

Міністерство освіти і науки України
Одеський національний політехнічний університет
Інститут комп'ютерних систем
Кафедра інформаційних систем

КУРСОВА РОБОТА

з дисципліни: «Технології створення програмних продуктів»

за темою: «Розробка органайзеру «Catch Everything»

Частина 2

Виконав:
студент 3-го курсу
групи AI-185
Вознюк Д.В.
Перевірив:
Блажко О. А.

Одеса-2020

Анотація

В курсовій роботі розглядається процес створення програмного продукту «Catch Everything». Робота виконувалась в команді з декількох учасників:

1. Волков С.В.;
2. Вознюк Д.В.;
3. Марков Д.

Тому в пояснювальній записці у розділах «Проектування» та «Конструювання» детальніше описано лише одну частину з урахуванням планів проведених робіт з розділу «Планування» з описом особливостей конструювання:

- структур даних реляційної моделі в системі керування базами даних pgAdmin4\$

- програмних модулів в інструментальному середовищі MVC з використанням фреймворку Spring boot та мови програмування Java.

Результати роботи розміщено на github-репозиторії за адресою:

<https://github.com/DavidMarkov/TCSP>

Перелік скорочень

ОС – операційна система

ІС – інформаційна система

БД – база даних

СКБД – система керування базами даних

ПЗ – програмне забезпечення

ПП – програмний продукт

UML – уніфікована мова моделювання

Зміст

1 Вимоги до програмного продукту	6
1.1 Визначення потреб споживача	6
1.1.1 Ієрархія потреб споживача	6
1.1.2 Деталізація матеріальної потреби	7
1.2 Бізнес-вимоги до програмного продукту.....	7
1.2.1 Опис проблеми споживача	7
1.2.1.1 Концептуальний опис проблеми споживача	7
1.2.1.2 Метричний опис проблеми споживача	7
1.2.2 Мета створення програмного продукту.....	7
1.2.2.1 Проблемний аналіз існуючих програмних продуктів.....	8
1.2.2.2 Мета створення програмного продукту.....	8
1.3 Вимоги користувача до програмного продукту.....	8
1.3.1 Історія користувача програмного продукту.....	8
1.3.2 Діаграма прецедентів програмного продукту	9
1.3.3 Сценаріїв використання прецедентів програмного продукту.....	9
1.4 Функціональні вимоги до програмного продукту.....	14
1.4.1. Багаторівнева класифікація функціональних вимог.....	14
1.4.2 Функціональний аналіз існуючих програмних продуктів	16
1.5 Нефункціональні вимоги до програмного продукту	17
1.5.1 Опис зовнішніх інтерфейсів.....	17
1.5.1.1 Опис інтерфейса користувача	17
1.5.1.1.1 Опис INPUT-інтерфейса користувача	17
1.5.1.1.2 Опис OUTPUT-інтерфейса користувача	18
1.5.1.2 Опис інтерфейсу із зовнішніми пристроями	21
1.5.1.3 Опис програмних інтерфейсів.....	21
1.5.1.4 Опис інтерфейсів передачі інформації.....	21
1.5.1.5 Опис атрибутів продуктивності	21
2.3 План розробки програмного продукту.....	22
2.3.1 Оцінка трудомісткості розробки програмного продукту	22
2.3.2 Визначення дерева робіт з розробки програмного продукту	24
2.3.3 Графік робіт з розробки програмного продукту	25
2.3.3.1 Таблиця з графіком робіт	25
2.3.3.2 Діаграма Ганта.....	26
3 Проектування програмного продукту	28
3.1 Концептуальне та логічне проектування структур даних програмного продукту	28

3.1.1 Концептуальне проектування на основі UML-діаграми концептуальних класів.....	28
3.1.2 Логічне проектування структур даних.....	28
3.2 Проектування програмних класів.....	29
3.3 Проектування алгоритмів роботи методів програмних класів	29

1 Вимоги до програмного продукту

1.1 Визначення потреб споживача

1.1.1 Ієрархія потреб споживача

Відомо, що в теорії маркетингу потреби людини можуть бути представлені у вигляді ієрархії потреб ідей американського психолога Абрахама Маслоу включають рівні:

- фізіологія (вода, їжа, житло, сон);
- безпека (особиста, здоров'я, стабільність),
- приналежність (спілкування, дружба, любов),
- визнання (повага оточуючих, самооцінка),
- самовираження (вдосконалення, персональний розвиток).

На рисунку 1.1 представлена ієрархія потреби споживача. Хотілося б задовольнити фізіологічні потреби, використовуючи майбутній програмний продукт.



Рис. 1.1 – Приклад ієрархії потреби споживача

1.1.2 Деталізація матеріальної потреби

Для деталізації матеріальних потреб була розроблена MindMap, що представлена на рисунку 1.2.

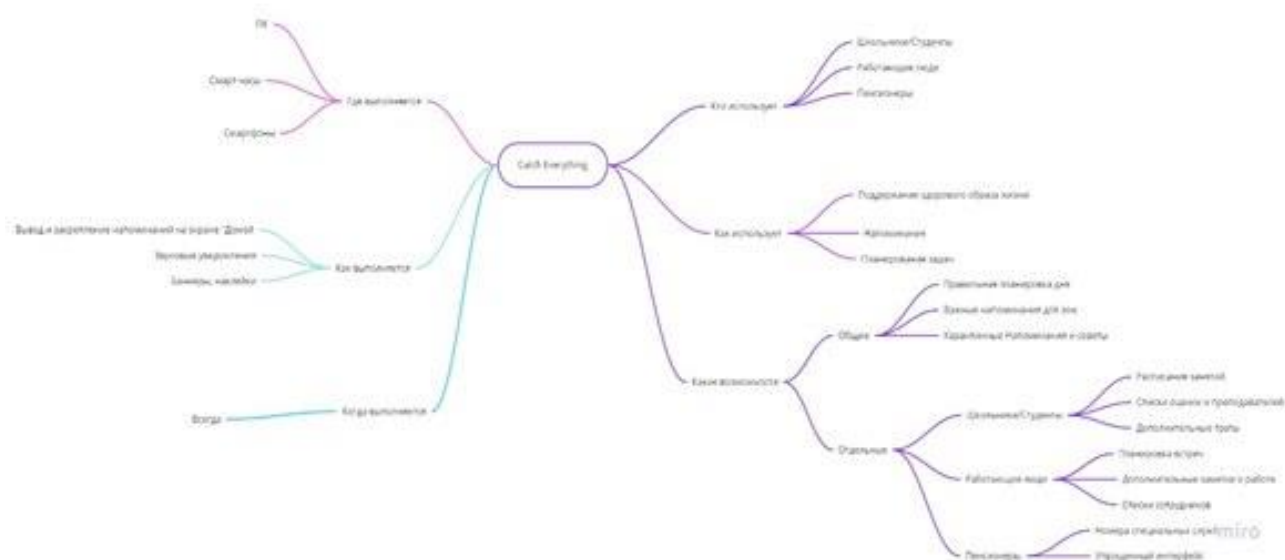


Рис. 1.2 – MindMap

1.2 Бізнес-вимоги до програмного продукту

1.2.1 Опис проблеми споживача

1.2.1.1 Концептуальний опис проблеми споживача

Неможливо запам'ятати:

1. Всі майбутні заплановані події;
2. З точністю стежити за водним балансом;
3. Адреса та номери телефонів та емейлів;
4. Розклад занять, рецепти блюд та т.д.

1.2.1.2 Метричний опис проблеми споживача

Метричний показник: Низький рівень продуктивності через неможливість запам'ятовування всієї інформації.

Рівень продуктивності LP (LP – the Level of Productivity) можна визначити як:

$$LP = N_x / N, \text{ де}$$

N_x – кількість потрібної інформації, яку користувач пам'ятає;

N – кількість всієї потрібної інформації.

