes, model.TimeMinuteL.Value, model.TimeMinuteR.Value, Limit.Minutes);


    // Binding Week
    Week = new Observe<int>(2);
    _preview.BindingItems(Week, model.DateWeek.Value, Limit.Week, false);


    // Binding Day and Month
    Month      = new Observe<int>(4);
    Day        = new Observe<int>(23);
    DateInline = new Observe<bool>(false);

    _preview.BindingDate(DateInline, Day, Month,
       model.DateDay.Value, model.DateMonth.Value);


    // Binding Distance
    Distance = new Observe<int>(10);
    _preview.BindingNumber(Distance, model.ActivityDistance.Value, Limit.Activity);


    // Binding Calories
    Calories = new Observe<int>(1000);
    _preview.BindingNumber(Calories, model.ActivityCalories.Value, Limit.Activity);


    // Binding Pulse
    Pulse = new Observe<int>(70);
    _preview.BindingNumber(Pulse, model.ActivityPulse.Value, Limit.Pulse);


    // Binding Steps
    Steps = new Observe<int>(10500);
    _preview.BindingNumber(Steps, model.ActivitySteps.Value, Limit.Activity);


    // Binding DoNotDisturb
    DoNotDisturb = new Observe<bool>(true);
    _preview.BindingSwitcher(DoNotDisturb, model.StatusNotDisturb.Value);


    // Binding Bluetooth
    Bluetooth = new Observe<bool>(true);
    _preview.BindingSwitcher(Bluetooth, model.StatusBluetooth.Value);


    // Binding Lock
    Lock = new Observe<bool>(true);
    _preview.BindingSwitcher(Lock, model.StatusLock.Value);


    // Binding Battery
    Battery = new Observe<int>(100);
    _preview.BindingBattery(Battery, model.BatteryIcon.Value, model.BatteryPercentage.Value);
}
```

```
        public ObservableCollection<PreviewItem> Container => _preview.Items;


        public Observe<int>  Hours       { get; private set; }
        public Observe<int>  Minutes     { get; private set; }
        public Observe<bool> DateInline  { get; private set; }
        public Observe<int>  Week        { get; private set; }
        public Observe<int>  Month       { get; private set; }
        public Observe<int>  Day         { get; private set; }
        public Observe<int>  Calories    { get; private set; }
        public Observe<int>  Distance    { get; private set; }
        public Observe<int>  Pulse       { get; private set; }
        public Observe<int>  Steps       { get; private set; }
        public Observe<bool> DoNotDisturb { get; private set; }
        public Observe<bool> Bluetooth   { get; private set; }
        public Observe<bool> Lock        { get; private set; }
        public Observe<int>  Battery     { get; private set; }
    }
}
// Solution Libs
using XMIWatch.Watch.Services;

namespace XMIWatch.Watch.Series.MiBand4.Project.Extentions
{
    public static class MiBand4ModelExt
    {
        /// <summary>
        /// Relink images for same items.
        /// </summary>
        public static T ReLink<T>(this T model)
            where T: MiBand4Model
        {
            var relinker = new ImageReLinker(model);

            relinker.ReLink(model.DateWeek.Value);

            return model;
        }
    }
}
// System Libs
using System.Windows.Controls;

namespace XMIWatch.Watch.Series.MiBand4.Project
{
    public partial class MiBand4Properties : Grid
    {
        public MiBand4Properties(MiBand4Model model, MiBand4Preview preview)
        {
            Model   = model;
            Preview = preview;

            InitializeComponent();
        }

        public MiBand4Model       Model   { get; private set; }
        public MiBand4Preview Preview { get; private set; }
    }
}
// Custom Libs

// Solution Libs
using XMIWatch.Services;
using XMIWatch.Watch.Base.InputModel.Classes.Containers;
using XMIWatch.Watch.Base.InputModel.Interfaces;
```

