Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут деревообробних та
комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних технологій
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

другий (магістерський)
(рівень вищої освіти)

на тему: "Розроблення системи тайм-менеджменту засобами технологій .Net та Angular"

Виконав: студент 6 курсу групи КН-61м
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Корчинський А. Г.
(прізвище та ініціали)

Керівник Процах Н. П.
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

Львів – 2021

РЕФЕРАТ

Дипломна робота містить 54 сторінки пояснювальної записки, 19 рисунків, 5 таблиць, 1 додаток, 12 джерел.

В даній роботі були досліджені популярні системи тайм менеджменту, їхні особливості, переваги та недоліки, а також було розроблено архітектуру та програмну реалізацію даної системи.

Програмне забезпечення реалізовано за допомогою TypeScript, Angular, HTML 5, SCSS, ASP.NET Core, Code First, Json Web Token, SignalR.

Ключові слова: TypeScript, Angular, HTML 5, SCSS, ASP.NET Core, Code First, Json Web Token, SignalR, тайм-менеджмент, розробка програмної системи.

ABSTRACT

The thesis contains 54 pages of explanatory note, 19 figures, 5 tables, 1 appendix, 12 used literary sources.

In this work were studied popular time management systems, their features, advantages and disadvantages, as well as were and developed the architecture and software implementation of this system. The software is implemented using TypeScript, Angular, HTML 5, SCSS, ASP.NET Core, Code First, Json Web Token, SignalR, Identity.

Keywords: TypeScript, Angular, HTML 5, SCSS, ASP.NET Core, Code First, Json Web Token, SignalR, Identity, time management, software development.

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно дослідити наявні системи тайм-менеджменту та розробити програмне забезпечення для реалізації даної системи:

- огляд технології Code First у .NET застосунках;
- розробити структуру серверної частини програмного застосунку, який повинен використовувати технології .NET Core та Code First;
- провести аналіз предметної області та наявних програмних систем;
- розробити архітектуру та структуру ПЗ, побудувати UML діаграми для системи;
- розробка системи.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ - програмне забезпечення

JWT - Json Web Token

ВСТУП

Грамотне використання часу – це схильність, яка потрібна всім. Індивід, обізнаний, що таке "time management", ніде не поспішає, і незмінно все робить вчасно, а отже, цілком захищений від нервозностей та стресу. А використовувати закони планування часом можуть всі – від першокласника до куратора міжнародної корпорації. Для цього потрібно всього лиш привести в марафет свій місяць і вкласти всі старання для того, щоб вникнути в керування часом.

Актуальність проблеми. Всі з нас набувають своїх накопичувачів часу, що не тільки не виробляють жодного продуктивного результату, але ще й вираховують із нашої повсякденності час. До таких концентраторів часу слід відносити:

- месенджери;
- пошук в Інтернеті;
- розважальні комп’ютерні програмні системи;
- телебачення.
- непотрібні розмови.
- погані звички.

Предмет дослідження – системи тайм-менеджменту в умовах сучасного суспільства.

Об’єкт дослідження – потреба в розробленні системи управління часом.

Мета роботи – створення інтернет-орієнтованого ПЗ, пристосованого для спрощення управління чиїмось часом. Що дозволить вдосконалити планування часом в управлінських структурах, а також в персональному житті людей.

Для досягнення поставленої мети передбачено розв’язання таких завдань:

- провести аналіз актуальних програм менеджменту власного часу
- розробити структуру та архітектуру застосунку, який повинен використовувати технології .NET Core та Code First
- розробити систему, на основі проведеного порівняльного аналізу продуктів аналогів та створеної архітектури системи
- описати етапи розроблення сервісу, його налагодження та тестування, а також перевірити роботу програми у режимі live
- проаналізувати отримані результати та зробити висновки.

Новизна роботи полягає у тому, що було вперше імплементовано технології SignalR, Code First та Identity для систем менеджменту власного часу.

Практичне значення одержаних результатів. Розроблено веб-застосунок для грамотного контролю власного часу, ПЗ запущено в тестовий режим та протестовано реальними користувачами.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗРОБЛЮВАНОЇ СИСТЕМИ

1.1. Керування власним часом у сучасному суспільстві

Керування часом – це різні види методик призначених для якнайкращої структуризації для досягнення цілей. Керування, перш за все утотожнюється з бізнес процесами чи певною виробничою діяльністю, але цей термін з плином часів набував усе інших значень й у даний період може вживатися в контексті власного повсякденного життя. Багато методів, процесів, технік й інструментів є складовою частиною системи управління часом. Різні способи, механізми, технології та інструментарії є невід'ємною складовою керування часом.

На думку Генадія Захаренка управління часом є розділом міждисциплінарної науки, яка основний свій акцент присвячує методам оптимізації витрат часу. На переконання Геннадія Захаренка, керування часом розглядається підрозділом методологічної науки, яка в основному акцентує увагу на способах диверсифікації керування часу.

Щоб ефективно керувати часом потрібно дотримуватись таких правил:

- щоб виробити дієвий та високоякісний графік, індивідум повинен працювати над ним автономно;
- спрямованість на особистісний дизайн, а не на загальноприняті вимоги;
- трудомісткості часу у різних областях життя повинні контролюватись для виявлення їх напівприхованих ресурсів;
- видозміна світосприйняття для поштовху його в першу чергу на результативність;
- запаси саморозвитку, становлення і результативності розглядаються здійсненними і бездонними, тобто необхідне вирішення є всьоголиш оперативно-тактичним і реальним запитанням.

Всі з нас набувають своїх накопичувачів часу, що не тільки не виробляють жодного продуктивного результату, але ще й вираховують із нашої повсякденності час. До таких концентраторів часу слід відносити:

- месенджери;
- пошук в Інтернеті;
- розважальні комп'ютерні програмні системи;
- телебачення.
- непотрібні розмови.
- погані звички.

Саме для таких поглиначів часу існують системи та методи, щоб їх відловлювати та виключати з робочого процесу. Отже, час, що ми витрачаємо без користі можна поступово зменшувати, а в результаті й повністю виключити з свого життя.

Для подібних концентраторів часу функціонують методології та принципи, для виявлення та прибирання їх з трудового механізму. Звідси, час, який безцільно спалюється, треба потроху відсікати, а зрештою цілковито відрахувати з власниого розпорядку дня.

Недостатність трудового часу зявляється у зв'язку з малоефективним управлінням часу. Отож диференціація справ що потребують першочегового пріорітету і тими, які можна поставити на другий план, є основним у нинішньому співтоваристві та дозволяє досягати більшої результативності.

Дивлячись на вищезазначену тематику, немислимо опустити принцип "Як зробити справи". Головна концепція вищезазначеного принципу це те що щоб добитися більш дієвого регулювання, ніж несвідоме, індивідууму слід звернути увагу на додаткові технології класифікації і додаткові переносники інформації. Діаграма переробки інформації що описує вищенаведену концепцією наведена на рисунку 1.1.

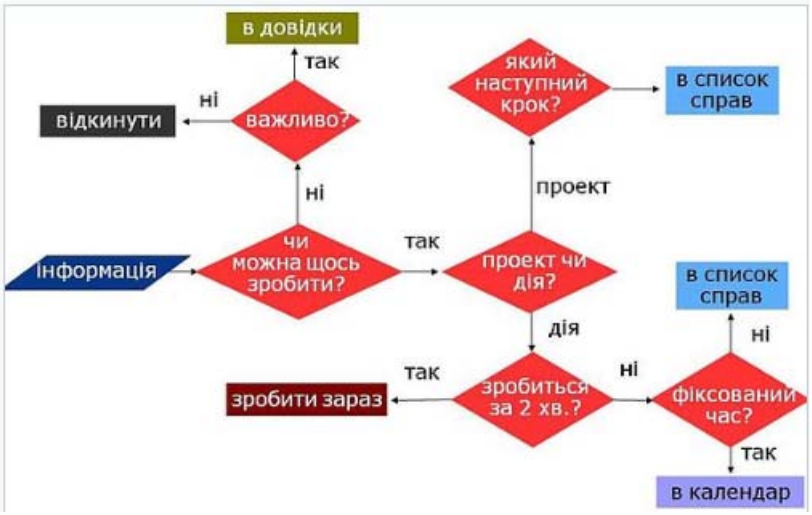
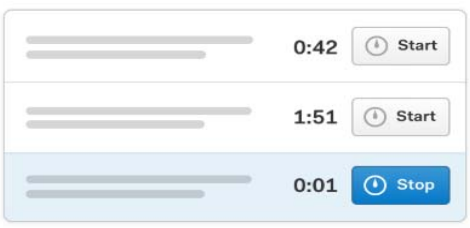


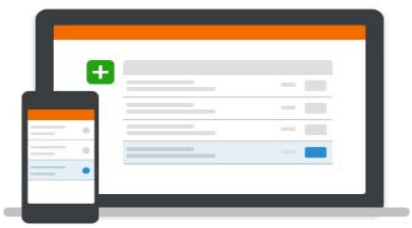
Рис. 1.1. Діаграма опрацювання інформації

Оглядаючи різні статті з дано тематики, мною також була виявлена технологія “Техніка Pomodoro”. Основна ідея методології є в тому що людина повинна розподіляти трудовий і відпочинковий час. Для прикладу можна засікти 25 хвилин реального часу на роботу, після чого повинен йти відпочинок у 5 хвилин. Коли назбирається 4 даних трудових відрізків, тоді ми повинні відпочити більше часу. В науковій термінології вищеописане називають присетами.

1.2. Програмні системи доступні на ринку

Харвест – відоме ПЗ виявлення та аналізу часу або ефективності, що набуває певна організація людей. Вищевказаний товар спрямований насамперед на кураторів бізнеспроцесів чи конструкторських адміністраторів. Є здатність вибудовувати доповіді, простежувати час витрачений на цілі, це робиться банальним набором всім доступним таймерам абоє секундомірам. Дає здатність формувати задуми на декілька кроків вперед. Дане ПЗ має кростплатформенну підтримку.





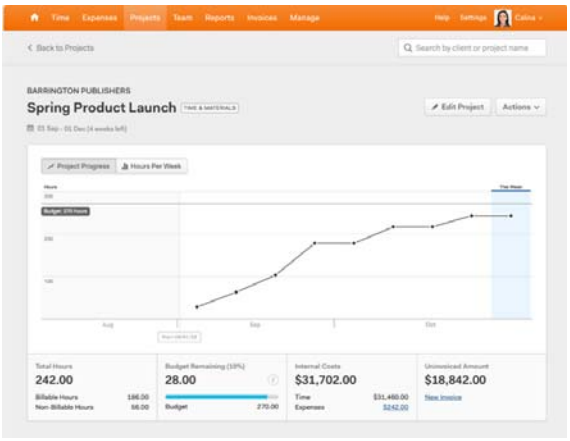


Рис. 1.2. ПЗ - Харвест

Плюси:

- здатність виявлення потраченого часу;
- стеження за витраченим часом відслідковуваної групи;
- здатність продукувати доповіді та діаграми;
- дозволяє прогнозувати витрати ресурсу часу;
- кросплатформенність.

Мінуси:

- спрямованість на кластер бізнес процеців;
- зайвий контент;
- не дозволяє користувачеві працювати з присетами.

Додаток рідного годинника – годинник що може бути як цифровим так і механічним. Він може бути встановлений на будь якому цифровому пристрої, а тому є найбільш доступним й поширеним. Попри найчастіше свій скупий функціонал, все-таки може вважатися системою тайм менеджменту, оскільки за правильним його використанням можна структурувати свій робочий графік.

Рис. 1.2. ПЗ - Додаток рідного годинника

Плюси:

- ПЗ присутнє майже на любому цифровому пристрої;
- при грамотному використанні має що треба аби управляти часом;
- не є комерційним продуктом.

Мінуси:

- відсутність можливості керування багатьма часовими фреймами;
- не дозволяє користувачеві працювати з присетами;
- не надає користувачеві діаграм та інших статистичних даних;
- потребує налагодженої додаткової системи для ефективного управління часом.

Таймер «Зробити справи» – ПЗ є прикладом реалізації методологій розглянутих в попередніх розділах. Основна особливість програмного продукту полягає в можливості грамотного налагодження роботи з присетами. 2015 рік є останнім коли виходили оновлення до даної системи. Є можливість створення нескладних діаграм. Сумісна з тільки з Windows.

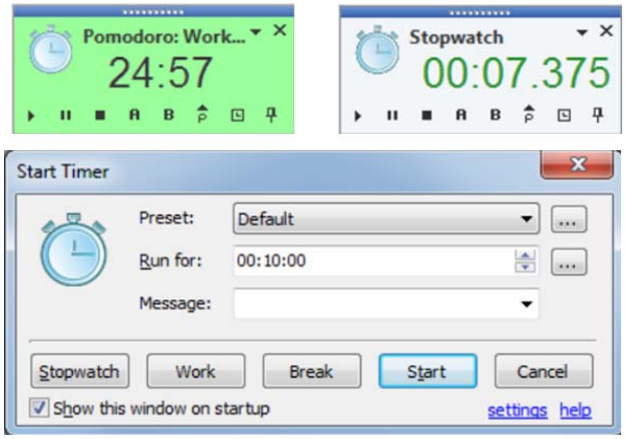


Рис. 1.3. ПЗ - Таймер «Зробити справи»

Переваги:

- наявність основних можливостей відліку часу: секундомір, таймер, будильник.
- можливість створювати нові, або використовувати базові пресети.

