МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»

Кафедра інженерії програмного забезпечення

**КУРСОВИЙ ПРОЕКТ**

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

з дисципліни: «Моделювання та аналіз програмного забезпечення»

на тему:

**«СИСТЕМА УПРАВЛІННЯ ПЕРСОНАЛОМ»**

студента IV курсу групи ІПЗ-20-4

спеціальності 121 «Інженерія програмного забезпечення»

Васьківського Віталія Юрійовича

(прізвище, ім’я та по-батькові)

Керівник доцент Сугоняк І.І.

Дата захисту: " \_\_\_ " \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ р.

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ І.І. Сугоняк \_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.В. Власенко .

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.М. Кравченко \_

(підпис) (прізвище та ініціали)

Житомир – 2023

ЗМІСТ

[ВСТУП 3](#_Toc154088678)

[1 АНАЛІЗ ВИМОГ КОРИСТУВАЧА ТА КОНЦЕПТУАЛЬНЕ МОДЕЛЮВАННЯ 4](#_Toc154088679)

[**1.1 Технічне завдання на розробку системи** 4](#_Toc154088680)

[**1.2 Обґрунтування вибору засобів моделювання** 5](#_Toc154088681)

[**1.3 Аналіз вимог до програмного продукту** 9](#_Toc154088682)

[2 РОЗРОБКА МОДЕЛІ ПРОГРАМНОГО КОМПЛЕКСУ НА ЛОГІЧНОМУ РІВНІ 13](#_Toc154088683)

[**2.1 Алгоритм роботи та стани програмної системи** 13](#_Toc154088684)

[**2.2 Об’єктно-орієнтована модель системи** 14](#_Toc154088685)

[**2.3 Взаємодія об’єктів системи** 17](#_Toc154088686)

[3 ФІЗИЧНА МОДЕЛЬ ТА ПРОТОТИП ПРОГРАМНОГО КОМПЛЕКСУ 19](#_Toc154088687)

[**3.1 Взаємодія компонентів системи** 19](#_Toc154088688)

[**3.2 Архітектура програмного комплексу та його розгортання** 20](#_Toc154088689)

[**3.3 Написання програмного коду для прототипу програми** 22](#_Toc154088690)

[ВИСНОВКИ 25](#_Toc154088691)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 26](#_Toc154088692)

[ДОДАТКИ 28](#_Toc154088693)

ВСТУП

В сучасних умовах динамічного розвитку бізнес-середовища ключовою складовою успіху підприємства є ефективне управління персоналом. Система управління персоналом визначає стратегічний курс компанії та грає важливу роль у формуванні корпоративної культури. Забезпечення оптимального використання людських ресурсів, стимулювання професійного зростання працівників та вирішення конфліктів – це лише кілька аспектів, які входять у сферу впливу системи управління персоналом.

Метою курсової роботи є дослідження особливостей моделювання та аналізу програмних комплексів за визначеним темою курсової роботи напрямком з використанням CASE-технологій.

Завданням на курсову роботу є:

* аналіз теоретичних засад моделювання програмного забезпечення;
* аналіз та опис вимог користування;
* методи модулювання функцій та поведінки системи;
* проектування об’єктної структури системи;
* фізичне моделювання;
* програмних комплексів;
* кодогенерація з моделей.

Предметом дослідження є можливості застосування CASE-засобів проектування програмного забезпечення.

Об’єктом дослідження є методи та засоби проектування програмного забезпечення та уніфікація процесу проектування.

В процесі роботи над курсовим проектом студент може використовувати монографічні, аналітичні, математичні, графічні методи, методи об’єктно-орієнтованого проектування та програмування та інші методи дослідження.

1 АНАЛІЗ ВИМОГ КОРИСТУВАЧА ТА КОНЦЕПТУАЛЬНЕ МОДЕЛЮВАННЯ

**1.1 Технічне завдання на розробку системи**

1. Назва системи:

Повне найменування системи: система управління персоналом.

Коротке найменування системи: Програма, Програмне забезпечення (ПЗ), Програмний комплекс, Система.

1. Призначення системи:

Система призначена для ефективного вибору, розвитку та утримання персоналу на підприємстві. Основні завдання: відслідковувавання відпрацьованого часу, створення лікарняних та відпусток.

1. Вимоги до системи:

Програмний продукт повинен мати наступні функціональні характеристики:

* доступ до системи за допомогою звичних для сучасної людини браузерів;
* зрозумілий та інтуїтивний дизайн;
* можливість одночасної роботи з системою як мінімум 50 користувачів;
* наявність двох ролей користувачів (адміністратор та працівник).

1. Вимоги до програмного забезпечення:

Програмний комплекс повинен бути побудований на клієнт-серверній архітектурі з використанням веб-технологій, що не вимагають додаткового ліцензування. При проектуванні програмного комплексу необхідно застосувати архітектуру SPA (Single-page application) для забезпечення користувачу досвіду, близького до користування настільною програмою. Запропонованою мовою програмування для створення серверної частини додатку, в нашому випадку, приватного API, є C#. При розробці передбачається використання фреймворків та

бібліотек: ASP.NET, GraphQL.NET, React, Redux.

Клієнтська частина повинна бути побудована за сучасними стандартами створення односторінкового застосунку, тобто весь необхідний код завантажується разом зі сторінкою, або динамічно довантажується за потребою, зазвичай у відповідь на дії користувача. Сторінка не оновлюється і не перенаправляє користувача до іншої сторінки у процесі роботи з нею. Також, дозволено використання готовових бібліотек (пакетів) для пришвидшення розробки застосунку та покращення його швидкодії.

1. Вимоги до інтерфейсу:

Взаємодія користувачів із програмним комплексом повинна відбуватися за допомогою веб-інтерфейсів, а точніше веб-браузерів. Інтерфейс програмного комплексу повинен бути інтуїтивно зрозумілим та зручним у використанні. Навігаційні елементи та компоненти інтерактивної взаємодії повинні бути виконані у зручній для користувача формі. Введення та виведення даних в систему, введення команд та результат їх виконання повинні виконуватися в інтерактивному режимі.

На інтерфейси користувача накладаються наступні вимоги:

* мінімальний час відгуку системи;
* повідомлення про критичні ситуації системи;
* отримання відповіді на будь-яку дію користувача;
* відображення процесу завантаження для ресурсомістких завдань.

**1.2 Обґрунтування вибору засобів моделювання**

В сучасному світі вже створено безліч додатків для будь-якої предметної області, тож і для створення UML-діаграм існує велика їх кількість.

Для аналізу кращого інструменту для вирішення заданої задачі було обрано наступні редактори: StarUML, Draw.IO, Creately.

