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 *Сєдих Максима Сергійовича*
(ПІБ)

академічної групи *121-19ск-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: Розробка web-додатку синхронізований трекер часу за допомогою технології JavaScript React

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	*доц.Кабак Л.В.*			
розділів:				
спеціальний				
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

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

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

« » _____ 2022 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-19ск-1* *Сєдих М.С.*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи Розробка web-додатку синхронізований трекер часу за допомогою технології JavaScript React

затверджена наказом ректора НТУ «ДП» від №

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2022 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2022 р.***

Завдання видав _____ *доц. Кабак Л.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Сєдих М.С.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *13.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 81 с., 18 рис., 1 табл., 3 дод., 20 джерел.

Об'єкт розробки: синхронізований трекер часу на технології React.

Мета кваліфікаційної роботи проекту: розробка синхронізованого трекера часу з гнучкою архітектурою для подальшого розвитку.

У вступі конкретизується мета створення роботи та постановка завдання, пояснюється актуальність проблеми та стан її вирішення, описується специфіка області дослідження.

У першому розділі аналізується предметна галузь, описуються конкретні показники, за якими визначається актуальність проблеми, наводяться основні терміни та поняття об’єкту дослідження, визначаються вимоги до продукту.

У другому розділі досліджуються інструменти, за допомогою яких вирішується завдання, описуються функціональні можливості розроблюємого програмного продукту, здійснюється проектування інформаційної системи, описується робота програми.

У третьому розділі визначається трудомісткість розробки програмного забезпечення, підраховуються витрати на створення програмного продукту і розраховується період розробки.

Практичне значення полягає у розробці системи, яка допомагає користувачу слідкувати за часом який він витрачає на розв'язання задач, завдяки чому він може оптимізувати його витрати, та здійснювати планування спираючись на реальні показники.

Продукт є актуальним через радикальні зміни у роботі IT галузі, викликані пандемією та появою необхідності оптимізувати робочий час співробітників, які працюють віддалено.

Список ключових слів: ТРЕКЕР ЧАСУ, ОПТИМІЗАЦІЯ, ЕФЕКТИВНІСТЬ, КЛІЄНТ, ІНТЕРНЕТ, СЕРВЕР, ВЕБ-САЙТ, БІЗНЕС, ІНФОРМАЦІЙНА СИСТЕМА, МОБІЛЬНИЙ ДОДАТОК, ФРЕЙМВОРК.

ABSTRACT

Explanatory note: 81 pages, 18 images, 1 table, 3 appendices, 20 sources.

Object of development: synchronized time tracker based on React technology.

The purpose of the qualification work of the project: development of a synchronized time tracker with a flexible architecture for further development.

The introduction specifies the purpose of creating the work and defines the task, explains the urgency of the problem and the state of its solution, describes the specifics of the study area.

The first section analyzes the subject area, describes the specific indicators that determine the relevance of the problem, provides basic terms and concepts of the object of study, defines requirements for the product.

The second section examines the tools used to solve the problem, describes the functionality of the developed software product, designs the information system, describes the work of the program.

The third section determines the complexity of software development, calculates the cost of creating the software product and calculates the development period.

The practical value is the development of a system that helps a user keep track of the time he spends on tasks, so he can optimize its costs, and carry out planning based on real indicators.

The product is relevant due to the radical changes in the IT industry caused by the pandemic and the need to optimize the working hours of employees who work remotely.

List of keywords: TIMER TRACKER, OPTIMIZATION, EFFICIENCY, CLIENT, INTERNET, SERVER, WEBSITE, BUSINESS, INFORMATION SYSTEM, MOBILE APP, FRAMEWORK.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ООП – об’єктно-орієнтоване програмування;

ПЗ – програмне забезпечення;

Клієнт – користувацька частина застосунку;

Сервер – серверна частина застосунку;

ORM – Об’єктно-Реляційне відображення. Технологія сфери комп'ютерної інженерії;

IT – сфера інформаційних технологій.

ВСТУП

У сучасному світі програмне забезпечення стало невід’ємною складовою життя, воно все більше проникає у побут людей та вносить корективи у наш розпорядок дня. Майже кожна побутова задача, або область професійної діяльності, може бути оптимізована за допомогою новітніх технологій.

Саме тому сьогодні з’являється все більше компаній, які спеціалізуються на розробці програмного забезпечення, та ставлять собі як мету оптимізацію витрат та прискорення циклу розробки, тобто організацію потоку створення нових програмних продуктів. Такий спосіб роботи є дуже вигідним, адже у такому ритмі є простір для мінімізації ризиків та максимізації прибутку, але водночас це є дуже складним процесом, який потребує багато експертизи та контролю.

Основною робочою одиницею таких компаній є програміст, час якого доцільно контролювати, адже він є дуже дорогим. Сучасні IT компанії мають у штаті і багато інших позицій, від менеджерів до тестувальників, які теж зіштовхуються з тими ж проблемами. Окрім того, такий застосунок буде майже єдиним об'єктивним джерелом інформації для планування часу, який потрібно витратити на майбутні проєкти.

Розроблюємий програмний продукт ставить перед собою мету вирішення цієї проблеми. Яка є дуже актуальною у цій сфері, та взагалі для будь-якої творчої професії, або навіть для оптимізації простих буденних задач поза роботою.

Ця область має свою специфіку, такі застосунки повинні постійно отримувати актуальну інформацію від користувача у впродовж дня, вони мають буди максимально зручні та швидкодоступні. Тому при розробці багато уваги виділяється саме доступності застосунку на багатьох пристроях, та простоті використання, натомість усю складну логіку є доцільним перекласти на серверну частину.

Завдання даної роботи: створення повного програмного продукту призначеного для контролю робочого часу. Продукт має складається з серверної частини, яка бере на себе усю логіку додатку, та клієнтських застосунків під всі популярні платформи, а саме: android, ios, web.

АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

Відстеження часу – це спосіб ефективно керувати робочим часом. Якщо виконувати його регулярно та акуратно, цей метод може допомогти вам підвищити продуктивність і, зрештою, підвищити зацікавленість і продуктивність співробітників вашої компанії у цілому.

На сьогоднішній час індустрія інструментів для відстеження часу очікує швидкого росту, це пов`язано на сам перед з пандемією, та необхідності IT індустрії переводити свій штат співробітників на віддалену роботу. Поряд з цим перед керівництвом компаній стоїть завдання контролю ризиків, які з’являються з переходом на віддалену роботу. Одним з рішень цієї проблеми і є контроль часу, який співробітники витрачають на те чи інше завдання.

За даними дослідження ринків від компанії Data Bridge, прогнозується, що в період з 2021 по 2028 рр. CAGR зросте на рівні 20,40% [1].

Зведений річний темп зростання (CAGR) – це норма прибутку, яка була б потрібна для зростання інвестицій від початкового балансу до кінцевого, якщо прибуток було реінвестовано наприкінці кожного періоду життєвого циклу інвестицій.

Найбільшою причиною зростання інтересу роботодавців до спостереження за робочим часом співробітників є їхня невизначеність і занепокоєння щодо статусу своєї компанії та того, чи роблять співробітники те, що необхідно робити для підтримки загальної ефективності бізнесу.

Так за даними опитування ExpressVPN серед власників бізнесу з теми спостереження за віддаленими працівниками [2] відомо що:

– 74% відчувають що їм бракує контролю над їх бізнесом.

– 69% відчувають себе неспокійно, томузо вони не можуть спостерігати своїх співробітників персонально.

– 57% не довіряють співробітникам без персонального контролю.

– 59% не довіряють співробітникам без програмного контролю над їх працею.

Рис. 1.1. Результати опитування ExpressVPN з теми спостереження за віддаленими працівниками

Такі опасіння є дуже обгрунтованими. IT працівникам з переходом на віддалену роботу дійсно може бракувати концентрації на робочому процесі.

Нове дослідження, проведене найбільшим у Великобританії, показало, що середній офісний працівник у Великобританії продуктивний лише 2 години 53 хвилини робочого дня.

За опитуванням Vouchercloud у 2022 році основними факторами, які відволікають працівників від роботи [3] є:

1. Перевірка соціальних мереж – 47%
2. Читання новинних сайтів – 45%
3. Обговорення позаробочої діяльності з колегами – 38%
4. Приготування гарячих напоїв – 31%
5. Перекури – 28%

6. Текстові/миттєві повідомлення – 27%

7. Вживання закусок – 25%

8. Приготування їжі в офісі – 24%

9. Здійснення дзвінків партнеру/друзям - 24%

10. Пошук нової роботи – 19%

Рис. 1.2. Результати опитування Vouchercloud з теми відволікаючих факторів, які впливають на працівників під час роботи

Дослідження Udemy у 2021 році показує, 67% працівників прагнуть вдосконалити себе, використовуючи додатковий час (без поїздок на роботу та з меншою кількістю соціальних занять), щоб опанувати нові навички або покращити наявні. 37% респондентів витрачали час на додаткову підготовку з професійних або певних навичок. Крім того, враховуючи, що віддалена робота, швидше за все, стане реальністю на довгий час, співробітники шукають способи покращити свою роботу з дому [4].

Але хоча автоматизоване відстеження часу і може полегшити життя, особливо коли мова йде про обробку заробітної плати, це також викликає проблеми.

Так за дослідженям QuickBooks у 2019 році [5] відомо що:

– 4% співробітників стверджують, що GPS-відстеження порушує їхнє право на конфіденційність

– 20% співробітників вдадуться до будь-якої форми юридичних позовів, якщо дізнаються, що за ними стежать поза робочим часом.

– 78% керівників і менеджерів вважають за краще довіряти своїм співробітникам, а не використовувати інструменти відстеження часу

Ці опитування дуже важливо враховувати під час проектування такого продукту, він має бути максимально прозорим та зрозумілим для кінцевого користувача, та разом з цим не повинен порушувати його свободи.

1.2. Призначення розробки та галузь застосування

Призначенням розробки даного програмного продукту є створення інформаційної системи, яка дозволяє кінцевому користувачеві слідкувати за продуктивністю роботи шляхом вимірювання часу який він витрачає на вирішення задач.

Це програмне рішення розробляється для персонального використання, та може бути встановлено на більшості популярних платформах.

Цільовою аудиторією цього продукту є робітники IT компаній віком 18-60, але базова концепція застосунку є дуже гнучкою та може використовуватися будь-якою людиною для власних цілей.

1.3. Підстави для розробки

Підставою до розробки продукту є наказ ректора Національного технічного університету «Дніпровська політехніка» завдання на дипломування та до кваліфікаційної роботи бакалавра на тему «Розробка web-додатку синхронізований трекер часу за допомогою технології JavaScript React».

1.4. Постановка завдання

Кінцевою ціллю розробки продукту є створення сервісу для слідкування за робочим часом користувача, складовими якого є:

- серверна частина з основною логікою продукту;
- мобільний додаток для платформи android;
- мобільний додаток для платформи ios;

– веб додаток.

Користувач застосунку повинен мати змогу:

– створити обліковий запис або скористатися вже існуючим;
– встановити або змінити ім’я користувача у застосунку;
– працювати з сутностями події у системі;
– синхронізувати застосунки на різних платформах між собою.

Програмний продукт повинен мати можливість швидко оновлюватися, адже його базова концепція є дуже гнучкою та має потенціал для розширення.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Оскільки клієнтська частина даного продукту реалізує технологію SPA, за якою додаток є односторінковим та динамічним, вона повинна складатися з таких модулів:

– сторінка реєстрації та авторизації у застосунок;
– сторінка налаштувань;
– головна сторінка.

Ця частина продукту повинна бути локалізована под англійську та українську мови.

Взаємодія користувача з веб-додатком не повинна викликати оновлення сторінки.

Серверна частина повинна виконувати роботу пов’язану з обробкою запитів, роботою з базою даних та забезпечувати можливість синхронізувати клієнтські застосунки, на різних платформах, між собою.

1.5.2. Вимоги до інформаційної безпеки

Для забезпечення інформаційної безпеки продукту потрібно дотримуватись наступних речей:

- забезпечити цілісність даних;
- здійснювати обробку даних на серверній частині;
- захистити всі запити до серверу токеном, тобдо видхиляти неавторизовані запити;
- не зберігати конфіденційну інформацію прокористувачів;

Готовий продукт повинен відповідати всим цим вимогам, натомість на етапі розробки доцільним є:

- використовувати тільки ліцензійне ПЗ;
- мати захист від зловмисних програм на етапі розробки;
- користуватися тільки перевіриними хмарними сервісами;
- не використовувати інструменти, які мають потенційні вразливості.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для розробки або оновлення програмного продукту можна використовувати будь-який компьютер. Продукт розроблювався та тестувався на ноутбуці з системою MacOS та архітектурою процесору M1, тому на етапі збірки прокту на іншій системі можуть з’явитися складнощі, але іх можна вирішити при достатній технічній підготовці.

Для роботи з застосунком на платформі IOS потрібно використовувати будь-який пристрій, який працює на цій системі. Для роботи з додатком для android потрібно використовувати будь-який пристрій який підтримує систему andrpoid версії 7.0 та вище. Для роботи з web застосунком підійде будь-який браузер.

Рекомендовані технічні характеристики для серверу:

- операційна система, яка працює на ядрі Linux;
- процессор Intel Xeon X3440 2.4GHz;

- оперативна пам’ять 8 GB DDR 3 1333hhz;
- жорсткий диск мінімальним об’ємом 40 GB.

1.5.4. Вимоги до інформаційної та програмної сумісності

Програмний продукт має бути написаний на мові програмування JavaScript, під вбудовані браузерні двигуни для цієї мови, та під двигун NodeJS.

Функціонування клієнтський застосунків повинно бути забезпечене на таких платформах:

- IOS 10 версії або вище;
- Android 7 версії або вище;
- Браузери сімейства Chrome, Internet Explorer, Mozilla Firefox, Opera.

Програмний продукт повинен оброблювати помилки пов’язані з несумісністю з платформою. Та доводити до користувача інформацію про необхідність використання JavaScript-у, у випадку якщо його браузер не підтримує цей функціонал, або якщо цей модуль відключено.

ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Система синхронізованого трекеру часу розроблюється з мету підвищення ефективності роботи співробітників IT компаній, шляхом слідкування за часом, який користувач витрачає на вирішення його робочих завдань, та подальшого аналізу його продуктивності. Програмний продукт забезпечує користувачеві платформу, на якій він може відстежувати історію своїх задач, та завдяки цьому планувати свою подальшу діяльність.

Продукт являє собою дуже просту у використанні систему, яка доступна на всіх популярних платформах, завдяки цьому поріг входу до неї істотно зменшується.

2.2. Опис застосованих математичних методів

Дана інформаційна система не потребує використання математичних методів та розробки математичної моделі. Всі операції які використовуються в алгоритмі програми відносяться до простих та не потребують опису.

2.3. Опис використаної архітектури та шаблонів проектування

Шаблоном проєктування вважається розв'язання певної проблеми, яке стало загальноприйнятим при розробці такої категорії програм. Саме шаблони проєктування слід розглядати як той самий цінний програмістський досвід, який дуже сильно рухає індустрію і який не слід ігнорувати.

При розробці даного програмного застосунку було задіяно такий шаблон як SPA.

Ідея цього підходу полягає у тому, щоб отримувати користувацький досвід, який схожий на досвід з використання повноцінних мобільних або десктопних застосунків, але реалізовувати це лише методами JavaScript, HTML, та CSS [7]. Тобто віддавати користувачеві веб-сторінку, яка поводиться як

повноцінний застосунок, який динамічно змінюється без потреби оновлювати сторінку.

Цей підхід передбачає проєктування застосунку як Rich client (Fat client, багатий клієнт), тобто велика частина функціоналу з серверної частини переноситься на клієнтську. Це насамперед стосується базової логіки застосунку, таку як роботу зі станом застосунку, перехід між сторінками, та не заважає здійснювати обробку бізнес-логіки на серверній частині

2.4. Опис використаних технологій та мов програмування

На перший погляд, при розробці веб-застосунку процес вибору мови програмування є простим та вже давно вирішеним, адже відомо що браузери вміють виконувати лише JavaScript.

JavaScript – високорівнева та дуже популярна мова програмування, яка на сьогодні стала ядром усих сучасних веб-застосунків. Вона має динамічну типізацію, та є інтерпретованою, тобто не переводиться у машинний код, а виконується послідовно по операторах.

Веб-застосунки дійсно працюють на ній, але сучасні сайти використовують її лише як мову виконання, тому програміст має змогу використовувати іншу мову для розробки, та компілювати її у JavaScript.

Існує багато мов, які працюють таким чином, але серед них виділяється одна – TypeScript. Це є розширенням від JavaScript, яке додає до неї статичну типізацію, має багато інструментів для роботи з типами, та сильно підвищує якість коду саме на етапі розробки. Після компіляції усі елементи TypeScript зникають та на виході програміст отримує чистий JavaScript, який він може виконувати у браузері і не тільки.

Наочне порівняння програмного модулю на мові JavaScript з тим самим модулем, але написаним на розширенні до цієї мови – TypeScript наведено на рисунку 2.1.