```
using XMIWatch.Watch.Base.InputModel.Interfaces.Extensions;
using XMIWatch.Watch.Base.InputModel.Interfaces.Parts;
using XMIWatch.Watch.Series.MiBand4.Input;
using XMIWatch.Watch.Series.MiBand5.Project;
using XMIWatch.Watch.Series.MiBand5.Project.Extentions;
using XMIWatch.Watch.Series.Services.InputConverting;
using XMIWatch.Watch.Services.Converting;

namespace XMIWatch.Watch.Series.MiBand5.Input
{
   public partial class InputMiBand5Model : IModelConvertible<InputMiBand5Model, MiBand5Model>
   {
      public MiBand5Model ToModel()
      {
         var model = new MiBand5Model();

         var indexChecker = GetIndexChecker();
         var indexConverter = new ImageIndexConverter(indexChecker);
         var itemConverter = new InputConverter(indexConverter);


         // Background
         itemConverter.ToMovableItem(Background?.Image, model.Background);
         itemConverter.ToMovableItem(Background?.Preview1, model.Preview);


         // Time
         itemConverter.ToMovableItems(Time?.Hours?.Tens, model.TimeHourL);
         itemConverter.ToMovableItems(Time?.Hours?.Ones, model.TimeHourR);
         itemConverter.ToMovableItems(Time?.Minutes?.Tens, model.TimeMinuteL);
         itemConverter.ToMovableItems(Time?.Minutes?.Ones, model.TimeMinuteR);


         // Date
         {
            var date = Date?.MonthAndDayAndYear;
            var inline = date?.OneLine;
            var separate = date?.Separate;

            if (inline != null)
            {

               itemConverter.ToNumberXY(inline?.Number, model.DateDay);
               itemConverter.ToNumberXY(inline?.Number, model.DateMonth);
               indexConverter.ToDelimiter(inline, model.DateDay.Value);

               model.DateSettings.Value.TwoDigitsDay.Set(true);
               model.DateSettings.Value.TwoDigitsMonth.Set(true);
            }
            else
            {
               itemConverter.ToNumberXY(separate?.Day, model.DateDay);
               itemConverter.ToNumberXY(separate?.Month, model.DateMonth);

               model.DateSettings.Value.IsSeparated.Set(true);
               model.DateSettings.Value.TwoDigitsDay.Set(date?.TwoDigitsDay ?? false);
               model.DateSettings.Value.TwoDigitsMonth.Set(date?.TwoDigitsMonth ?? false);
            }
         }

         itemConverter.ToMovableItems(Date?.ENWeekDays, model?.DateWeek);


         // Activity
         {
```

```
            var from = Activity?.Distance;
            var to = model.ActivityDistance;

            if (from != null)
            {
                itemConverter.ToNumberXY(from.Number, to);
                indexConverter.ToSuffix(from, to?.Value);
                indexConverter.ToDecimal(from, to?.Value);
            }
        }
        {
            var from = Activity?.Calories;
            var to = model.ActivityCalories;

            if (from != null)
            {
                itemConverter.ToNumberXY(from.Number, to);
                indexConverter.ToSuffix(from, to?.Value);
            }
        }
        {
            var from = Activity?.Pulse;
            var to = model.ActivityPulse;

            if (from != null)
            {
                itemConverter.ToNumberXY(from.Number, to);
                indexConverter.ToNoData(from, to?.Value);
            }
        }

        itemConverter.ToNumberXY(Activity?.Steps?.Number, model.ActivitySteps);


        // Status
        itemConverter.ToSwitcher(Status?.Bluetooth, model.StatusBluetooth);
        itemConverter.ToSwitcher(Status?.Lock, model.StatusLock);
        itemConverter.ToSwitcher(Status?.DoNotDisturb, model.StatusNotDisturb);
        itemConverter.ToNumberXY(Battery?.BatteryText?.Coordinates, model.BatteryPercentage);
        itemConverter.ToMovableItems(Battery?.BatteryIcon, model.BatteryIcon);


        ImageLoader.Load(FolderPath, indexChecker.Build(), model.ImagesPool);

        return model.ReLink();
    }


    /// <summary>
    /// Create input model instance from <see cref="MiBand4Model"/>
    /// </summary>
    public InputMiBand5Model(MiBand5Model model)
    {
        // Background
        {
            var background = OutputConverter.CreateImageItem(model.Background.Value);
            var preview = OutputConverter.CreateImageItem(model.Preview.Value);

            if (background != null)
            {
                Background = new Background
                {
                    Image = background,
                    Preview1 = preview,
                    Preview2 = preview,
```