1.2.2 Мета створення програмного продукту

1.2.2.1 Проблемний аналіз існуючих програмних продуктів

Таблиця 1.2.2.1 – Аналіз існуючих ПП

№	Назва продукту	Вартість	Ступінь готовності	Примітка
1	Органайзер Calendars	Безкоштовно/ платно	1	Просунуте додаток, підійде лише людям працюючим у великих компаніях
2	Щоденник Tappsk	Безкоштовно/ платно	1	Цікава система звичок, зручний інтерфейс
3	Список справ і завдань Todoist	Безкоштовно/ платно	1	Можливість розставляти пріоритети, захищений початковий екран

1.2.2.2 Мета створення програмного продукту

Метою роботи є створення органайзера з зручним інтуїтивно-зрозумілим користувацьким інтерфейсом. Підвищення продуктивності шляхом запису всієї необхідної інформації, яка при потребі буде виводиться у вигляді нагадувань.

1.3 Вимоги користувача до програмного продукту

1.3.1 Історія користувача програмного продукту

1. Як користувач, я можу управляти завданнями (створення, редагування, видалення).
2. Як користувач, я можу управляти подією (створення, редагування, видалення).
3. Як користувач, я можу управляти телефонною книгою (створення, редагування, видалення).
4. Як користувач, я можу управляти нотатками (створення, редагування, видалення).
5. Як користувач, я можу отримувати нагадування про карантинні заходи.
6. Як користувач, я можу управляти особистими записами (особистий щоденник).
7. Як користувач, я можу використовувати менеджер паролів, для збереження паролів.
8. Як користувач, я можу купити підписку на преміум, для відключення реклами.

9. Як гость, я можу зареєструватися у додатку

1.3.2 Діаграма прецедентів програмного продукту

Діаграма прецедентів зображена на рисунку 1.3.2.

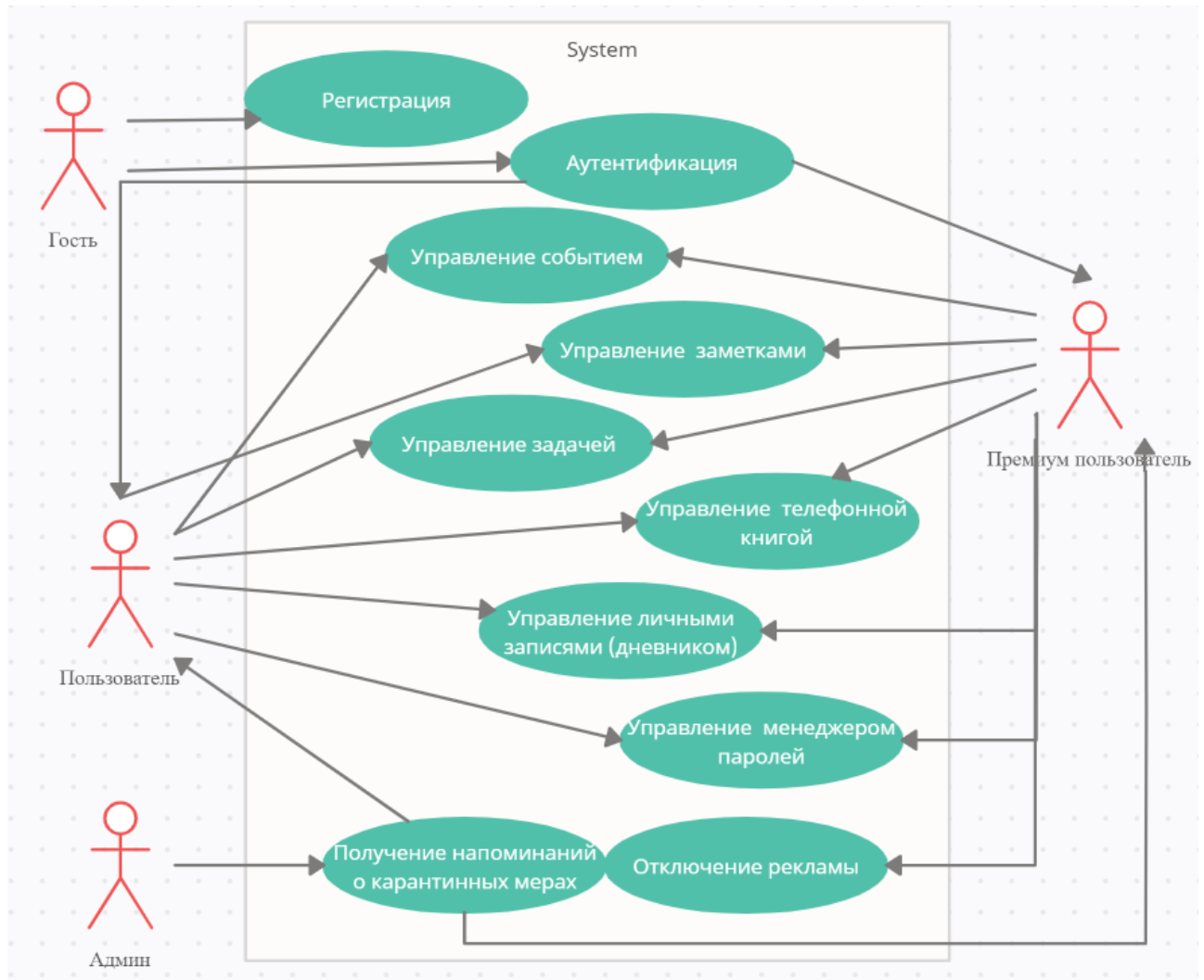


Рис. 1.3.2 – Діаграма прецедентів

1.3.3 Сценарії використання прецедентів програмного продукту

Сценарій №1.

Назва прецеденту: «Управління подією».

Передумова початку виконання сценарію: Потреба у створенні, редагуванні і видаленні події.

Актор: Користувач / Преміум-користувач.

Гарант успіху: Додавання нового події і можливість його редагування.

Приклад основного успішного сценарію прецеденту «Управління подією»:

1. Користувач передає ПП текстове повідомлення з точним зазначенням часу його настання для додавання події.

2. ПП зберігає подія і дає можливість редагувати або видалити його.
Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Управління подією»:

3.1 ПП виявляє, що користувач не вказав час настання події.

3.а.1 ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №2.

Назва прецеденту: «Управління завданням».

Передумова початку виконання сценарію: Потреба у створенні, редагуванні і видаленні завдання.

Актор: Користувач / Преміум-користувач.

Гарант успіху: Додавання нового завдання і можливість її редагування.

Приклад основного успішного сценарію прецеденту «Управління завданням»:

2. Користувач передає ПП опис справ, які йому необхідно виконати для додавання завдання.

2. ПП зберігає завдання і дає можливість редагувати або видалити її.

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Управління завданням»:

3.1 ПП виявляє, що користувач не вказав жодної справи.

3.а.1 ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №3.

Назва прецеденту: «Отримання нагадування про карантинні заходи».

Передумова початку виконання сценарію: Запобігання поширенню вірусної інфекції.

Актор: ПП.

Гарант успіху: Оповіщення користувача.

Приклад основного успішного сценарію прецеденту «Отримання нагадування про карантинні заходи»:

1. ПП відправляє користувачеві текстове повідомлення про необхідність дотримання карантинних заходів з певним інтервалом часу.

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Отримання нагадування про карантинні заходи»:

1.1. Користувач не отримує повідомлення про дотримання карантинних заходів через те, що відключив раніше цю функцію.

Сценарій №4.

Назва прецеденту: «Управління замітками».

Передумова початку виконання сценарію: Потреба у створенні, редагуванні і видаленні замітки.