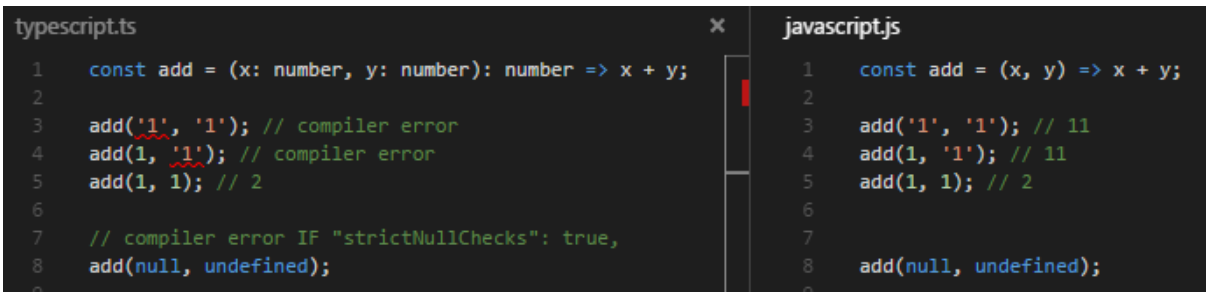

Рис. 2.1. Порівняння коду JavaScript та TypeScript

Розробка ПЗ може займати дійсно багато часу, але цей процес можна пришвидшити за допомогою інструментів. На сьогодні програмісту вже не потрібно на пряму спілкуватися з машиною яка виконує його код, за нього це роблять компілятори та інтерпретатори, переводячи високорівневі мови програмування у машинний код. Але і цього не достатньо, тому інженери навчилися об’єднувати типові для кожної програми модулі у бібліотеки, які у свою чергу розвинулися у дуже потужні продукти, такі як React.

React – це велика JavaScript бібліотека для розробки веб-застосунків. Вона має багато інструментів для роботи з інтерфейсом браузерів, у тому числі і віртуальне дерево об’єктів веб-сторінки. Ця бібліотека створена для розробки додатків за технологією SPA (Single Page Application) – це такий спосіб відображення веб-документу, коли користувач працює з однією динамічною сторінкою, тобто її зміст змінюється у зв’язку з потребами користувача.

Даний застосунок повинен працювати одразу на декількох популярних платформах (web, android, ios), тому і бібліотеки обираються з огляду на це. React має інструменти і для розробки мобільних застосунків, а саме бібліотеку React-Native. Яка теж працює на JavaScript та має дуже схожі правила використання.

Однак робота з декількома платформами все ще забирає багато часу, як на етапі розробки, так і на етапі підтримки продукту. Тут на поміч може прийти ще одна перевага середовища React – react-native-web. За допомогою цього набору інструментів можна перекласти код написаний на React-native, для мобільних застосунків, у компоненти бібліотеки React. Таким чином стає

можливе створення одного застосунку для подальшої компіляції на декількох платформах.

Під час такої розробки виникає багато нюансів, які потрібно вирішувати, адже такий додаток не є звичайною веб-сторінкою, яка виконується у вбудованому браузері мобільного пристрою, а є повноцінною програмою, яка працює з інтерфейсом платформи напряму.

Оскільки для такого застосунку стоїть завдання перевикористання коду, є проблема організації структури проекту. Для розв'язання цієї проблеми існує така технологія як yarn workspaces.

Yarn workspaces – складова пакетного менеджера yarn, яка дозволяє об'єднувати програмні модулі в окремі пакети, таким чином їх можливо повторно використовувати у декількох окремих частинах програми. У цьому випадку у частинах, які відповідають за різні платформи.

Приклад компіляції одного програмного застосунку під різні платформи наведено на рисутку 2.2.

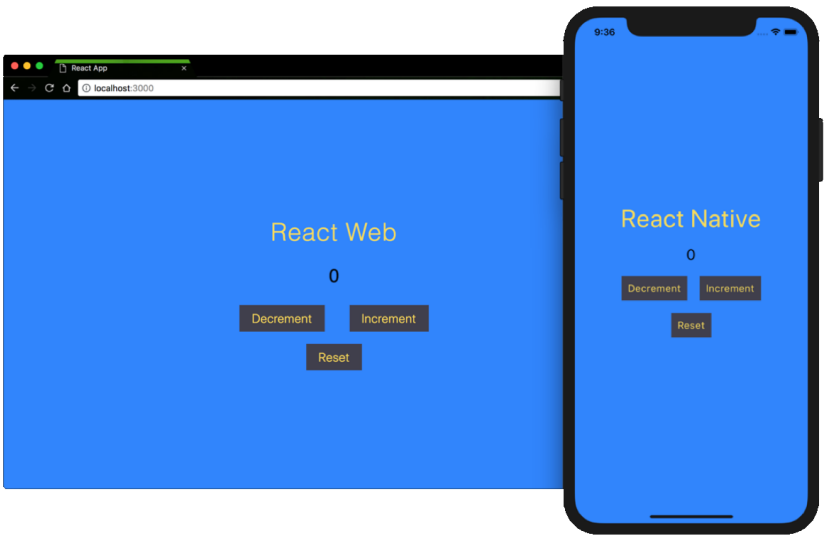

Рис. 2.2. Перевикористання програмного коду для декількох платформ

Програмні продукти у своїй більшості створюються для роботи с даними, навіть звичайний калькулятор, щоб виконати складання двох чисел, потрібно отримати ці два числа від користувача, це і називається даними з якими працює додаток. Більш складні програмні застосунки часто роблять обробку не

миттєво, а оперують вже наявнийми даними. Для цих цілей існують бази даних (БД), які дають змогу додаткам зберігати дані для подальшої обробки.

Бази даних поділяються на два основні типи:

– Реляційні – у таких БД подання даних залежить насамперед від їх фізичного представлення на диску. Основним поняттям яким інженери оперують при розробці додатків з використанням цього типу БД є Таблиця. Дані представляються у таблицях, програміст має можливість створювати, видаляти та редагувати таблиці.

– Нереляційні – виходячи з назви цього типу можна зрозуміти що він сильно відрізняється від першого. Дані у ньому представлені Колекціями та Записами та не мають прив’язки до фізичної структури на диску. Основною перевагою БД такого типу є можливість швидкого розширення Запису, розробнику не потрібно редагувати усі записи, щоб додати нове поле в існуючу модель.

Даний програмний продукт використовує БД нереляційного типу – MongoDB, яка використовує JSON як мову представлення даних. Мову JSON можна визначити як відкритий формат представлення даних, який має зручний для користувача синтаксис, інформація зберігається парами атрибут – значення. Ця мова є розширенням для мови програмування JavaScript.

Для доступу до БД з програмного коду дані повинні бути представлені сутностями, якими оперує обрана мова програмування. JavaScript має підтримку об’єктно-орієнтованого програмування, тому для роботи з БД зазвичай використовуються класи та об’єкти цих класів.

Об’єктно-орієнтоване програмування – підхід, за якого програма представляється об’єктами, класами об'єктів та зв’язками між ними. Цей підхід вважається одним з найвідоміших та підтримується багатьма сучасними мовами програмування.

Для відображення моделей бази даних у програмних застосунках використовуються ORM. ORM (Object-Relational Mapping, Об’єктно-Реляційне

відображення) – технологія програмування, яка зв’язує БД з сутностями об’єктно–орієнтованих мов програмування.

Для розробки цього програмного застосунку було обрано ORM “Prisma”, ця ORM підтримує багато БД, має зручну мову представлення даних та багато інструментів для роботи з graphql.

На даний час існує декілька підходів за якими клієнтська частина продукту спілкується з серверною. При проектуванні цього застосунку перевага надалася саме graphql.

Graphql – сучасна мова, створена для спілкування серверної частини програми з клієнтською. На відміну від аналогів, сервери, які працюють з цією технологією, надають повний опис структури даних якими може скористуватися клієнтська частина, та дає змогу запросити тільки ті дані, які їй потрібні.

На сервері graphql підхід реалізовано за допомогою бібліотеки Apollo Server, а на клієнті за допомогою Apollo Client, який водночас бере на себе роботу з управління даними застосунку та кешування інформації.

Оскільки Apollo Server бере на себе тільки роботу з реалізації підходу graphql, тобто надає змогу серверу приймати запити цією мовою та відповідати на них, було створено багато інструментів які покращують його роботу. Тому як основну серверну бібліотеку було обрано typegraphql, вона дозволяє створювати обробники запитів та, що важливо, має підтримку вже відомої ORM Prisma за допомогою розширення typegraphql-prisma.

2.5. Опис структури програми та алгоритмів її функціонування

Даний програмний продукт складається з двох основних модулів, які у свою чергу можуть поділятися на другорядні модулі.

Таким чином існує клієнтська та серверна частина. Почнемо з серверної, яка знаходиться у директорії /api. Структуру директорії /api зображено на рисунку 2.3.

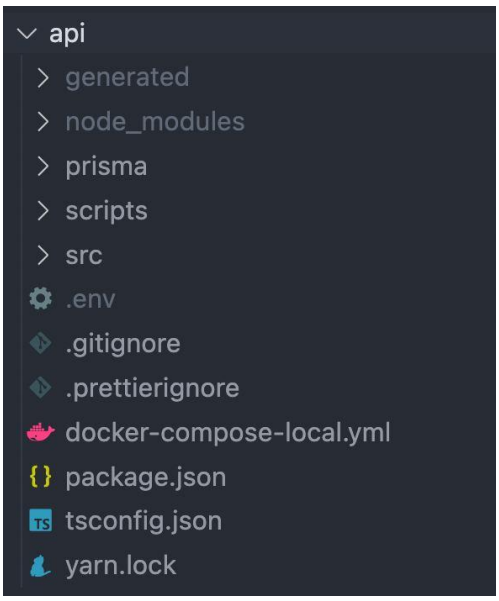

Рис. 2.3. Структура директорії api

Цей модуль складається з таких директорій як:

– /src – основний розділ у якому зберігаються весь вихідний код який стосується серверної логіки застосунку;

– /prisma – містить схему бази даних, яка використовується ORM Prisma;

– /node_modules – тут знаходяться всі модулі, які використовуються програмним застосунком;

– /scripts – містить допоміжні скрипти, які потрібні на етапі розробки чи розгортання на віддаленій машині.

Модуль, який відповідає за клієнтську частину застосунку є більш складним та налічує три основні розділи:

– /web – ця частина відповідає за роботу застосунку у веб-серидовищі;

– /app – по аналогії з /web, відповідає за роботу мобільних застосунків;

– /shared – є основним розділом цього модулю, тобто є ядром всіх середовищ розгортування застосунку.

Оскільки весь код додатку пишеться спільно для всіх платформ, він знаходиться саме тут. Повний перелік розділів в проекті зображено на рисунку 2.4.

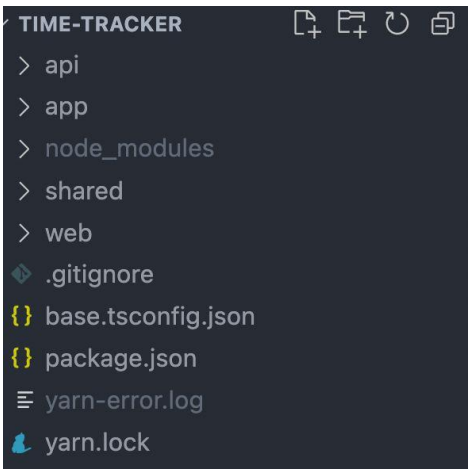

Рис. 2.4. Повна структура проекту

Детально розглянемо /web та /app. Її структура є дуже схожою на /api, але ці розділи не містять вихідного коду, який працює з логікою застосунку. Вони відповідають лише за розгортання програми на відповідних платформах. Вміст цих директорій наведено на рисунку 2.5.

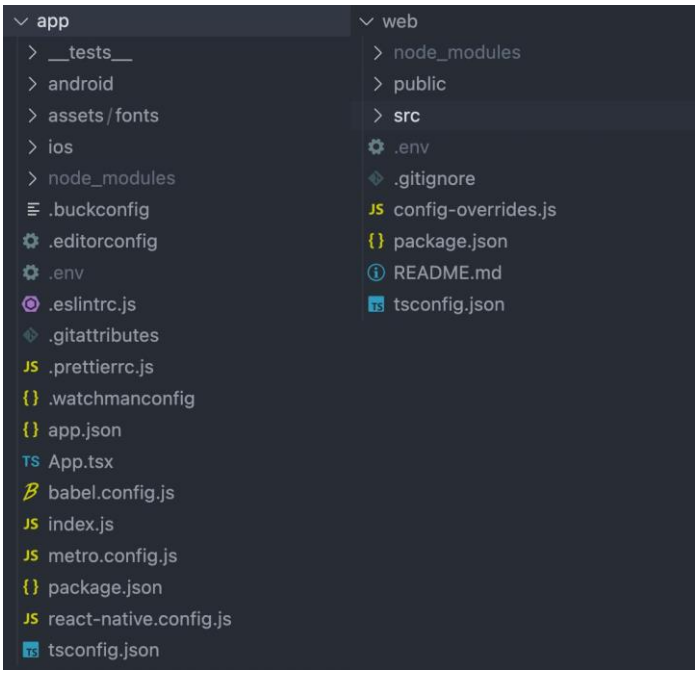

Рис. 2.5. Вміст директорій /app та /web

Натомість розділ /shared містить лише логіку застосунку, вихідний код якої розділено по категоріях у директорії /src. Детальний вміст директорії /shared зображено на рисунку 2.6

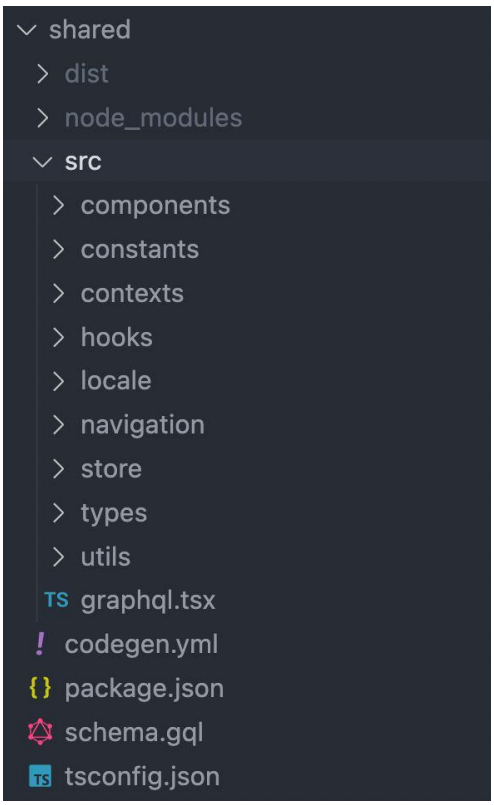

Рис. 2.6. Вміст директорії /shared

Всі ці розділи мають багато спільного. Під час розробки продукту код писався відповідно до принципом DRY (Do not repeat yourself, не повторюй себе), тобто проект написаний таким чином, щоб спільні частини між розділами не повторювались, а перевикористовувались. Таким чином ми маємо спільну конфігурацію TypeScript, яка перевикористовується у кожному розділі, та спільну директорію node_modules, яка відповідає за залежності, які використовуються обома /web та /app розділами.

Повернимося до серверної частини продукту. База даних інформаційної системи налічує чотири основні таблиці, які зв’язані між собою. Схему бази даних системи наведено на рисунку 2.7

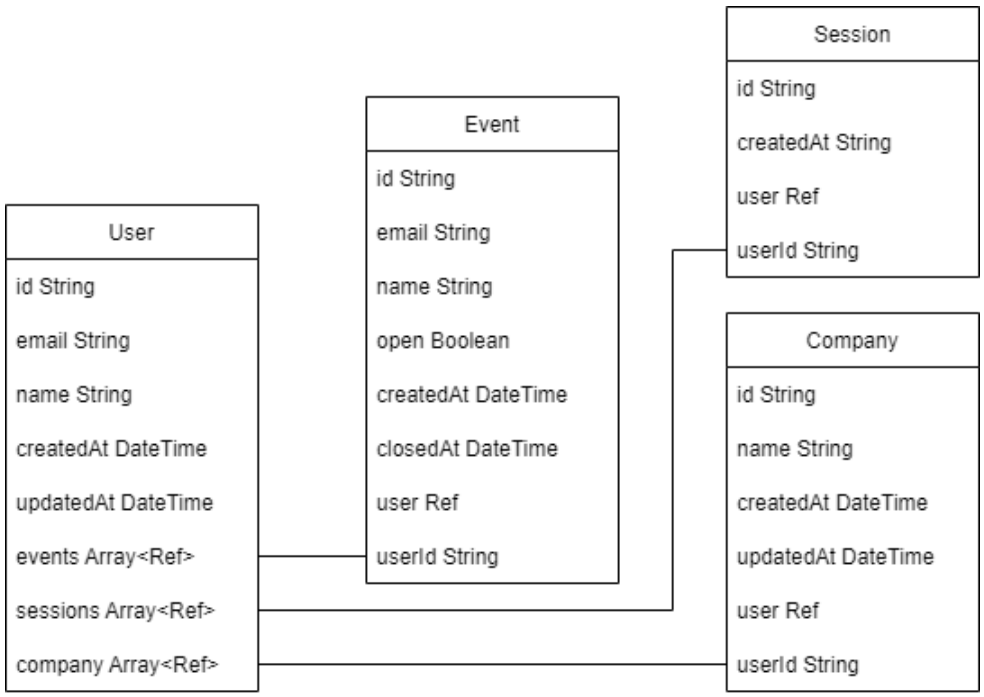

Рис. 2.7. Схема бази даних інформаційної системи

На схемі присутні тількі основні моделі, які безпосередньо відображають логіку роботи системи. Інші таблиці, які описують другорядну логіку, або допоміжні таблиці для сутності Enum на схемі опущені.