```
            Preview3 = preview
        };
    }
}


// Time
{
    var hours = new TimeBlock
    {
        Ones = OutputConverter.CreateImageItems(model.TimeHourR.Value),
        Tens = OutputConverter.CreateImageItems(model.TimeHourL.Value)
    };
    if ((hours?.Ones ?? hours?.Tens) == null) hours = null;


    var minutes = new TimeBlock
    {
        Ones = OutputConverter.CreateImageItems(model.TimeMinuteR.Value),
        Tens = OutputConverter.CreateImageItems(model.TimeMinuteL.Value)
    };
    if ((minutes?.Ones ?? minutes?.Tens) == null) minutes = null;


    if ((hours ?? minutes) != null)
    {
        Time = new Time
        {
            Hours = hours,
            Minutes = minutes
        };
    }
}


// Date
{
    var day = OutputConverter.CreateNumberXY(model.DateDay.Value);
    var week = OutputConverter.CreateImageItems(model.DateWeek.Value);
    var enWeek = new InputImageItems()
    {
        ImageIndex = week.ImageIndex,
        ImagesCount = 7,
        X = week.X,
        Y = week.Y
    };
    var cnWeek = new InputImageItems()
    {
        ImageIndex = enWeek.ImageIndex + 7,
        ImagesCount = 7,
        X = week.X,
        Y = week.Y
    };
    var cn2Week = new InputImageItems()
    {
        ImageIndex = cnWeek.ImageIndex + 7,
        ImagesCount = 7,
        X = week.X,
        Y = week.Y
    };

    var date = new DateMonthDay();
    var isInit = false;

    if (model.DateSettings.Value.IsSeparated.Value)
```

```
    {
        var month = OutputConverter.CreateNumberXY(model.DateMonth.Value);
        if ((day ?? month) != null)
        {
            date.Separate = new DateSeparate
            {
                Day = day,
                Month = month
            };

            isInit = true;
        }
    }
    else
    {
        if (day != null)
        {
            date.OneLine = new DateLine
            {
                Number = day
            };
            date.OneLine.CopyDelimiter(model.DateDay.Value);

            isInit = true;
        }
    }


    if (isInit)
    {
        date.TwoDigitsDay = model.DateSettings.Value.TwoDigitsDay.Value;
        date.TwoDigitsMonth = model.DateSettings.Value.TwoDigitsMonth.Value;
    }
    else date = null;


    if (date != null || week != null)
    {
        Date = new Date
        {
            MonthAndDayAndYear = date,
            ENWeekDays = enWeek,
            CNWeekDays = cnWeek,
            CN2WeekDays = cn2Week
        };
    }
}


// Activity
{
    var activity = new Activity();
    var isInit = false;

    // Calories
    {
        var item = new ActivityCalories();
        var from = model.ActivityCalories.Value;

        item.Number = OutputConverter.CreateNumberXY(from);

        if (item?.Number != null)
        {
            item.CopySuffix(from);
```

```
            activity.Calories = item;
            isInit = true;
        }
    }

    // Distance
    {
        var item = new ActivityDistance();
        var from = model.ActivityDistance.Value;

        item.Number = OutputConverter.CreateNumberXY(from);


        if (item?.Number != null)
        {
            item
                .CopySuffix(from)
                .CopyDecimal(from);

            activity.Distance = item;
            isInit = true;
        }
    }

    // Steps
    {
        var item = new ActivitySteps();
        var from = model.ActivitySteps.Value;

        item.Number = OutputConverter.CreateNumberXY(from);

        if (item?.Number != null)
        {
            activity.Steps = item;
            isInit = true;
        }
    }

    // Pulse
    {
        var item = new ActivityPulse();
        var from = model.ActivityPulse.Value;

        item.Number = OutputConverter.CreateNumberXY(from);

        if (item?.Number != null)
        {
            item.CopyNoData(from);

            activity.Pulse = item;
            isInit = true;
        }
    }

    if (isInit) Activity = activity;
}


// Status
{
    var status = new Status();
    var isInit = false;


    status.Bluetooth = OutputConverter.CreateSwitcher(model.StatusBluetooth.Value);
```