**StarUML**

StarUML – це редактор, що доступний на всіх популярних платформах, таких як Windows та Linux. Це додаток має як безкоштовну, так і комерційну версію. До переваг StarUML можна віднести використання плагінів, які дозволяють генерувати код з моделей для мов програмування. Не потребує ніякої реєстрації. Редактор підтримує усі стандарти UML і оновлюється відповідно до нових (2.0). Має доволі зрозумілий і зручний інтерфейс, при використанні яким не повинно виникнути проблем.

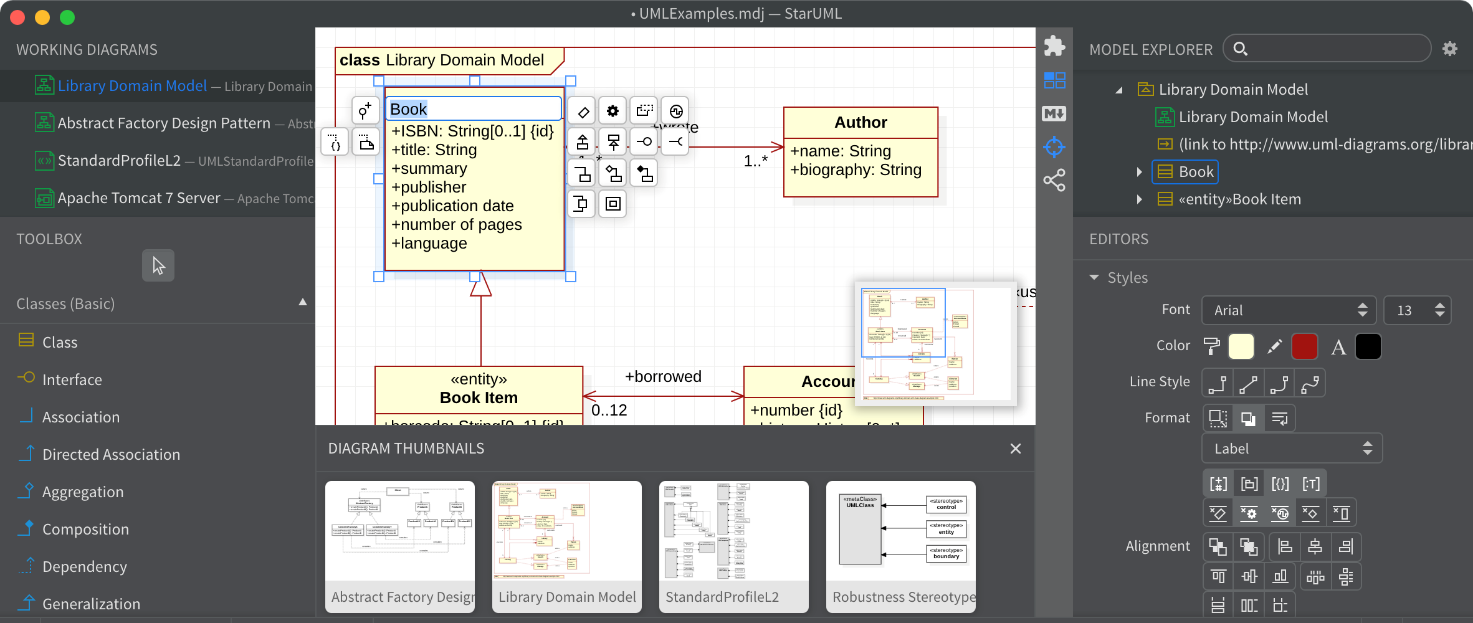


Рисунок 1.1 – Інтерфейс програми StarUML

**Draw.IO**

Draw.IO – це безкоштовне міжплатформне програмне забезпечення для малювання графіків із відкритим кодом, розроблене на HTML5 та JavaScript. Його інтерфейс можна використовувати для створення діаграм, таких як блок-схеми, каркасні схеми, діаграми UML, організаційні діаграми та мережеві діаграми. Доступний як онлайн у вигляді кросбраузерного веб-додатку, так і як автономний настільний додаток для Linux, macOS і Windows. Веб-додаток не потребує реєстрації, його можна відкривати та зберігати на локальному жорсткому диску. Підтримувані формати зберігання та експорту для завантаження включають PNG, JPEG, SVG і PDF. Має достатній обсяг елементів для побудови відповідних діаграм, але менший в порівняні з StarUML.

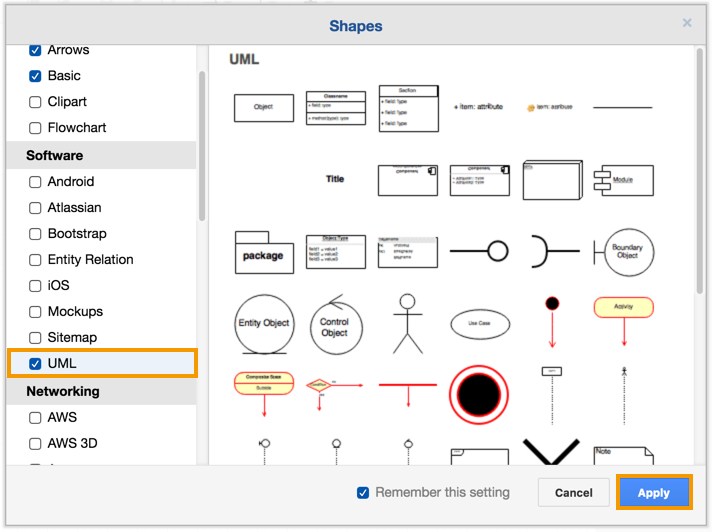


Рисунок 1.2 – Інтерфейс програми Draw.IO

**Creately**

Creately — це візуальний інструмент для співпраці SaaS із можливостями створення діаграм і дизайну, розроблений Cinergix. Creately має дві версії: хмарну онлайн-версію та офлайн-версію для комп’ютера, яка сумісна з Windows, Mac і Linux. Програма здебільшого відома тим, що створює блок -схеми, організаційні діаграми,діаграми проектів, діаграми UML, інтелектуальні карти та інші бізнес-візуальні матеріали. Має зручний інтерфейс і велику кількість шаблонів діаграм, але не всіх з них є безкоштовними. Також, певні функції є платними. Для початку роботи з діаграмами необхідно створити акаунт.