Таким чином ми маємо основну модель User, яка описує користувача та має посилання на інші моделі, такі як:

- Company – компанія, до якої користувач має відношення.
- Event – подія користувача
- Session – поточна сессія користувача у застосунку

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Спілкування з серверною частиною застосунку відбувається за допомогою підходу graphql. Клієнтська частина отримує дані від користувача та формує запити за допомогою бібліотеки Apollo Client, яка у свою чергу переводить дані у формат JSON та відсилає запит на сервер. Далі запит оброблюється, та сервер надсилає відповідь у форматі теж JSON.

JSON – це формат обміну даними, який є розширенням мови програмування JavaScript, тобто це текстовий формат, який використовується для обміну даними між платформами [8].

Перевагою такого інструменту як мова graphql є те що програмісту не потрібно виконувати ризикову операцію переведення JSON-у у об'єкт мови JavaScript, це за нього робить бібліотека. Ця операція вважається ризиковою, тому що у відповіді серверу може бути відсутня частина даних, або дані можуть прийти у неправильному форматі. Контроль цієї операції бере на себе graphql.

2.7. Опис розробленого програмного продукту

2.7.1. Використані технічні засоби

Для розробки даної інформаційнох системи був використаний персональний компьютер фірми Apple – MacBook air 13 inch з наступними характеристиками:

- процесор M1 3.2 ГГц 8 ядер вбудованим GPU
- 16 GB RAM LPDDR4X частотою 3733МГц
- 256 GB ssd

Для коректної роботи застосунку потрібен сервер з наступними мінімальними зарактерискиками:

- 20 GB фізичної пам’яті
- 512 MB оперативної пам’яті
- 1 або більше ядер процессора частотою 2.4 ГГц

2.7.2. Використані програмні засоби

При розробці програмних застосунків важливу роль відіграє обране середовище програмування, воно буде впливати як на швидкість та комфорт роботи, так і на потенційну кількість помилок.

Мови програмування та мови представлення даних завжди мають власний синтаксис та правила використання, тому при проектуванні стоїть завдання

обрати правильне середовище розробки, яке має інструменти для роботи з ними.

Виходячи з цього було обране середовище розробки програмних застосунків Visual Studio Code (VSCode). Воно має вбудовані методи для роботи з основними інструментами програмування, та водночас має можливість для розширення функціонала за допомогою технології “Extensions” (Розширення).

Дамо визначення, Visual Studio Code – сучасний редактор вихідного коду, призначений для розробки програмних застосунків. Він поставляється з вбудованими інструментами для дебагінгу, форматування коду, підсвіченням синтаксису та розумною системою автозакінчення програмних операторів.

При розробці даного застосунку було використано такі розширення для VSCode:

– Prettier – покращує вбудовану у VSCode систему форматування коду.

– GitLens – має ряд інструментів для роботи з системою контролю версій.

– Prisma – доповнює вбудовану систему представлення коду, додає підтримку специфічних файлів конфігурації для модуля Prisma.

Встановлені розширення у інтерфейсі редактору вихідного коду Visual Studio Code зображено на рисунку 2.8.

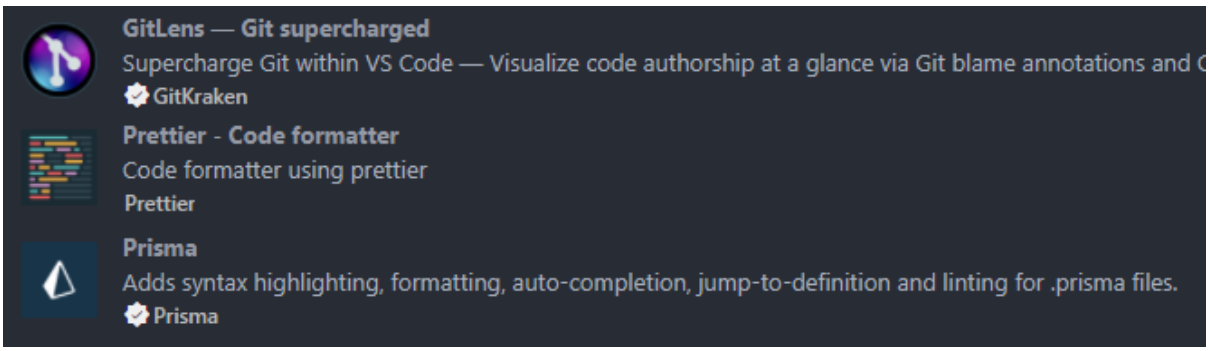

Рис. 2.8. Розширення для редактору Visual Studio Code

Сервер та веб додаток потрібно розміщувати на віддаленій машині, щоб вона працювала цілодобово. Для вирішення цього завдання було обрано платформу Heroku, яка реалізує принцип PaaS (Platform as a service, платформа

як сервіс) та дає можливість збирати, запускати та оперувати додатками на віддаленому сервісі.

Інтерфейс платформи Heroku для серверної частини веб застосунку наведено на рисунку 2.9.

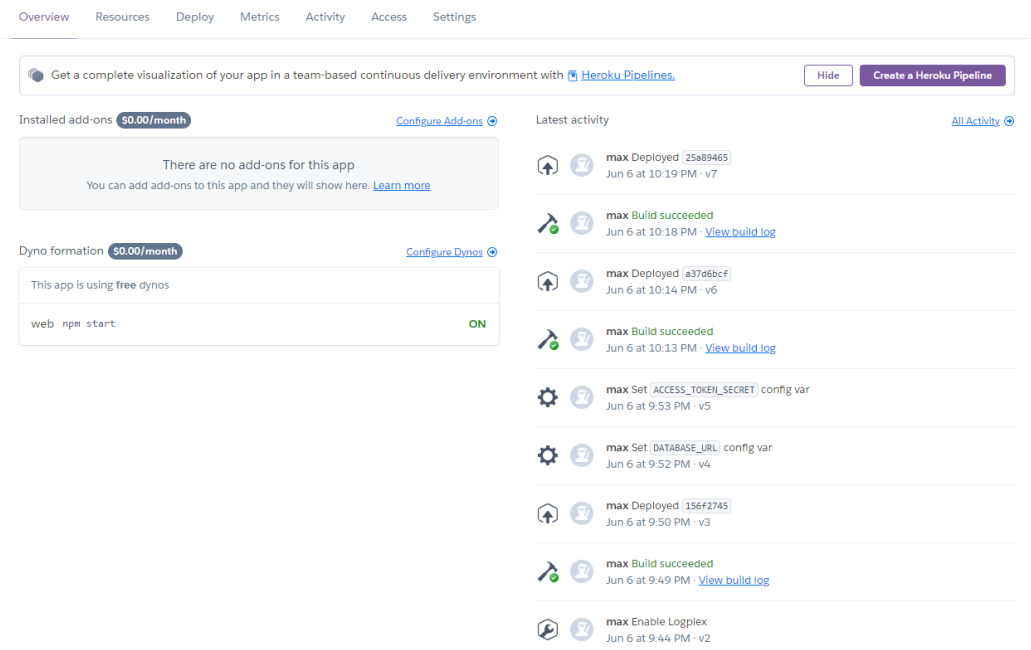

Рис. 2.9. Оглядова сторінка серверної частини продукту на платформі Heroku

2.7.3. Виклик та завантаження програми

Для виклику програми на веб-платформі потрібно завантажити і встановити будь-який браузер та перейти за посиланням https://time-tracker001-app.herokuapp.com/.

Натомість для роботи з мобільними версіями застосунку, потрібно мати відповідні до платформи файли встановлення додатків. Після інсталяції потрібно запустити додаток.

Для роботи з застосунками у режимі розробника повинно встановити на персональний компьютер рушій NodeJS.

Node.js – це JavaScript–оточення побудоване на JavaScript–рушієві Chrome V8 [9]

Далі потрібно інсталювати всі залежності основних модулів продукту, зібрати їх за допомогою команди build та запустити за допомогою команди start.

2.7.4. Опис інтерфейсу користувача

Розроблюємий додаток поставляється на багато платформ, тому для демонстрації інтерфейсу продукту доцільним буде показувати одразу і веб-застосунок і мобільний.

Відкриваючи посилання на трекер часу, або додаток, у перший раз користувач одразу потрапляє на сторінку авторизації у додаток. Екрани авторизації наведено на рисунку 2.10.

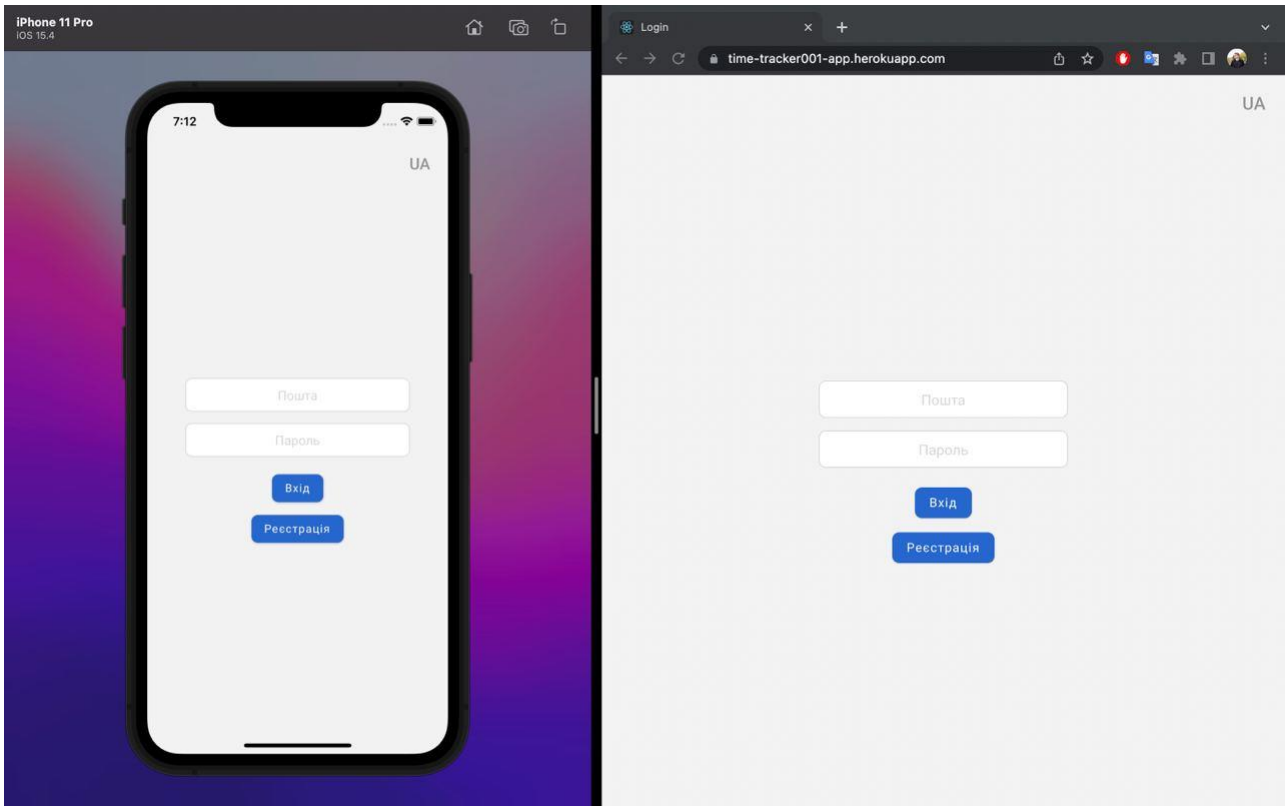

Рис. 2.10. Екран авторизації у застосунок

На цьому екрані користувачу пропонується ввести його пошту та пароль. У випадку, якщо користувач помилився при заповненні цих полів, йому на екран виведеться помилка обраною мовою застосунку. Повідомлення про помилку наведено на рисунку 2.11.

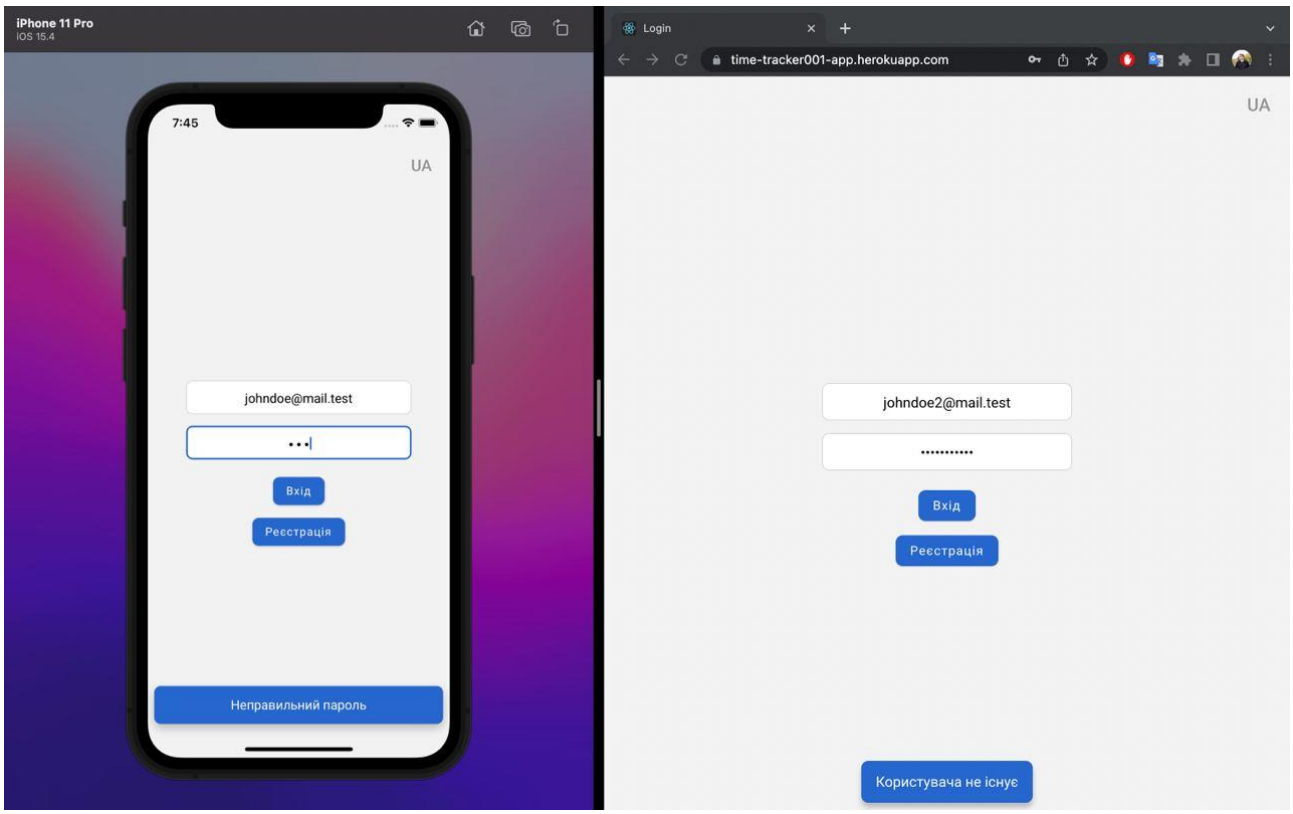

Рис. 2.11. Сторінка авторизації з повідомленням про помилку

Після успішної авторизації користувач потрапляє на головну сторінку додатка. Застосунок реалізує технологію SPA, тому сайт не оновлюється, усі дії проходять на одній сторінці. Головна сторінка додатка наведена на рисунку 2.12.