```
        status.Lock = OutputConverter.CreateSwitcher(model.StatusLock.Value);
        status.DoNotDisturb = OutputConverter.CreateSwitcher(model.StatusNotDisturb.Value);

        if ((status?.Bluetooth ?? status?.Lock ?? status?.DoNotDisturb) != null) isInit = true;

        if (isInit) Status = status;
    }


    // Battery
    {
        var item = new Battery
        {
            BatteryText = new BatteryText()
            {
                Coordinates = OutputConverter.CreateNumberXY(model.BatteryPercentage.Value)
            },
            BatteryIcon = OutputConverter.CreateImageItems(model.BatteryIcon.Value)
        };

        if (item.BatteryText?.Coordinates == null) item.BatteryText = null;

        if (item?.BatteryText != null || item?.BatteryIcon != null)
        {
            Battery = item;
        }
    }
}


private ImageIndexChecker GetIndexChecker()
{
    var counter = new ImageIndexChecker(FolderPath);

    // Background
    counter.Add(Background?.Image);
    counter.Add(Background?.Preview1);


    // Time
    counter.Add(Time?.Hours?.Tens);
    counter.Add(Time?.Hours?.Ones);
    counter.Add(Time?.Minutes?.Tens);
    counter.Add(Time?.Minutes?.Ones);


    // Date
    {
        var date = Date?.MonthAndDayAndYear;
        var inline = date?.OneLine;
        var separate = date?.Separate;

        if (inline != null)
        {
            counter.Add(inline);
            counter.Add(inline?.Number);
        }
        else if (separate != null)
        {
            counter.Add(separate?.Day);
            counter.Add(separate?.Month);
        }
    }
    counter.Add(Date?.ENWeekDays);
```

```
            // Activity
            counter.Add(Activity?.Steps?.Number);
            {
                var item = Activity?.Calories;

                counter.Add(item);
                counter.Add(item?.Number);
            }
            {
                var item = Activity?.Distance;

                counter.Add(item?.Number);
                counter.Add((IInputSourceIndexDecimal)item);
                counter.Add((IInputSourceIndexSuffix)item);
            }
            {
                var item = Activity?.Pulse;

                counter.Add(item);
                counter.Add(item?.Number);
            }


            // Status
            counter.Add(Status?.Bluetooth);
            counter.Add(Status?.Lock);
            counter.Add(Status?.DoNotDisturb);
            counter.Add(Battery?.BatteryIcon);
            counter.Add(Battery?.BatteryText?.Coordinates);


            return counter;
        }
    }
}
// Custom Libs
using Newtonsoft.Json;

// Solution Libs
using XMIWatch.Watch.Base.InputModel.Classes.Containers;
using XMIWatch.Watch.Base.InputModel.Interfaces;
using XMIWatch.Watch.Base.InputModel.Interfaces.Parts;
using XMIWatch.Watch.Series.MiBand4.Input;
using XMIWatch.Watch.Series.MiBand5.Project;

namespace XMIWatch.Watch.Series.MiBand5.Input
{
    public partial class InputMiBand5Model : IModelConvertible<InputMiBand5Model, MiBand5Model>
    {
        public InputMiBand5Model() { }

        public string FolderPath { get; set; }
        public Background Background { get; set; }
        public Activity Activity { get; set; }
        public Battery Battery { get; set; }
        public Status Status { get; set; }
        public Time Time { get; set; }
        public Date Date { get; set; }
    }

    #region Background Model

    public class Background
    {
```

```
    public InputImageItem Image { get; set; }
    public InputImageItem Preview1 { get; set; }
    public InputImageItem Preview2 { get; set; }
    public InputImageItem Preview3 { get; set; }
}

#endregion

#region Date Model

public class DateSeparate
{
    public InputNumberXY Day { get; set; }
    public InputNumberXY Month { get; set; }
}

public class DateLine : IInputSourceIndexDelimiter
{
    public InputNumberXY Number { get; set; }
    public int? DelimiterImageIndex { get; set; }
}

public class DateMonthDay
{
    public DateSeparate Separate { get; set; }
    public DateLine OneLine { get; set; }
    public bool TwoDigitsMonth { get; set; } = true;
    public bool TwoDigitsDay { get; set; } = true;
}

public class Date
{
    public DateMonthDay MonthAndDayAndYear { get; set; }
    public InputImageItems ENWeekDays { get; set; }
    public InputImageItems CNWeekDays { get; set; }
    public InputImageItems CN2WeekDays { get; set; }
}