Актор: Користувач / Преміум-користувач.

Гарант успіху: Додавання нової замітки і можливість її редагування.

-Приклад основного успішного сценарію прецеденту «Управління замітками»:

1. Користувач передає ПП текстове повідомлення для додавання замітки.
2. ПП зберігає замітку і дає можливість редагувати або видалити її.

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Управління замітками»:

3.1 ПП виявляє, що користувач нічого не заповнив.

3.а.1 ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №5.

Назва прецеденту: «Управління телефонною книгою».

Передумова початку виконання сценарію: Потреба в запису, редагуванні і видаленні номера телефону.

Актор: Користувач / Преміум-користувач.

Гарант успіху: Додавання нового запису в телефонній книзі і можливість її редагування.

Приклад основного успішного сценарію прецеденту «Управління телефонною книгою»:

1. Користувач передає ПП контактні дані для додавання номера телефону.
2. ПП зберігає запис з номером телефону і дає можливість редагувати або видалити його.

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Управління телефонною книгою»:

3.1 ПП виявляє, що користувач передав ПП не всі необхідні контактні дані.

3.а.1 ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №6.

Назва прецеденту: «Управління особистими записами (щоденником)».

Передумова початку виконання сценарію: Потреба у веденні, редагуванні і видаленні особистих записів.

Актор: Користувач / Преміум-користувач.

Гарант успіху: Додавання новий особистий записи і можливість її редагування.

Приклад основного успішного сценарію прецеденту «Управління особистими записами (щоденником)»:

1. Користувач передає ПП текстове повідомлення із зазначенням часу його додавання для додавання особистої записи.

2. ПП зберігає особисту запис і дає можливість редагувати або видалити її.
Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Управління особистими записами (щоденником)»:

3.1 ПП виявляє, що користувач не вказав час.

3.а.1 ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №7.

Назва прецеденту: «Управління менеджером паролів».

Передумова початку виконання сценарію: Потреба в запису, редагування і видалення паролів.

Актор: Користувач / Преміум-користувач.

Гарант успіху: Додавання нового паролі і можливість його редагування.

Приклад основного успішного сценарію прецеденту «Управління менеджером паролів»:

1. Користувач передає ПП умовне слово для додавання пароля.

2. ПП зберігає пароль і дає можливість редагувати або видалити його.
Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Управління менеджером паролів»:

3.1 ПП виявляє, що користувач нічого не заповнив.

3.а.1 ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №8.

Назва прецеденту: «Відключення реклами».

Передумова початку виконання сценарію: Потреба в запису, редагування і видалення паролів.

Актор: Користувач.

Гарант успіху: Використання організатора без реклами.

Приклад основного успішного сценарію прецеденту «Відключення реклами»:

1. Користувач передає дані про свою карту і оплачує замовлення на покупку преміум.

2. ПП обробляє дані.

3. ПП змінює роль користувача на «Преміум-користувач» і відправляє квитанцію про оплату на пошту користувача.

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Відключення реклами»:

3.1 ПП виявляє, що користувач передав ПП не всі дані карти.

3.а.1. ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

3.2. ПП виявляє, що у користувача недостатня сума на рахунку.

3.а.2. ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №9.

Назва прецеденту: «Аутентифікація».

Передумова початку виконання сценарію: Потреба у вході в свою особисту сторінку.

Актор: Гість.

Гарант успіху: Вхід на особисту сторінку.

Приклад основного успішного сценарію прецеденту «Аутентифікація»:

1. Користувач вводить логін і пароль.

2. ПП обробляє дані.

3. ПП вітає користувача про успішне вході на особисту сторінку

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Аутентифікація»:

3.1 ПП виявляє, що користувач передав ПП не коректний логін або пароль, або логін і пароль.

3.а.1. ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

Сценарій №10.

Назва прецеденту: «Реєстрація».

Передумови початку виконання сценарію: Потреба у створенні особистої сторінки.

Актор: Гість.

Гарантії успіху: Реєстрація користувача.

Приклад основного успішного сценарію прецеденту «Реєстрація»:

1. Користувач вводить необхідні дані для реєстрації.

2. ПП обробляє дані.

3. ПП вітає користувача про успішну реєстрацію.

Приклад альтернативного сценарію для попереднього прикладу основного успішного сценарію прецеденту «Реєстрація»:

3.1 ПП виявляє, що користувач передав ПП не коректні дані.

3.а.1. ПП видає повідомлення про помилку і переходить до кроку 1 основного успішного сценарію.

1.4 Функціональні вимоги до програмного продукту

1.4.1. Багаторівнева класифікація функціональних вимог

Багаторівнева класифікація функціональних вимог представлена в таблиці 1.4.1. На рисунку 1.4.1 опис ієрархічної класифікація функціональних вимог.

Таблиця 1.4.1. Багаторівнева класифікація функціональних вимог

Ідентифікатор функції	Назва функції
FR1	Управління подіями(д)/ завданнями(с)/ нотатками(Д) / паролями(д) / номерами(С)
FR1.1	Користувач переходить у розділ Події/Менеджер паролів і вибирає дію "Додати/змінити подію/пароль"
FR1.2	Користувач надсилає/редагує необхідні дані ПП для додавання/змінення елемента
FR1.3	ПП зберігає внесені дані
FR2	Управління Завданнями/Номерами
FR2.1	Користувач переходить у розділ Завданнями/Телефона книга і вибирає дію "Додати/змінити завдання/запис в телефонії книзі"
FR2.2	Користувач надсилає/редагує необхідні дані ПП для додавання/змінення елемента
FR2.3	ПП зберігає внесені дані
FR3	Управління Нотатками/Особистим щоденником
FR3.1	Користувач переходить у розділ Нотатки/Особистий щоденник і вибирає дію "Додати/змінити Нотаток/Запис в особистому щоденнику"
FR3.2	Користувач надсилає/редагує необхідні дані ПП для додавання/змінення елемента

FR3.3	ПП зберігає внесені дані
FR4	Реєстрація
FR4.1	Користувач переходить у розділ Реєстрація
FR4.2	Користувач надсилає необхідні дані ПП для реєстрації
FR4.3	ПП зберігає дані користувача
FR5	Аутентифікація
FR5.1	Користувач переходить у вікно «Вхід»
FR5.2	Користувач надсилає логін та пароль ПП для входу
FR5.3	ПП перевіряє логін та пароль на відповідність в БД
FR5.4	Користувач заходить на особисту сторінку
FR6	Отримання нагадування про карантинні заходи
FR6.1	ПП надсилає користувачеві текстове повідомлення про те, що карантинних заходів потрібно дотримуватися через певний проміжок часу
FR7	Вимкнення реклами
FR7.1	Користувач заходить в розділ «Придбати Преміум» і вибирає акцію «придбати преміум».
FR7.2	Користувач передає необхідні дані для придбання преміум
FR7.3	ПП обробляє дані
FR7.4	ПП змінює роль користувача на "Преміум користувач" і відправляє квитанцію про оплату поштою користувача.