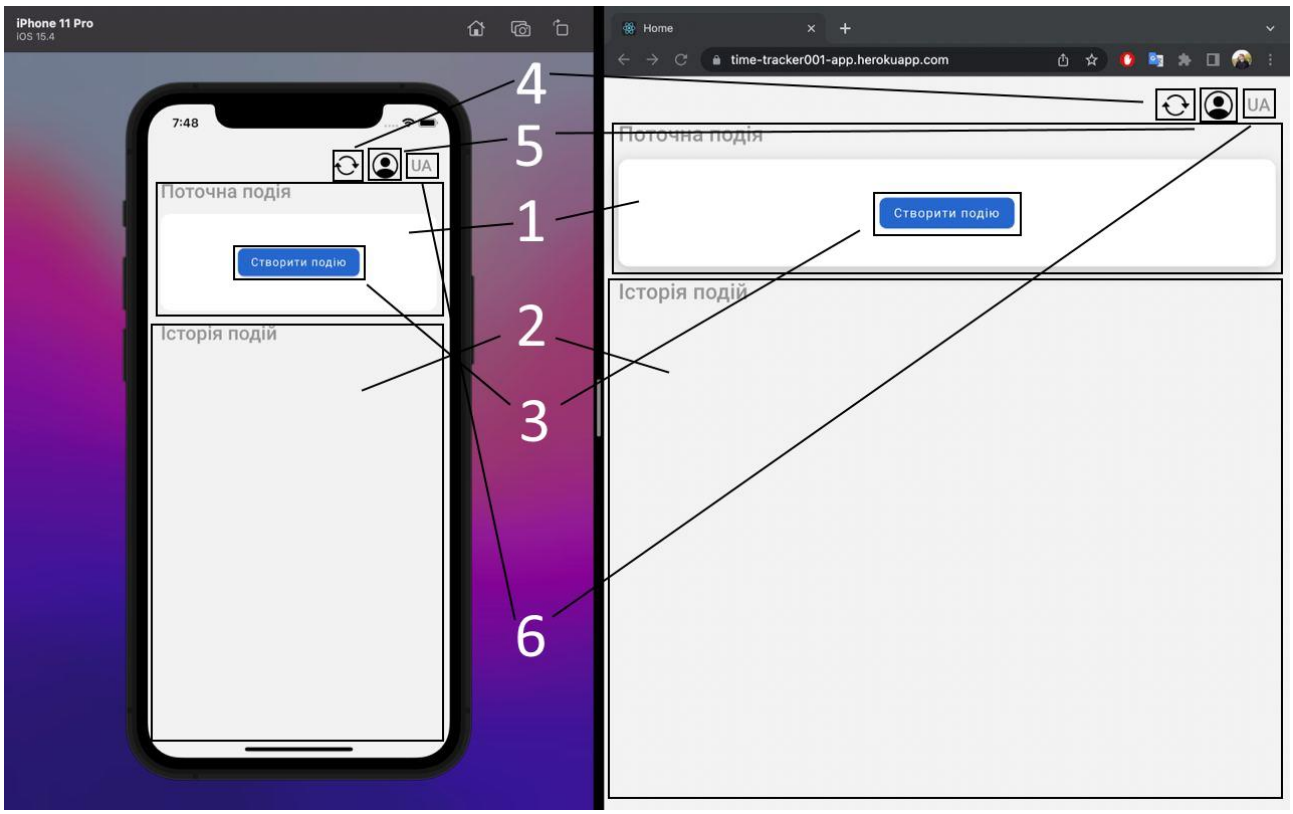

Рис. 2.12. Головна сторінка додатка з підписаними блоками інтерфейсу

Розберемо елементи, які користувач бачить на цій сторінці:

1. Блок поточних подій. Тут буде зображена поточна подія, час який вона триває та елементи управління подією.
2. Блок минулих подій. У цьому блоці зображуються вже закінчені події. Таку подію можна видалити або змінити її ім’я.
3. Елемент управління для створення нової події
4. Елемент управління для оновлення списку подій, він потрібен для синхронізації подій між застосунками.
5. Елемент управління для відкриття вікна налаштувань.
6. Елемент управління для зміни мови застосунку.

Створимо нову подію для користувача. На початку вона не має ім’я, але його можна одразу вказати. Встановимо назву: “Робота над ПЗ”. Щоб подія з’явилася у застосунку на іншій платформі натиснемо на елемент управління для оновлення списку подій. Створена подія зображена на рисунку 2.13.

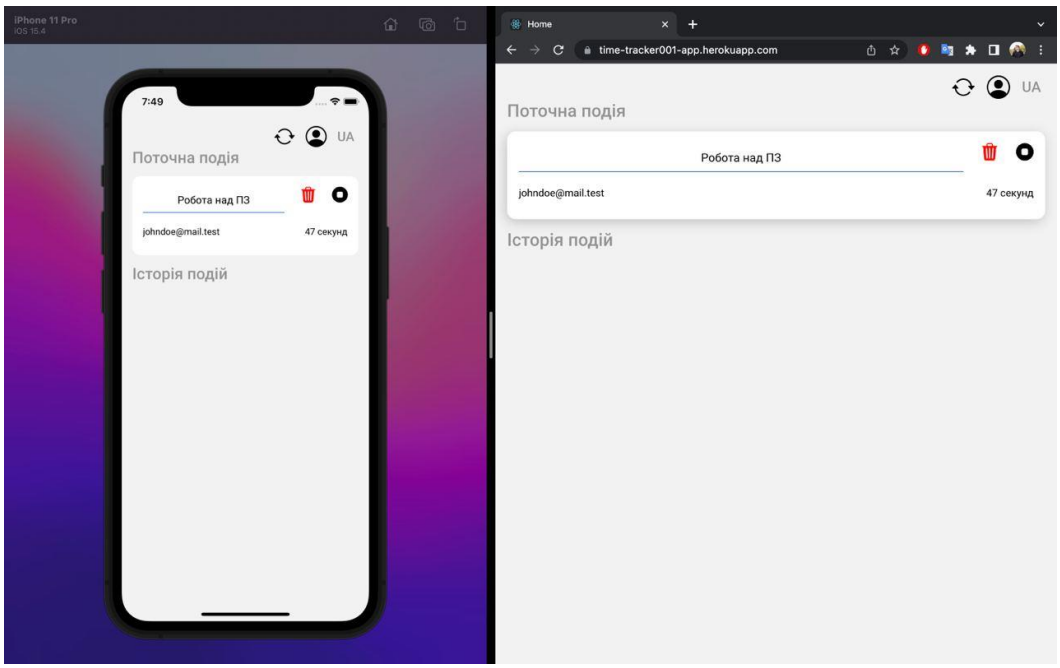

Рис. 2.13. Створення першої події для користувача

Щоб змінити ім’я користувача треба перейти у вікно налаштувань. Зробити це можна відповідним елементом управління.

На цьому екрані ми бачимо пошту користувача та його ім’я, яке ми можемо змінити. На додаток на цьому екрані знаходиться елемент управління для виходу з облікового запису користувача.

Екран налаштувань наведено на рисунку 2.14.

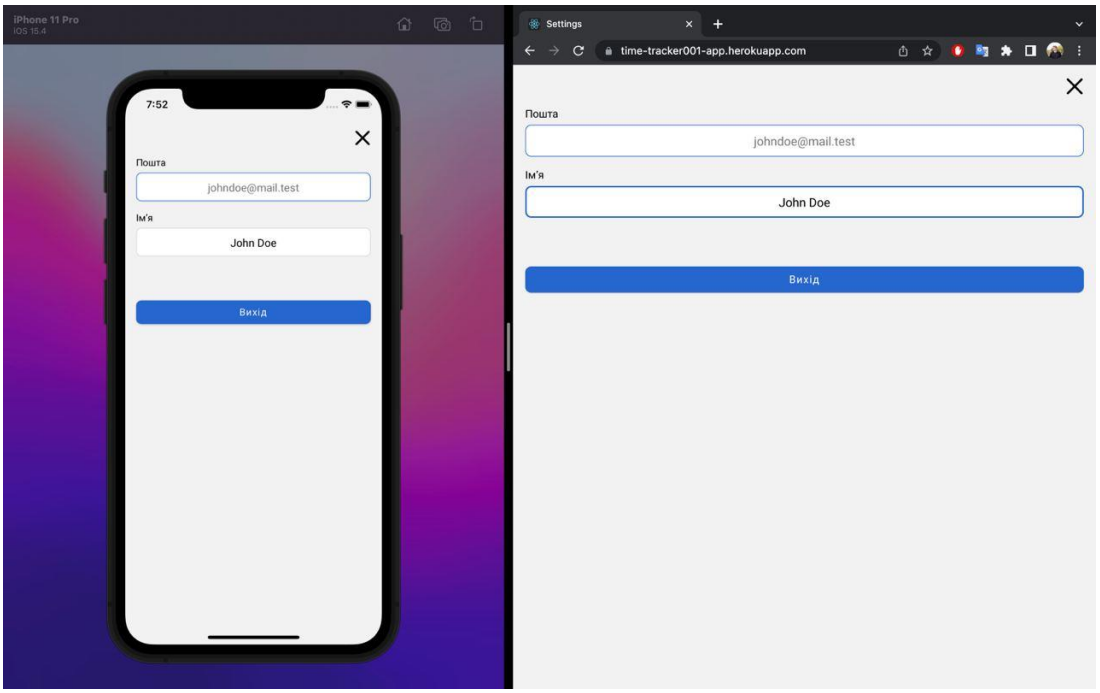

Рис. 2.14. Екран налаштувань користувача

Додамо більше подій для користувача. Історія подій зображена під блоком актуальної події. Заповнена історія подій користувача наведена на рисунку 2.15

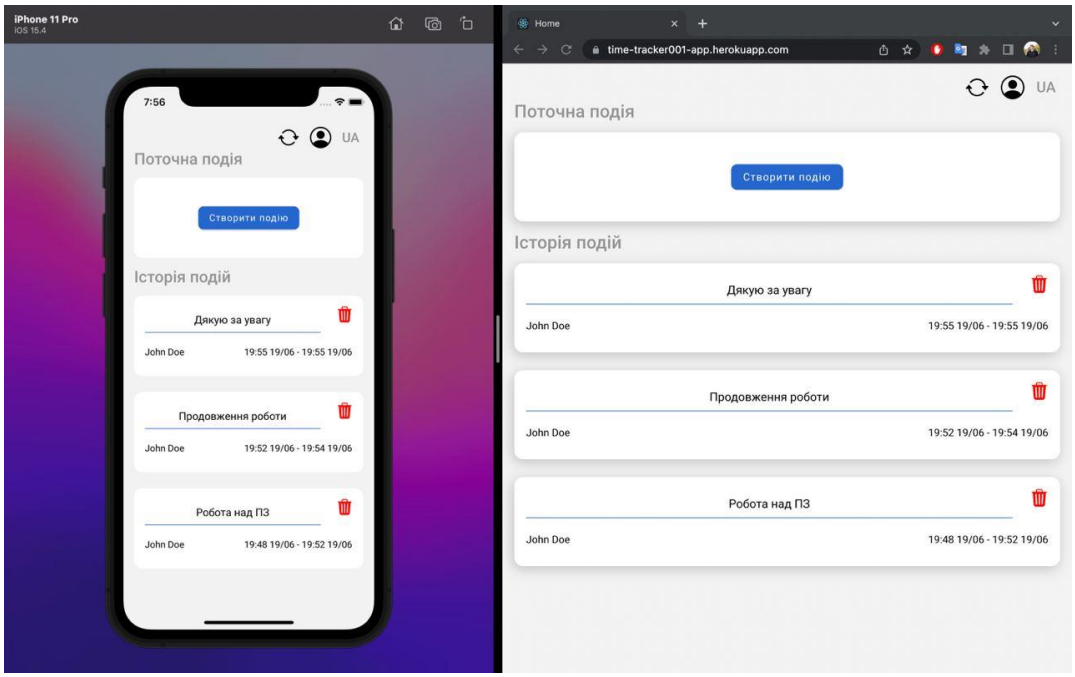

Рис. 2.15. Заповнена історія подій користувача

У додатку можна змінити мову інтерфейсу, це буде впливати як на підписи блоків так і на мову, яка використовується для наведення часу, який триває подія. Інтерфейс застосунку на англійській мові наведено на рисунку 2.16

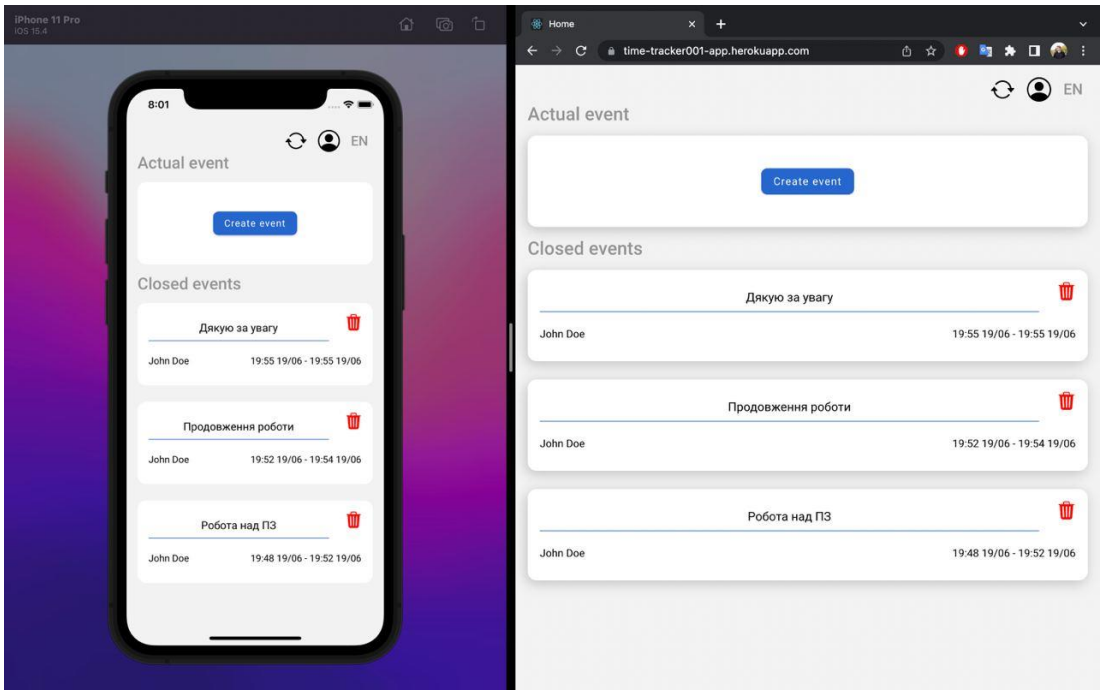

Рис. 2.16. Англомовний інтерфейс застосунку

ЕКОНОМІЧНИЙ РОЗДІЛ

Готовий програмний застосунок повинен виправдовувати витрачені на нього ресурси, тому важливим етапом планування розробки продукту є розрахунок трудомісткості та витрат на створення програмного забезпечення.

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

- q – передбачуване число операторів – 1857.
- C – коефіцієнт складності програми – 1,7.
- p – коефіцієнт корекції програми в ході її розробки – 0,09;
- B – коефіцієнт збільшення витрат – 1,3;
- k – коефіцієнт кваліфікації програміста – 1;
- $С_{ПР}$– середня годинна заробітна плата програміста, грн/год. Спираючись на опитування, проведені командою DOU серед IT фахівців України, середня заробітна плата Juniour JavaScript розробника зі стажем роботи 2 роки складає 900$ на місяць [6]. З урахуванням 40 годинного робочого тижня та при чинному курсі Національного банку України 1 долар = 29,46 гривень, отримаємо значення 166 грн/год;
- B_k– число виконавців – 1;
- F_p– місячний фонд робочого часу (при 40 годинному робочому тижні $F_p = 176$ годин);
- $С_{Мч}$– вартість машино-години ЕОМ, грн/год – 5.

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 (3.1):

$$t = t_o + t_u + t_a + t_n + t_{\text{отл}} + t_{\text{д}}, \text{ людино-годин,} \quad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50),

t_u – витрати праці на дослідження алгоритму рішення задачі,

t_a – витрати праці на розробку блок-схеми алгоритму,

t_n – витрати праці на програмування по готовій блок-схемі,

$t_{\text{отл}}$ – витрати праці на налагодження програми на ЕОМ,

$t_{\text{д}}$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється. Для взначення умовного числа операторів використовується формула (3.2):

$$Q = q * C * (1 + p), \quad (3.2)$$

де q – передбачуване число операторів,

C – коефіцієнт складності програми,

p – коефіцієнт кореляції програми в ході її розробки.

$$Q = 1430 * 1{,}5 * (1 + 0{,}08) = 2{,}316.60$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста, за формулою (3.3):

$$t_u = \frac{Q * B}{(75..85) * k}, \text{людино} - \text{годин,} \quad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі,

k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

Витрати праці на вивчення опису задачі:

$$t_u = \frac{2{,}316.60 * 1.5}{80 * 1} = 50{,}68, \text{людино} - \text{годин}.$$

Витрати праці на розробку алгоритму рішення задачі зозраховуються формулою (3.4):

$$t_a = \frac{Q}{(20..25) * k}, \text{людино} - \text{годин}. \quad (3.4)$$

$$t_a = \frac{2{,}316.60}{20 * 1} = 144{,}79, \text{людино} - \text{годин}.$$

Витрати на складання програми по готовій блок–схемі можна вирахувати формулою (3.5):

$$t_n = \frac{Q}{(20..25) * k}, \text{людино} - \text{годин}. \quad (3.5)$$

$$t_n = \frac{2{,}316.60}{23 * 1} = 125{,}90, \text{людино} - \text{годин}.$$

Витрати праці на налагодження програми на ЕОМ за умови автономного налагодження одного завдання розраховуються за формулою (3.6):

$$t_{\text{отл}} = \frac{Q}{(4..5) * k}, \text{людино} - \text{годин}. \quad (3.6)$$

$$t_{\text{отл}} = \frac{2{,}316.60}{4 * 1} = 723{,}94, \text{людино} - \text{годин}.$$

Витрати за умови комплексного налагодження завдання обчислюються формулою (3.7):

$$t_{\text{отл}}^{k} = 1{,}5 * \ t_{\text{отл}}, \text{людино} - \text{годин}. \quad (3.7)$$

$$t_{\text{отл}}^{k} = 1.5 * \ 723{,}94 = 1085{,}91, \text{людино} - \text{годин}.$$

Витрати праці на підготовку документації можна обчислити за формулою (3.8):

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}}, \text{людино} - \text{годин} \quad (3.8)$$

де $t_{\text{др}}$ – трудомісткість підготовки матеріалів і рукопису, , що розраховується за формулою (3.9).

$$t_{\text{др}} = \frac{Q}{(15..20) * k}, \text{людино} - \text{годин}, \quad (3.9)$$

$$t_{\text{др}} = \frac{2{,}316.60}{15 * 1} = 193{,}05, \text{людино} - \text{годин}.$$

$t_{\text{до}}$ – трудомісткість редагування, печатки й оформлення документації, що розраховується за формулою 3.10.