- дає змогу створювати прості звіти.
- можливість запускати декілька таймерів, або секундомірів одночасно.

Недоліки:

- орієнтована тільки під систему Windows.
- застарілий дизайн та інтерфейс.
- неможливість генерування графіків та діаграм.

Таблиця 1.1

Порівняльна характеристика

Функціонал	Harvest	Native clock app	Getting Things Done timer
Можливість побудови звітів	+	-	+
Можливість побудови графіків	+	-	-
Секундомір, таймер, будильник	+	+	+
Підтримка функціональності присетів	-	-	+
Кросплатформенність	+	+	-
Орієнтація продукту на загальний ринок	-	+	+
Не надлишковість	-	+	+

Проаналізувавши таблицю можна сказити що розглянути продукти мають свої мінуси. Що сигналізує про відсутність ПЗ, яке б могло надати всі потрібні функції користувачам.

1.3. Висновки до розділу 1

Аналізуючи тайм менеджмент було виявлено декілька технологій за допомогою яких можна полегшити управління часом. За допомогою даних технологій було сформовано список основних правил формування системи керування часом. Детальніше розглянуто такі техніки як “Техніка Pomodoro” і “Як зробити справи”. З розібраних робіт можна констатувати потрібні ПЗ по управлінню часом у сучасному суспільстві та те що такі програмні системи є гостро потрібними в динамічно розвиваючомусь ринку.

На підставі проведеного детального аналізу продуктів аналогів та визначення їх недоліків та переваг, як результат стало зрозуміло, що додатків орієнтованих на управління часом хоч і є багато проте вони мають ті чи інші недоліки. А саме: надлишкова функціональність, відсутність можливості створення чи вибору присетів, залежність від певної операційної системи. Саме тому часто доводиться переплачувати за непотрібний функціонал, або комбінувати декілька програмних систем.

З детального розгляду аналогових систем і визначення їх мінусів і плюсів, було визначено, програмних систем по управлінні часом хоч й існує значна кількість, але вони мають свої мінуси. Серед таких мінусів: надлишковість, робота із наборами, відсутність кросплатформенності. Тому часто доводиться переплачувати за непотрібні функції або комбінувати кілька програмних систем. Ось чому користувачі дуже часто витрачають лишні кошти на непотрібний функціонал або використовують декілька ПЗ одночасно.

Отже, на даний момент в суспільстві є потреба в системі, що полегшувала б управління часом, проте наявні аналоги не задовольняють всіх потреб або є надлишковими, саме це і є причиною створення інтуїтивної та високо ефективної системи менеджменту власного часу.

Отож, в теперішній час у людей існує потребність в ПЗ, яке б полегшувало тайм-менеджмент, але існуючі програми не відповідають всім потребам або є надлишковими, це і є причиною розробки доної дипломної роботи.

РОЗДІЛ 2. ЗАГАЛЬНА ПОСТАНОВКА ЗАВДАННЯ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Завдання

Сформовано завдання по розробці ПЗ, яке надавало б можливість полегшити роботу по тайм-менеджменту як для пересічних громадян так і для працівників у сфері менеджменту. Потрібно розробити систему, яка б мала змогу працювати як з зареєстрованими так і незареєстрованими користувачами. Люди, які ще не зареєструвались в даній системи мають змогу доступитися всього до трьох вкладок: таймер, будильник, секундомір. Зареєстрований ж користувач отримує куда більший набір можливостей. Він може створювати різні набори таймерів та секундомірів(присети), або використовувати вже наявні в системи. Також має доступ до вкладки завдань, на якій може оперувати з багатьма завданнями(по суті окремими іменованими таймерами або секундомірами), що були створенні, виконати вигризку або загрузку завдань. Має мати доступ до вкладки із статистичними даними наведеними за допомогою діаграм. Такому користувачу має відображатись і вкладка історії роботи з системою. Зареєстрований користувач має мати змогу оперувати власними обліковими даними, наприклад для заміни існуючого паролю або видалення своїх облікових даних із системи.

Система також повинна мати змогу підтримувати різні ролі певних користувачів. Так повинен існувати адміністратор, що забезпечував би функції по видаленню облікових даних деяких користувачів, або зворотній зв'язок з ними.

Щоб грамотно реалізувати авторизацію та автентифікацію користувачів повинні використатися найкращі відповідні практики. Для прикладу авторизація користувача через існуючу соціальну мережу.

Одна з основних вимог це підтримка системи всіх можливих девайсів незалежно від їхньої операційної системи.

2.2. Інструментальні засоби та технології розробки

Як основне середовище розробки було обрана Microsoft Visual Studio 2017. Для роботи з фронтендовою частиною програми було обрано середовище Microsoft Visual Code. Основна технологія що буде використовуватись під час розробки - .NET Core. Angular буде використано для розробки клієнтської частини ПЗ. Було вирішено що TypeScript буде найбільш вдалим вибором мови програмування під веб частину ПЗ.

Методології що будуть застосовуватись при безпосередній розробці системи:

- **Angular** – фронтендовий набір бібліотек, що був створений як зусиллями команди розробників Angular, так і звичайними рядовими програмістами бажаючими покращити бібліотеки;
- **ASP.NET Core** – це кросплатформна платформа з відкритим вихідним кодом для створення сучасних хмарних додатків, підключених до Інтернету, таких як веб-програми, програми Інтернету речей та мобільні серверні програми;
- **MS SQL Server** – це система повязана з реляційною базою даних (RDBMS), яка дає можливість користуватись застосунками по роботі з транзакцями, бізнес-аналітикою;
- **Swagger API –** дозволяє генерувати корисну інформацію про ПЗ та формує певний звіт на її основі;
- **Postman –** в основному потрібний для надсилання тестових запитів для перевірки арі системи;
- **LINQ –** бібліотека що реалізує роботу з базою даних, імітуючи роботу запитів;
- **Entity Framework Core** – сукупність бібліотек, які дають доступ до сховищ даних;

- **NUnit, Moq** – бібліотеки основним призначенням яких є допомога у тестуванні програмної системи;
- **SignalR** – фреймворк що забезпечує негайне надання потоків інформації, без додаткового очікування з клієнтської сторони. Основна перевага це надання інструментарію по роботі в реальному часі.
- **JSON Web Token (JWT)** – переважно застосовується для формування та передачі токенів безпеки, чим реалізує надійність передачі даних;
- **Code First –** бібліотека з допомогою якої можна спочатку розробляти сутності, а вже на основі них створювати відповідні таблиці в базі даних.

Також для ефективної реалізації обміну даними та робити з базою даних було вирішено використати патерни програмування Unit of work та Repository.

Щоб забезпечити надійне підключення та роботу з будь якою базою даних обрано патерни “Одиниця роботи” і “Репозиторій”.

На роль розумного помічника в відслідковування якості коду було обрано Resharper.

2.3. Формування структурної моделі ПЗ

Серверний застосунок потрібно поділити на декілька проектів. А також розділити саму реалізацію та тестові проекти. В результаті було сформовано по 4 основні та тестові проекти. Кожен з даних застосунків буде відповідальний за свій набір функціональних можливостей малюнок 2.3.

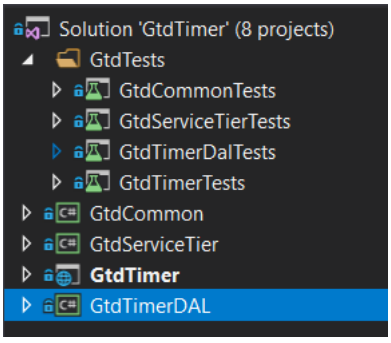


Рис. 2.3. Розподіл серверу

Даний поділ був зумовлений фактором відповідних вимог до грамотного розроблення програмних систем за вибраними нами раніше технологіями. Тобто контролери повинні бути відокремлені від сервісів, а також свої власні проекти повинні мати й класи що реалізують доступ та взаємодію з сховищем даних. Детальний розподіл:

1) Застосунок що містить сутності та реалізує основну взаємодію з сховищем даних.

2) Застосунок покликаний реалізувати можливості api системи.

3) Застосунок що формує всю бізнес логіку проектної системи й забезпечує відповідні обчислення

4) Застосунок в якому повинні міститися загальні елементи що не відносяться до вищеназваних структур

Застосунки для тестування системи були створенні у відповідності до основних застосунків.

Також нище наведу опис спроектованих компонентів для програмної системи:

- Контролер будильників – дана частина системи буде відповідати за всю функціональність по прийманні запитів щодо будильників.
- Контролер авторизації – дана частина системи буде відповідати за всю функціональність по прийманні запитів щодо авторизації або реєстрації.
- Контролер присетів – дана частина системи буде відповідати за всю функціональність по прийманні запитів з наборами(присетами).
- Контролер користувачів – дана частина системи буде відповідати за всю функціональність по прийманні запитів які відповідають за роботк з користувачами ПЗ.
- Контролер завдань – дана частина системи буде відповідати за всю функціональність по прийманні запитів з певними користувацькими завданнями.

2.4. Висновки до розділу 2

На основі розглянутої предметної області було створено відповідні вимоги та цілі по проектуванню та розробці програмної системи, що має мати можливість використовувати основні технології управління часом. Завдяки створеному ПЗ користувач має мати можливість полегшити формування особистого розпорядку дня.

Також зазначу вибір середовищ розробки, а саме для серверної частини застосунку було обрано Microsoft Visual Studio 2017, в свою чергу для веб частини застосунку було обрано Microsoft Visual Code. За роботу з базами даних буде відповідати SQL Server Management Studio 2017.

Під час структуризації системи було сформовано по 4 основні та тестові програми, а також описані деякі компоненти системи.

РОЗДІЛ 3. РОЗРОБЛЕННЯ АРХІТЕКТУРНОЇ МОДЕЛІ І ПРОЕКТУВАННЯ СТРУКТУРНИХ ЕЛЕМЕНТІВ

3.1. Розроблення архітектурної моделі ПЗ

Для виконання поставлених завдань найкраще підійде клієнт-серверна архітектурна модель програмної системи. Великою перевагою саме даною моделі в нашому випадку буде те, що вона дозволяє розробляти і підтримувати мультиплатформенні рішення. Основна ідея даного підходу полягає в існуванні серверної частини застосунку, основна мета якого передавати потрібні дані на клієнтську частину ПЗ. В свою чергу клієнтська частина програмного застосунку відповідає за коректну обробку та відображення отриманої інформації, а також за основні інтерактивні взаємодії з користувачами системи. Як саме виглядає в нашому випадку дана архітектурна модель зображено на малюнку 3.1.

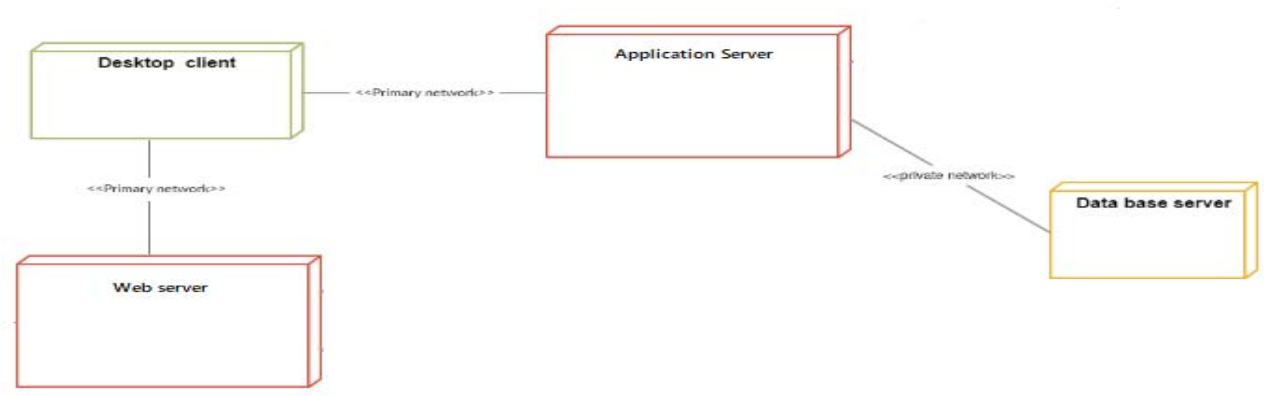


Рис. 3.1. Архітектурна модель

Підтримка мультиплатформенності вимагала вибору відповідного програмного фреймфорку, що забезпечував би можливість розробляти програмні системи заточені під будь які девайси. Саме таким і є обраний .NET Core фреймворк. Дана технологія працює зо допомогою нагет пакетів, що

зумовлює також можливість модулізації застосунку. Відпадає потреба у використанні біблікотеки Web.dll. Враховуючи що технологія є розбудованою на основі незалежних компонент то вона володіє прекрасною розширюваністю.