#endregion

#region Activity Models

public class ActivitySteps : IInputSourceIndexPrefix
{
    public InputNumberXY Number { get; set; }
    public int? PrefixImageIndex { get; set; }
}

public class ActivityPulse : ActivitySteps, IInputSourceIndexNoData
{
    public int? NoDataImageIndex { get; set; }
}

public class ActivityCalories : ActivitySteps, IInputSourceIndexSuffix
{
    [JsonProperty("Text")]
    public new InputNumberXY Number { get; set; }
    public int? SuffixImageIndex { get; set; }
}

public class ActivityDistance : ActivitySteps, IInputSourceIndexDecimal, IInputSourceIndexSuffix
{
    [JsonProperty("KmSuffixImageIndex")]
    public int? SuffixImageIndex { get; set; }
    public int? DecimalPointImageIndex { get; set; }
```

```
    }

    public class Activity
    {
        public ActivitySteps Steps { get; set; }
        public ActivityPulse Pulse { get; set; }
        public ActivityDistance Distance { get; set; }
        public ActivityCalories Calories { get; set; }
    }

    #endregion

    #region Status Model

    public class Coordinates : IInputContainerPartAxis
    {
        public int X { get; set; }
        public int Y { get; set; }
    }

    public class BatteryText
    {
        public InputNumberXY Coordinates { get; set; }
    }

    public class Battery
    {
        public BatteryText BatteryText { get; set; }
        public InputImageItems BatteryIcon { get; set; }
    }

    public class Status
    {
        public InputSwitcher DoNotDisturb { get; set; }
        public InputSwitcher Lock { get; set; }
        public InputSwitcher Bluetooth { get; set; }
    }

    #endregion
}
// System Libs
using System;

// Solution Libs
using XMIWatch.Services;
using XMIWatch.Watch.Base.ProjectModel.Interfaces;
using XMIWatch.Watch.Base.ProjectModel.Services;
using XMIWatch.Watch.Base.ProjectModel.Services.Containers;
using XMIWatch.Watch.Series.MiBand4.Project;

namespace XMIWatch.Watch.Series.MiBand5.Project
{
    public class MiBand5Model : WatchBase, ICloneable
    {
        /// <summary>
        /// Init all items to default.
        /// </summary>
        public MiBand5Model() : base(new WatchConfig(WatchSeries.MiBand5))
        {
            // Background
            Background = new Observe<MovableItem>(new MovableItem(WatchItemType.Background));
            Preview    = new Observe<MovableItem>(new MovableItem(WatchItemType.Preview));


            // Time
```

```
            TimeMinuteL = new Observe<MovableItems>(new MovableItems(WatchItemType.MinuteL));
            TimeMinuteR = new Observe<MovableItems>(new MovableItems(WatchItemType.MinuteR));
            TimeHourL   = new Observe<MovableItems>(new MovableItems(WatchItemType.HourL));
            TimeHourR   = new Observe<MovableItems>(new MovableItems(WatchItemType.HourR));


            // Date
            DateWeek     = new Observe<MovableItems>(new MovableItems(WatchItemType.Week));
            DateDay      = new Observe<NumberDelimiter>(new NumberDelimiter(WatchItemType.Day));
            DateMonth    = new Observe<Number>(new Number(WatchItemType.Month));
            DateSettings = new Observe<DateSettings>().Init();


            // Activity
            ActivitySteps    = new Observe<Number>(new Number(WatchItemType.Steps));
            ActivityPulse    = new Observe<NumberNoData>(new NumberNoData(WatchItemType.Pulse));
            ActivityCalories = new Observe<NumberSuffix>(new NumberSuffix(WatchItemType.Calories));
            ActivityDistance = new Observe<NumberSuffixDecimal>(new
NumberSuffixDecimal(WatchItemType.Distance));