Отримаємо трудомісткість розробки програмного забезпечення:

$$t_{\text{до}} = 0{,}7 - 5 * t_{\text{др}}, \text{людино} - \text{годин}. \quad (3.10)$$

$$t_{\text{до}} = 0{,}75 * 193{,}05 = 144{,}79, \text{людино} - \text{годин}.$$

$$t_{\text{д}} = 193{,}05 + 144{,}79 = 337{,}84, \text{людино} - \text{годин}.$$

$$t = 50 + 50{,}68 + 144{,}79 + 125{,}90 + 723{,}94 + 337{,}84$$
$$= 1433{,}15, \text{людино} - \text{годин}$$

У результаті ми розрахували, що в загальній складності необхідно $1433{,}15, \text{людино} - \text{годин}$ для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

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. Розраховується $К_{ПО}$ за формулою (3.11):

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{грн.} \quad (3.10)$$

Заробітна плата виконавців визначається за формулою (3.11):

$$З_{ЗП} = t * С_{ПР}, \text{грн}, \quad (3.12)$$

де: t – загальна трудомісткість, людино-годин,

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

$$З_{ЗП} = 1433{,}15 * 166 = 237\,901.29, \text{грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ визначається за формулою (3.13):

$$З_{МВ} = t_{отл} * С_{мч}, \text{грн}, \quad (3.13)$$

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

$С_{мч}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 723{,}94 * 5 = 3\,619.69, \text{грн.}$$

$$К_{ПО} = 237\,901.29 + 3\,619.69 = 241\,520.98, \text{грн.}$$

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУП.

Очікуваний період створення ПЗ визначається за формулою (3.14):

$$T = \frac{t}{B_k * F_p}, \text{міс}, \quad (3.14)$$

де B_k - число виконавців,

F_p - місячний фонд робочого часу.

$$T = \frac{1433{,}15}{1 * 176} = 8{,}14, \text{міс.}$$

Висновки: обчислено трудомісткість розробленого додатку (1433,15 людино годин), проведено обчислення вартості роботи по створенню програми (241 520.98 грн.) та визначено час на її створення (8,14 місяців).

ВИСНОВКИ

В результаті проведеної кваліфікаційної роботи била досягнута початкова мета, а саме було розроблено синхронізований трекер часу, який працює на всіх найбільш популярних на даний час платформах.

Програмне забезпечення призначено оптимізувати процес професійної діяльності користувача або покращити його ефективність у буденних задачах, шляхом планування часу, який він витрачає на їх вирішення.

Як результат було створено гнучкий продукт, який працює на сучасних технологіях, та має великий потенціал для розвитку. Практичне використання якого допоможе користувачеві оптимально розраховувати свій час та дасть зрозумілу оцінку ефективності працівника його роботодавцю.

Під час розробки були досліджені та реалізовані такі етапи:

- проаналізовано предметну галузь задачі та досліджено ринок подібних продуктів;
- спроектовано комплексну програмну систему з урахуванням подальшого розвитку;
- реалізовано базову концепцію продукту;
- досліджені вектори розвитку даної інформаційної системи.

Продукт створено з урахуванням сучасних тенденцій такого роду систем. Для реалізації використовувався новітній підхід за яким спілкуються серверна та клієнтська частина застосунків - graphql. Проект працює на популярній бібліотеці React, та використовує всі її переваги, а саме: розширення для мобільної розробки — React-Native, можливість перевикористовувати існуючі модулі для декількох платформ за допомогою react-native-web.

Для кваліфікаційної роботи було вирахувано трудомісткість розробки продукту - 1433.15 людино-годин, та потенційні витрати на його створення — 241 520.98 грн.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Global Time Tracking Software Market – Industry Trends and Forecast to 2028. URL: https://www.databridgemarketresearch.com/reports/global-time-tracking-software-market.

2. ExpressVPN survey reveals the extent of surveillance on the remote workforce. URL: https://www.expressvpn.com/blog/expressvpn-survey-surveillance-on-the-remote-workforce/.

3. Survey Reveals Employee Productivity Averages 2 Hours, 23 Minutes in a Work Day. URL: https://www.vouchercloud.com/resources/office-worker-productivity.

4. Udemy In Depth: The Portrait of a Pandemic at Work. URL: https://research.udemy.com/portrait-of-a-pandemic-at-work/.

5. GPS tracking and location-aware tech survey 2019. URL: https://quickbooks.intuit.com/time-tracking/gps-survey/2019-survey/

6. Статистика зарплат програмістів, тестувальників і PM в Україні | DOU URL: https://jobs.dou.ua/salaries/?period=2021-12&position=Junior%20SE&technology=JavaScript (станом на 10.06.2022).

7. Emmit Scott. SPA Design and Architecture: Understanding Single Page Web Applications 1st edition – 2015. – 5 с.

8. Lindsay Bassett. Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON – 2015. – 15 с.

9. Офиційний сайт рушія NodeJS. Головна сторінка. URL: https://nodejs.org/uk/

10. JSON Web Token Introductio. URL: https://jwt.io/introduction

11. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності «Комп'ютерні системи» / О.Г. Вагонова, О.Б. Нікітіна, Н.Н. Романюк; М-во освіти і науки України, ДВНЗ «Нац. гірн. ун- т». – Д.: НГУ, 2013. – 11 с.

12. Ірина Бородкіна, Георгій Бородкін. Інженерія програмного забезпечення. Посібник для студентів вищих навчальних закладів, 2018 – 204 с.

13. Юрій Грицюк. Аналіз вимог до програмного забезпечення, 2018 – 456 с.

14. Schwaber K. Agile Project Management with Scrum Developer Best Practices. Microsoft Press, 2004. – 192 p.

15. Алгоритми і структура даних: Навчальний посібник / В.М.Ткачук. - ІваноФранківськ : Видавництво Прикарпатського національного університету імені Василя Стефаника. – 2016. – 286 с.

16. JavaScript API documentation. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

17. Introduction to GraphQL. Complete GrahpQL documentation. URL: https://graphql.org/learn/

18. NodeJs. Довідкова документація про API. URL: https://nodejs.org/uk/docs/

19. Todd Fredrich. REST API Tutorial. – 2012. – p. 13-15.

20. Алгоритми та структури даних. Навчальний посібник / Т. О. Коротєєва. Львів : Видавництво Львівської політехніки. – 2014. – 280 с.

ДОДАТОК А

Код програми

Серверна частина застосунку:

schema.prisma

```
generator client {
  provider        = "prisma-client-js"
  previewFeatures = ["mongoDb"]
}

datasource db {
  provider = "mongodb"
  url      = env("DATABASE_URL")
}

generator typegraphql {
  provider = "typegraphql-prisma"
  output   = "../generated/type-graphql"
}

model User {
  id String @id @default(auto()) @map("_id") @db.ObjectId

  email    String @unique
  password String
  name     String

  sessions Session[]
  events   Event[]
}

model Session {
  id String @id @default(auto()) @map("_id") @db.ObjectId

  refreshToken String @unique

  user   User   @relation(fields: [userId], references: [id], onDelete: Cascade)
  userId String @db.ObjectId
}

model Event {
  id String @id @default(auto()) @map("_id") @db.ObjectId

  open      Boolean
  name      String
  createdAt DateTime  @default(now())
  closedAt  DateTime?

  user   User   @relation(fields: [userId], references: [id], onDelete: Cascade)
  userId String @db.ObjectId
}
```

Index.ts

```
import 'reflect-metadata';
import './utils/resolversEnhanceMap';
import {ApolloServer} from 'apollo-server';
import {ApolloServerPluginLandingPageGraphQLPlayground} from 'apollo-server-core';
import {buildSchema} from 'type-graphql';
import path from 'path';
import context, {prisma} from './context';
import {resolvers} from '../generated/type-graphql';
import customResolvers from './resolvers';
import authChecker from './utils/authChecker';
```

```
  const bootstrap = async () => {
    await prisma.$connect();

    const schema = await buildSchema({
      emitSchemaFile: path.resolve(__dirname, '../../shared/schema.gql'),
      resolvers: [...resolvers, ...customResolvers],
      authChecker,
    });

    const server = new ApolloServer({
      schema,
      plugins: [ApolloServerPluginLandingPageGraphQLPlayground()],
      introspection: true,
      context,
    });

    server.listen().then(({url}) => {
      console.log(`🚀  Server is ready at ${url}`);
    });
  };

bootstrap();
```

contex.ts

```
import {PrismaClient} from '@prisma/client';
import {ContextFunction} from 'apollo-server-core';
import {ExpressContext} from 'apollo-server-express';
import ContextType from './types/Context';

export const prisma = new PrismaClient();

const makeContext: ContextFunction<ExpressContext, ContextType> = async ({
  req,
}) => ({prisma, req});

export default makeContext;
```

auth.resolver.ts

```
import {Args, Ctx, Mutation, Query, Resolver} from 'type-graphql';
import {uid} from 'rand-token';
import ContextType from '../types/Context';
import {
  AuthArgs,
  AuthPayload,
  LogoutPayload,
  TokenArgs,
} from '../types/graphql/auth.types';
import {accessTokenSecret, refreshTokenSize} from '../utils/env';
import {getAccessToken} from '../utils/token';
import {User} from '../../generated/type-graphql';
import {bearer} from '@borderless/parse-authorization';
import {verify} from 'jsonwebtoken';
import Token from '../types/Token';

@Resolver()
class AuthResolver {
  @Mutation(() => AuthPayload)
  async login(
    @Args() {email, password}: AuthArgs,
    @Ctx() {prisma}: ContextType
  ) {
    const user = await prisma.user.findUnique({where: {email}});

    if (!user) throw new Error('User not found');
    if (password !== user.password) throw new Error('Password is incorrect');

    const session = await prisma.session.create({
      data: {refreshToken: uid(refreshTokenSize), userId: user.id},
    });
```

```
  const payload: AuthPayload = {
    refreshToken: session.refreshToken,
    accessToken: getAccessToken(user.id),
  };

  return payload;
}

@Mutation(() => AuthPayload)
async register(
  @Args() {email, password}: AuthArgs,
  @Ctx() {prisma}: ContextType
) {
  const findUser = await prisma.user.findUnique({where: {email}});

  if (findUser) {
    throw new Error('User already exists');
  }

  const user = await prisma.user.create({data: {name: '', email, password}});

  const session = await prisma.session.create({
    data: {refreshToken: uid(refreshTokenSize), userId: user.id},
  });

  const payload: AuthPayload = {
    refreshToken: session.refreshToken,
    accessToken: getAccessToken(user.id),
  };

  return payload;
}

@Mutation(() => AuthPayload)
async token(@Args() {refreshToken}: TokenArgs, @Ctx() {prisma}: ContextType) {
  const session = await prisma.session.findUnique({
    where: {refreshToken},
  });

  if (!session) {
    throw new Error('Session not found');
  }

  const payload: AuthPayload = {
    refreshToken,
    accessToken: getAccessToken(session.userId),
  };

  return payload;
}

@Mutation(() => LogoutPayload)
async logout(
  @Args() {refreshToken}: TokenArgs,
  @Ctx() {prisma}: ContextType
) {
  const session = await prisma.session.delete({where: {refreshToken}});

  const payload: LogoutPayload = {
    userId: session.userId,
  };

  return payload;
}

@Query(() => User)
async me(@Ctx() {req, prisma}: ContextType) {
  const token = bearer(req.headers?.['authorization'] ?? '');

  if (!token) {
```

```
            throw new Error('Token is not provided');
          }

          const verified = verify(token, accessTokenSecret) as Token | string;

          if (typeof verified !== 'string') {
            const user = await prisma.user.findUnique({where: {id: verified.userId}});

            if (!user) {
              throw new Error('User not found');
            }

            return user;
          }

          throw new Error('invalid token');
        }
      }

export default AuthResolver;
```

event.resolver.ts

```
import {Args, Ctx, Query, Resolver} from 'type-graphql';
import ContextType from '../types/Context';
import {HasOpenEventsArgs} from '../types/graphql/event.types';

@Resolver()
class EventResolver {
  @Query(() => Boolean)
  async hasOpenEvents(
    @Args() {userId}: HasOpenEventsArgs,
    @Ctx() {prisma}: ContextType
  ) {
    const count = await prisma.event.count({where: {userId, open: true}});

    return !!count;
  }
}

export default EventResolver;
```

auth.types.ts

```
import {ArgsType, Field, ObjectType} from 'type-graphql';

// Args

@ArgsType()
export class AuthArgs {
  @Field(() => String!)
  email: string;

  @Field(() => String!)
  password: string;
}

@ArgsType()
export class TokenArgs {
  @Field(() => String!)
  refreshToken: string;
}

// Objects

@ObjectType()
export class AuthPayload {
  @Field(() => String!)
  accessToken: string;
```

```
  @Field(() => String!)
  refreshToken: string;
}

@ObjectType()
export class LogoutPayload {
  @Field(() => String!)
  userId: string;
}
```

event.types.ts

```
import {ArgsType, Field} from 'type-graphql';

@ArgsType()
export class HasOpenEventsArgs {
  @Field(() => String!)
  userId: string;
}
```

Context.ts

```
import {PrismaClient} from '@prisma/client';
import {ExpressContext} from 'apollo-server-express';

type ContextType = {
  req: ExpressContext['req'];
  prisma: PrismaClient;
};

export default ContextType;
```

Token.ts

```
interface Token {
  userId: string;
}

export default Token;
```

authChecker.ts

```
import {bearer} from '@borderless/parse-authorization';
import {verify} from 'jsonwebtoken';
import {AuthChecker} from 'type-graphql/dist/interfaces/AuthChecker';
import ContextType from '../types/Context';
import {accessTokenSecret} from './env';

const authChecker: AuthChecker<ContextType> = async ({context: {req}}) => {
  const token = bearer(req.headers?.['authorization'] ?? '');

  if (!token) {
    throw new Error('Token is not provided');
  }

  const verified = verify(token, accessTokenSecret);

  return !!verified;
};

export default authChecker;
```

env.ts

```
import {Env} from '@humanwhocodes/env';
```

```
import dotenv from 'dotenv';

dotenv.config();

const env = new Env();

export function validate() {
  env.require('ACCESS_TOKEN_SECRET');
}

export const refreshTokenSize = 15;
export const accessTokenSecret = env.get('ACCESS_TOKEN_SECRET')!;
```

resolversEnhanceMap.ts

```
import {
  ResolversEnhanceMap,
  applyResolversEnhanceMap,
} from '../../generated/type-graphql';
import {Authorized} from 'type-graphql';

const resolversEnhanceMap: ResolversEnhanceMap = {
  Event: {_all: [Authorized()]},
  User: {_all: [Authorized()]},
  Session: {_all: [Authorized()]},
};

applyResolversEnhanceMap(resolversEnhanceMap);
```

token.ts

```
import {Request} from 'express';
import {sign, verify} from 'jsonwebtoken';
import {bearer} from '@borderless/parse-authorization';
import {accessTokenSecret} from './env';
import Token from '../types/Token';

export function getToken(request: Request) {
  const {authorization} = request.headers;

  if (authorization) {
    const token = bearer(authorization);

    if (token) {
      return verify(token, accessTokenSecret) as Token;
    }
  }
}

export function getAccessToken(userId: string) {
  const token: Token = {userId};

  return sign(token, accessTokenSecret, {
    expiresIn: '15m',
  });
}
```