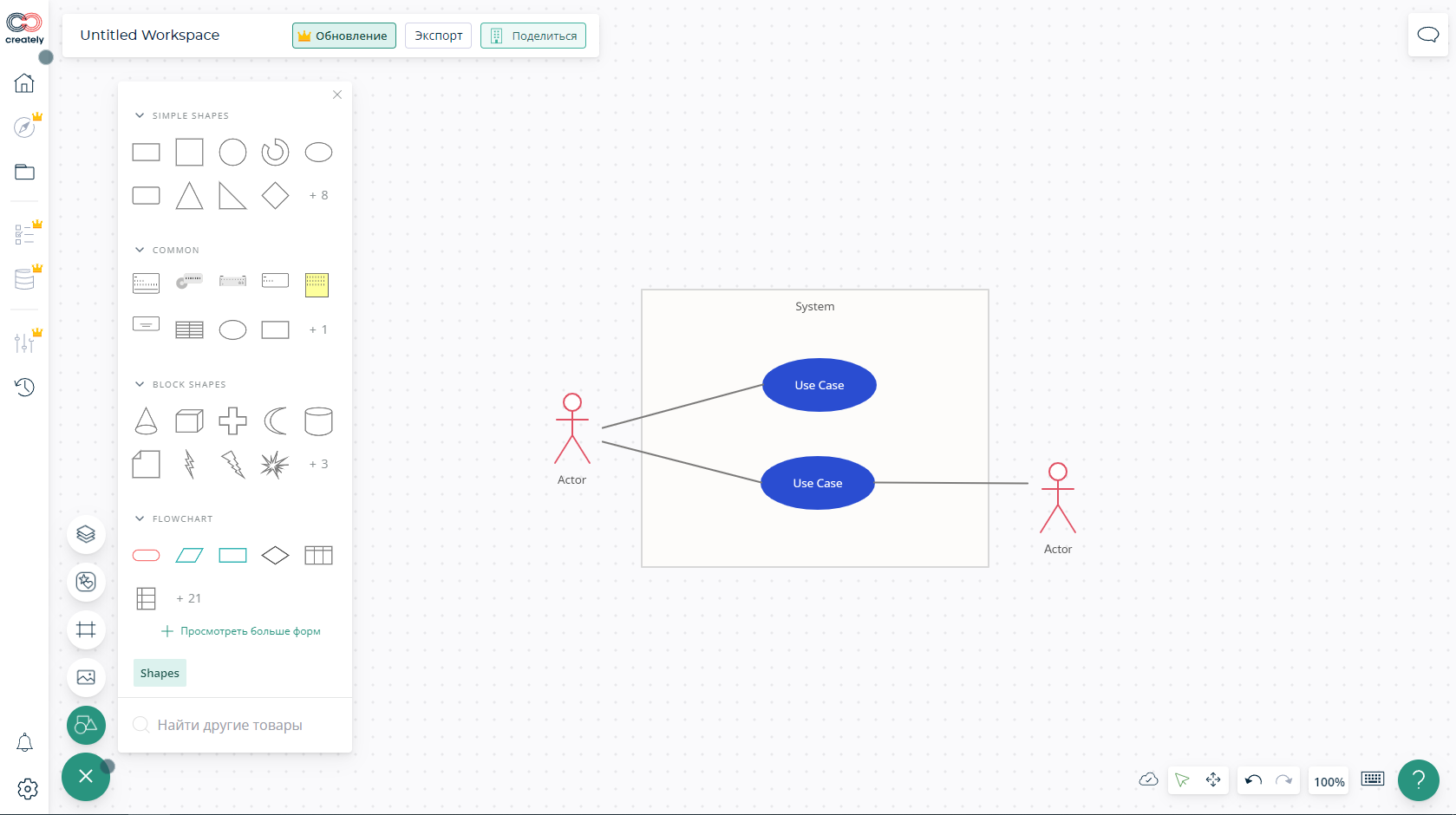


Рисунок 1.3 – Інтерфейс програми Creately

Для наочності порівняння отримані результати подані в таблиці 1.1.

Таблиця 1.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Простота використання | Наявність всіх необхідних діаграм | Генерація коду | Експорт | Імпорт | Підтримка |
| StarUML | + | + | + | + | + | + |
| Draw.IO | + | +- | - | + | + | + |
| Creately | + | +- | - | + | + | + |

Зважаючи на отримані результати, було вирішено використовувати Draw.io для реалізації поставленої задачі.

**1.3 Аналіз вимог до програмного продукту**

Розглянемо високорівневі вимоги, якими повинна володіти система, щоб мати можливість безперервно виконувати покладені на неї фунцій, тим самим відповідно задовольнивши всі зазначені стандарти, специфікації та інші формальні документи. Зазначені умови, яких повинен дотримуватись користувач для роботи з даною інформаційною системою та вказані основні ділові переваги данної ІС. Приведені можливості системи, обмеження та показники якості.

**Бізнес-вимоги:**

1. Основні цілі: проект створюється з метою управління персоналом.
2. Можливості: перегляд відпрацьованого часу, створення відпусток та лікарняних.
3. Представлення проекту: проект буде реалізовано у вигляді веб-додатку, що містить зручний інтерфейс та необхідну інформацію за даною тематикою.

**Вимоги користувачів**

* **Працівник**

1. Можливість створення та перегляд відпрацьованого часу;
2. Можливість переглядати та створювати запит на відпустку;
3. Можливість переглядати та створювати запит на лікарняний;
4. Можливість переглядати календар з відміченими вихідними, скороченими днями та святами;
5. Можливість перегляду профілю;

* **Адміністратор компанії**

1. Створення працівників;
2. Надання відповідний дозволів;
3. Покупка підписок;

* **Супер адміністратор**

1. Створення та редагування компаній;

**Характеристика об`єкту комп`ютеризації:**

Користувач буде мати можливість відслідковувати відпрацьований час, створювати відпустки та лікарняні. Адміністратор через веб-інтерфейс буде мати можливість видавати дозволи та керувати підпискою.

**Функціональні вимоги:**

1. Підключення сторонніх API для реалізації оплати і авторизації за допомогою Google.
2. Створення модуля для менеджменту часом.
3. Створення модуля для менеджменту відпустками.
4. Створення модуля для менеджменту лікарняними.

**Нефункціональні вимоги:**

1. Сприйняття

* час, потрібний для навчання інструментами роботи з інформаційною системою для звичайних користувачів – 1 години, для досвідчених – 0.5 година;
* час відповіді системи для звичайних запитів не повинен перевищувати 1 секунд, а для більш складних запитів – 2 сек.
* інтерфейс повинен бути інтуїтивно зручним для користувача та не вимагати додаткової підготовки.

1. Продуктивність

* Система повинна підтримувати мінімум 50 користувачів, що одночасно працюють, пов’язаних з спільною базою даних.