            // Status
            StatusLock       = new Observe<Switcher>(new Switcher(WatchItemType.Lock));
            StatusBluetooth  = new Observe<Switcher>(new Switcher(WatchItemType.Bluetooth));
            StatusNotDisturb = new Observe<Switcher>(new Switcher(WatchItemType.NotDisturb));


            // Battery
            BatteryIcon      = new Observe<MovableItems>(new MovableItems(WatchItemType.BatteryIcon));
            BatteryPercentage = new Observe<Number>(new Number(WatchItemType.BatteryPercentage));
        }


        /// <summary>
        /// Init copy of <see cref="MiBand5Model"/>.
        /// </summary>
        public MiBand5Model(MiBand5Model model) : base(model)
        {
            // Background
            Background = Background.Clone();
            Preview    = Preview.Clone();

            // Time
            TimeMinuteL = TimeMinuteL.Clone();
            TimeMinuteR = TimeMinuteR.Clone();
            TimeHourL   = TimeHourL.Clone();
            TimeHourR   = TimeHourR.Clone();


            // Date
            DateWeek     = DateWeek.Clone();
            DateDay      = DateDay.Clone();
            DateMonth    = DateMonth.Clone();
            DateSettings = DateSettings.Clone();


            // Activity
            ActivitySteps    = ActivitySteps.Clone();
            ActivityPulse    = ActivityPulse.Clone();
            ActivityCalories = ActivityCalories.Clone();
            ActivityDistance = ActivityDistance.Clone();


            // Status
            StatusLock       = StatusLock.Clone();
```

```
            StatusBluetooth  = StatusBluetooth.Clone();
            StatusNotDisturb = StatusNotDisturb.Clone();


            // Battery
            BatteryIcon      = BatteryIcon.Clone();
            BatteryPercentage = BatteryPercentage.Clone();
        }


        public Observe<MovableItem>         Background        { get; private set; }
        public Observe<MovableItem>         Preview           { get; private set; }
        public Observe<MovableItems>        TimeMinuteL       { get; private set; }
        public Observe<MovableItems>        TimeMinuteR       { get; private set; }
        public Observe<MovableItems>        TimeHourL         { get; private set; }
        public Observe<MovableItems>        TimeHourR         { get; private set; }
        public Observe<MovableItems>        DateWeek          { get; private set; }
        public Observe<NumberDelimiter>     DateDay           { get; private set; }
        public Observe<Number>              DateMonth         { get; private set; }
        public Observe<DateSettings>        DateSettings      { get; private set; }
        public Observe<Number>              ActivitySteps     { get; private set; }
        public Observe<NumberNoData>        ActivityPulse     { get; private set; }
        public Observe<NumberSuffix>        ActivityCalories  { get; private set; }
        public Observe<NumberSuffixDecimal> ActivityDistance  { get; private set; }
        public Observe<Switcher>            StatusLock        { get; private set; }
        public Observe<Switcher>            StatusBluetooth   { get; private set; }
        public Observe<Switcher>            StatusNotDisturb  { get; private set; }
        public Observe<MovableItems>        BatteryIcon       { get; private set; }
        public Observe<Number>              BatteryPercentage { get; private set; }


        /// <summary>
        /// Returns full copy of <see cref="MiBand5Model"/>.
        /// </summary>
        public override object Clone()
            => new MiBand5Model(this);
    }
}
// System Libs
using System.Collections.ObjectModel;
using System.Windows.Media.Imaging;

// Solution Libs
using XMIWatch.Services;
using XMIWatch.UIControls.Previewer.Classes;
using XMIWatch.Watch.Base.ProjectModel.Classes.Extentions;
using XMIWatch.Watch.Base.ProjectModel.Services;

namespace XMIWatch.Watch.Series.MiBand5.Project
{
    public class MiBand5Preview
    {
        private readonly PreviewManager _preview;

        /// <summary>
        /// Binding all items for <paramref name="model"/>.
        /// </summary>
        public MiBand5Preview(MiBand5Model model)
        {
            _preview = new PreviewManager(model);

            // Load Background
            Container.Add(new PreviewItem(model.Background.Value,
                new Observe<BitmapSource>(model.GetAutoSource(model.Background.Value).Source)));
```