docker-compose-local.yml

```
version: '3.8'

services:
  mongo-setup:
    container_name: mongo-setup
    image: mongo
    restart: on-failure
    networks:
      default:
    volumes:
```

```
    - ./scripts:/scripts
  entrypoint: ['/scripts/setup.sh']
  depends_on:
    - mongo1
    - mongo2
    - mongo3

mongo1:
  hostname: mongo1
  container_name: localmongo1
  image: mongo
  expose:
    - 27017
  ports:
    - 27017:27017
  restart: always
  entrypoint:
    [
      '/usr/bin/mongod',
      '--bind_ip_all',
      '--replSet',
      'rs0',
      '--journal',
      '--dbpath',
      '/data/db',
    ]
  volumes:
    - ~/apps/mongo:/data/db
    - ~/apps/mongo/data1/configdb:/data/configdb

mongo2:
  hostname: mongo2
  container_name: localmongo2
  image: mongo
  expose:
    - 27017
  ports:
    - 27018:27017
  restart: always
  entrypoint:
    [
      '/usr/bin/mongod',
      '--bind_ip_all',
      '--replSet',
      'rs0',
      '--journal',
      '--dbpath',
      '/data/db',
    ]
  volumes:
    - ~/apps/mongo/data2/db:/data/db
    - ~/apps/mongo/data2/configdb:/data/configdb
mongo3:
  hostname: mongo3
  container_name: localmongo3
  image: mongo
  expose:
    - 27017
  ports:
    - 27019:27017
  restart: always
  entrypoint:
    [
      '/usr/bin/mongod',
      '--bind_ip_all',
      '--replSet',
      'rs0',
      '--journal',
      '--dbpath',
      '/data/db',
    ]
```

```
    volumes:
      - ~/apps/mongo/data3/db:/data/db
 - ~/apps/mongo/data3/configdb:/data/configdb
```

package.json

```
{
  "name": "api",
  "version": "1.0.0",
  "main": "index.js",
  "license": "MIT",
  "scripts": {
    "build": "run-s build:*",
    "build:clean": "rm -rf dist",
    "build:prisma": "prisma generate",
    "build:typescript": "tsc",
    "start": "node dist/src",
    "dev": "ts-node-dev -r dotenv/config src/index"
  },
  "devDependencies": {
    "@types/graphql-fields": "^1.3.4",
    "@types/jsonwebtoken": "^8.5.8",
    "@types/node": "^17.0.36",
    "@types/ramda": "^0.28.13",
    "eslint": "^8.16.0",
    "npm-run-all": "^4.1.5",
    "prettier": "^2.6.2",
    "prisma": "3.10.0",
    "ts-node": "^10.8.0",
    "ts-node-dev": "^2.0.0",
    "typegraphql-prisma": "0.19.1",
    "typescript": "^4.7.2"
  },
  "dependencies": {
    "@borderless/parse-authorization": "^1.0.0",
    "@humanwhocodes/env": "^2.2.0",
    "@nexus/schema": "^0.20.1",
    "@prisma/client": "3.10.0",
    "apollo-server": "^3.8.1",
    "class-validator": "^0.13.2",
    "dotenv": "^16.0.1",
    "graphql": "15.3.0",
    "graphql-fields": "^2.0.3",
    "graphql-scalars": "^1.17.0",
    "jsonwebtoken": "^8.5.1",
    "ramda": "^0.28.0",
    "rand-token": "^1.0.1",
    "reflect-metadata": "^0.1.13",
    "type-graphql": "^1.1.1",
    "uuid": "^8.3.2"
  }
}
```

Web-додаток

Index.tsx

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

```
reportWebVitals();
```

App.tsx

```
import './App.css';
import Shared from '../../shared/src/components/Shared';

function App() {
  return (
    <div className='App'>
      <Shared />
    </div>
  );
}

export default App;
```

config-overrides.js

```
const path = require('path');
const {
  override,
  babelInclude,
  addBabelPlugins,
  removeModuleScopePlugin,
  addWebpackModuleRule,
} = require('customize-cra');

module.exports = override(
  // This will remove the CRA plugin that prevents to import modules from outside
the src directory
  removeModuleScopePlugin(),
  // Overwrites the include option for babel loader to include our packages
  babelInclude([path.resolve('src'), path.resolve(__dirname, '../shared/src')]),
  addWebpackModuleRule({
    test: [/(@?react-(navigation|native))*\.(ts|js)x?$/],
    exclude: [/react-native-web/, /\.(native|ios|android)\.(ts|js)x?$/],
    loader: 'babel-loader',
    options: {
      plugins: ['babel-plugin-transform-class-properties'],
      presets: [
        ['@babel/preset-react'],
        ['babel-preset-react-app', {typescript: true, runtime: 'automatic'}],
        [
          require.resolve('babel-preset-react-app/dependencies'),
          {helpers: true},
        ],
      ],
      cacheDirectory: true,
      cacheCompression: false,
    },
  }),
  ...addBabelPlugins(
    'babel-plugin-react-native-web',
    '@babel/plugin-transform-react-jsx'
  )
);
```

package.json

```
{
  "name": "web",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.11.4",
    "@testing-library/react": "^11.1.0",
    "@testing-library/user-event": "^12.1.10",
```

```
    "@types/jest": "^26.0.15",
    "@types/node": "^12.0.0",
    "@types/react-dom": "^17.0.0",
    "customize-cra": "^1.0.0",
    "react-dom": "^17.0.2",
    "react-native-web": "^0.17.5",
    "react-scripts": "4.0.3",
    "typescript": "^4.1.2",
    "use-state-persist": "^0.3.0",
    "web-vitals": "^1.0.1"
  },
  "scripts": {
    "start": "react-app-rewired start",
    "build": "react-app-rewired build && mv build ../spa/build",
    "test": "react-app-rewired test",
    "eject": "react-app-rewired eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "devDependencies": {
    "@babel/preset-react": "^7.16.0",
    "babel-loader": "^8.2.3",
    "babel-plugin-react-native-web": "^0.17.5",
    "babel-preset-react-app": "^10.0.0",
    "customize-cra": "^1.0.0",
    "react-app-rewired": "^2.1.8"
  }
}
```

tsconfig.json

```
{
  "extends": "../base.tsconfig.json",
  "compilerOptions": {
    "target": "es5",
    "lib": ["dom", "dom.iterable", "esnext"],
    "allowJs": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "noFallthroughCasesInSwitch": true,
    "module": "esnext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx"
  },
  "include": ["src"]
}
```

Мобільний додаток:

App.tsx

```
import React from 'react';
import Shared from '../shared/src/components/Shared';

const App = () => <Shared />;

export default App;
```

package.json

```
{
  "name": "app",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "android": "react-native run-android",
    "ios": "react-native run-ios --simulator=\"iPhone 11 Pro\"",
    "start": "react-native start",
    "test": "jest",
    "lint": "eslint . --ext .js,.jsx,.ts,.tsx"
  },
  "dependencies": {
    "@react-native-community/async-storage": "^1.12.1",
    "react": "17.0.2",
    "react-native": "0.66.3",
    "react-native-safe-area-context": "^4.2.5",
    "react-native-screens": "^3.13.1",
    "react-native-use-persist-storage": "^1.0.2"
  },
  "devDependencies": {
    "@babel/core": "^7.12.9",
    "@babel/runtime": "^7.12.5",
    "@react-native-community/eslint-config": "^2.0.0",
    "@types/jest": "^26.0.23",
    "@types/react-native": "^0.66.4",
    "@types/react-test-renderer": "^17.0.1",
    "babel-jest": "^26.6.3",
    "eslint": "^7.14.0",
    "jest": "^26.6.3",
    "metro-react-native-babel-preset": "^0.66.2",
    "react-native-config": "^1.4.5",
    "react-test-renderer": "17.0.2",
    "typescript": "^4.4.4"
  },
  "resolutions": {
    "@types/react": "^17"
  },
  "jest": {
    "preset": "react-native",
    "moduleFileExtensions": [
      "ts",
      "tsx",
      "js",
      "jsx",
      "json",
      "node"
    ]
  }
}
```

tsconfig.ts

```
{
  "compilerOptions": {
    "target": "esnext",
    "module": "commonjs",
    "lib": ["es2017"],
```

```
        "allowJs": true,

        "jsx": "react-native",

        "noEmit": true,

        "isolatedModules": true,

        "strict": true,

        "moduleResolution": "node",

        "allowSyntheticDefaultImports": true,
        "esModuleInterop": true,

        "skipLibCheck": false,
        "resolveJsonModule": true
      },
      "exclude": [
        "node_modules",
        "babel.config.js",
        "metro.config.js",
        "jest.config.js"
      ]
}
```

Спільна частина клієнту:

package.json

```
{
  "name": "shared",
  "version": "1.0.0",
  "main": "index.js",
  "license": "MIT",
  "private": true,
  "scripts": {
    "dev": "tsc -p tsconfig.json --watch",
    "build": "tsc -p tsconfig.json",
    "codegen": "graphql-codegen -r dotenv/config --config codegen.yml"
  },
  "dependencies": {
    "@graphql-codegen/cli": "^2.6.2",
    "@graphql-codegen/typescript": "^2.4.11",
    "@graphql-codegen/typescript-operations": "^2.4.0",
    "@graphql-codegen/typescript-react-apollo": "^3.2.14"
  }
}
```

copdegen.yml

```
overwrite: true
schema:
  - ./schema.gql
documents:
  - 'src/**/*.gql'
generates:
  src/graphql.tsx:
    plugins:
      - 'typescript'
      - 'typescript-operations'
 - 'typescript-react-apollo'
```

tsconfig.json

```
{
  "extends": "../base.tsconfig.json",
  "include": ["src"],
```

```
    "compilerOptions": {
      "outDir": "dist"
    }
}
```

apolloLink.ts

```
import {
  ApolloClient,
  createHttpLink,
  NormalizedCacheObject,
} from '@apollo/client';
import {fromPromise} from '@apollo/client';
import {onError as onErrorLink} from '@apollo/client/link/error';
import {setContext} from '@apollo/client/link/context';
import {TokenMutation, TokenMutationVariables, TokenDocument} from '../graphql';

export const makeHttpLink = (uri: string) => createHttpLink({uri});

export const makeAuthLink = (getAccessToken: () => string | undefined) =>
  setContext(({operationName}, context) => {
    if (
      !['token', 'login', 'register'].some(
        (operation) => operation === operationName
      )
    ) {
      const token = getAccessToken();

      if (token) {
        context.headers = {Authorization: `Bearer ${token}`};
      }
    }

    return context;
  });

async function getObservablePromise(
  client: ApolloClient<NormalizedCacheObject>,
  getRefreshToken: () => string | undefined
) {
  const refreshToken = getRefreshToken();

  if (!refreshToken) {
    throw new Error('Refresh token was not provided');
  }

  const result = await client.mutate<TokenMutation, TokenMutationVariables>({
    mutation: TokenDocument,
    variables: {refreshToken},
  });

  if (!result.data?.token) {
    throw new Error('Failed to obtain new token');
  }

  return result.data.token;
}

export function makeErrorLink(
  client: ApolloClient<NormalizedCacheObject>,
  options: {
    getRefreshToken: () => string | undefined;
    login: (refreshToken: string, accessToken: string) => Promise<void>;
    logout: () => Promise<void>;
    onError?: (error: string) => void;
  }
) {
  return onErrorLink(({graphQLErrors, operation, forward}) => {
    if (graphQLErrors?.some((error) => error.message === 'jwt expired')) {
      const observablePromise = getObservablePromise(
```

```
            client,
            options.getRefreshToken
          )
            .then(async (tokens) => {
              await options.login(tokens.accessToken, tokens.refreshToken);
              return tokens;
            })
            .catch(async (error) => {
              if (error instanceof Error) {
                options.onError?.(error.message);
              }
              await options.logout?.();
            });

          return fromPromise(observablePromise).flatMap((tokens) => {
            if (tokens) {
              operation.setContext((context: Record<string, any>) => {
                context.headers = {Authorization: `Bearer ${tokens.accessToken}`};
                return context;
              });
            }

            return forward(operation);
          });
        }
      });
}
```

createContext.ts

```
import React from 'react';

function createContext<T extends {}>(
  name: string
): readonly [React.Context<T | undefined>, () => T] {
  const context = React.createContext<T | undefined>(undefined);

  const useContext = () => {
    const value = React.useContext(context);

    if (value === undefined) {
      throw new Error(`${name} context provider is missing`);
    }

    return value;
  };

  return [context, useContext];
}

export default createContext;
```

DatasourceContext.ts

```
interface DatasourceContext {
  isAuthorized: boolean;
  login: (accessToken: string, refreshToken: string) => Promise<void>;
  logout: () => Promise<void>;
}

export default DatasourceContext;
```

EventContex.ts

```
import {EventFragment} from '../graphql';

interface EventsContext {
  items: EventFragment[];
  actualEvent: EventFragment | undefined;
```

```
  loading: boolean;
  refresh: () => void;
  onNext: () => void;
}

export default EventsContext;
```

UseTokenRefs.ts

```
import DatasourceContext from './DatasourceContext';

interface UseTokenRefsReturnType {
  loading: boolean;
  isAuthorized: boolean;
  accessTokenRef: React.MutableRefObject<string | undefined>;
  refreshTokenRef: React.MutableRefObject<string | undefined>;
  setTokens: DatasourceContext['login'];
  clearTokens: DatasourceContext['logout'];
}

export default UseTokenRefsReturnType;
```

RootNavigator.ts

```
import React, {useMemo} from 'react';
import {
  createNativeStackNavigator,
  NativeStackNavigationOptions,
} from '@react-navigation/native-stack';
import Routes from '../constants/Routes';
import Home from '../components/Home';
import Login from '../components/Login';
import {useDatasourceContext} from '../contexts/DatasourceContext';
import Settings from '../components/Settings';

const SCREEN_OPTIONS: NativeStackNavigationOptions = {headerShown: false};

export type RootNavigatorParamList = {
  [Routes.Login]: undefined;
  [Routes.Home]: undefined;
  [Routes.Settings]: undefined;
};

const Stack = createNativeStackNavigator<RootNavigatorParamList>();

const RootNavigator: React.FC = () => {
  const {isAuthorized} = useDatasourceContext();

  const Screen = useMemo<React.ReactElement>(
    () =>
      isAuthorized ? (
        <>
          <Stack.Screen name={Routes.Home} component={Home} />
          <Stack.Screen name={Routes.Settings} component={Settings} />
        </>
      ) : (
        <Stack.Screen name={Routes.Login} component={Login} />
      ),
    [isAuthorized]
  );

  return (
    <Stack.Navigator
      initialRouteName={Routes.Login}
      screenOptions={SCREEN_OPTIONS}>
      {Screen}
    </Stack.Navigator>
  );
};
```

```
export default RootNavigator;
```

locale.ts

```
import i18n from 'i18next';
import {initReactI18next} from 'react-i18next';
import en from './resources/en.json';
import ua from './resources/ua.json';

const resources = {
  en: {translation: en},
  ua: {translation: ua},
};

i18n.use(initReactI18next).init({
  compatibilityJSON: 'v3',
  resources,
  lng: 'ua',
  interpolation: {escapeValue: false},
});

export default i18n;
```

en.json

```
{
  "page": {
    "login": "Log in"
  },
  "input": {
    "email": "Email",
    "password": "Password",
    "name": "Name"
  },
  "button": {
    "login": "Sign in",
    "register": "Sign up",
    "createEvent": "Create event",
    "logout": "Logout"
  },
  "text": {
    "actualEvent": "Actual event",
    "closedEvents": "Closed events"
  }
}
```

ua.json

```
{
  "page": {
    "login": "Авторизація"
  },
  "input": {
    "email": "Пошта",
    "password": "Пароль",
    "name": "Ім’я"
  },
  "button": {
    "login": "Вхід",
```

```
    "register": "Реєстрація",
    "createEvent": "Створити подію",
    "logout": "Вихід"
  },
  "text": {
    "actualEvent": "Поточна подія",
    "closedEvents": "Історія подій"
  },
  "Token is not provided": "Невідома помилка",
  "User not found": "Користувача не існує",
  "Password is incorrect": "Неправильний пароль",
  "User already exists": "Користувач вже існує"
}
```

useApolloClient.ts

```
import React, {useEffect, useState} from 'react';
import {ApolloClient, from, InMemoryCache} from '@apollo/client';
import storage from '../constants/storage';
import {persistCache} from 'apollo-cache-persist';
import DatasourceContext from '../types/DatasourceContext';
import {apiUri} from '../constants/env';
import {makeAuthLink, makeErrorLink, makeHttpLink} from '../utils/apolloLink';
import {EventFragment} from '../graphql';

const cache = new InMemoryCache({
  typePolicies: {
    Query: {
      fields: {
        events: {
          keyArgs: ['where'],
          merge: (
            existing: EventFragment[] | undefined,
            incoming: EventFragment[],
            {args}
          ) => (!args?.skip ? incoming : (existing ?? []).concat(incoming)),
        },
      },
    },
  },
});

const client = new ApolloClient({cache});

interface UseApolloClientOptions {
  accessTokenRef: React.MutableRefObject<string | undefined>;
  refreshTokenRef: React.MutableRefObject<string | undefined>;
  setTokens: DatasourceContext['login'];
  clearTokens: DatasourceContext['logout'];
}

const useApolloClient = ({
  accessTokenRef,
  refreshTokenRef,
  setTokens,
  clearTokens,
}: UseApolloClientOptions) => {
  const [isPersisted, setIsPersisted] = useState<boolean>(false);

  useEffect(() => {
    persistCache({cache, storage}).then(() => setIsPersisted(true));
  }, []);

  useEffect(() => {
    client.setLink(
      from([
        makeErrorLink(client, {
          getRefreshToken: () => refreshTokenRef.current,
          login: setTokens,
          logout: clearTokens,
```

```
            onError: console.error,
          }),
          makeAuthLink(() => accessTokenRef.current),
          makeHttpLink(apiUri),
        ])
      );
    }, [setTokens, clearTokens]);

    return isPersisted ? client : undefined;
  };

export default useApolloClient;
```