**Системні вимоги:**

1. **Вимоги до середовища виконання:**

Система повинна задовольняти зазначеним вимогам на комп’ютері в наступній мінімальній комплектації:

* підтримка браузера Google Chrome версії 45+ або
* підтримка браузера Mozilla Firefox версії 38+ або
* підтримка браузера Microsoft Edge версії 12+ або
* підтримка браузера Opera версії 30+ або
* підтримка браузера Safari версії 9+

1. **Вимоги до СУБД та доступу до даних:**

У ядрі системи повинна бути представлена СУБД реляційного доступу;

**Бізнес правила:**

Система повинна відповідати всім стандартам побудови сучасних веб-застосунків, що використовує архітектуру SPA (Single-page application).

**Антивимоги:**

* не використовує інформацію, заборонену нормами моралі та заканодавством;
* не використовує інформацію, що є недійсною або може зашкодити психіці користувачів;
* не містить реклами.

**Бізнес-логіка:**

Підсумовуючи, побудуємо діаграму варіантів використання (Use Case Diagram).

Специфікацію прецендентів наведено в додатку А.

На рисунку 1.4 зображено побудовану діаграму варіантів використання.

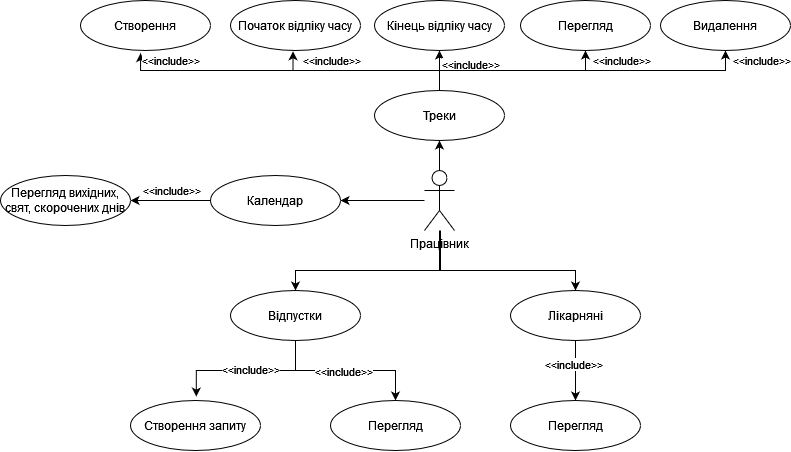


Рисунок 1.4 – Діаграма варіантів використання

2 РОЗРОБКА МОДЕЛІ ПРОГРАМНОГО КОМПЛЕКСУ НА ЛОГІЧНОМУ РІВНІ

**2.1 Алгоритм роботи та стани програмної системи**

Під час згадки алгоритму роботи та можливих станів програмної системи відразу приходить на думку побудова діаграми діяльності (activity diagram).

Діаграма діяльності є однією важливою діаграмою поведінки для опису динамічних аспектів системи. Діаграма діяльності – це, по суті, вдосконалена версія блок-схеми, яка моделює потік від однієї діяльності до іншої.

На рисунку 2.1 зображено діаграму діяльності.

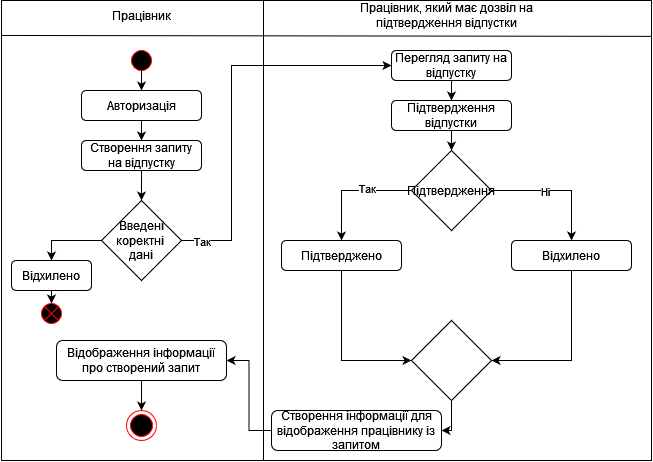


Рисунок 2.1 – Діаграма діяльності

**Опис діаграми діяльності:**

* авторизація;
* створення запиту на відпустку;
* перегляд запиту на відпустку;
* підтвердження запиту на відпустку(так – підтверджено, ні - відхилено);
* створення інформації для відображення працівнику із запитом;
* відображення інформації про створений запит;
* кінець роботи веб-застосунку;

**2.2 Об’єктно-орієнтована модель системи**

Об’єктно-орієнтовану модель системи прийнято розглядати у вигляді діаграми класів (class diagram). Для побудови даної діаграми спочатку потрібно розібратися, що воно собою представляє. Діаграма класів – це тип діаграми статичної структури, який описує структуру системи, показуючи класи системи, їхні атрибути, операції (або методи) і зв’язки між об’єктами.

На рисунку 2.2 зображено побудовану діаграму класів.

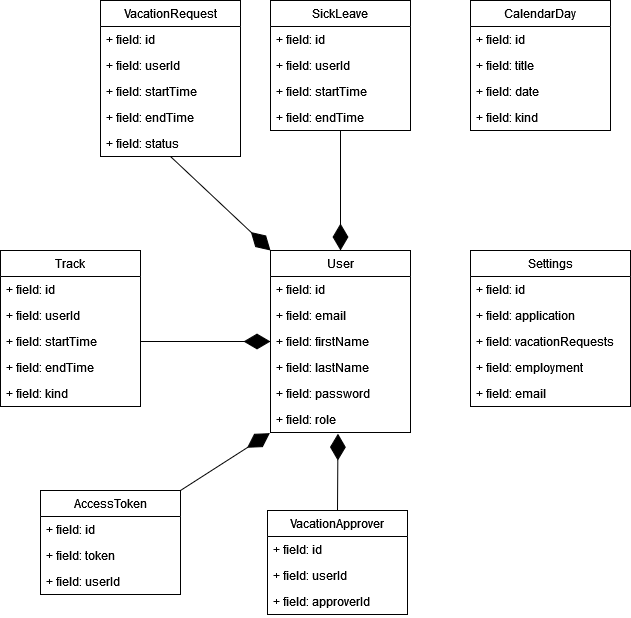


Рисунок 2.2 – Діаграма класів

Нижче наведено опис класів, а також їх атрибутів та методів:

* Клас *User* містить інформацію про зареєстрованого користувача та реалізовує методи
* Клас *Track* містить запис з інформацієї про відпрацьований час. Зв’язок з класом User реалізований у вигляді агрегації.
* Клас *VacationRequest* містить інформацію про відпустку. Зв’язок з класом User реалізований у вигляді агрегації.
* Клас *SickLeave* містить інформацію про лікарняний. Зв’язок з класом User реалізований у вигляді агрегації.
* Клас *AccessToken* містить інформацію про токен для авторизації. Зв’язок з класом User реалізований у вигляді агрегації.
* Клас *Settings* містить налаштування додатку
* Клас *CalendayDay* мітить інформацію про вихідні, скорочені дні та свята.
* Клас *VacationApprover* мітстить користувачів, які можуть підтверджувати відпустку. Зв’язок з класом User реалізований у вигляді агрегації.