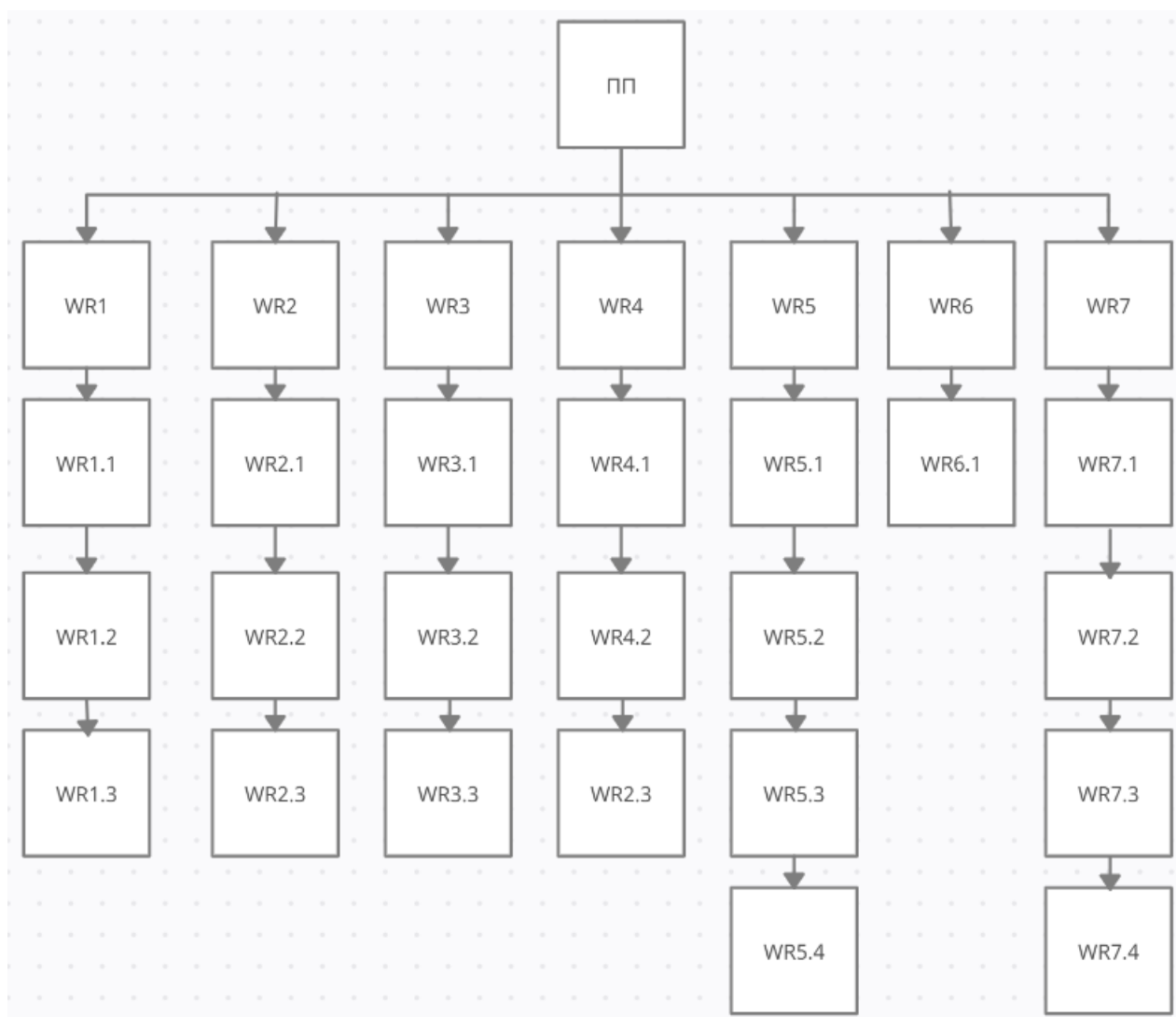


Рис. 1.4.1 – Опис ієрархічної класифікація функціональних вимог

1.4.2 Функціональний аналіз існуючих програмних продуктів

Таблиця 1.4.2 – Функціональний аналіз існуючих програмних продуктів

Ідентифікатор функції	onlinemschool.com	yaclass.ru	geogebra.org
FR1	≈	≈	≈
FR2	-	-	-
FR3	+	+	+
FR4	+	-	-
FR5	+	+	+
FR6	-	-	-

Продовження таблиці 1.4.2 – Функціональний аналіз існуючих програмних продуктів

FR7	+	-	+
-----	---	---	---

1.5 Нефункціональні вимоги до програмного продукту

1.5.1 Опис зовнішніх інтерфейсів

1.5.1.1 Опис інтерфейса користувача

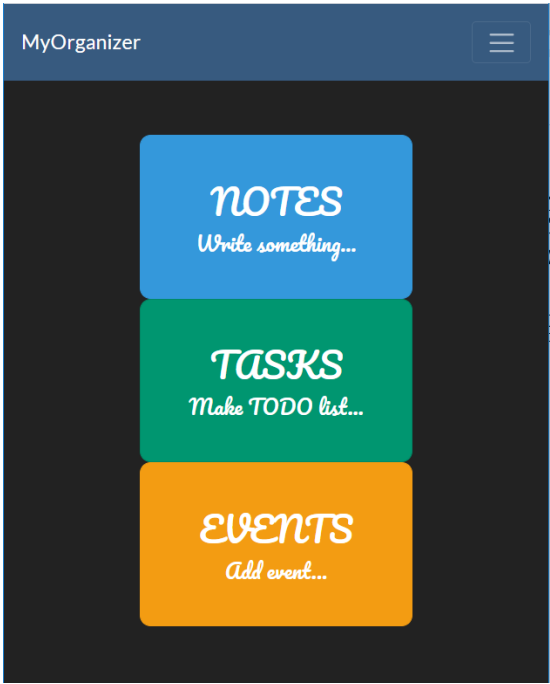
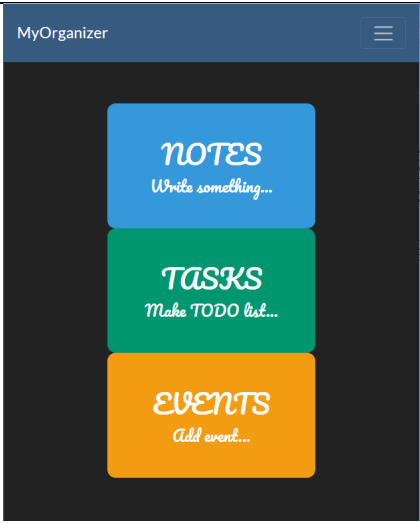
1.5.1.1.1 Опис INPUT-інтерфейса користувача

Таблиця 1.5.1.1.1 – Опис INPUT-інтерфейса користувача

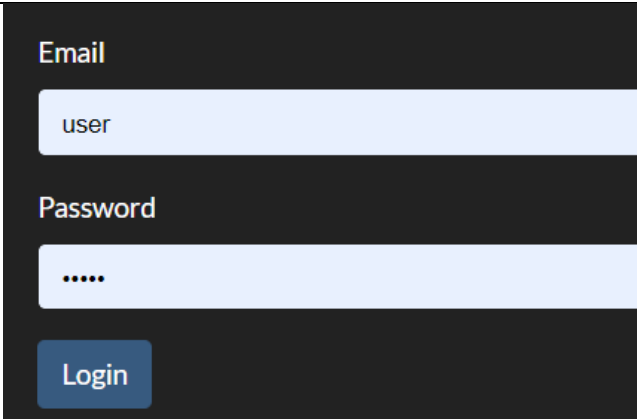
Ідентифікатор функції	Засіб INPUT потоку	Особливості використання
FR1	Стандартна клавіатура і миша, сенсорний екран"	
FR2	Стандартна клавіатура і миша, сенсорний екран	
FR3	Стандартна клавіатура і миша, сенсорний екран	
FR4	Стандартна клавіатура і миша, сенсорний екран	
FR5	Стандартна клавіатура і миша, сенсорний екран	
FR7	Стандартна клавіатура і миша, сенсорний екран	