Особливістю розроблюваної системи є робота з часом, тому створивши діаграму випадків використання, з даною діаграмою можна ознайомитися на малюнку 3.2, й проаналізувавши отримані результати стало очевидно що ПЗ повинно не залежити від стабільності з'єднання з мережею інтернет.

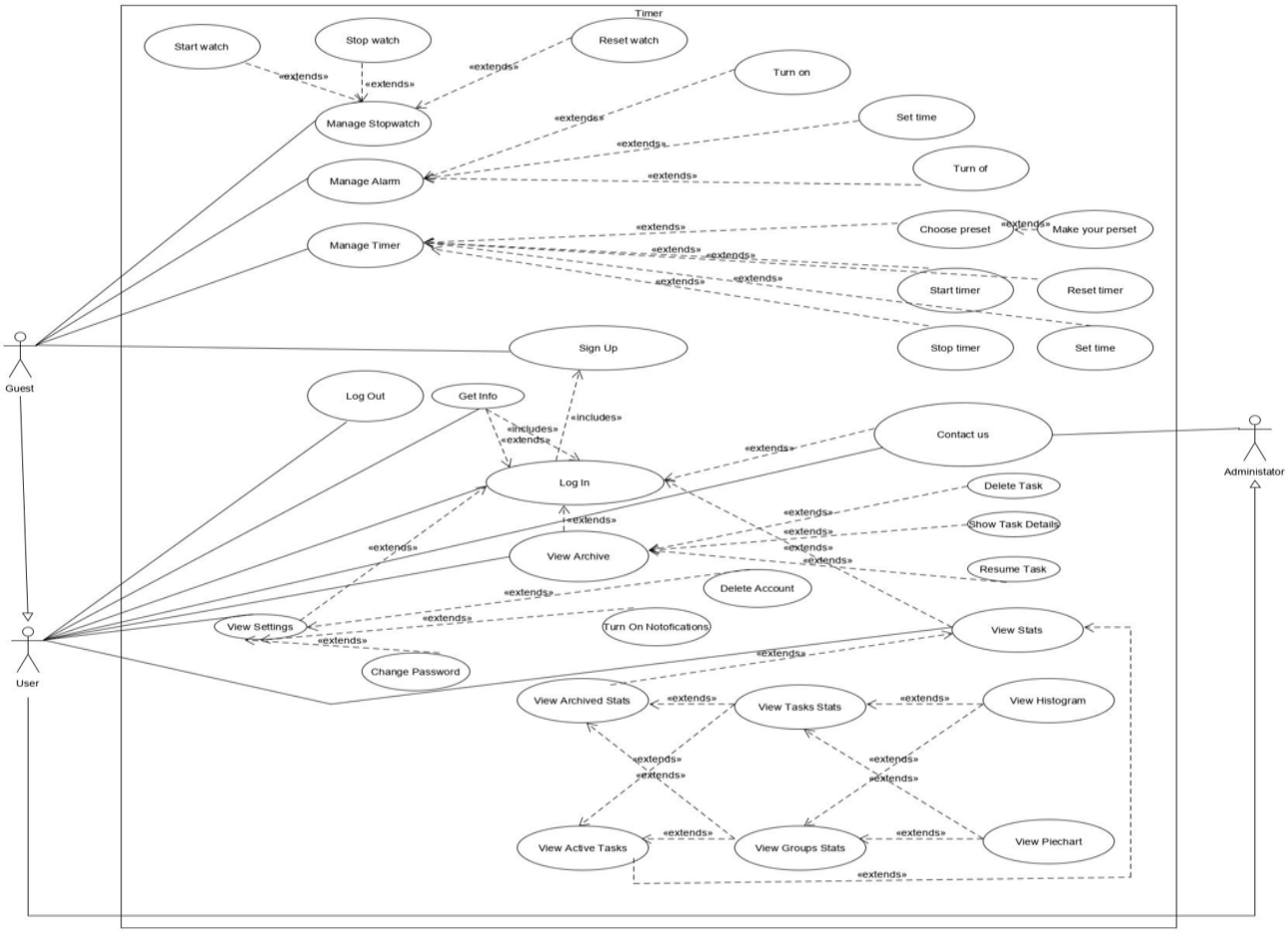


Рис. 3.2. Діаграма випадків використання системи

Саме тому була вибрана технологія Angular, яка є прикладом так званого "single page application". Це означає що більшість інформації вивантажується під час першого відкриття сайту. Далі запити на сервер відправляться тільки при крайній необхідності, в такий спосіб ми можемо переміщатися між

вкладками та запускати таймери/секундоміри не надсилаючи запитів до серверу і не залежачи від якості з’єднання з мережею Інтернет.

Під наведені вище вимоги якнайкраще підходить веб-технологія Angular, що є прикладом односторінкового застосунку. Отже дана технологія дозволить нам вигрузити дані, при першому відкритті сайту, а надалі буде працювати з мінімальними звертаннями до серверної частини. Таким чином оброка часових проміжків не буде залежити від якості інтернету, що могло б сутєво понизити якість нашого продукту.

Для передачі даних було вирішено скористатися REST API, оскільки системні ресурси змінюються не надто часто. Дана технологія реалізує обмін даними за допомогою формату JSON. Саме завдяки цьому швидкість передачі даних є достатньо високою та розмір файлів що передаються є малим. Ця технологія підходить також для використання мобільної мережі Інтернет, що повністю задовольняє нашу умову щодо кросплатформеності.

Щоб обмінюватися інформацією було обрано REST API. Такий вибір зумовлений першочергово низькою частотою зміни системних ресурсів. В цій технології потоки даних передаються як файли JSON. JSON формат забезпечує для нашого програмного продукту високу швидкість передачі. Але не менш вагомим аргументом на користь саме цієї технології є її використання на мобільних пристроях, що персікається з вимогами до нашого ПЗ.

Для забезпечення безпеки даних користувачів було вирішено скористатися технологією JSON Web Token. Дана технологія дає змогу надавати користувачеві токен доступу при авторизації й тільки при наявності такого токену користувач може продовжувати виконувати дії з особистими даними в системі. Детальніше з роботою даної технології можна ознайомитися на рис. 3.3.

В системах орієнтованих на інтернет завжди потрібно надавати велику увагу безпеці. Щоб вберегтися від перехоплення даних користувачів було вирішено скористатися веб токенами JSON малюнок 3.3. Суть даного метоту

полягає у використанні певного токену, що надає можливість користувачеві працювати з системою. Токен надається тільки вразі успішної авторизації.

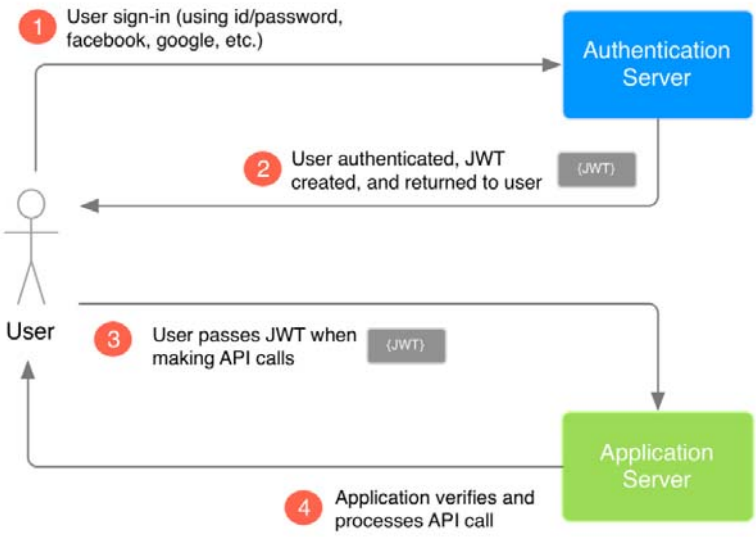


Рис. 3.3. Робота JSON Web Token

Для паралельної роботи з ПЗ на різних вкладках веб браузера, а також на різних девайсах потрібно використати фреймворк SignalR. Цей фреймвор за допомогою вбудованих технологій дозволяє асинхронно працювати з різними повідомленнями що курсують від серверної до клієнтської частини.

Варті уваги також патерни, які було вирішено використати в програмній системі. Такими патернами є Unit Of Work та Repository. Repository це патерн на основі якого створюються класи або компоненти, які інкапсулюють логіку, необхідну для доступу до джерел даних. В такий спосіб відбувається централізація основних функцій доступу до даних, завдяки чому ремонтопридатність стає вищою, та полегшує можливість зміни технології що використовується для доступу до бази даних. Патерн Unit Of Work водночас дає змогу нам оперувати багатьма репозиторіями й дає нам впевненість в тому що всі вони будуть використовувати один й той самий контекст даних.

Не можна не згадати про роботу зі сховищем даних. В нашій системі для коректної роботи було вирішено скористатися патернами програмування одиниця роботи та сховище. Основною ідеєю, яку несуть в собі дані патерни

програмування, є висока ремонтноздатність системи та можливість легкого переходу між різними типами сховищ даних.

3.2 Архітектура сховища даних

Для створення бази даних було вирішено використати технологію Code First, також для керування ролями в системі було вирішено використати технологію Identity. Використання Code First в нашому проекті означає те що спочатку мають бути створені класи доменних сутностей в програмній компоненті системи, а вже на їхній основі згенеруються відповідні сутності та зв’язки в базі даних. Отже, діаграма класів та діаграма бази даних будуть практично однаковими.

База в застосунку буде створена за допомогою фреймворку Code First. Він надає нам такі переваги:

- Легкі класи об'єктів або POCO.
- Більше контролю за класами сутностей, оскільки ви самі їх кодуєте, а не залежно від EF для їх створення. Це означає, що вам не потрібно визначати часткові класи для анотацій даних.
- Опція ніколи не повинна вказувати відображення будь-де. Конвенція приймає конфігурацію.
- DbContext слідує шаблону репозиторію.
- Lazy loading, пов'язаний з ним об'єкт, що завантажує все, що потрібно. Наприклад, модель Post може оголошувати модель автора в коді POCO та EF, спочатку автоматично відображатиме це відношення. Знов-таки, використання конвенції робить нас настільки продуктивними.
- Прекрасно підходить для програм з новими полями.
- Генерація уявлень ASP.NET MVC відмінно працює.
- ModelBinder працює як завжди.

Також для коректної роботи з різними користувачами вирішено застосувати фреймворк Identety.

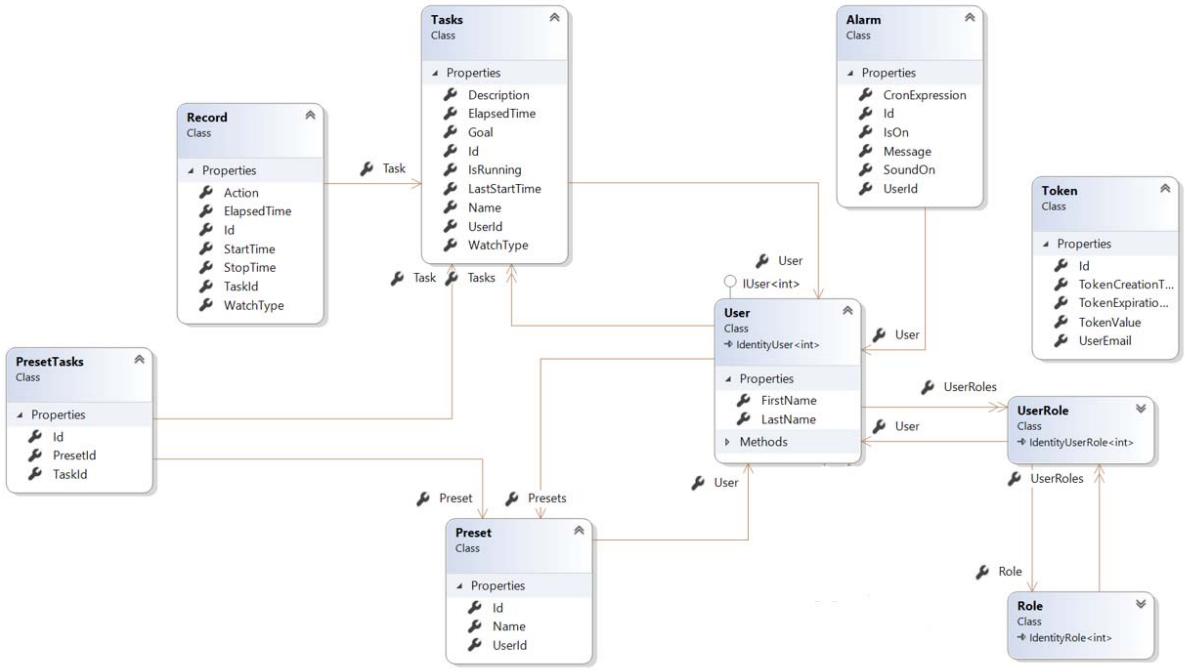


Рис. 3.4. Схема бази даних ПЗ

Детальніше про таблиці:

1) Завдання присету – Дана сутність повинна накопичувати інформацію про відповідні присет та завдання, для індентифікації залежності між ними.