**Патерни проектування**

Нині IT-сфера розвивається дуже швидко, тому рішення багатьох проблем, що часто зустрічаються, вже знайдено і надано назву “патерн проектування”. Без їх використання важко уявити сучасні застосунки. Тому використання патернів продумується ще на стадії моделювання майбутньої системи.

Отже, було прийнято рішення використовувати патерн “Репозиторій”, який напевно є найпопулярнішим рішенням для застосунків, розроблених за архітектурою SPA.

“Репозиторії” — це класи або компоненти, які інкапсулюють логіку, необхідну для доступу до джерел даних. Вони централізують спільну функціональність доступу до даних, забезпечуючи кращу зручність   
  
обслуговування та відокремлюючи інфраструктуру або технологію, що використовується для доступу до баз даних, від рівня моделі предметної області.

На рисунку 2.3 зображено загальний вигляд патерну “Репозиторій ”.



Рисунок 2.3 – Загальний вигляд патерну “Репозиторій”

На рисунку 2.4 зображено реалізацію патерну “Репозиторій”.

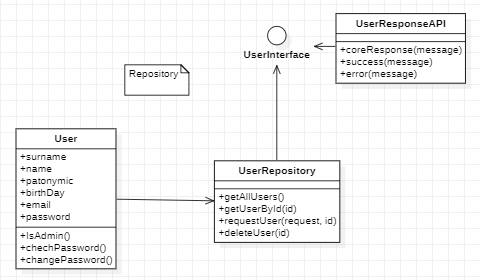


Рисунок 2.4 – Реалізація патерну “Репозиторій”

**2.3 Взаємодія об’єктів системи**

Взаємодію об’єктів системи в UML прийнято зображати у вигляді діаграми послідовності.

Діаграми послідовності – це діаграми взаємодії, які детально описують, як виконуються операції. Вони фіксують взаємодію між об’єктами в контексті співпраці. Діаграми послідовності зосереджені на часі, і вони візуально показують порядок взаємодії за допомогою вертикальної осі діаграми, щоб відобразити час, які повідомлення надсилаються та коли.

Було вирішено побудувати діаграму послідовності для модуля дискусій, щоб продемонструвати процес обговорення певного питання.

На рисунку 2.6 зображена діаграма послідовності для модуля дискусій.

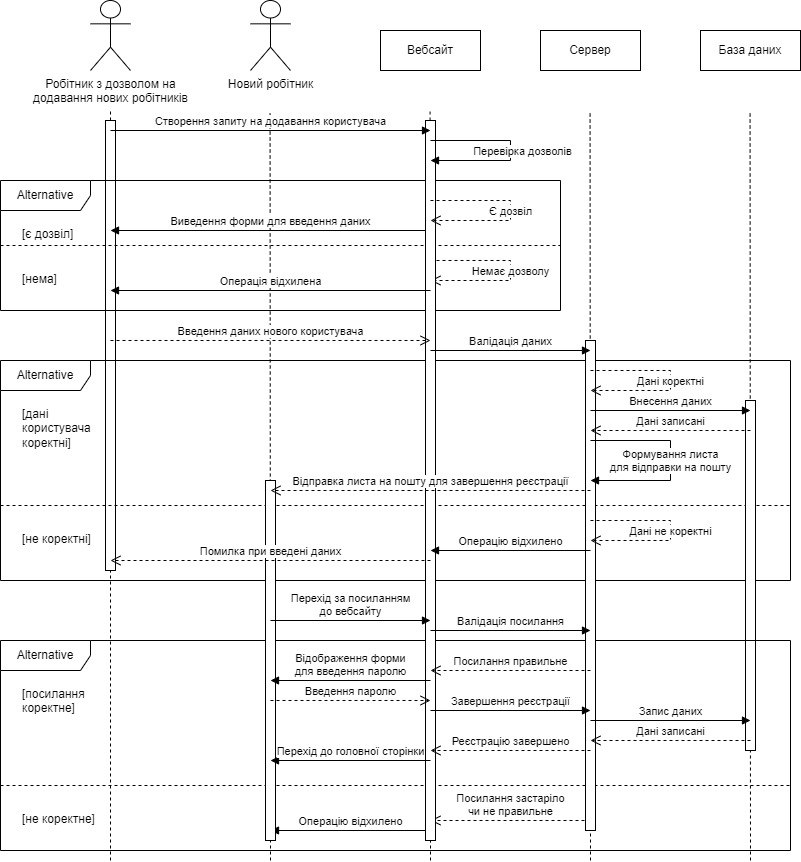


Рисунок 2.6 – Діаграма послідовності для модуля дискусій

**Опис діаграми послідовності:**

1. Працівник з дозволом на додавання нових працівників намагається створити нового працівника.
2. Відбувається перевірка дозволу(якщо є – виведення форми для створення, якщо ні – операція відхилена).
3. Після заповнення даних відбувається валідація(якщо успішна – створення користувача, якщо є помилки – відображення помилок).
4. Відправка листа на пошту.
5. Новий робітник переходить за посиланням на пошті і створює пароль.

3 ФІЗИЧНА МОДЕЛЬ ТА ПРОТОТИП ПРОГРАМНОГО КОМПЛЕКСУ

**3.1 Взаємодія компонентів системи**

Для зображення взаємодії компонентів системи мовою UML передбачено окремий вид діаграм – діаграму компонентів (component diagram).

Цей тип діаграм використовується для моделювання фізичних аспектів об’єктно-орієнтованих систем, які використовуються для візуалізації, специфікації та документування компонентних систем, а також для побудови виконуваних систем шляхом прямого та зворотного проектування. Діаграми компонентів – це, по суті, діаграми класів, які зосереджуються на компонентах системи, які часто використовуються для моделювання статичної реалізації системи.

На рисунку 3.1 зображено діаграму компонентів.