1.5.1.1.2 Опис OUTPUT-інтерфейса користувача

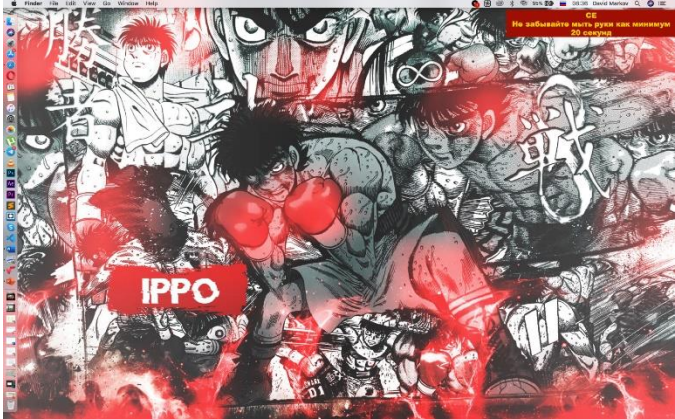
Таблиця 1.5.1.1.2 – Опис OUTPUT-інтерфейса користувача

Ідентифікатор функції	Засіб OUTPUT потоку	Особливості використання
FR1	Графічний інтерфейс	
FR2		

Продовження таблиці 1.5.1.1.2 – Опис OUTPUT-інтерфейса користувача

FR3		
FR4		
FR5		

Продовження таблиці 1.5.1.1.2 – Опис OUTPUT-інтерфейса користувача

FR6	Графічний інтерфейс	
FR7	Графічний інтерфейс	

1.5.1.2 Опис інтерфейсу із зовнішніми пристроями

Таблиця 1.5.1.2 – Опис інтерфейсу із зовнішніми пристроями

Ідентифікатор функції	Зовнішній пристрій
FR1	Смартфон, ПК, годинник
FR2	Смартфон, ПК, годинник
FR3	Смартфон, ПК, годинник
FR4	Смартфон, ПК, годинник
FR5	Смартфон, ПК, годинник
FR6	Смартфон, ПК, годинник
FR7	Смартфон, ПК, годинник

1.5.1.3 Опис програмних інтерфейсів

Для реалізації більшості функцій програмного продукту необхідно:

- HTML.
- CSS.
- JavaScript.
- Java.

1.5.1.4 Опис інтерфейсів передачі інформації

Рекомендується використовувати

Провідні інтерфейси:

1. Ethernet
2. GigabitEthernet

Бездротові інтерфейси:

1. Wi-Fi

1.5.1.5 Опис атрибутів продуктивності

Ідентифікатор функції	Максимальний час реакції програмного продукту на дії користувача(секунди)
FR1	2
FR2	3
FR3	3
FR4	4
FR5	2
FR6	2
FR7	8

2.3 План розробки програмного продукту

2.3.1 Оцінка трудомісткості розробки програмного продукту

Всі актори діляться на три типи: прості, середні і складні. Простий актор представляє зовнішню систему з чітко визначеним програмним інтерфейсом. Середній актор представляє або зовнішню систему, що взаємодіє з ПП за допомогою мережових протоколів, або особистість, що користується текстовим інтерфейсом (наприклад, алфавітно-цифровим терміналом). Складний актор представляє особистість, що користується графічним інтерфейсом. Загальна кількість акторів кожного типу помножується на відповідний ваговий коефіцієнт, потім обчислюється загальний ваговий показник(рис. 2.3.1.1).

Тип актора	Ваговий коефіцієнт
Простий	1
Середній	2
Складний	3

Рис. 2.3.1.1 – Вагові коефіцієнти акторів

Всі прецеденти діляться на три типи; прості, середні і складні в залежності від кількості кроків успішних сценаріїв (основних і альтернативних). Загальна кількість прецедентів кожного типу помножується на відповідний ваговий коефіцієнт, потім обчислюється загальний ваговий показник (рис. 2.3.1.2).

Тип прецедента	Кількість кроків сценарію	Ваговий коефіцієнт
Простий	≤ 3	5
Середній	4-7	10
Складний	> 7	15

Рис. 2.3.1.2 – Вагові коефіцієнти прецедентів

$$UCPP = 3 \cdot 1 + 8 \cdot 5 = 43$$

Технічна складність проекту (TCF – Technical Complexity Factor) обчислюється з урахуванням показників технічної складності (табл. 2.3.1.1). Кожному показнику присвоюється значення STi в діапазоні від 0 до 5: 0 означає відсутність значимості показника для даного проекту, 5 - високу значимість). Значення TCF обчислюється за формулою – $TCF = 0,6 + (0,01 * (ST_i * Вага_i))$

Таблиця 2.3.1.1 – TCF-таблиця

Показник	Опис показника	STi	Вага
T1	Розподільна система	0	2
T2	Висока продуктивність	3	1
T3	Робота користувачів онлайн	3	1
T4	Складна обробка даних	1	-1
T5	Повторне використання коду	4	1
T6	Простота інсталювання	5	0.5
T7	Простота використання	5	0.5
T8	Переносимість	1	2
T9	Простота внесення змін	1	1
T10	Паралелізм	0	1
T11	Спеціальні вимоги безпеки	2	1
T12	Беспосередній доступ до системи зі сторони зовнішніх користувачів	0	1
T13	Спеціальні вимоги до навчання користувачів	0	1

$$TCF = 0.79$$

Рівень кваліфікації розробників (EF - Environmental Factor) обчислюється з урахуванням наступних показників (табл. 2.3.1.2).

Таблиця 2.3.1.2 – Рівень кваліфікації розробників

Показник	Опис показника	STi	Вага
F1	Знайомство з технологією	2	1.5
F2	Досвід розробки додатків	0	0.5
F3	Досвід використання ООП	0	1
F4	Наявність провідного аналітика	2	0.5
F5	Мотивація	2	1
F6	Стабільність вимог	5	2

F7	Часткова зайнятість	2	-1
F8	Складні мови програмування	1	-1

EF = 1.01

UCP = 33.51

2.3.2 Визначення дерева робіт з розробки програмного продукту

При створенні дерева робіт (Work BreakDown Structure- WBS) використовується дерево функцій.

Кожна функція 1-го рівня ієрархії перетворюється в Work Package (WP)

Кожна функція 2-го рівня ієрархії перетворюється в Work Task (WT).

Для кожної задачі визначаються підзадачі - Work Sub Task (WST) з урахуванням базових процесів розробки програмних модулів: проектування, конструювання, модульне тестування, збірка та системне тестування(рис. 2.3.2).