- айді –поле для ідентифікації;
- айді присету – поле для ідентифікації присету;
- айді завдання – поле для ідентифікації завдання;

2) Присет – Дана сутність повинна накопичувати інформацію про присети.

- айді – поле для ідентифікації;
- ім’я – користувацька назва;
- айді користувача – поле для ідентифікації користувача;

3) Запис – Дана сутність повинна накопичувати інформацію про записи в історії.

- дія – зберігає назву проведеної операцію над записом;
- айді – поле для ідентифікації;

- час початку – зберігає час коли запис розпочався;
- час зупинки – зберігає час коли запис був зупинений;
- айді завдання – поле для ідентифікації завдання;

4) Завдання – Дана сутність повинна накопичувати інформацію про завдання.

- опис – містить додаткову інформацію;
- затрачений час – скільки часу було витрачено на завдання;
- ціль – бажаний час виконання;
- айді – поле для ідентифікації;
- чи запущене – дозволяє перевірити статус завдання;
- останній час запуску – коли востаннє стартувалось;
- ім’я – користувацька назва;
- айді користувача – поле для ідентифікації користувача;
- тип годинника – показує якого саме типу є дане завдання.

5) Корстувач – Дана сутність повинна накопичувати інформацію про користувачів. Доповнює поля реалізовані в фреймворку Identity.

- ім’я – ім’я користувача;
- прізвище – прізвище користувача;

6) Будильник – Дана сутність повинна накопичувати інформацію про будильники.

- крон вираз – спеціальний код що визначає коли спрацює будильник;
- айді – поле для ідентифікації;
- чи включени – перевіряє робочість в даний момент;
- повідомлення – додаткова інформація при включенні;
- чи звук включений – інформує про наявність звукового ефекту;
- айді користувача – поле для ідентифікації користувача;

7) Роль користувача – Дана сутність повинна накопичувати інформацію про певну роль даного користувача. Використовує поля імплементовані за допомогою Identety.

8) Роль – Дана сутність повинна накопичувати інформацію про ролі користуваців, що наявні в системі. . Використовує поля імплементовані за допомогою Identety.

9) Токен – Дана сутність повинна накопичувати інформацію про JWT токени.

- айді – поле для ідентифікації;
- час створення токену –коли був створений;
- час виходу токену – коли закінчиться термін користування;
- значення токену – певна унікальна комбінація що передається;
- емейл користувача – визначає до якого користувача буде прив’язано даний токен;

3.3. Розроблення вигляду користувацького інтерфейсу

Враховуючи специфікацію вимог, а також евристичні характеристики було розроблено мокапи програмної системи. Основне завдання даних мокапів – спрощення розроблення користувацького інтерфейсу в майбутньому. Основним засобом навігації в програмній системі було вирішено зробити вкладки, що відображатимуться у верхній частині екрану, реалізацію даної задумки можна побачити на рис. 3.6.

Щоб спростити завдання розробки інтерфейсу програмної системи було вирішено створити мокапи. Мокап– це повнорозмірна модель будь-якого дизайну, що використовується для демонстрації та оцінки стилю ще не випущеного продукту. Щоб переміщатися між основними функціональними можливостями ПЗ розроблено систему вкладок вигляд якою можна побачити на малюнку 3.5.

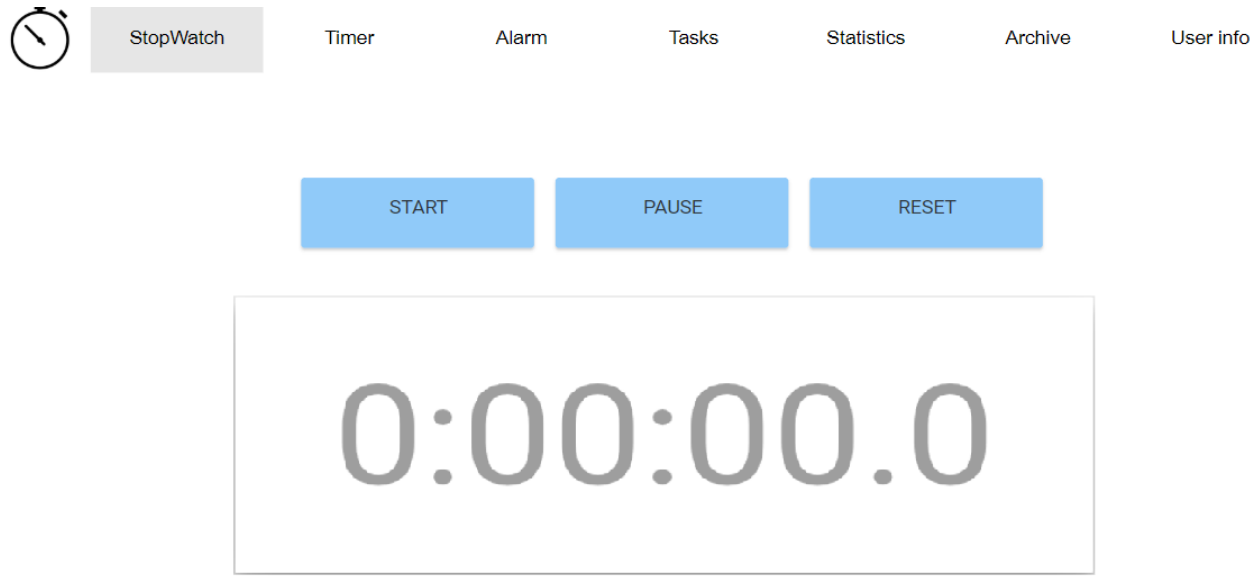


Рис. 3.5. Вигляд системи вкладок

ПЗ повинно інтуїтивно й не надто нагромаджено відображати вкладку завдань. Для цього було розроблено відповідний мокап, який можна побачити на малюнку 3.6.

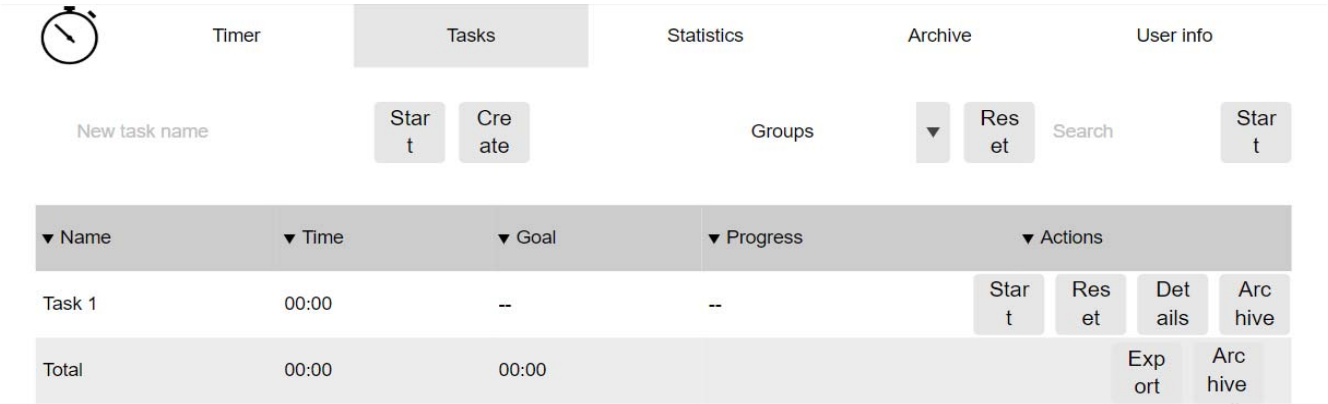


Рис. 3.6. Мокап вкладки завдань

Для ПЗ вибраний мінімалістичний стиль, що зумавлено тим що основне завдання даної системи це управління часом, тобто нічого зайвого не повинне відволікати користува. В даному розділі є надлишковим показувати всі

розроблені мокапи, для детального ознайомлення з ними скористайтеся додатком А.

3.4. Висновки до розділу 3

Оптимальною архітектурною моделю для нашого застосунку є клієнт-серверна модель. Відштовхуючись від даної моделі було вибрано й технології для реалізації системи. ASP.NET Core стала фундаментом для проектування серверу нашого застосунку, так як володіє всіма потрібними нам характеристиками. Angular став фундаментом для проектування веб частини застосунку, впершу чергу через можливість працювати в застосунку при нестабільному інтернет з'єднанні.

Проектуючи базу даних було вирішено скористатися технологією Code First для її створення та подальшої експлуатації. Також було вирішено скористатися технологією Identity для якісного та ефективного керування ролями користувачів у системі. Було розроблено основні домені сутності, які будуть використовуватися в програмному продукті.

Code First став основною технологією для розробки баз даних. Також було розроблено сутності та зв'язки між ними які в подальшому повинні використовуватися для зберігання інформації в сховищах даних. Технологія Identety була вибрана як надійний спосіб реалізації набору користувачів що мають різні ролі в системі.

У висновку відзначимо, що для полегшення розроблення інтерфейсу, а також для розуміння поставлених задач, було розроблено мокапи на основі евристичних характеристик.

Зазначемо, що були створенні мокапи для розробки на їх основі користувацького інтерфейсу.

РОЗДІЛ 4. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1. Процес розроблення ПЗ

Як основне середовище розробки використовувалась Microsoft Visual Studio 2017. Для роботи з фронтендовою частиною програми використовувалось середовище Microsoft Visual Code. Основна технологія що була використана під час розробки - .NET Core. Angular був використаний для розробки клієнтської частини ПЗ.

Першим етапом розробки було створення серверної частини програмного застосунку. Оскільки ми використали Code First, то для початку потрібно було створити класи, що слугують прототипом для таблиць баз даних. При потребі змінити створену базу даних використовувались міграції.

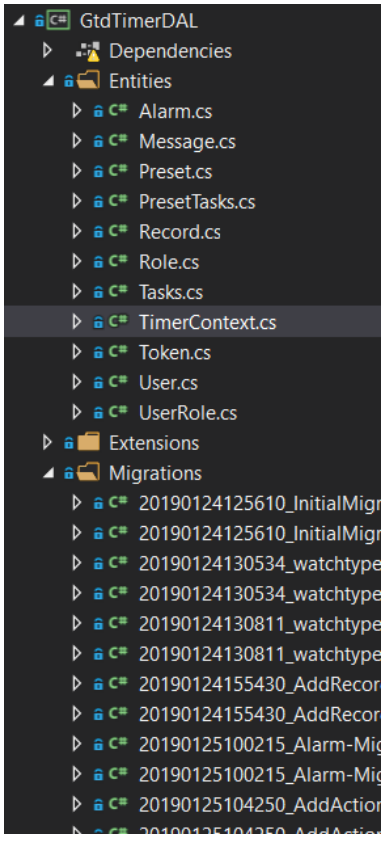


Рис. 4.1. Класи прототипи таблиць та міграції до бази даних

В ході даної фази, відповідно до побудованої раніше, схеми бази даних було розроблено наступні сутності: Alarm, Preset, PresetTask, Record, Role, Task, TimerContext, Token, User, UserRole.

Дивлячись на рисунок 4.1. можна побачити класи що реалізують сутності що були описані ще під час розробки архітектури бази даних.

Наступний крок був у реалізації патернів одиниця роботи та репозиторій. В результаті чого було створено відповідні класи. Також на даному етапі було впроваджено технологію Identety.

Три класи, а також їхні інтерфейси змогли вмістити всю функціональність, яка є повязана з обробкою даних.

Клас репозиторій був використаний щоб обробляти всі сутності, окрім тих що пов’язані з технологією Identity. Щоб добитися токого результати в даному класі було використано generic. Щоб детальніше ознайомитися з реалізацією даного класу можна скористатися додатком Б.

Класи UserStore та ApplicationUserManager реалізують технологію Identity та дозволяє працювати з сутностями що пов’язані з користувачами та їх ролями в програмній системі.

Класи користувацький магазин і менеджер користувацького застосунку працють із сутностями повязаними з технологією Identity. Що в свою чергу означає, що дані класи реалізують функціональність користувачів та їхніх ролей.

Клас та інтерфейс UnitOfWork та IUnitOfWork реалізують патерн програмування Unit Of Work.

Клас одиниця роботи є реалізацією відповідного патерну програмування, який дозволяє працювати з репозиторіями.

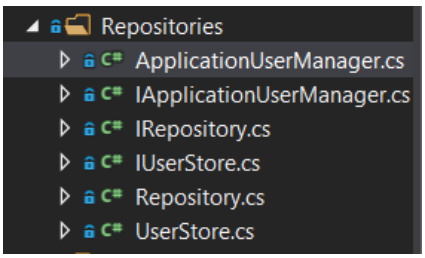


Рис. 4.2. Реалізація взаємодії з сховищем даних.