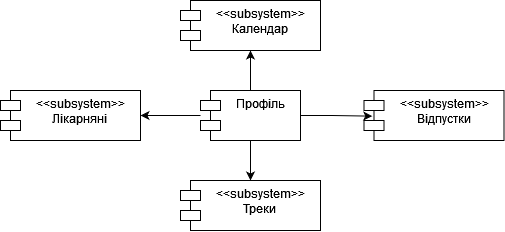


Рисунок 3.1 – Діаграма компонентів

**Опис діаграми компонентів:**

* Для роботи з основними компонентами користувач повинен бути авторизованим у системі.
* Систему було розділено на три підкомпоненти
* Підсистема *Треки* представляє собою відслідковування відпрацьованого часу.
* Підкомпонент *Відпустки* відповідає за створення та перегляд відпусток.
* Підкомпонент *Лікарніні* відповідає за створення та перегляд лікарняних.
* Компонент *Календар* відповідає за перегляд вихідних, свят та скорочених днів.

**3.2 Архітектура програмного комплексу та його розгортання**

При проектуванні програмного комплексу було вирішено застосувати архітектуру SPA (Single-page application) для забезпечення користувачу досвіду, близького до користування настільною програмою.

**До переваг цієї архітектури належать:**

* *швидке реагування на дії користувача*;

Щоб виконати запит користувача, SPA завантажує невеликий файл JSON замість нової веб-сторінки. Навіть із кешуванням і відкладеним завантаженням MFA потребує більше часу, щоб отримати нові дані з сервера, ніж SPA. Крім того, той факт, що сторінку не потрібно перезавантажувати, робить робочі процеси, які включають кілька екранів, більш оптимізованими в SPA, ніж у звичайних веб-додатках.

* *висока продуктивність і стабільність*;

Правильно виконані асинхронні запити можуть зменшити навантаження на сервер і вимагати меншої пропускної здатності. Основна логіка односторінкової програми зазвичай повністю завантажується за першим запитом, що дає змогу виконувати більшість операцій у автономному режимі, забезпечуючи чудове рішення для поганого підключення до Інтернету.

* *швидкий розвиток*;

Під час розробки SPA фронтенд і бекенд розділені, тому розробники можуть працювати паралельно. Крім того, зміна фронтенду не впливає на бекенд, і навпаки.

* *зручний для мобільних пристроїв*.

Сервер SPA можна використовувати для створення мобільної програми, ідентичної веб-програмі. Це не вимагає особливої ​​адаптації, оскільки SPA вже розроблено як додаток, а не веб-сайт. Це значно прискорює мобільну розробку.

Для зображення розгортання системи існує однойменна діаграма – діаграма розгортання (deployment diagram).

Діаграма розгортання – це діаграма, яка показує конфігурацію вузлів обробки часу виконання та компонентів, які на них живуть. Діаграми розгортання – це різновид структурної діаграми, яка використовується для моделювання фізичних аспектів об’єктно-орієнтованої системи. Вони часто використовуються для моделювання статичного вигляду розгортання системи (топології апаратного забезпечення).

На рисунку 3.2 зображено діаграму розгортання.

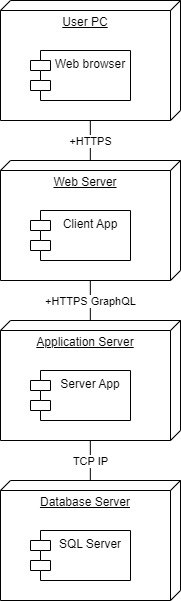


Рисунок 3.2 – Діаграма розгортання

Спроектована система передбачає реалізацію як фронтенд, так і бекенд (приватне API) частин додатку з використанням сучасних фреймоворків.

**3.3 Написання програмного коду для прототипу програми**

Написано класи для проектованої системи зображені на рисунку 3.3, а лістинг коду кожного класу наведено в додатку Б.

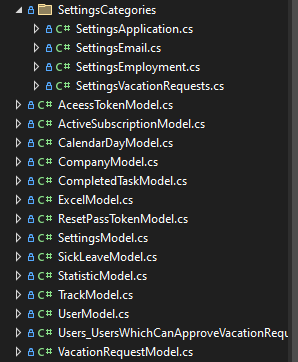


Рисунок 3.3 – Файли класів

На рисунках 3.4–3.5 зображені розроблені прототипи додатку.