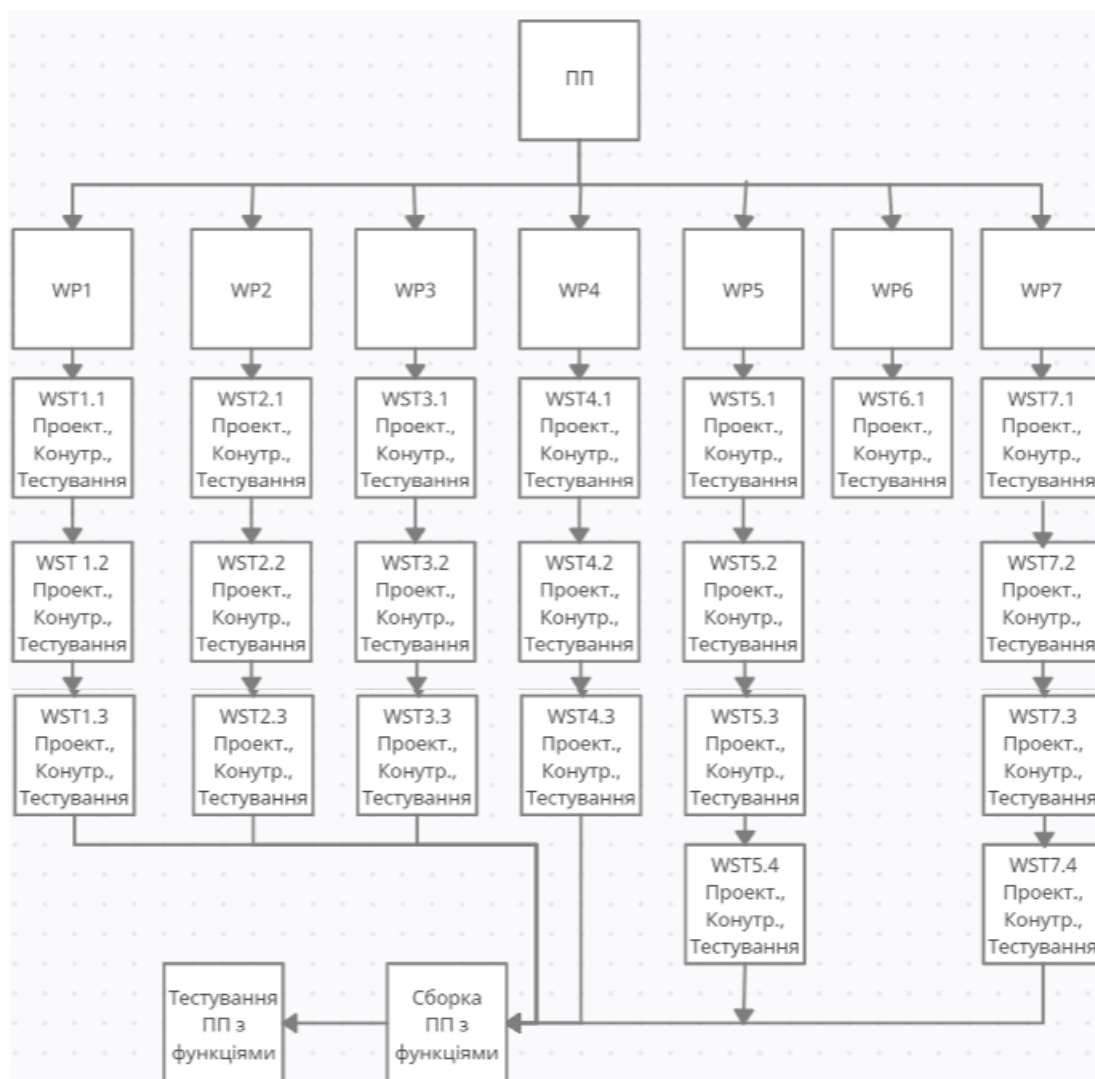


Рис. 2.3.2 – Дерево робіт з розробки ПП

2.3.3 Графік робіт з розробки програмного продукту

2.3.3.1 Таблиця з графіком робіт

Для кожної підзадачі визначається виконавець, що фіксується у вигляді таблиці, приклад якої представлено в таблиці 2.3.3.1.

Таблиця 2.3.3.1 – Таблиця графіку робіт

WST	Дата початку	Дні	Дата завершення	Виконавець
WST 1	17.10.20	3	20.10.20	Марков Д.
WST 1.1	20.10.20	3	23.10.20	Марков Д.
WST 1.2	23.10.20	4	27.10.20	Марков Д.
WST 1.3	27.10.20	3	30.10.20	Марков Д.
WST 2	30.10.20	3	02.11.20	Волков С.
WST 2.1	02.10.20	4	06.11.20	Волков С.
WST 2.2	06.10.20	3	09.11.20	Волков С.
WST 2.3	09.11.20	3	12.11.20	Волков С.
WST 3	12.11.20	4	16.11.20	Вознюк Д.
WST 3.1	17.10.20	3	20.10.20	Вознюк Д.
WST 3.2	20.10.20	3	23.10.20	Вознюк Д.
WST 3.3	23.10.20	4	27.10.20	Вознюк Д.
WST 4	27.10.20	3	30.10.20	Волков С.
WST 4.1	30.10.20	3	02.11.20	Волков С.
WST 4.2	02.10.20	4	06.11.20	Волков С.
WST 4.3	06.10.20	3	09.11.20	Волков С.
WST 5	09.11.20	3	12.11.20	Вознюк Д.
WST 5.1	12.11.20	4	16.11.20	Вознюк Д.
WST 5.2	17.10.20	3	20.10.20	Вознюк Д.
WST 5.3	20.10.20	3	23.10.20	Вознюк Д.
WST 5.4	23.10.20	4	27.10.20	Вознюк Д.
WST 6	27.10.20	3	30.10.20	Марков Д.
WST 6.1	30.10.20	3	02.11.20	Марков Д.
WST 7	02.10.20	4	06.11.20	Марков Д.
WST 7.1	06.10.20	3	09.11.20	Вознюк Д.
WST 7.2	09.11.20	3	12.11.20	Волков С.
WST 7.3	12.11.20	4	16.11.20	Волков С.
WST 7.4	17.10.20	3	20.10.20	Вознюк Д.

2.3.3.2 Діаграма Ганта

Діаграма Ганта (складається із смуг (вісь Y), орієнтованих уздовж осі часу (вісь X). Кожна смуга – окрема підзадача в проєкті, її кінці - моменти початку і завершення роботи, її протяжність - тривалість роботи. Мета діаграми - візуально показати послідовність процесів та можливість паралельного виконання робіт(таблиця 2.3.3.2.1).

Таблиця 2.3.3.2.1. Діаграма Ганта

W ST	Дата поча тку	Д ні	Дата заверш ення	Викона вець	17. 10	20. 10	23. 10	27. 10	30. 10	02. 11	06. 11	09. 11	12. 11
W ST 1	17.1 0.20	3	20.10.2 0	Марко в Д.									
W ST 1.1	20.1 0.20	3	23.10.2 0	Марко в Д.									
W ST 1.2	23.1 0.20	4	27.10.2 0	Марко в Д.									
W ST 1.3	27.1 0.20	3	30.10.2 0	Марко в Д.									
W ST 2	30.1 0.20	3	02.11.2 0	Волков С.									
W ST 2.1	02.1 0.20	4	06.11.2 0	Волков С.									
W ST 2.2	06.1 0.20	3	09.11.2 0	Волков С.									
W ST 2.3	09.1 1.20	3	12.11.2 0	Волков С.									
W ST 3	12.1 1.20	4	16.11.2 0	Возню к Д.									
W ST 3.1	17.1 0.20	3	20.10.2 0	Возню к Д.									

Продовження таблиці 2.3.3.2.1. Діаграма Ганта

W ST 3.2	20.1 0.20	3	23.10.2 0	Возню к Д.									
W ST 3.3	23.1 0.20	4	27.10.2 0	Возню к Д.									
W ST 4	27.1 0.20	3	30.10.2 0	Волков С.									
W ST 4.1	30.1 0.20	3	02.11.2 0	Волков С.									
W ST 4.2	02.1 0.20	4	06.11.2 0	Волков С.									
W ST 4.3	06.1 0.20	3	09.11.2 0	Волков С.									
W ST 5	09.1 1.20	3	12.11.2 0	Возню к Д.									
W ST 5.1	12.1 1.20	4	16.11.2 0	Возню к Д.									
W ST 5.2	17.1 0.20	3	20.10.2 0	Возню к Д.									
W ST 5.3	20.1 0.20	3	23.10.2 0	Возню к Д.									
W ST 5.4	23.1 0.20	4	27.10.2 0	Возню к Д.									
W ST 6	27.1 0.20	3	30.10.2 0	Марко в Д.									
W ST 6.1	30.1 0.20	3	02.11.2 0	Марко в Д.									
W ST 7	02.1 0.20	4	06.11.2 0	Марко в Д.									

Продовження таблиці 2.3.3.2.1. Діаграма Ганта

W ST 7.1	06.1 0.20	3	09.11.2 0	Возню к Д.									
W ST 7.2	09.1 1.20	3	12.11.2 0	Волков С.									
W ST 7.3	12.1 1.20	4	16.11.2 0	Волков С.									
W ST 7.4	17.1 0.20	3	20.10.2 0	Возню к Д.									