На наступній стадії було створено такі класи для контролерів:

- Контролер будильників – містить логіку пов’язану з функціональністю будильників;
- Контролер автентифікації – містить логіку що відповідає за реєстрацію або авторизацію користувача в розробленому ПЗ;
- Контролер присетів – містить логіку пов’язану з функціональністю присетів;
- Контролер користувачів – містить логіку пов’язану з функціональністю користувачів ПЗ;
- Контролер завдань – містить логіку пов’язану з функціональністю завдань програмної системи.

Завершальним етапом розроблення серверної частини програмного застосунку стало створення класів та інтерфейсів сервісів, що реалізують логіку роботи. Були створені наступні класи сервісів:

Останнім кроком в побудові сервера було створення класів які б відповідали за розрахунки бізнес логіки ПЗ. Для цього було розроблено такі сервіси:

- Сервіс будильників – містить реалізацію бізнес логіки функціональності будильників у програмої системи;
- Сервіс автентифікації – містить реалізацію бізнес логіки функціональності для авторизації або реєстрації в програмній системі;
- Сервіс присерів – містить реалізацію бізнес логіки функціональності присетів;
- Сервіс користувачів – містить реалізацію бізнес логіки функціональності пов’язаної з користувачами ПЗ;
- Сервіс завдань – містить реалізацію бізнес логіки функціональності пов’язаної з завданнями;

- Сервіс токенів – містить реалізацію бізнес логіки функціональності токенів;
- Сервіс користувачів айдентеті – містить реалізацію бізнес логіки функціональності Identity в ПЗ.

Коли серверна частина програми було розроблена, настав час розробки фронтедної частини. Технологія Angular Material стада основою для побудови вигляду інтерфейсу.

Іконки що використовує Angular Material зберігаються в хмарних сховищах, тому для не потрібно зберігати їхні копії в себе на сервері, вони автоматично будуть стягнуті при підключенні до мережі Інтернет.

Початковим кроком було запуск команди термінала npm install в директорії що відповідає за веб програмної системи. Подібною командою встановлено всі потрібні пакети.

Angular являє собою компонентно орієнтовану технологію розробки. Щоб відповідати цьому всі елементи розроблялися як окремі елементи. Практично кожний такий елемент є складовою частиною більшого елемента.

Структуризація веб рішення було сворено за рахунок відокремлення кожного елемента в свою власну папку. В даній папці для кожного елемента створювалися чотири файли таких розширень:

- .ts – реалізує сам елемент і його основні характеристики.
- .spec.ts – реалізує тести.
- .html – містить розмітку елемента.
- .css – містить інформацію про стилі елемента.

Функції які не є частиною елементу були поміщені в сервіси.

Елемент navigation став основою для всієї функціональності що пов’язана з преміщенням між різними компонентами ПЗ.

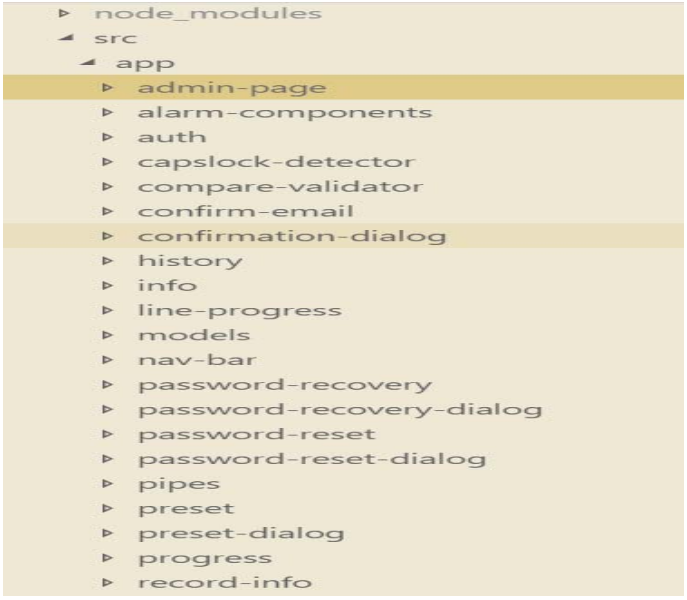


Рис. 4.3. Елементи клієнтської частини ПЗ

Після завершення всіх робіт для паралелізації та роботи на різних пристроях та вкладках одночасно було імплементовано методологію SignalR, а саме у класі хаб завдання. Детальніше з даною реалізацією можна ознайомитись в додатку В.

4.2. Реалізація елементів зовнішнього дизайну

Вкладки таймеру, секундоміра та будильника є спільними для незареєстрованого та зареєстрованого користувача, проте функціональні можливості дещо відрізняються.

В наведених нище малюнках можна побачити що функціонал для незареєстрованих користувачів є неповним, що відповідає вимогам, які були сформовані.

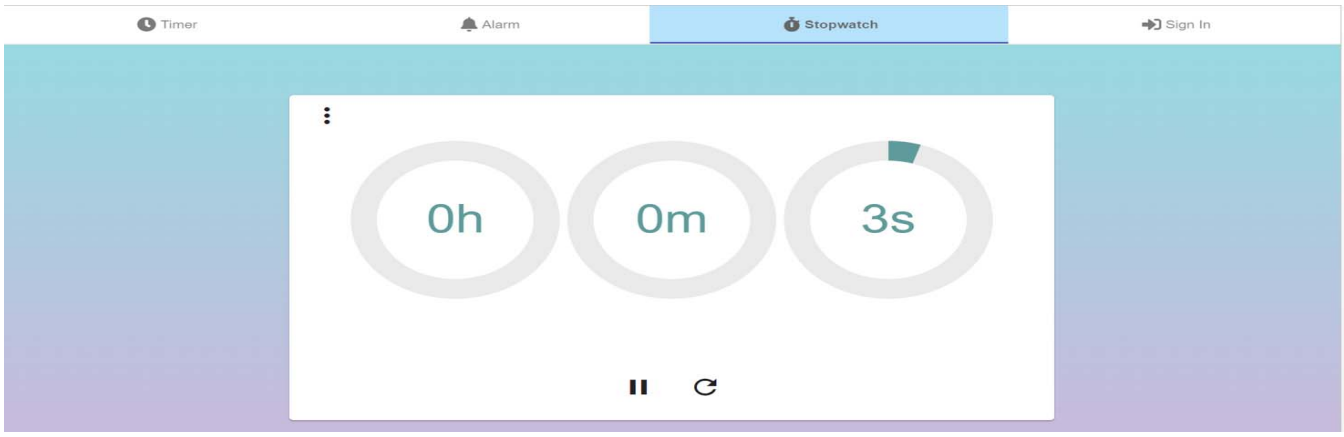


Рис. 4.4. Вкладки ПЗ незареєстрованого користувача

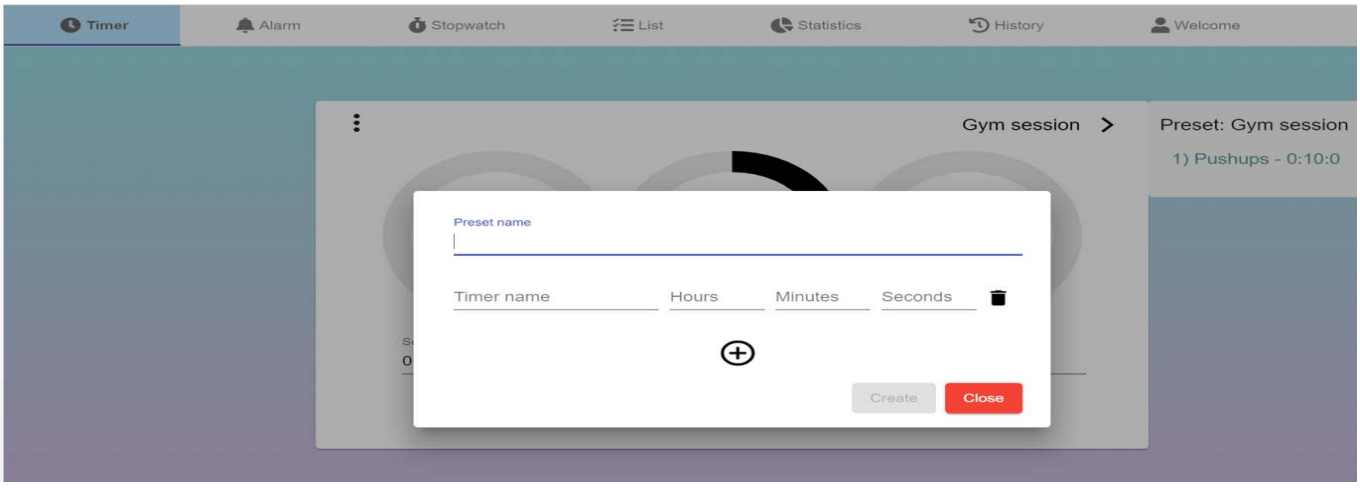


Рис. 4.5. Вкладки ПЗ зареєстрованого користувача

Також створена вкладка завдань. Дана вкладка дозволяє створювати та відслідковувати різні завдання, що реалізуються за допомогою секундоміра, або таймера. Таких завдань може бути надвичайно багато.

Цікавою вкладкою може слугувати статистика в якій відображені діаграми що дозволяють аналізувати пропорційність використання часу на різні завдання.

Щоб отримати детальнішу інформацію про все що відображено на вкладці статистики можна перейти на вкладку історії, де відображаються все що відбувалося із завданнями.

На сторінці налаштувань користувач може працювати з власним обліковим записом.

Окрема сторінка існує для користувачів що мають роль адміністратора. Де вони мають додаткові можливості.

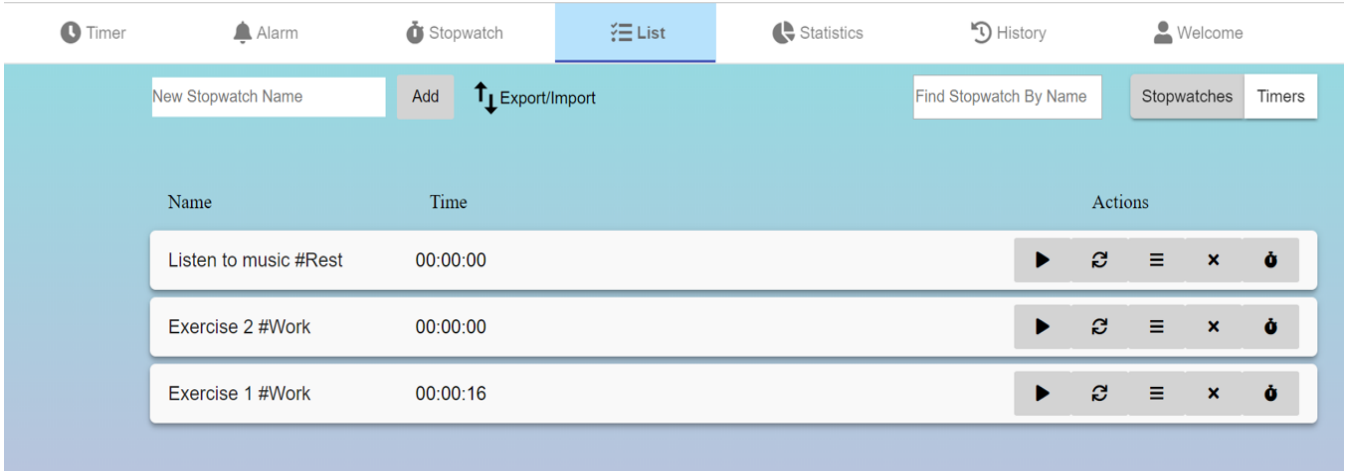


Рис. 4.6. Вигляд вкладки завдань

Рис. 4.7. Вигляд вкладки статистики

Авторизація або автентифікація також відбувається на оремій вкладці.

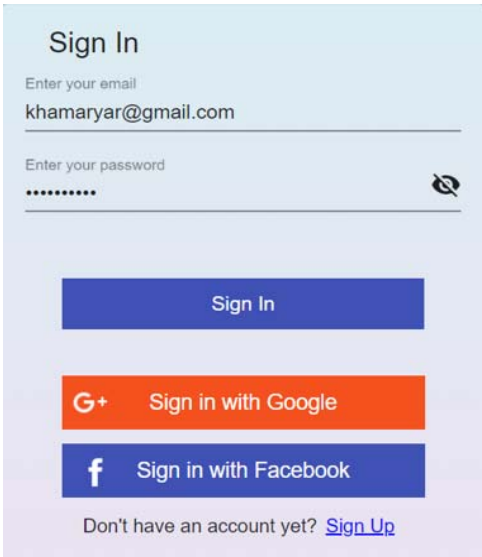


Рис. 4.8. Вигляд процесу автентифікації

На мобільних пристроях ПЗ адаптується та дещо змінює свій вигляд.