useMe.ts

```
  import {useMeQuery} from '../graphql';

  const useMe = () => {
    const {data} = useMeQuery({fetchPolicy: 'cache-only'});
    return data?.me;
  };

export default useMe;
```

useTokenRefs.ts

```
  import {useCallback, useRef} from 'react';
  import {usePersistStorage} from 'react-native-use-persist-storage';
  import UseTokenRefsReturnType from '../types/UseTokenRefs';

  const useTokenRefs = (): UseTokenRefsReturnType => {
    const [accessToken, setAccessToken, accessRestored] = usePersistStorage<
      string | undefined
    >('accessToken', undefined);

    const [refreshToken, setRefreshToken, refreshRestored] = usePersistStorage<
      string | undefined
    >('refreshToken', undefined);

    const accessTokenRef = useRef<string | undefined>(accessToken);
    const refreshTokenRef = useRef<string | undefined>(refreshToken);

    accessTokenRef.current = accessToken;
    refreshTokenRef.current = refreshToken;

    const setTokens = useCallback<UseTokenRefsReturnType['setTokens']>(
      async (accessToken, refreshToken) => {
        setAccessToken(accessToken);
        setRefreshToken(refreshToken);
      },
      [setAccessToken, setRefreshToken]
    );

    const clearTokens = useCallback<
      UseTokenRefsReturnType['clearTokens']
    >(async () => {
      setAccessToken(undefined);
      setRefreshToken(undefined);
    }, [setAccessToken, setRefreshToken]);

    return {
      loading: !accessRestored || !refreshRestored,
      isAuthorized: !!accessToken && !!refreshToken,
      accessTokenRef,
      refreshTokenRef,
      setTokens,
      clearTokens,
    };
  };
```

```
export default useTokenRefs;
```

useTokenRefs.web.ts

```
import {useCallback, useRef} from 'react';
import UseTokenRefsReturnType from '../types/UseTokenRefs';
import {useStatePersist} from 'use-state-persist';

const useTokenRefs = (): UseTokenRefsReturnType => {
  const [accessToken, setAccessToken] = useStatePersist<string | undefined>(
    'accessToken'
  );

  const [refreshToken, setRefreshToken] = useStatePersist<string | undefined>(
    'refreshToken'
  );

  const accessTokenRef = useRef<string | undefined>(accessToken);
  const refreshTokenRef = useRef<string | undefined>(refreshToken);

  accessTokenRef.current = accessToken;
  refreshTokenRef.current = refreshToken;

  const setTokens = useCallback<UseTokenRefsReturnType['setTokens']>(
    async (accessToken, refreshToken) => {
      setAccessToken(accessToken);
      setRefreshToken(refreshToken);
    },
    [setAccessToken, setRefreshToken]
  );

  const clearTokens = useCallback<
    UseTokenRefsReturnType['clearTokens']
  >(async () => {
    setAccessToken(undefined);
    setRefreshToken(undefined);
  }, [setAccessToken, setRefreshToken]);

  return {
    loading: false,
    isAuthorized: !!accessToken && !!refreshToken,
    accessTokenRef,
    refreshTokenRef,
    setTokens,
    clearTokens,
  };
};

export default useTokenRefs;
```

DatasourceContex.tsx

```
import {ApolloProvider} from '@apollo/client';
import React, {useEffect} from 'react';
import {MeDocument, MeQuery} from '../graphql';
import useApolloClient from '../hooks/useApolloClient';
import useTokenRefs from '../hooks/useTokenRefs';
import DatasourceContext from '../types/DatasourceContext';
import createContext from '../utils/createContext';

const [Context, useDatasourceContext] =
  createContext<DatasourceContext>('Datasource Context');

export {useDatasourceContext};

export const DatasourceContextProvider: React.FC<
  React.PropsWithChildren<{}>
> = ({children}) => {
  const {isAuthorized, loading, ...tokens} = useTokenRefs();
```

```
  const client = useApolloClient(tokens);

  useEffect(() => {
    if (client && isAuthorized) {
      client.query<MeQuery>({query: MeDocument, fetchPolicy: 'network-only'});
    }
  }, [client, isAuthorized]);

  if (loading || !client) {
    // TODO: add loading state
    return null;
  }

  return (
    <Context.Provider
      value={{
        isAuthorized,
        login: tokens.setTokens,
        logout: tokens.clearTokens,
      }}>
      <ApolloProvider client={client}>{children}</ApolloProvider>
    </Context.Provider>
  );
};
```

EventsContext.tsx

```
import React, {useCallback, useState} from 'react';
import {useActualEventQuery, useEventsQuery} from '../graphql';
import useMe from '../hooks/useMe';
import EventsContext from '../types/EventsContext';
import createContext from '../utils/createContext';

const TAKE = 5;

const [Context, _useEventsContext] =
  createContext<EventsContext>('Events Context');

export const useEventsContext = _useEventsContext;

export const EventsContextProvider: React.FC<React.PropsWithChildren<{}>> = ({
  children,
}) => {
  const [skip, setSkip] = useState<number>(0);

  const user = useMe();

  const {data, loading, refetch} = useEventsQuery({
    variables: {
      userId: user?.id ?? '',
      take: TAKE,
      skip,
    },
    skip: !user,
    fetchPolicy: 'cache-and-network',
    nextFetchPolicy: 'cache-and-network',
  });

  const {
    data: actualEventData,
    loading: actualEventLoadings,
    refetch: refetchActualEvent,
  } = useActualEventQuery({
    variables: {userId: user?.id ?? ''},
    skip: !user,
    fetchPolicy: 'network-only',
  });

  const onNext = useCallback<() => void>(() => {
    if (data) {
```

```
            setSkip(data.events.length);
          }
        }, [data?.events.length, setSkip]);

        const refresh = useCallback<() => void>(() => {
          setSkip(0);
          if (user) {
            refetch({userId: user.id, take: TAKE, skip: 0});
            refetchActualEvent();
          }
        }, [user, refetch, refetchActualEvent, setSkip]);

        return (
          <Context.Provider
            value={{
              items: data?.events ?? [],
              actualEvent: actualEventData?.events?.[0],
              loading: loading || actualEventLoadings,
              onNext,
              refresh,
            }}>
            {children}
          </Context.Provider>
        );
};
```

ThemeContex.ts

```
import React from 'react';
import {Provider as PaperProvider} from 'react-native-paper';
import theme from '../constants/theme';

const ThemeContext: React.FC<React.PropsWithChildren<{}>> = ({children}) => {
  return <PaperProvider theme={theme}>{children}</PaperProvider>;
};

export default ThemeContext;
```

TranslationContex.ts

```
import React from 'react';
import {I18nextProvider} from 'react-i18next';
import i18n from '../locale/index';

const TranslationContext: React.FC<React.PropsWithChildren<{}>> = ({
  children,
}) => {
  return <I18nextProvider i18n={i18n}>{children}</I18nextProvider>;
};

export default TranslationContext;
```

Routes.ts

```
enum Routes {
  Login = 'Login',
  Home = 'Home',
  Settings = 'Settings',
}

export default Routes;
```

theme.ts

```
const theme: ReactNativePaper.Theme = {
  colors: {
    primary: '#2666CF',
    accent: '#2666CF',
    text: '#000',
```

```
        backdrop: '#2666CF',
        disabled: '#2666CF',
        error: '#2666CF',
        notification: '#2666CF',
        onSurface: '#2666CF',
        placeholder: '#d6d6d6',
        surface: '#2666CF',
        background: '#FFF',
      },
      fonts: {
        regular: {fontFamily: 'Roboto'},
        light: {fontFamily: 'Roboto'},
        medium: {fontFamily: 'Roboto'},
        thin: {fontFamily: 'Roboto'},
      },
      animation: {scale: 1},
      roundness: 8,
      mode: 'adaptive',
      dark: false,
    };

export default theme;
```

ActualEvent.tsx

```
    import React from 'react';
    import {useEventsContext} from '../contexts/EventsContext';
    import EmptyCard from './EmptyCard';
    import EventCard from './EventCard';

    const ActualEvent: React.FC = () => {
      const {actualEvent} = useEventsContext();

      return actualEvent ? <EventCard item={actualEvent} actual /> : <EmptyCard />;
    };

export default ActualEvent;
```

EmptyCard.tsx

```
    import {getOperationName} from '@apollo/client/utilities';
    import React, {memo, useCallback} from 'react';
    import {useTranslation} from 'react-i18next';
    import {StyleProp, ViewStyle, View} from 'react-native';
    import {useEventsContext} from '../contexts/EventsContext';
    import {
      ActualEventDocument,
      useCreateEventMutation,
      useHasOpenEventsLazyQuery,
    } from '../graphql';
    import useMe from '../hooks/useMe';
    import Button from './ui/Button';

    const EmptyCard: React.FC = () => {
      const {t} = useTranslation();

      const user = useMe();

      const [hasOpenEvents] = useHasOpenEventsLazyQuery({
        fetchPolicy: 'network-only',
      });

      const {refresh} = useEventsContext();

      const [createEvent, {loading}] = useCreateEventMutation();

      const onPress = useCallback<() => Promise<void>>(async () => {
        if (user) {
          const {data} = await hasOpenEvents({variables: {userId: user.id}});
```

```
        if (!data || data.hasOpenEvents) {
          refresh();
        } else {
          createEvent({
            variables: {userId: user.id},
            refetchQueries: [getOperationName(ActualEventDocument) ?? ''],
          });
        }
      }
    }, [user, refresh, hasOpenEvents, createEvent]);

    return (
      <View style={style}>
        <Button
          title={t('button.createEvent')}
          onPress={onPress}
          disabled={loading}
        />
      </View>
    );
  };

  const style: StyleProp<ViewStyle> = {
    minHeight: 120,
    maxHeight: 120,
    height: 120,
    marginHorizontal: 16,
    marginVertical: 12,
    padding: 16,
    backgroundColor: 'white',
    borderRadius: 12,
    overflow: 'hidden',
    shadowColor: '#000',
    shadowOffset: {width: -2, height: 4},
    shadowOpacity: 0.2,
    shadowRadius: 16,
    elevation: 16,
    justifyContent: 'center',
    alignItems: 'center',
    flex: 1,
  };

export default memo(EmptyCard);
```

EventCard.tsx

```
  import React, {memo, useCallback, useEffect, useState} from 'react';
  import {
    ActualEventDocument,
    EventFragment,
    useDeleteEventMutation,
    useUpdateEventMutation,
  } from '../graphql';
  import {StyleProp, ViewStyle, View, TouchableOpacity} from 'react-native';
  import {Text} from 'react-native-paper';
  import Input from './ui/Input';
  import {format, formatDistanceToNowStrict} from 'date-fns';
  import DeleteIcon from './icons/DeleteIcon';
  import {getOperationName} from '@apollo/client/utilities';
  import StopIcon from './icons/StopIcon';
  import {useEventsContext} from '../contexts/EventsContext';
  import {useTranslation} from 'react-i18next';
  import {uk, enGB} from 'date-fns/locale';

  interface EventCardProps {
    actual?: boolean;
    item: EventFragment | undefined;
  }

  const EventCard: React.FC<EventCardProps> = ({item, actual = false}) => {
```

```
const [distance, setDistance] = useState<string>('');

const {
  i18n: {language},
} = useTranslation();

useEffect(() => {
  const interval =
    actual && item
      ? setInterval(() => {
          setDistance(
            formatDistanceToNowStrict(new Date(item.createdAt), {
              locale: language === 'ua' ? uk : enGB,
            })
          );
        }, 500)
      : undefined;

  return () => {
    if (interval) {
      clearInterval(interval);
    }
  };
}, [setDistance, item?.createdAt, language]);

const [text, setText] = useState<string>(item?.name ?? '');

const {refresh} = useEventsContext();

const [updateEventMutation, {loading: updateLoading}] =
  useUpdateEventMutation({onError: refresh});

const [deleteEventMutation, {loading: deleteLoading}] =
  useDeleteEventMutation({onError: refresh});

const updateName = useCallback<() => void>(() => {
  if (item) {
    updateEventMutation({
      variables: {id: item.id, data: {name: {set: text}}},
    });

    refresh();
  }
}, [item, text, refresh, updateEventMutation]);

const closeEvent = useCallback<() => Promise<void>>(async () => {
  if (item) {
    await updateEventMutation({
      variables: {
        id: item.id,
        data: {open: {set: false}, closedAt: {set: new Date()}},
      },
      refetchQueries: [getOperationName(ActualEventDocument) ?? ''],
    });

    refresh();
  }
}, [item, refresh, updateEventMutation]);

const deleteEvent = useCallback<() => Promise<void>>(async () => {
  if (item) {
    await deleteEventMutation({
      variables: {id: item.id},
      refetchQueries: [getOperationName(ActualEventDocument) ?? ''],
    });

    refresh();
  }
}, [item, refresh, deleteEventMutation]);

useEffect(() => {
```

```
    if (item) {
      setText(item.name);
    }
  }, [item?.name]);

  return item ? (
    <View style={wrapperStyle}>
      <View style={headWrapperStyle}>
        <View style={inputWrapperStyle}>
          <Input
            mode='flat'
            value={text}
            disabled={updateLoading}
            onChange={setText}
            onBlur={updateName}
          />
        </View>

        <View style={verticalSeparatorStyle} />

        <TouchableOpacity onPress={deleteEvent} disabled={deleteLoading}>
          <View style={iconWrapperStyle}>
            <DeleteIcon />
          </View>
        </TouchableOpacity>

        {actual && (
          <>
            <View style={verticalSeparatorStyle} />

            <TouchableOpacity onPress={closeEvent} disabled={updateLoading}>
              <View style={iconWrapperStyle}>
                <StopIcon />
              </View>
            </TouchableOpacity>
          </>
        )}
      </View>
      <View style={horizontalSeparatorStyle} />
      <View style={bodyWrapperStyle}>
        <Text>{item.user.name || item.user.email}</Text>
        <Text>
          {actual
            ? distance
            : format(new Date(item.createdAt), 'HH:mm dd/MM') +
              (item.closedAt
                ? ' - ' + format(new Date(item.closedAt), 'HH:mm dd/MM')
                : '')}
        </Text>
      </View>
    </View>
  ) : (
    <View style={emptyStateStyle} />
  );
};

const emptyStateStyle: StyleProp<ViewStyle> = {height: 120};

const bodyWrapperStyle: StyleProp<ViewStyle> = {
  flex: 1,
  flexDirection: 'row',
  justifyContent: 'space-between',
};

const headWrapperStyle: StyleProp<ViewStyle> = {flex: 1, flexDirection: 'row'};

const iconWrapperStyle: StyleProp<ViewStyle> = {width: 24, height: 24};

const inputWrapperStyle: StyleProp<ViewStyle> = {flexGrow: 1};

const verticalSeparatorStyle: StyleProp<ViewStyle> = {width: 24};
```

```
  const horizontalSeparatorStyle: StyleProp<ViewStyle> = {height: 32};

  const wrapperStyle: StyleProp<ViewStyle> = {
    height: 120,
    minHeight: 120,
    maxHeight: 120,
    marginHorizontal: 16,
    marginVertical: 12,
    padding: 16,
    backgroundColor: 'white',
    borderRadius: 12,
    overflow: 'hidden',
    shadowColor: '#000',
    shadowOffset: {width: -2, height: 4},
    shadowOpacity: 0.2,
    shadowRadius: 16,
    elevation: 16,
  };

export default memo(EventCard);
```

Events.tsx

```
  import React, {useCallback, memo} from 'react';
  import {FlatList, FlatListProps} from 'react-native';
  import {useEventsContext} from '../contexts/EventsContext';
  import {EventFragment} from '../graphql';
  import EventCard from './EventCard';

  const keyExtractor: FlatListProps<EventFragment>['keyExtractor'] = (value) =>
    value.id;

  const Events: React.FC = () => {
    const {items, loading, onNext, refresh} = useEventsContext();

    const renderItem = useCallback<
      NonNullable<FlatListProps<EventFragment>['renderItem']>
    >(({item}) => <EventCard key={item.id} item={item} />, []);

    return (
      <FlatList
        data={items}
        keyExtractor={keyExtractor}
        renderItem={renderItem}
        onEndReached={onNext}
        showsVerticalScrollIndicator={false}
        onRefresh={refresh}
        refreshing={loading}
      />
    );
  };

export default memo(Events);
```