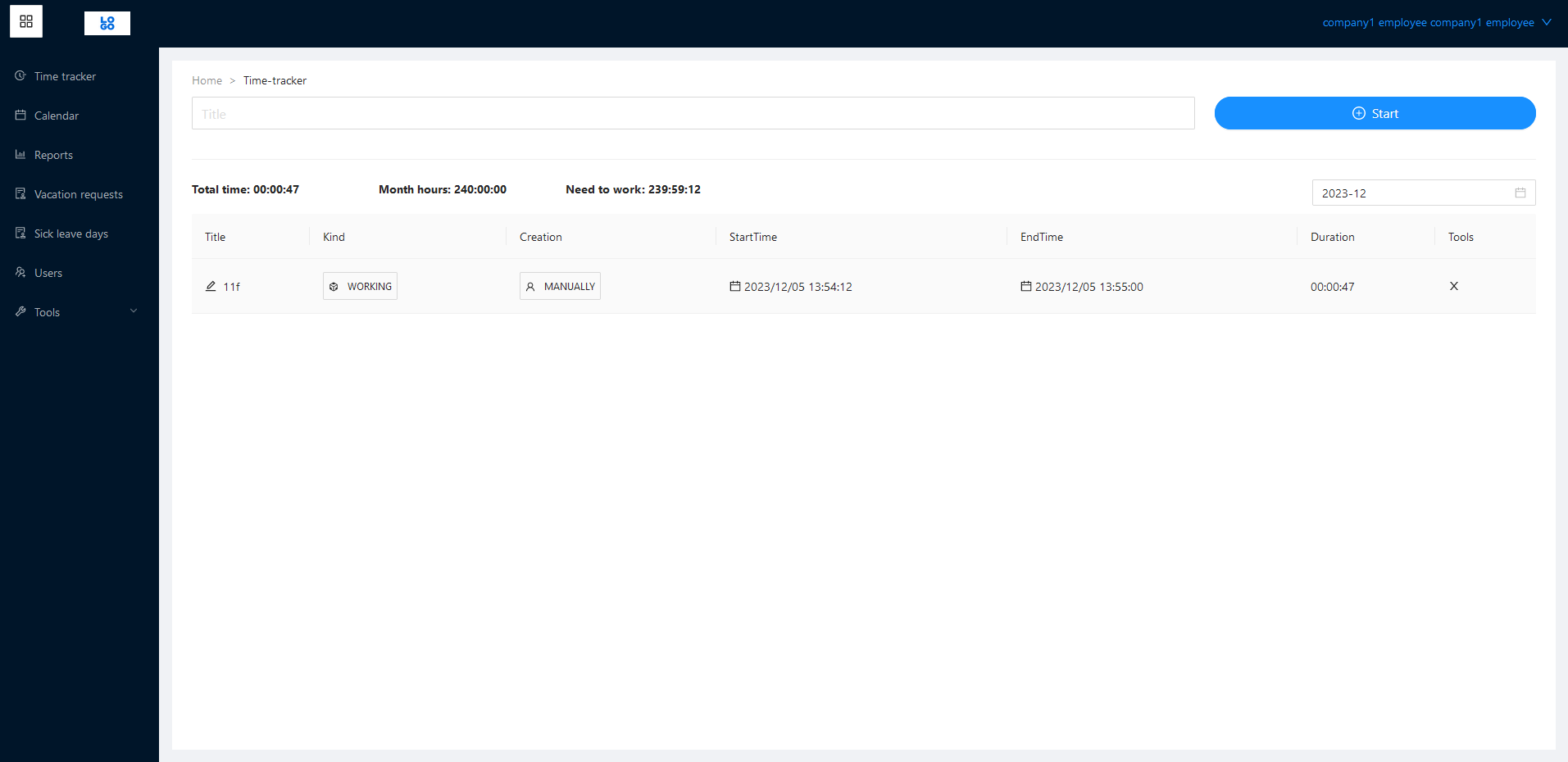


Рисунок 3.4 – Прототип сторінки треків

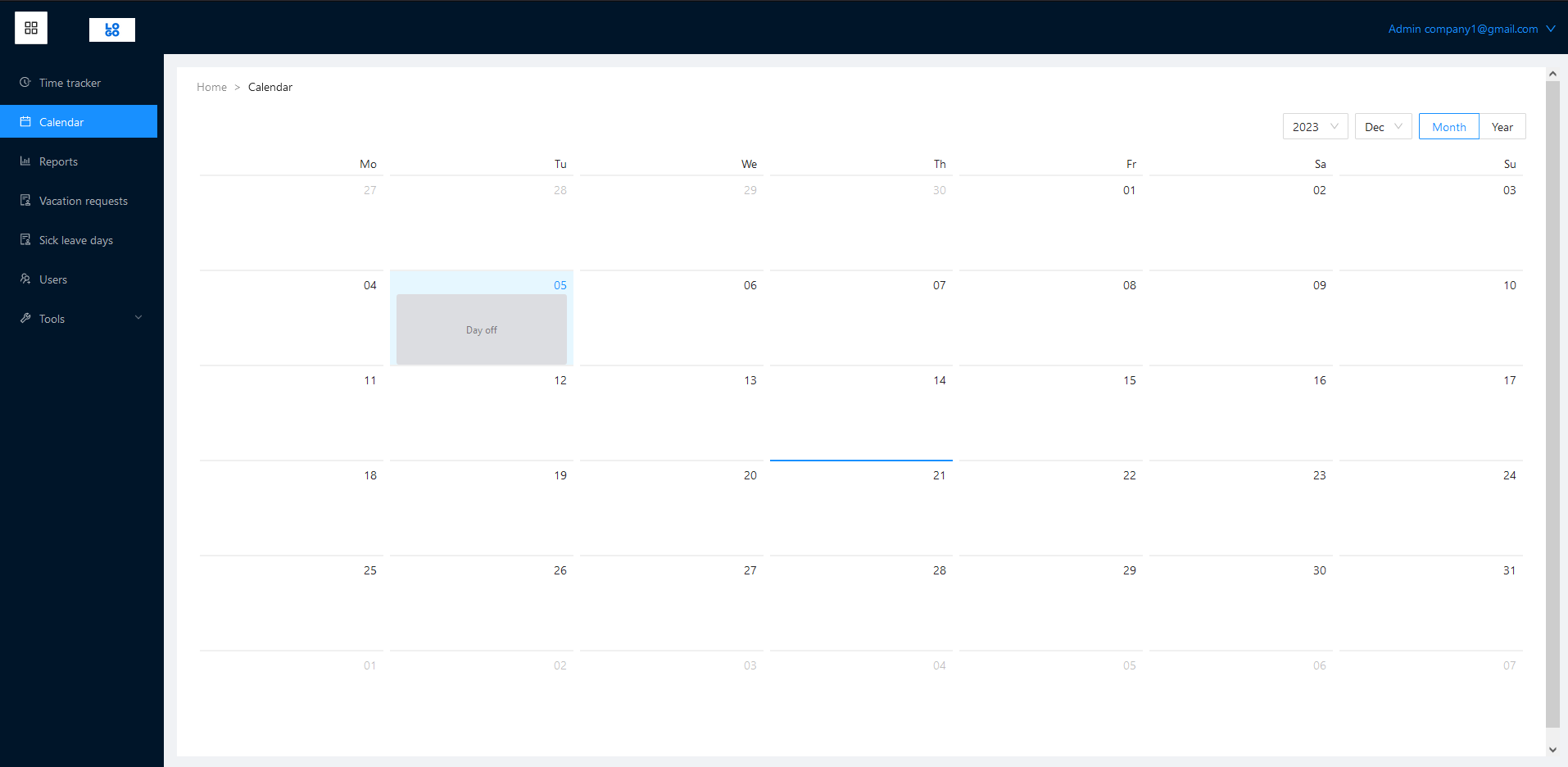


Рисунок 3.5 – Прототип сторінки з календарем

ВИСНОВКИ

Отже, під час виконання даної курсової роботи було закріплено пройдений матеріал, який вивчався протягом курсу, а саме: моделювання та аналіз програмного забезпечення за допомогою мови UML. Виконання курсового проекту було розділено на 3 етапи.

У першому розділі було отримано технічне завдання на розробку системи. Було обрано засіб моделювання для вирішення поставленого завдання. Також, було виокремлено та проаналізовано вимоги до програмного продукту.

У другому розділі було розроблено модель програмного комплексу на логічному рівні. Було змодельовано діаграму активності для демонстрації алгоритму роботи та можливих станів програмної системи та діаграму класів для презентування об’єктно-орієнтованої моделі системи. Імплементовано патерни проектування. Також, було спроектовано діаграму послідовності для зображення взаємодії об’єктів системи.

У третьому розділі було створено фізичну модель та прототип програмного комплексу. Було продемонстровано взаємодію компонентів системи за допомогою побудови діаграми компонентів. Було обґрунтовано вибір архітектури програмного комплексу. Також, було побудовано діаграму розгортання. Після написання коду з діаграми класів були розроблені прототипи додатку.