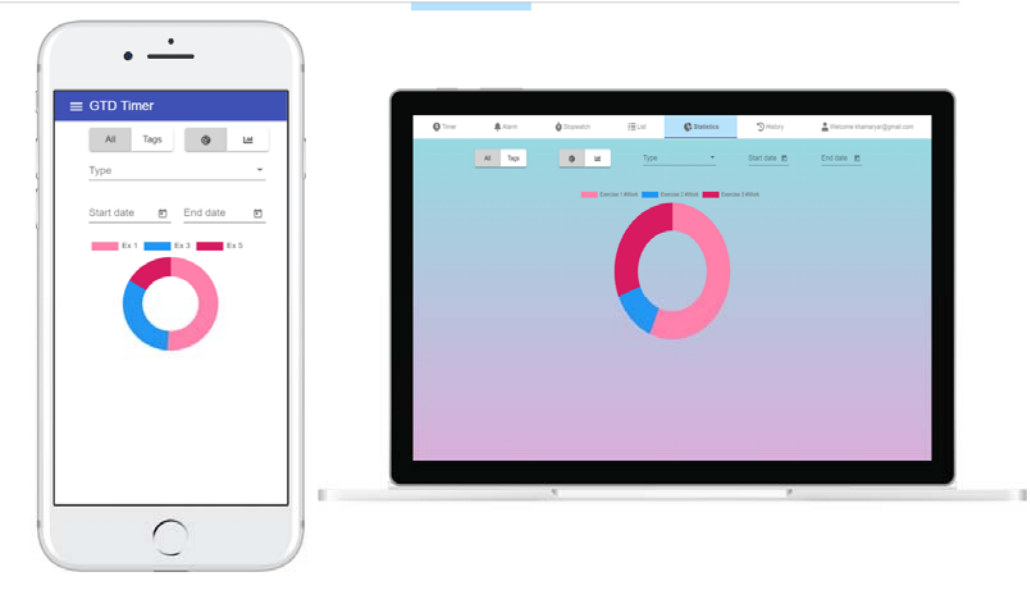


Рис. 4.8. Різниція інтерфейсів на різних дивайсах

4.3. Розробка тестових випадків ПЗ

Для стабільної роботи системи, а також для легшої маштабованості та розширюваності ПЗ потрібно покривати тестами. Спираючись на коректно написані тести можна стверджувати про правильну роботу усіх систем програмного забезпечення.

Такі тести можна запускати багаторазово. Успішне виконання тестів покаже розробнику, що зміни не зламали нічого, що ламати не планувалося.

Тест, що провалився, дозволить виявити, що в коді зроблено зміни, які змінюють або ламають його поведінку. Дослідження помилки, яку видає тест, що провалився, і порівняння очікуваного результату з отриманим дасть можливість зрозуміти, де виникла помилка, будь вона в коді або в вимогах.

NUnit — це платформа модульного тестування для всіх мов .Net. Спершу портована з JUnit, поточна виробнича версія, версія 3, була повністю переписана з багатьма новими функціями та підтримкою широкого спектру платформ .NET. Саме даний фреймворк було вирішено використати при написанні серверних тестових проектів, яких було створено 4, для кожного з основних проектів відповідно.

Функціональне тестування охопило всю функціональність застосунку, що в свою чергу дозволяє судити про правильність роботи програмної системи. Було розроблено 121 тестовий випадок для серверної частини програмного застосунку та 32 тестові випадки для веб-частини програмної системи.

Завдяки функціональним тестам, які змогли охопити всю логіку застосунку, можна стверджувати що ПЗ працює коректно. Всього було створено 121 серверний тест і 32 веб тести.

Postman — це клієнт API, який дозволяє розробникам легко створювати, ділитися, тестувати та документувати API. Саме дана програма використовувалася спочатку для валідації запитів, але пізніше до проекту був також доданий swagger, що ще більше підвищело ефективність перевірки справності запитів.

Як розподілилися тести по різних елементах ПЗ можна побачити в таблиці 4.1.

Таблиця 4.1

Поділ тестів для різних компонент ПЗ

Варіанти використання	**Тестові випадки**
Сервіс будильника	4
Сервіс аутентифікації	4
Сервіс присетів	9
Сервіс завдань	19
Сервіс роботи з токеном	3
Сервіс роботи з користувачами	21
Запити до бази даних	3
Контролер будильника	4
Контролер аутентифікації	4
Контролер присетів	6
Контролер завдань	26
Контролер роботи з користувачами	16
Error Handling Middleware	2
Веб компоненти	32
Загалом	**153**

За допомогою Resharper та lint можна було витрачати менше часу на рутинну, повторювану ручну роботу і замість цього зосередитися на поставленому завданні. Надійний набір функцій для автоматичної перевірки помилок і виправлення коду скорочував час розробки та підвищував ефективність. Дані утиліти реалізовували статичний аналіз коду та підвищували продуктивність розробники і покращували якость коду.

4.4. Висновки до розділу 4

Закінчивши роботу над даним розділом можна стверджувати про успішне дотримання всіх проектних рішень. Розробка відбувалася в запланованих програмних середовищах таких як:

- Microsoft Visual Studio 2017
- Visual Studio Code
- Microsoft SQL Server 2017

Та за допомогою запланованих технологій:

- .Net Core
- Angular

Хочеться відзначити успішне впровадження методології Code First а також Identity.

Інтерфейс користувача був розроблений згідно всіх наявних евристичних правил. Для клієнтської частини застосунку використовувалась бібліотека Angular Material. А сама структура веб частини є елементною й складається з компонентів.

Програма повністю покрита тестами яких в загальному налічується 153.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП ПРОЕКТУ

5.1 Інформаційна карта проекту

Термін стартап означає компанію на перших етапах діяльності. Стартапи засновані одним або кількома підприємцями, які хочуть розробити продукт або послугу, на які, на їхню думку, є попит. Ці компанії зазвичай починають з високих витрат і обмеженого доходу, тому вони шукають капітал з різних джерел, таких як венчурні капіталісти.

Розроблена нами в попередніх розділах програмна система також може вважатися стартапом.

І перш за все перед розголошення про даний стратап було сформовано інформаційну карта проекту.

Таблиця 5.1.

Інформаційна карта проекту.

Назва номінації	Web application (Інтернет-застосунок)
Назва проекту	«GTDTimer»
Назва ВНЗ, факультету, спеціальності	НЛТУ, кафедра інформаційних технологій, комп’ютерні науки
Прізвище, ім’я, по батькові автора	Корчинський Андрій Григорович
Цілі і задачі проекту	• огляд технології Code First у .NET застосунках; • розробити структуру серверної частини програмного застосунку, який повинен використовувати технології .NET Core та Code First; • провести аналіз предметної області та наявних програмних систем; • розробити архітектуру та структуру ПЗ, побудувати UML діаграми для системи;

	• розробка системи;
Короткий зміст проекту	Основна ідея даного проекту це створення інтернет-орієнтовант програмне забезпечення, пристосованого для спрощення управління чиїмось часом. Що дозволить вдосконалити планування часом в управлінських структурах, а також в персональному житті людей.
Терміни виконання проекту	4 місяці
Бюджет проекту	104 935 грн

5.2 Стратегії розвитку проекту

У бізнесі ціна, якість та обслуговування, як і раніше, визначають споживчий вибір. Перш ніж споживачі віддадуть перевагу вашій ідеї (а не іншій компанії), ви повинні переконати їх, чим ваш продукт (послуга) відрізняється і чим він кращий. Якщо ви можете довести, чому ви унікальні, ви залучите клієнтів, які ідеально підходять для вашого бізнесу. Мало кому подобається конкурувати за ціною. Усім хочеться виставити її вище та отримати відповідний дохід. Однак, коли ви тільки починаєте і ще не на 100% упевнені у своїх навичках, встановіть нижчу ціну, ніж у досвідченіших конкурентів, і ви не затримаєтеся в низькому ціновому діапазоні надовго. Зате спочатку низька ціна допоможе вам отримати ваших перших клієнтів і змусить їх привести своїх друзів. Однак варто підкреслити, що якість вашої роботи має бути чудовою. Це важливо, тому що більшість людей підсвідомо очікують, що низька ціна дорівнює низькій якості. Здивуйте їх високою якістю.

Якщо ваш продукт зручний у плані питання доступу або отримання, це дає вам величезну перевагу. Полегшіть та прискоріть процес отримання вашого товару для споживача – і ви зможете вигідно виділитись. Це справді має значення! Зручність добре оплачується, і вона затребувана.

Ми віддаємо перевагу якимсь певним брендам, тому що вони надійніші, менше ламаються і симпатичніші виглядають. Якість та увага до деталей, які ви вкладаєте у свої продукти та послуги, є потужними факторами при ухваленні рішення про покупку з боку споживача. Якщо ваш продукт виглядає або працює краще, ніж у конкурента, його куплять.

Якщо ви можете запропонувати один пиріжок із сотнею начинок, то це спрацює. Так, це просто банальне «розмноження» наповнювачів, а не показник переваги тіста пиріжків, але це ефективно.

Напевно, більшість конкурентів у вашому секторі грішать посереднім обслуговуванням клієнтів. Вони працюють, і навіть обіцяють швидкість, але не терміновість. Мало того, вони пропонують правила повернення або обміну, але ті, які призначені для захисту інтересів компанії, а не клієнтів. І тут вам з'являється відмінна можливість піднятися на рівень вище і забезпечити першокласний сервіс. Якщо ваш конкурент дає 30-денну гарантію повернення грошей, зробіть свою гарантію 60 днів! Якщо вони зазвичай відповідають клієнтам протягом одного робочого дня, відповідайте на будь-які запити того ж дня через чотири години або навіть швидше.

Ці, здавалося б, невеликі хитрощі для вашої унікальної торгової пропозиції негайно відкладаються у пам'яті клієнтів, причому надовго. Крім того, вони не вимагають істотних витрат у реалізації, то чому б до них не вдатися? Саме так на ринку і з'являються справді визначні та проривні компанії.

5.3. Розробка маркетингової моделі стартап проекту

Маркетингові стратегії, що ви починаєте використовувати, коли ваш бізнес тільки починається, визначають зростання вашого бізнесу. Щоб

забезпечити стабільне зростання, вам необхідно створити маркетинговий бюджет та виділити кошти на правильні маркетингові канали, які залучають клієнтів у ваш бізнес та створюють гарну впізнаваність бренду.

Використання правильних маркетингових ідей для стартапів та аналіз маркетингових стратегій інших стартапів може допомогти вашому бізнесу залучити потенційних клієнтів з обмеженим бюджетом.

Також важливо визначити основні переваги вашого проекту над конкурентами.

Таблиця 5.2.

Визначення переваг проекту

№ п/п	Потреба	Вигода проекту	Ключові переваги перед конкурентами
1	Програма що працює на всіх пристроях одинаково добре	Мультиплатформеність з підтримкою будь якого розміру екрану	Кросплатформкність
2	Можливість перегляду історії та створення статистики	Програма автоматично створює графіки даних, які легко аналізувати та надає повну історію взаємодії	Створення діаграм, наявність історії
3	Можливість адміністрування користвачів	Відсіювання користувачів що не використовують, або зловживають системою, для зменшення навантаження на неї	Можливість адміністрування користувачів

Монетизація – це можливість отримувати прибуток із вашого інтернет-ресурсу.

На сьогоднішній день існує багато способів монетизації сайту.

Поширеним шляхом отримання прибутку стала контекстна реклама. Насправді це розміщення посилань на сторонні сайти серед статей, проте текст

реклами цілком або частково відповідає темі статті, де він розміщений. У наш час – це найпрестижніший і, можливо, найбільш легальний вид заробітку в інтернеті.

Саме таким способом ми і скористаємося

5.4. Елементи фінансової моделі стартапу

Фінансове моделювання є важливим інструментом планування стартапу, що дозволяє визначити доцільність запуску проекту та залучення інвестицій, ефективність діяльності компанії, правильність стратегії розвитку. Відсутність економічного макету бізнесу може призвести до негативних наслідків різного ступеня тяжкості — від нерозумних і необґрунтованих витрат до повного провалу ідеї.

Таблиця 5.3

Складові фінансової моделі стартапу

Статті витрат	**Одиниці виміру**	**Фактична кількість, шт.**	**Ціна одиниці, грн.**	**Разом, грн.**
1	**2**	**3**	**4**	**5**
Сировина і матеріали	шт	-	-	-
Купівельні напівфабрикати та комплектуючі вироби	шт	5	-	20 323
Зворотні відходи (вираховуються)	грн	-	-	-
Основна заробітна плата	грн	6	-	49 900
Додаткова заробітна плата	грн	-	-	-
Відрахування на соціальне страхування (ЄСВ 22%)	грн	6	-	10 978
Витрати на утримання й експлуатацію устаткування	грн	-	-	-
Загальновиробничі витрати, у т.ч.:				
- змінні;	грн	4	-	4 870
- постійні;	грн	2	-	1 100
Разом виробничих витрат:	грн	6	-	5 970
Адміністративні витрати	грн	3	-	3 860
Витрати на збут	грн	3	-	13 904
Інші операційні витрати	грн	-	-	-
Разом виробничих і операційних витрат:	грн	32	-	104 935

5.5. Висновки до розділу 5

В ході даного розділу було розроблено стартап нашого проекту. Створено інформаційну карту проекту, розрозбено стратегії розвитку, маркетингову модель та створено елементи фінансової моделі.