```
    // Binding Hours
    Hours = new Observe<int>(12);
    _preview.BindingTime(Hours, model.TimeHourL.Value, model.TimeHourR.Value, Limit.Hours);


    // Binding Minutes
    Minutes = new Observe<int>(12);
    _preview.BindingTime(Minutes, model.TimeMinuteL.Value, model.TimeMinuteR.Value, Limit.Minutes);


    // Binding Week
    Week = new Observe<int>(2);
    _preview.BindingItems(Week, model.DateWeek.Value, Limit.Week, false);


    // Binding Day and Month
    Month      = new Observe<int>(4);
    Day        = new Observe<int>(23);
    DateInline = new Observe<bool>(false);

    _preview.BindingDate(DateInline, Day, Month,
       model.DateDay.Value, model.DateMonth.Value);


    // Binding Distance
    Distance = new Observe<int>(10);
    _preview.BindingNumber(Distance, model.ActivityDistance.Value, Limit.Activity);


    // Binding Calories
    Calories = new Observe<int>(1000);
    _preview.BindingNumber(Calories, model.ActivityCalories.Value, Limit.Activity);


    // Binding Pulse
    Pulse = new Observe<int>(70);
    _preview.BindingNumber(Pulse, model.ActivityPulse.Value, Limit.Pulse);


    // Binding Steps
    Steps = new Observe<int>(10500);
    _preview.BindingNumber(Steps, model.ActivitySteps.Value, Limit.Activity);


    // Binding DoNotDisturb
    DoNotDisturb = new Observe<bool>(true);
    _preview.BindingSwitcher(DoNotDisturb, model.StatusNotDisturb.Value);


    // Binding Bluetooth
    Bluetooth = new Observe<bool>(true);
    _preview.BindingSwitcher(Bluetooth, model.StatusBluetooth.Value);


    // Binding Lock
    Lock = new Observe<bool>(true);
    _preview.BindingSwitcher(Lock, model.StatusLock.Value);


    // Binding Battery
    Battery = new Observe<int>(100);
    _preview.BindingBattery(Battery, model.BatteryIcon.Value, model.BatteryPercentage.Value);
}
```

```
        public ObservableCollection<PreviewItem> Container => _preview.Items;


        public Observe<int>  Hours        { get; private set; }
        public Observe<int>  Minutes      { get; private set; }
        public Observe<bool> DateInline   { get; private set; }
        public Observe<int>  Week         { get; private set; }
        public Observe<int>  Month        { get; private set; }
        public Observe<int>  Day          { get; private set; }
        public Observe<int>  Calories     { get; private set; }
        public Observe<int>  Distance     { get; private set; }
        public Observe<int>  Pulse        { get; private set; }
        public Observe<int>  Steps        { get; private set; }
        public Observe<bool> DoNotDisturb { get; private set; }
        public Observe<bool> Bluetooth    { get; private set; }
        public Observe<bool> Lock         { get; private set; }
        public Observe<int>  Battery      { get; private set; }
    }
}
// System Libs
using System.Windows.Controls;

namespace XMIWatch.Watch.Series.MiBand5.Project
{
    public partial class MiBand5Properties : Grid
    {
        public MiBand5Properties(MiBand5Model model, MiBand5Preview preview)
        {
            Model   = model;
            Preview = preview;

            InitializeComponent();
        }

        public MiBand5Model   Model   { get; private set; }
        public MiBand5Preview Preview { get; private set; }
    }
}
// Solution Libs
using XMIWatch.Watch.Services;

namespace XMIWatch.Watch.Series.MiBand5.Project.Extentions
{
    public static class MiBand5ModelExt
    {
        /// <summary>
        /// Relink images for same items.
        /// </summary>
        public static T ReLink<T>(this T model)
            where T: MiBand5Model
        {
            var relinker = new ImageReLinker(model);

            relinker.ReLink(model.DateWeek.Value);

            return model;
        }
    }
}
```

ДОДАТОК Б

Відгук керівника економічного розділу

ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Єгоров.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Єгоров.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Єгоров.zip	Архів. Містить коди програми і скомпільовану програму
Презентація	
Єгоров.ppt	Презентація кваліфікаційної роботи