3 Проектування програмного продукту

3.1 Концептуальне та логічне проектування структур даних програмного продукту

3.1.1 Концептуальне проектування на основі UML-діаграми концептуальних класів

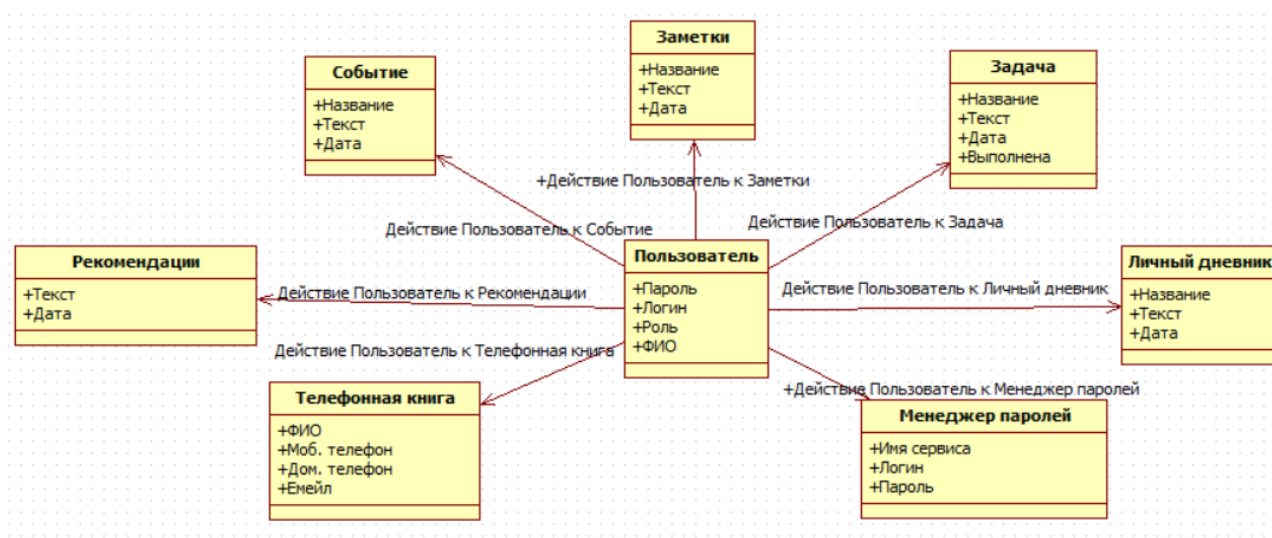


Рис. 3.1.1 – UML-діаграма концептуальних класів

3.1.2 Логічне проектування структур даних

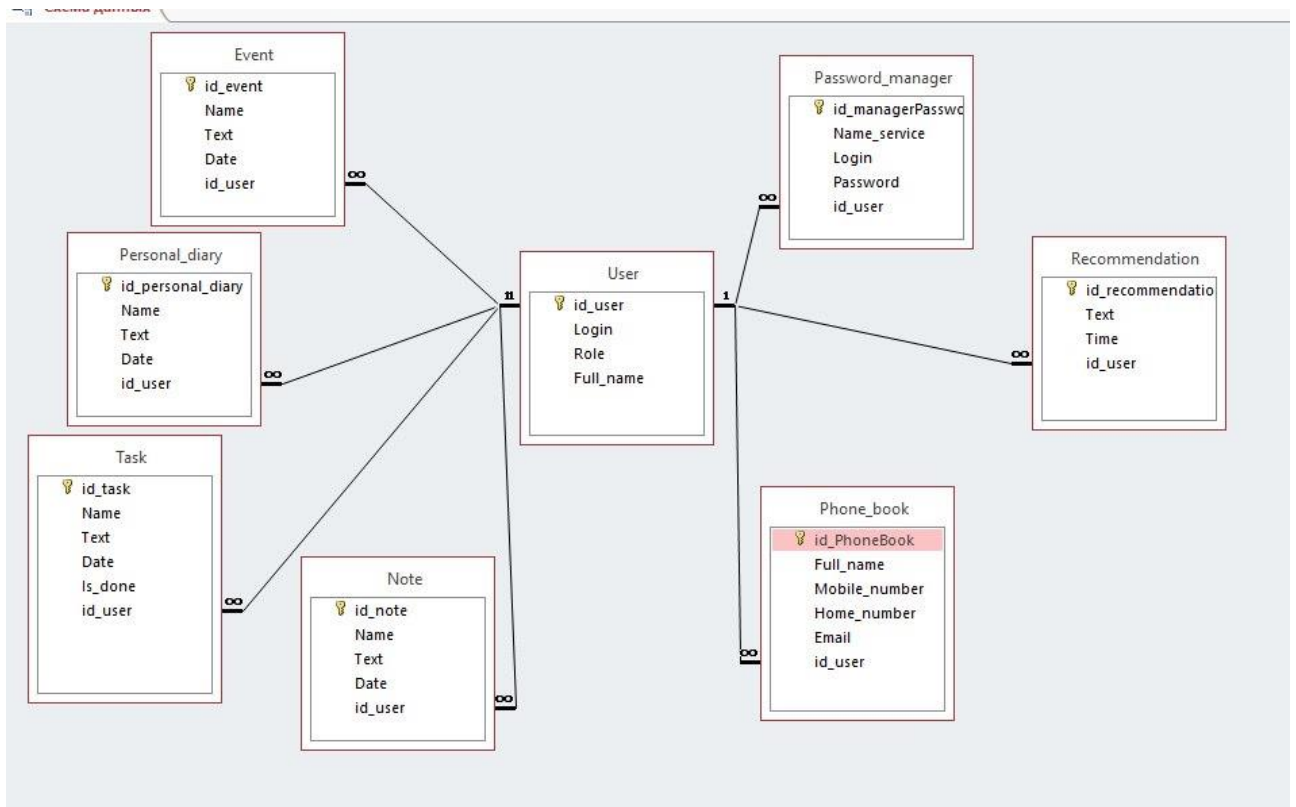


Рис. 3.1.2 – UML-діаграма структурних класів

3.2 Проектування програмних класів

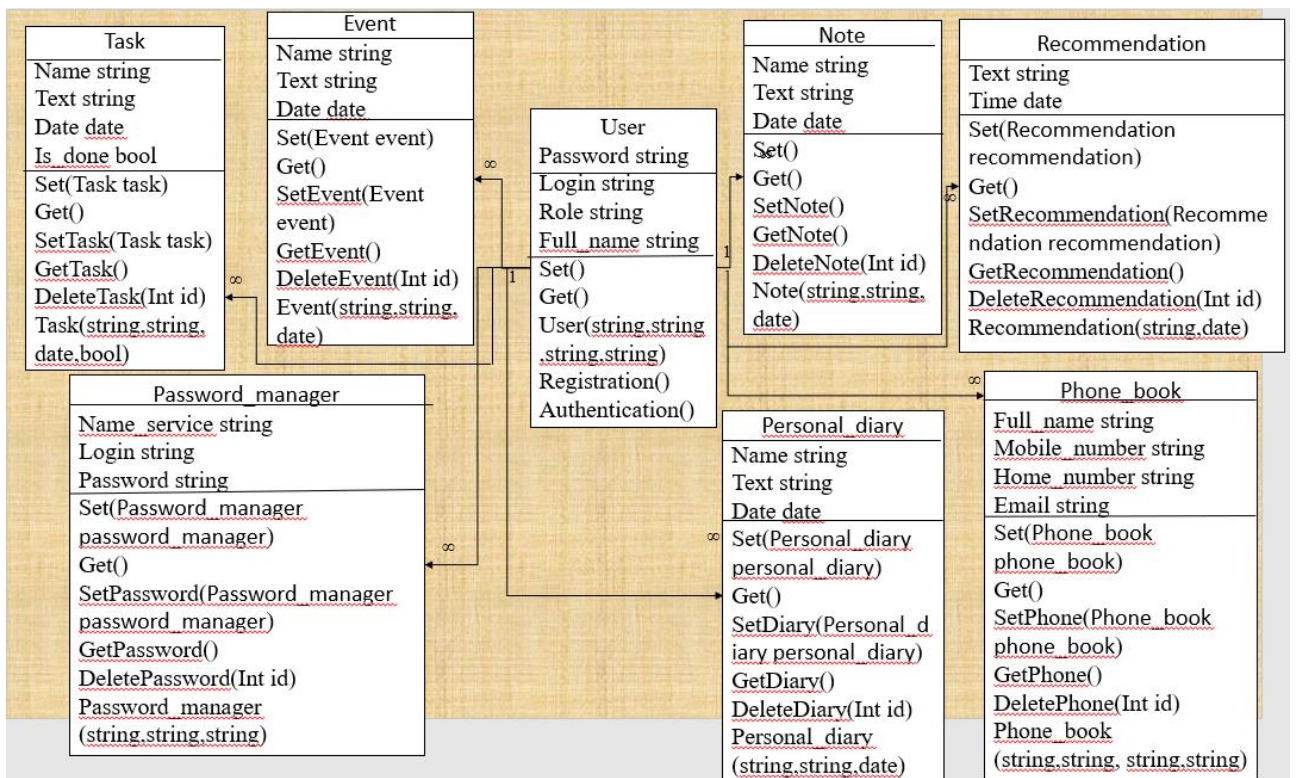


Рис. 3.2 – Діаграма програмних класів

3.3 Проектування алгоритмів роботи методів програмних класів

Алгоритм метода Authentication(string Login, string password) изображений на рисунку 3.3.1.

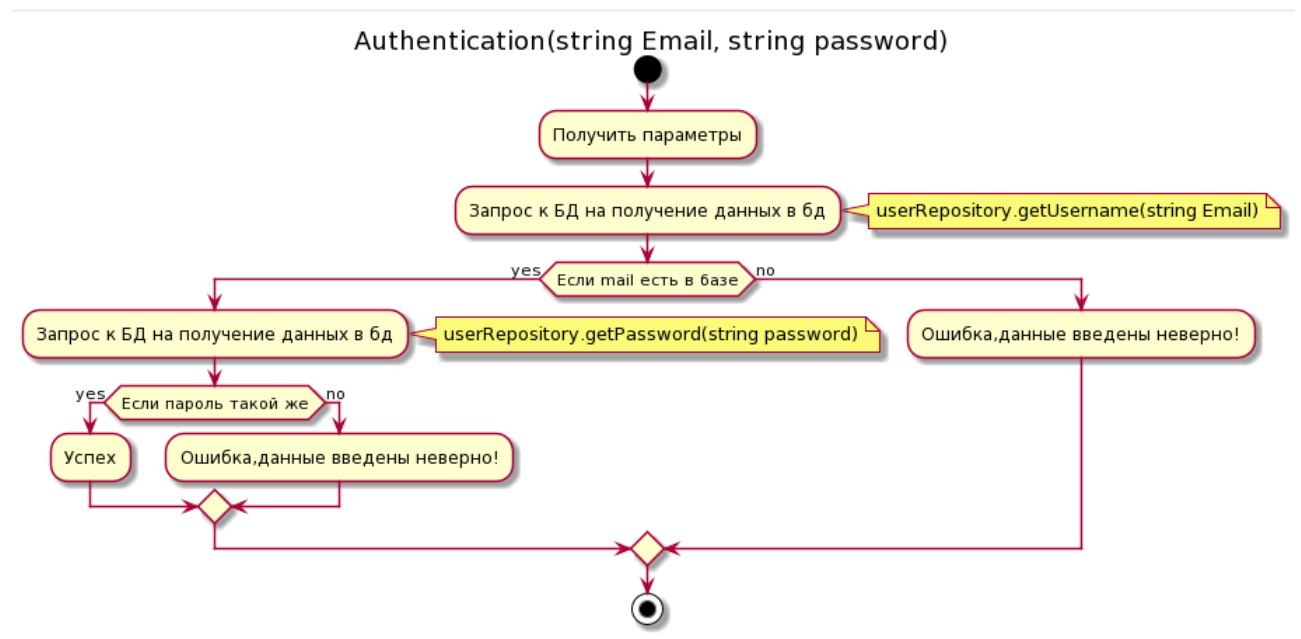


Рисунок 3.3.1 – Алгоритм метода Authentication(User user)

Код до метода Authentication(string Login, string password):

```

@startuml
start
title Authentication(string Email, string password)
:Получить параметры;
:Запрос к БД на получение данных в бд;
note right
userRepository.getUsername(string Email)
end note