ВИСНОВКИ

Отже, внаслідок виконання магістерської кваліфікаційної роботи було розроблено інтернет-орієнтовану систему менеджменту власного часу. Дане ПЗ дозволяє полегшити управління часом в особистому житті, а також в проектному менеджменті.

В результаті проведених робіт у межах даної дипломної роботи створено систему тайм-менеджменту засобами технологій .Net та Angular. Дана система відповідає всім створеним вимогам, та покликана допомагти людям в полегшенні проблем тайм-менеджменту.

Отже, основні заключення що можна винести з даної роботи:

- Проблема керування часом є як ніколи актуальною в нашому суспільстві, що в свою чергу й спонукало до написання даної кваліфікаційної роботи. Проте провівши досконалий аналіз наявних на ринку програмних систем було з’ясовано, що всі вони в тій чи іншій мірі є недосконалими, а отже нами було створено та записано основні завдання та вимоги до програмної системи, яка б надала вичерпну функціональність роботи з керуванням власного часу для користувачів.
- Сформовано завдання з створення кросплатформенного ПЗ, яке повинно реалізовувати можливість для користувача використовувати основні методології тайм менеджменту. Щоб реалізувати задумку було вибрано середовища розробки Microsoft Visual Studio 2017, Microsoft Visual Code, SQL Server Management Studio 2017. В розробзці вирішено використовувати такі технології: Angular, ASP.NET Core, SignalR. Була створена специфікація вимог. На основі якої й проектувалося та створювалося ПЗ.
- Було вирішено, що як основне середовище розробки буле використовуватися Microsoft Visual Studio 2017. Для роботи з фронтендовою частиною програми буде використовуватися середовище Microsoft Visual Code. Основна технологія що буде використовуватись під час розробки - .NET Core. Angular буде використано для розробки клієнтської частини ПЗ.

- Була вибрана клієнт-серверна архітектура. ASP.NET Core стала основною технологією розробки серверу. Фронтент було вирішено створювати за допомогою технології Angular. Вибрано реляційну базу даних для роботи з інформацією, а саме Microsoft SQL server. Були використанні найкращі практики щодо побудови програмних систем на базі .Net Core, а також патерни програмування.
- Розроблено основні компоненти та структурні елементи ПЗ. Під час роботи постійно створювались та оновлювались тести для системи. Усі тести пройшли успішно. Розроблення ПЗ пройшла успішно. Під час розробки компонентів і структурних елементів програмної системи постійно відбувалося написання тестів та їх оновлення, що дає можливість стверджувати успішність етапу розробки ПЗ, оскільки всі тести пройшли успішно.
- Створено модель стартапу та розроблено основні методи маркетинну та стратегій, а також фіноносової моделі, для успішного втілення даного стартаму в життя.

З цього всього вважаю дану роботу закінченою успішно адже система розроблена та є конткурентно спроможною на ринку

СПИСОК ЛІТЕРАТУРИ

1. Google Play [Електронний ресурс]: – Режим доступу: https://play.google.com. (2020)
2. Lucier K. Time Management Systems – and How to Use Them. [Електронний ресурс]: – Режим доступу: http://collegelife.about.com/od/TimeManagement/a/Time-ManagementSystems-And-How-To-Use-Them.htm. (2020)
3. SAE J1939 [Електронний ресурс]: – Режим доступу: Електронні дані з Вікіпедії — вільна енциклопедії. – Посилання: https://en.wikipedia.org/wiki/SAE_J1939. (2020)
4. Архангельский Г. А. Корпоративный тайм-менеджмент: Энциклопедия решений / Г. А. Архангельский. – М. : Альпина Бизнес Букс, 2008. – 160 с.
5. Власов П. К. Психология менеджмента / Власов П. К., Липницкий А. В., Лущихина И. М. и др.; под ред. Никифорова Г. С. — [3-е изд.]. — Х. : Гуманит. центр, 2007. — 510 с.
6. Г. А. Архангельский. Тайм-менеджмент. Полный курс : учебное пособие / С. В. Бехтерев, 2012. – 311 с.
7. Гаврилюк А. М. Тайм-менеджмент як складова успішної комунікативної взаємодії в індустрії туризму України А. М. Гаврилюк, Х. В. Плецан // Ефективна економіка. – 2016. – № 1. [Електронний ресурс]: – Режим доступу: http://www.economy.nayka.com.ua/pdf/1_2016/14.pdf. (2020)
8. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектноориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2007. – 440 с.
9. Історія тайм менеджменту [Електронний ресурс]: – Режим доступу:: https://upravlenievremenem.com/scheho-nashalos/. (2020)
10. Конноли Т., Бегг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика. : Пер. с англ. – М.: Изд. Дом «Вильямс», 2010.

11. Коротко про тайм менеджмент [Електронний ресурс]: – Режим доступу: https://www.omatic-hacker.cc/ua//2010. (2020)
12. Куликова В. Н. Заставьте время работать на вас / В. Н. Куликова. – М. : ЗАО Центрполиграф, 2008. – 192 с.
13. Мандрикова Е. Ю. Разработка опросника самоорганизации деятельности (ОСД). Психологическая диагностика, 2010. – № 2. – С. 87–111.
14. Трейси Б. Результативный тайм-менеджмент: эффективная методика управления собственным временем / Брайан Трейси; [пер. с англ. А. Евтеева]. – М. : СмартБук, 2007. – 79 с.
15. Управління часом [Електронний ресурс]: – Режим доступу: https://www.timesever.com/articles/a1760.php. (2020)
16. Корчинський А., Процах Н.П. Розроблення системи тайм-менеджменту засобами технологій .Net та Angular // Комп'ютерне моделювання та інформаційні технології: матеріали третьої науково-практичної конференції студентів, аспірантів та молодих вчених (Львів, 14-16 жовтня 2021 р.). – Львів: кафедра інформаційних технологій НЛТУ України, 2021. – С. 61-63