Отже, виконана робота показала, що за допомогою вивченого матеріалу за курс можливо успішно моделювати програмні комплекси, використовуючи мову UML.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. StarUML documentation: Introduction. URL: <https://docs.staruml.io>.
2. UML для бізнес-моделювання: для чого потрібні діаграми процесів [Електронний ресурс] – Режим доступу до ресурсу: <https://evergreens.com.ua/ua/articles/uml-diagrams.html>.
3. Single-page application [Електронний ресурс] – Режим доступу до ресурсу: <https://en.wikipedia.org/wiki/Single-page_application>.
4. Графічний редактор DRAW.IO [Електронний ресурс] – Режим доступу до ресурсу: <https://app.diagrams.net/>.
5. Графічний редактор Creately [Електронний ресурс] – Режим доступу до ресурсу: <https://creately.com/>.
6. Use Case Diagram [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>.
7. Activity Diagram [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/>.
8. Class Diagram [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>.
9. Repository Design Pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@pererikbergman/repository-design-pattern-e28c0f3e4a30>.
10. Laravel API — Repository Pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.devgenius.io/laravel-api-repository-pattern-make-your-code-more-structured-the-simple-guide-5b770da766d7>.
11. Unit Of Work pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/Unit_Of_Work>.
12. Sequence Diagram [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>.
13. Deployment Diagram [Електронний ресурс] – Режим доступу до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/>.
14. Графічний редактор Figma [Електронний ресурс] – Режим доступу до ресурсу: <https://www.figma.com/>.
15. Unified Modeling Language [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/Unified_Modeling_Language>.

ДОДАТКИ

Додаток А

Специфікація прецедентів

Таблиця А.1

|  |  |  |  |
| --- | --- | --- | --- |
| **Ім’я варіанту використання: Перегляд відпустки** | | | **Рівень важливості: високий** |
| **Основний актор**: користувач | | | |
| **Короткий опис:** | | Перегляд повної інформації про відпустку | |
| **Передумови:** | | Ініціалізація в системі | |
| **Пост-умови:** | |  | |
| **Відносини:** | |  | |
|  | **Асоціація:** |  | |
|  | **Включення:** | Ініціалізація | |
|  | **Розширення:** |  | |
|  | **Узагальнення:** |  | |
| **Основний потік:** | | Якщо сталася помилка під час перегляду інформації, то необхідно проінформувати користувача. | |
| **Альтернативні/виключні потоки:** | | Виведена форма для повідомлення помилки | |

Таблиця А.2

|  |  |  |  |
| --- | --- | --- | --- |
| **Ім’я варіанту використання: Створення треку** | | | **Рівень важливості: високий** |
| **Основний актор**: користувач | | | |
| **Короткий опис:** | | Створення треку для відслідковування відпрацьованого часу | |
| **Передумови:** | | Бути авторизованим | |
| **Пост-умови:** | |  | |
| **Відносини:** | |  | |
|  | **Асоціація:** |  | |
|  | **Включення:** |  | |
|  | **Розширення:** |  | |
|  | **Узагальнення:** |  | |
| **Основний потік:** | | Якщо сталася помилка під час створення треку, то необхідно проінформувати користувача. | |
| **Альтернативні/виключні потоки:** | | Виведена форма для повідомлення помилки | |

Додаток Б

UserModel.cs

Лістинг програми:

using Newtonsoft.Json;

using Newtonsoft.Json.Converters;

using TimeTracker.Business.Abstractions;

using TimeTracker.Business.Enums;

namespace TimeTracker.Business.Models

{

public class UserModel : BaseModel

{

public string Email { get; set; }

public string Password { get; set; }

public string Salt { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string MiddleName { get; set; }

public int RoleNumber { get; private set; }

public Guid? CompanyId { get; set; }

public Role Role

{

get => (Role)RoleNumber;

set => RoleNumber = (int)value;

}

public string PermissionsString { get; private set; }

public IEnumerable<Permission> Permissions

{

get => JsonConvert.DeserializeObject<IEnumerable<Permission>>(PermissionsString);

set => PermissionsString = JsonConvert.SerializeObject(value, new StringEnumConverter());

}

public Employment Employment { get; set; }

public IEnumerable<UserModel> UsersWhichCanApproveVacationRequest { get; set; } = new List<UserModel>();

public ExcelModel ToExcelModel()

{

ExcelModel excelModel = new ExcelModel()

{

UserInitials = $"{this.LastName} {this.FirstName} {this.MiddleName}",

Employment = this.Employment

};

return excelModel;

}

}

}

TrackModel.cs

Лістинг програми:

using TimeTracker.Business.Abstractions;

using TimeTracker.Business.Enums;

namespace TimeTracker.Business.Models

{

public class TrackModel : BaseModel

{

public Guid UserId { get; set; }

public string? Title { get; set; }

public TrackKind Kind { get; set; }

public TrackCreation Creation { get; set; }

public string? EditedBy { get; set; }

public DateTime? StartTime { get; set; }

public DateTime? EndTime { get; set; }

}

}

VacationRequest.cs

Лістинг програми:

using TimeTracker.Business.Abstractions;

using TimeTracker.Business.Enums;

namespace TimeTracker.Business.Models

{

public class VacationRequestModel : BaseModel

{

public DateTime DateStart { get; set; }

public DateTime DateEnd { get; set; }

public string? Comment { get; set; }

public VacationRequestStatus Status { get; set; }

public Guid UserId { get; set; }

public UserModel User { get; set; }

public Guid CompanyId { get; set; }

}

}

SickLeave.cs

Лістинг програми:

using Newtonsoft.Json;

using TimeTracker.Business.Abstractions;

namespace TimeTracker.Business.Models

{

public class SickLeaveModel : BaseModel

{

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

public Guid UserId { get; set; }

public string Comment { get; set; } = "";

public string FilesString { get; private set; }

public IEnumerable<string> Files

{

get => JsonConvert.DeserializeObject<IEnumerable<string>>(FilesString ?? "[]");

set => FilesString = JsonConvert.SerializeObject(value);

}

public Guid CompanyId { get; set; }

}

}

CalendarDay.cs

Лістинг програми:

using TimeTracker.Business.Abstractions;

using TimeTracker.Business.Enums;

namespace TimeTracker.Business.Models

{

public class CalendarDayModel : BaseModel

{

public string? Title { get; set; }

public DateTime Date { get; set; }

public DayKind Kind { get; set; }

public int WorkHours { get; set; }

public Guid CompanyId { get; set; }

}

}