Home.tsx

```
  import React, {memo, useCallback} from 'react';
  import {SafeAreaView} from 'react-native-safe-area-context';
  import Events from './Events';
  import {Text} from 'react-native-paper';
  import {
    StyleProp,
    TextStyle,
    TouchableOpacity,
    View,
    ViewStyle,
  } from 'react-native';
  import {useTranslation} from 'react-i18next';
  import {EventsContextProvider} from '../contexts/EventsContext';
```

```
import ProfileIcon from './icons/ProfileIcon';
import {NativeStackScreenProps} from '@react-navigation/native-stack';
import {RootNavigatorParamList} from '../navigation/RootNavigator';
import Routes from '../constants/Routes';
import ActualEvent from './ActualEvent';
import RefreshButton from './RefreshButton';
import LanguageSwitch from './LanguageSwitch';

type HomeProps = NativeStackScreenProps<RootNavigatorParamList, Routes.Home>;

const Home: React.FC<HomeProps> = ({navigation: {navigate}}) => {
  const {t} = useTranslation();

  const openSettings = useCallback<() => void>(() => {
    navigate(Routes.Settings);
  }, [navigate]);

  return (
    <SafeAreaView style={wrapperStyle}>
      <EventsContextProvider>
        <View
          style={{alignItems: 'flex-end', paddingTop: 16, paddingRight: 16}}>
          <View style={{flexDirection: 'row'}}>
            <RefreshButton />
            <View style={{width: 16}} />
            <TouchableOpacity onPress={openSettings}>
              <View style={{height: 32, width: 32}}>
                <ProfileIcon />
              </View>
            </TouchableOpacity>
            <View style={{width: 16}} />
            <LanguageSwitch />
          </View>
        </View>

        <Text style={textStyle}>{t('text.actualEvent')}</Text>
        <ActualEvent />

        <Text style={textStyle}>{t('text.closedEvents')}</Text>
        <Events />
      </EventsContextProvider>
    </SafeAreaView>
  );
};

const textStyle: StyleProp<TextStyle> = {
  color: '#949494',
  fontSize: 24,
  fontWeight: '500',
  paddingHorizontal: 16,
  paddingVertical: 2,
};

const wrapperStyle: StyleProp<ViewStyle> = {
  flex: 1,
};

export default memo(Home);
```

LanguageSwitch.tsx

```
import React, {memo, useCallback} from 'react';
import {useTranslation} from 'react-i18next';
import {TouchableOpacity, View} from 'react-native';
import {Text} from 'react-native-paper';

const LanguageSwitch: React.FC = () => {
  const {
    i18n: {language, changeLanguage},
  } = useTranslation();
```

```
  const switchLanguage = useCallback(() => {
    changeLanguage(language === 'ua' ? 'en' : 'ua');
  }, [language, changeLanguage]);

  return (
    <TouchableOpacity onPress={switchLanguage}>
      <View style={{width: 32, height: 32}}>
        <Text
          style={{
            fontWeight: '500',
            fontSize: 20,
            color: '#949494',
            lineHeight: 32,
            textTransform: 'uppercase',
          }}>
          {language}
        </Text>
      </View>
    </TouchableOpacity>
  );
};

export default memo(LanguageSwitch);
```

Login.tsx

```
import React, {memo, useCallback, useState} from 'react';
import {StyleProp, View, ViewStyle, Text, TextStyle} from 'react-native';
import {NativeStackScreenProps} from '@react-navigation/native-stack';
import {SafeAreaView} from 'react-native-safe-area-context';
import {useTranslation} from 'react-i18next';
import {RootNavigatorParamList} from '../navigation/RootNavigator';
import Routes from '../constants/Routes';
import Input from './ui/Input';
import Button from './ui/Button';
import {useDatasourceContext} from '../contexts/DatasourceContext';
import {useLoginMutation, useRegisterMutation} from '../graphql';
import {Snackbar} from 'react-native-paper';
import LanguageSwitch from './LanguageSwitch';

type LoginProps = NativeStackScreenProps<RootNavigatorParamList, Routes.Login>;

const Login: React.FC<LoginProps> = ({navigation: {navigate}}) => {
  const [email, setEmail] = useState<string>('');
  const [password, setPassword] = useState<string>('');
  const [snackbarVisible, setSnackbarVisible] = useState<boolean>(false);

  const {t} = useTranslation();

  const {login} = useDatasourceContext();

  const [loginMutation, {error: loginError}] = useLoginMutation({
    onCompleted: (data) => {
      login(data.login.accessToken, data.login.refreshToken);
    },
    onError: () => setSnackbarVisible(true),
  });

  const [registerMutation, {error: registerError}] = useRegisterMutation({
    onCompleted: (data) => {
      login(data.register.accessToken, data.register.refreshToken);
    },
    onError: () => setSnackbarVisible(true),
  });

  const onLoginButtonPress = useCallback<() => void>(() => {
    loginMutation({variables: {email, password}});
  }, [email, password, loginMutation]);

  const onRegisterButtonPress = useCallback<() => void>(() => {
```

```
    registerMutation({variables: {email, password}});
  }, [email, password, navigate]);

  const onSnackbarDismiss = useCallback<() => void>(() => {
    setSnackbarVisible(false);
  }, [setSnackbarVisible]);

  return (
    <SafeAreaView style={{flex: 1, flexDirection: 'column'}}>
      <View style={{alignItems: 'flex-end', padding: 16}}>
        <LanguageSwitch />
      </View>
      <View
        style={{
          flexGrow: 1,
          flex: 1,
          alignItems: 'center',
          justifyContent: 'center',
        }}>
        <View style={inputsWrapperStyle}>
          <Input
            placeholder={t('input.email')}
            keyboardType='email-address'
            value={email}
            onChange={setEmail}
          />
          <View style={inputSeparatorStyle} />
          <Input
            placeholder={t('input.password')}
            secureTextEntry
            value={password}
            onChange={setPassword}
          />
        </View>

        <View style={separatorStyle} />

        <Button title={t('button.login')} onPress={onLoginButtonPress} />

        <View style={separatorStyle} />

        <Button title={t('button.register')} onPress={onRegisterButtonPress} />
      </View>

      <Snackbar
        visible={snackbarVisible}
        duration={5000}
        onDismiss={onSnackbarDismiss}
        wrapperStyle={snackbarWrapperStyle}>
        <Text style={snackbarTextStyle}>
          {loginError?.message || registerError?.message
            ? t(loginError?.message ?? registerError?.message ?? '')
            : ''}
        </Text>
      </Snackbar>
    </SafeAreaView>
  );
};

const snackbarTextStyle: StyleProp<TextStyle> = {
  color: '#fff',
  fontSize: 16,
  textAlign: 'center',
};

const snackbarWrapperStyle: StyleProp<ViewStyle> = {
  alignItems: 'center',
};

const inputsWrapperStyle: StyleProp<ViewStyle> = {
  height: 110,
```

```
    width: 280,
  };

  const inputSeparatorStyle: StyleProp<ViewStyle> = {
    height: 8,
  };

  const separatorStyle: StyleProp<ViewStyle> = {
    height: 16,
  };

export default memo(Login);
```

RefreshButton.tsx

```
  import React, {memo} from 'react';
  import {TouchableOpacity, View} from 'react-native';
  import {useEventsContext} from '../contexts/EventsContext';
  import RefreshIcon from './icons/RefreshIcon';

  const RefreshButton: React.FC = () => {
    const {refresh} = useEventsContext();

    return (
      <TouchableOpacity onPress={refresh}>
        <View style={{height: 32, width: 32}}>
          <RefreshIcon />
        </View>
      </TouchableOpacity>
    );
  };

export default memo(RefreshButton);
```

Settings.tsx

```
  import React, {memo, useCallback, useState} from 'react';
  import {NativeStackScreenProps} from '@react-navigation/native-stack';
  import Routes from '../constants/Routes';
  import {RootNavigatorParamList} from '../navigation/RootNavigator';
  import {TouchableOpacity, View} from 'react-native';
  import {Text} from 'react-native-paper';
  import {useTranslation} from 'react-i18next';
  import CloseIcon from './icons/CloseIcon';
  import {SafeAreaView} from 'react-native-safe-area-context';
  import Input from './ui/Input';
  import {useUpdateUserNameMutation} from '../graphql';
  import useMe from '../hooks/useMe';
  import Button from './ui/Button';
  import {useDatasourceContext} from '../contexts/DatasourceContext';
  import {useApolloClient} from '@apollo/client';

  type SettingsProps = NativeStackScreenProps<
    RootNavigatorParamList,
    Routes.Settings
  >;

  const Settings: React.FC<SettingsProps> = ({navigation: {goBack}}) => {
    const {t} = useTranslation();

    const client = useApolloClient();

    const {logout} = useDatasourceContext();

    const [updateName, {loading: nameLoading}] = useUpdateUserNameMutation();

    const user = useMe();

    const [name, setName] = useState<string>(user?.name ?? '');
```

```
    const captureNameChange = useCallback<() => void>(() => {
      if (user) {
        updateName({variables: {id: user.id, name}});
      }
    }, [user, name, updateName]);

    const onLogoutPress = useCallback<() => Promise<void>>(async () => {
      logout();
      client.clearStore();
    }, [client, logout]);

    return (
      <SafeAreaView style={{flex: 1, padding: 16, justifyContent: 'flex-start'}}>
        <View style={{alignItems: 'flex-end'}}>
          <TouchableOpacity onPress={goBack}>
            <View style={{height: 24, width: 24}}>
              <CloseIcon />
            </View>
          </TouchableOpacity>
        </View>
        {!!user && (
          <>
            <View style={{height: 16}} />
            <Text>{t('input.email')}</Text>
            <Input disabled value={user.email} />
            <View style={{height: 16}} />
            <Text>{t('input.name')}</Text>
            <Input
              disabled={nameLoading}
              value={name}
              onChange={setName}
              onBlur={captureNameChange}
            />
            <View style={{height: 64}} />
            <Button title={t('button.logout')} onPress={onLogoutPress} />
          </>
        )}
      </SafeAreaView>
    );
  };

export default memo(Settings);
```

Shared.tsx

```
  import React, {memo} from 'react';
  import {View, StyleProp, ViewStyle} from 'react-native';
  import {NavigationContainer} from '@react-navigation/native';
  import RootNavigator from '../navigation/RootNavigator';
  import ThemeContext from '../contexts/ThemeContext';
  import TranslationContext from '../contexts/TranslationContext';
  import {DatasourceContextProvider} from '../contexts/DatasourceContext';

  const Shared: React.FC = () => (
    <DatasourceContextProvider>
      <TranslationContext>
        <ThemeContext>
          <View style={wrapperStyle}>
            <NavigationContainer>
              <RootNavigator />
            </NavigationContainer>
          </View>
        </ThemeContext>
      </TranslationContext>
    </DatasourceContextProvider>
  );

  const wrapperStyle: StyleProp<ViewStyle> = {
    flex: 1,
    backgroundColor: '#ecf0f1',
```

```
};

export default memo(Shared);
```

Button.tsx

```
import React, {memo} from 'react';
import {StyleProp, TextStyle} from 'react-native';
import {Button as PaperButton} from 'react-native-paper';

type ButtonProps = {
  title: string;
  disabled?: boolean | undefined;
  onPress: () => void;
};

const Button: React.FC<ButtonProps> = ({title, ...props}) => (
  <PaperButton {...props} mode='contained' labelStyle={labelStyle}>
    {title}
  </PaperButton>
);

const labelStyle: StyleProp<TextStyle> = {
  textTransform: 'none',
};

export default memo(Button);
```

Input.tsx

```
import React, {memo} from 'react';
import {StyleProp, TextInputProps, TextStyle} from 'react-native';
import {TextInput} from 'react-native-paper';

type InputProps = Pick<
  TextInputProps,
  'value' | 'placeholder' | 'secureTextEntry' | 'keyboardType' | 'onBlur'
> & {
  disabled?: boolean;
  mode?: 'outlined' | 'flat';
  onChange?: TextInputProps['onChangeText'];
};

const Input: React.FC<InputProps> = ({
  onChange,
  mode = 'outlined',
  disabled = false,
  ...props
}) => (
  <TextInput
    {...props}
    disabled={disabled}
    style={inputStyle}
    mode={mode}
    autoCapitalize='none'
    onChangeText={onChange}
  />
);

const inputStyle: StyleProp<TextStyle> = {
  backgroundColor: '#fff',
  height: 40,
  textAlign: 'center',
};

export default memo(Input);
```

actualEvent.gql

```
query actualEvent($userId: String!) {
  events(where: {userId: {equals: $userId}, open: {equals: true}}) {
    ...Event
  }
}
```

Events.gql

```
query events($userId: String!, $take: Int = 10, $skip: Int = 0) {
  events(
    where: {userId: {equals: $userId}, open: {equals: false}}
    take: $take
    skip: $skip
    orderBy: {createdAt: desc}
  ) {
    ...Event
  }
}
```

hasOpenEvents.gql

```
query hasOpenEvents($userId: String!) {
  hasOpenEvents(userId: $userId)
}
```

me.gql

```
query me {
  me {
    ...User
  }
}
```

createEvent.gql

```
mutation createEvent($userId: String!) {
  createEvent(data: {user: {connect: {id: $userId}}, open: true, name: ""}) {
    ...Event
  }
}
```

deleteEvent.gql

```
mutation deleteEvent($id: String!) {
  deleteEvent(where: {id: $id}) {
    id
  }
}
```

login.gql

```
mutation login($email: String!, $password: String!) {
  login(email: $email, password: $password) {
    accessToken
    refreshToken
  }
}
```

register.gql

```
mutation register($email: String!, $password: String!) {
  register(email: $email, password: $password) {
    accessToken
```

```
    refreshToken
  }
}
```

token.gql

```
mutation token($refreshToken: String!) {
  token(refreshToken: $refreshToken) {
    accessToken
    refreshToken
  }
}
```

updateEvent.gql

```
mutation updateEvent($id: String!, $data: EventUpdateInput!) {
  updateEvent(where: {id: $id}, data: $data) {
    ...Event
  }
}
```

updateUserName.gql

```
mutation updateUserName($id: String!, $name: String!) {
  updateUser(where: {id: $id}, data: {name: {set: $name}}) {
    ...User
  }
}
```

Event.gql

```
fragment Event on Event {
  id
  name
  open
  createdAt
  closedAt
  user {
    id
    email
    name
  }
}
```

User.gql

```
fragment User on User {
  id
  email
  name
}
```

Корінь системи:

base.tsconfig.json

```
{
  "compilerOptions": {
    "target": "esnext",
    "module": "commonjs",
    "moduleResolution": "Node",
    "lib": [
      "esnext",
      "dom",
      "dom.iterable"
```

```
    ],
    "allowJs": true,
    "allowSyntheticDefaultImports": true,
    "esModuleInterop": true,
    "isolatedModules": true,
    "strict": true,
    "jsx": "react",
    "resolveJsonModule": true,
    "skipLibCheck": true,
    "sourceMap": true
  }
}
```

package.json

```
{
  "name": "time-tracker",
  "version": "1.0.0",
  "main": "index.js",
  "license": "MIT",
  "private": true,
  "scripts": {
    "build": "run-s build:*",
    "build:clean": "rm -rf web/build",
    "build:codegen": "yarn workspace shared codegen",
    "build:shared": "yarn workspace shared build",
    "build:web": "yarn workspace web build",
    "start": "node spa/server.js"
  },
  "workspaces": {
    "packages": [
      "web",
      "app",
      "shared"
    ]
  },
  "dependencies": {
    "@apollo/client": "^3.6.6",
    "@react-navigation/native": "^6.0.10",
    "@react-navigation/native-stack": "^6.6.2",
    "apollo-cache-persist": "^0.1.1",
    "babel-plugin-transform-class-properties": "^6.24.1",
    "date-fns": "^2.28.0",
    "graphql": "^16.5.0",
    "npm-run-all": "^4.1.5",
    "react": "17.0.2",
    "react-native": "0.66.3",
    "react-native-paper": "^4.12.1",
    "react-native-safe-area-context": "^4.2.5",
    "react-native-screens": "^3.13.1",
    "react-native-svg": "^12.3.0",
    "react-native-use-persist-storage": "^1.0.2",
    "react-native-vector-icons": "^9.1.0"
  },
  "devDependencies": {
    "@types/react": "17.0.20",
    "@types/react-native": "0.66.4",
    "i18next": "^21.8.8",
    "react-i18next": "^11.17.0"
  }
}
```

.gitignore

```
# dependencies
**/node_modules/
**/.pnp
**/.pnp.js
```

```
      # env
      **/.env

      # production
      **/build/
      **/dist/

      # yarn
      **/npm-debug.log*
      **/yarn-debug.log*
      **/yarn-error.log*
      **/debug.log*

      # ios
      app/ios/Pods

      # android
      **/build/
      **/.idea
      **/.gradle
      **/local.properties
      *.iml
      **/*.hprof

      # fastlane
      **/fastlane/report.xml
      **/fastlane/Preview.html
      **/fastlane/screenshots

      # vs code
      **/.vscode/

      # buck
      **/buck-out/
      **/\.buckd/
      *.keystore
!debug.keystore
```

ДОДАТОК Б

Відгук керівника економічного розділу на кваліфікаційну роботу бакалавра на тему: «Розробка web-додатку синхронізований трекер часу за допомогою технології JavaScript React» студента групи 121-19ск-1 Сєдих Максима Сергійовича

ДОДАТОК В

Перелік файлів на диску

Ім’я файлу	Опис
Кваліфікаційна робота Сєдих.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Сєдих.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Сєдих.rar	Архів, який містить вихідний код продукту.
Сєдих.ppt	Презентація кваліфікаційної роботи