Додаток А

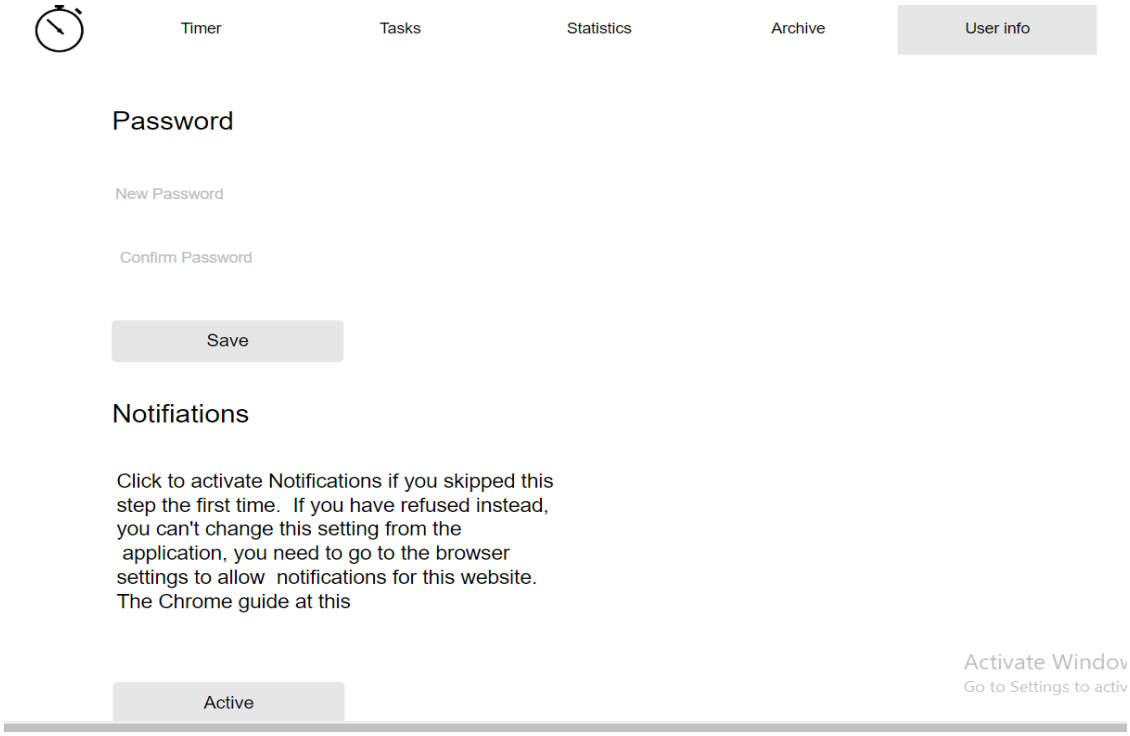


Рис. А.1. Сторінка з налаштуваннями

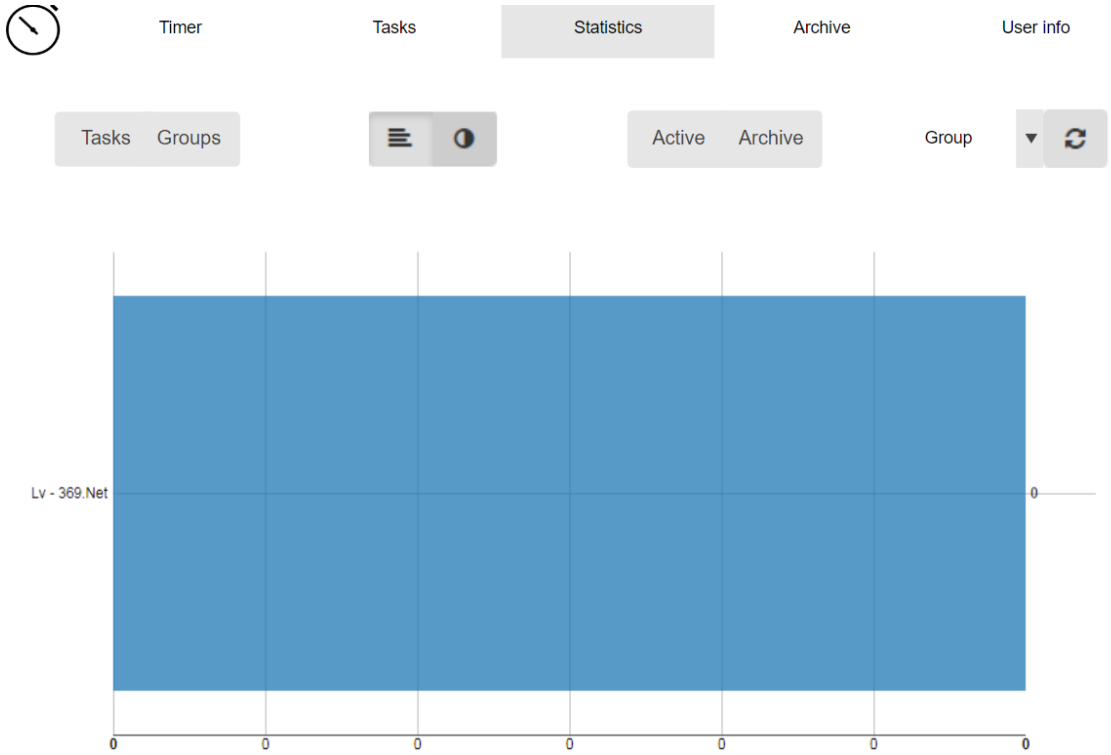


Рис. А.2. Вкладка з статистикою

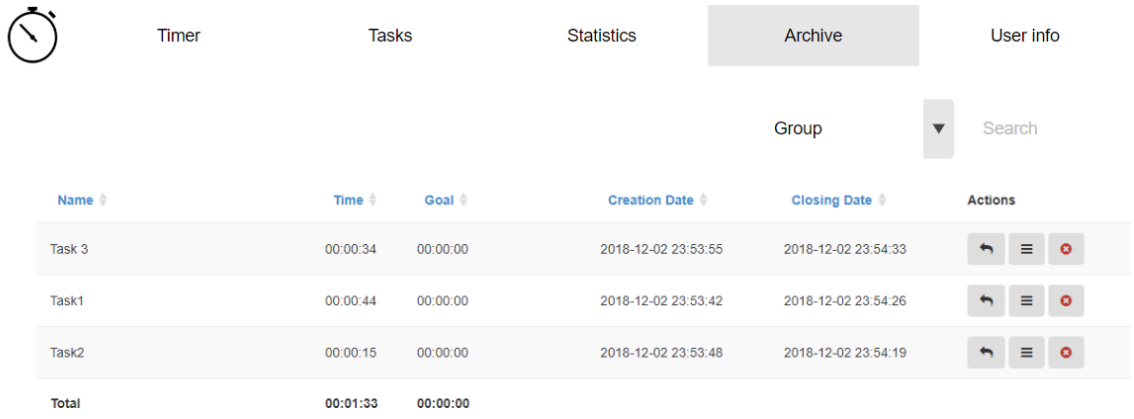


Рис. А.3. Вкладка з архівом

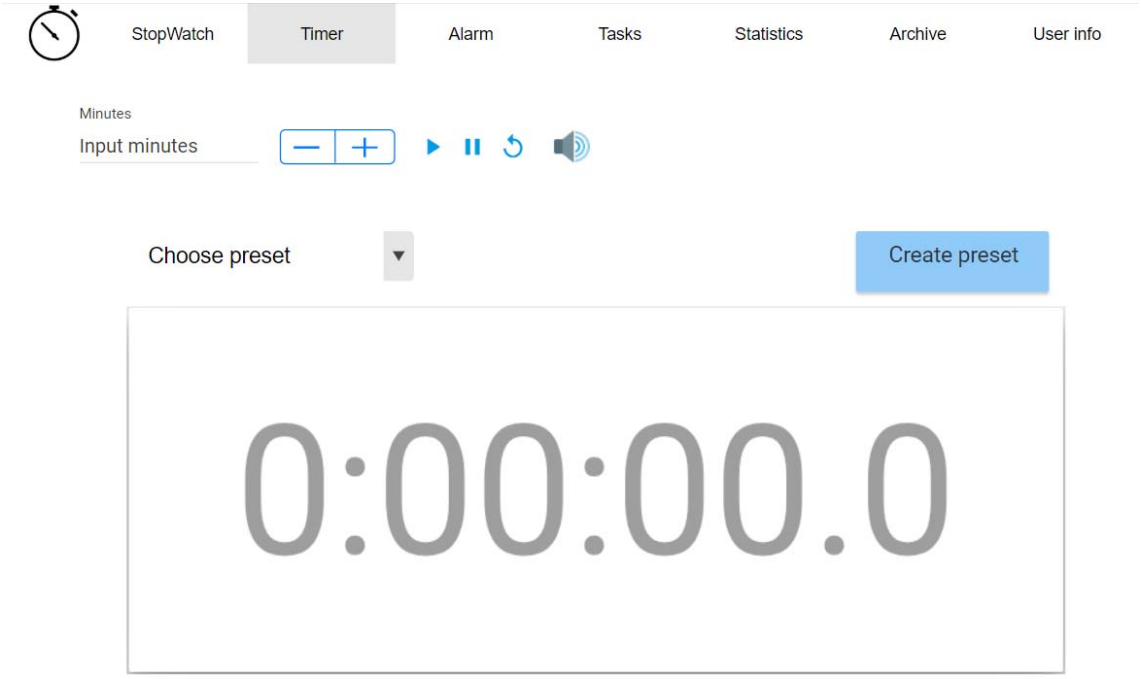


Рис. А.4. Вкладка з таймером

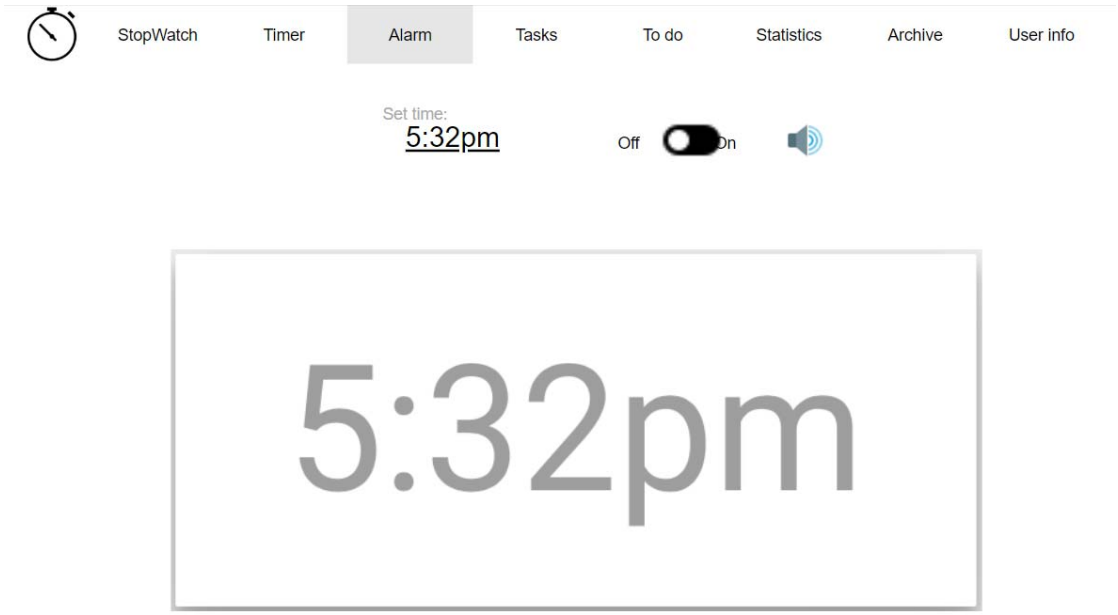


Рис. А.5. Вкладка з будильником

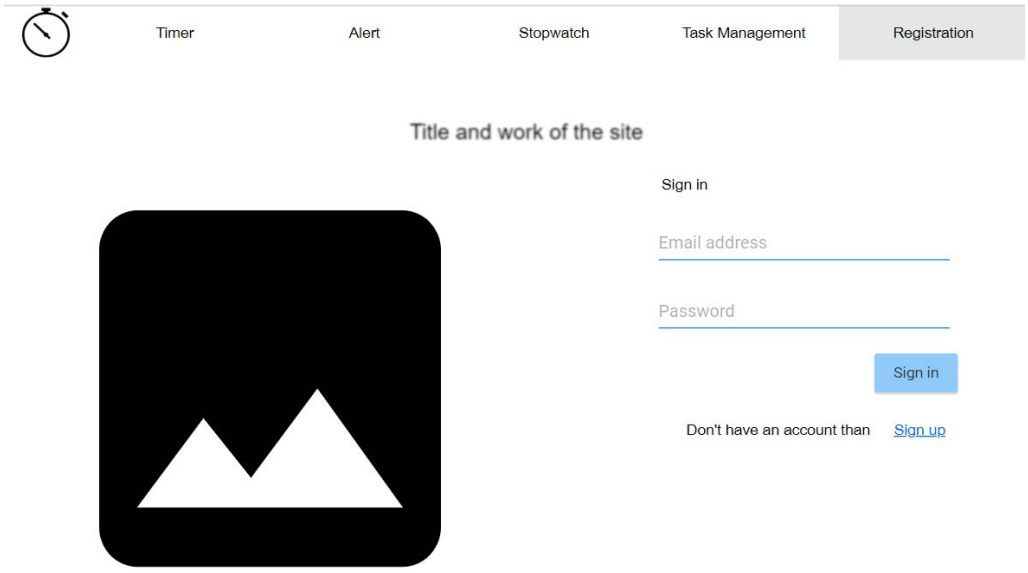


Рис. А.6. Вкладка з авторизацією

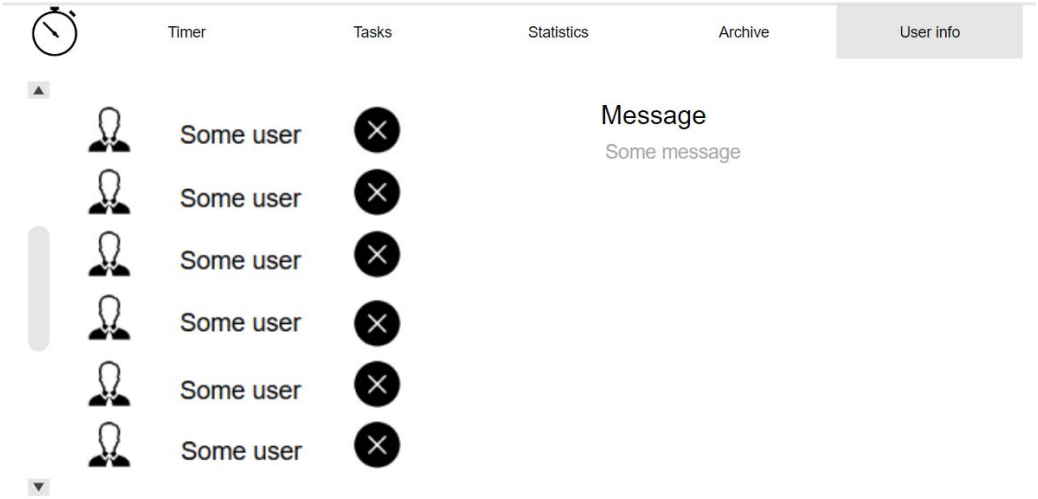


Рис. А.7. Сторінка адміністратора системи

Додаток Б

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;

using GtdTimerDAL.Entities;

namespace GtdTimerDAL.Repositories
{
    /// <summary>
    /// Class which implements IRepository interface
    /// </summary>
    /// <typeparam name="TEntity"> generic entity </typeparam>
    public class Repository<TEntity> : IRepository<TEntity> where TEntity : class
    {
        /// <summary>
        /// Gets or sets database set of entity type
        /// </summary>
        private readonly DbSet<TEntity> dbSet;

        /// <summary>
        /// Initializes a new instance of the <see cref="Repository{TEntity}" /> class.
        /// </summary>
        /// <param name="context">timer context instance</param>
        public Repository(TimerContext context)
        {
            TimerContext = context;
            dbSet = context.Set<TEntity>();
        }

        /// <summary>
        /// Gets or sets timer context
        /// </summary>
        public TimerContext TimerContext { get; set; }

        public IEnumerable<TEntity> GetAllEntities()
        {
            return dbSet;
        }

        public IEnumerable<TEntity> GetAllEntitiesByFilter(Func<TEntity, bool> filter)
        {
            return dbSet.Where(filter);
        }

        public TEntity GetByID(object id)
        {
            return dbSet.Find(id);
        }

        public void Create(TEntity entity)
        {
            dbSet.Add(entity);
        }

        public void Delete(object id)
        {
            TEntity entityToDelete = dbSet.Find(id);
            Delete(entityToDelete);
        }

        public void Delete(TEntity entityToDelete)
        {
            if (TimerContext.Entry(entityToDelete).State == EntityState.Detached)
            {
                dbSet.Attach(entityToDelete);
            }

            dbSet.Remove(entityToDelete);
        }

        public void Update(TEntity entityToUpdate)
        {
            dbSet.Attach(entityToUpdate);
            TimerContext.Entry(entityToUpdate).State = EntityState.Modified;
        }

        public void Save()
        {
            TimerContext.SaveChanges();
        }
    }
}
```

Рис. Б.1. Вигляд реалізованого репозиторію

Додаток В

```
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.SignalR;

using GtdCommon.ModelsDto;
using GtdServiceTier.Services;

namespace GtdTimer.Hubs
{
    /// <summary>
    /// class for task hub
    /// </summary>
    [Authorize]
    public class TaskHub : Hub<ITaskClient>
    {
        /// <summary>
        /// Instance of task service
        /// </summary>
        private readonly ITaskService taskService;

        /// <summary>
        /// Instance of user identity service
        /// </summary>
        private readonly IUserIdentityService userIdentityService;

        /// <summary>
        /// Initializes a new instance of the <see cref="TaskHub" /> class.
        /// </summary>
        /// <param name="taskService">instance of task service</param>
        /// <param name="userIdentityService">instance of user identity service</param>
        public TaskHub(ITaskService taskService, IUserIdentityService userIdentityService)
        {
            this.taskService = taskService;
            this.userIdentityService = userIdentityService;
        }

        public async Task CreateTask(TaskDto model)
        {
            model.UserId = GetUserId();
            taskService.CreateTask(model);
            await Clients.User(Context.UserIdentifier).CreateTask(model);
        }

        public async Task StartTask(TaskDto model)
        {
            model.UserId = GetUserId();
            taskService.StartTask(model);
            await Clients.GroupExcept(Context.UserIdentifier, Context.ConnectionId).StartTask(model);
        }

        public async Task PauseTask(TaskDto model)
        {
            model.UserId = GetUserId();
            taskService.PauseTask(model);
            await Clients.GroupExcept(Context.UserIdentifier, Context.ConnectionId).PauseTask(model);
        }

        public async Task ResetTask(TaskDto model)
        {
            model.UserId = GetUserId();
            taskService.ResetTask(model);
            await Clients.GroupExcept(Context.UserIdentifier, Context.ConnectionId).ResetTask(model);
        }

        public async Task DeleteTask(int taskId)
        {
            taskService.DeleteTaskById(taskId);
            await Clients.GroupExcept(Context.UserIdentifier, Context.ConnectionId).DeleteTask(taskId);
        }

        public async Task UpdateTask(TaskDto model)
        {
            model.UserId = GetUserId();
            taskService.UpdateTask(model);
            await Clients.GroupExcept(Context.UserIdentifier, Context.ConnectionId).UpdateTask(model);
        }

        public override async Task OnConnectedAsync()
        {
            await Groups.AddToGroupAsync(Context.ConnectionId, Context.UserIdentifier);
            await base.OnConnectedAsync();
        }

        public override async Task OnDisconnectedAsync(Exception exception)
        {
            await Groups.RemoveFromGroupAsync(Context.ConnectionId, Context.UserIdentifier);
            await base.OnDisconnectedAsync(exception);
        }

        private int GetUserId()
        {
            var userId = userIdentityService.GetUserId(Context.User.Identity);

            return userId;
        }
    }
}
```

Рис. В.1. Вигляд реалізованої системи SignalR