if (Если mail есть в базе) then (yes)
:Запрос к БД на получение данных в бд;
note right
userRepository.getPassword(string password)
end note
if (Если пароль такой же) then (yes)
:Успех;
else (no)
:Ошибка, данные введены неверно!;
endif
else (no)

```

```

:Ошибка, данные введены неверно!;
endif
stop
@enduml

```

Алгоритм методу getNote() изображений на рисунку 3.3.2.

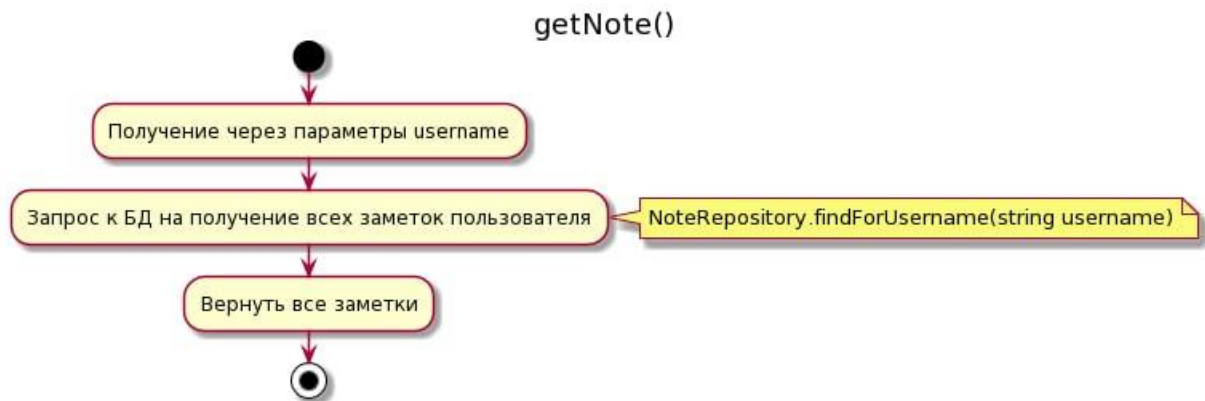


Рисунок 3.3.2 – Алгоритм методу getNote()

Код до методу getNote(string username):

```

@startuml
start
title getNote()
:Получение через параметры username;
:Запрос к БД на получение всех заметок пользователя;
note right
NoteRepository.findForUsername(string username)
end note
:Вернуть все заметки;
stop
@enduml

```

Алгоритм методу getDiary() изображений на рисунку 3.3.3.



Рисунок 3.3.3 – Алгоритм метода `getDiary()`

Код до метода `getDiary()`:

```
@startuml
start
title getDiary()
:Получение через параметры username;
:Запрос к БД на получение всех личных записей пользователя;
note right
DiaryRepository.findForUsername(string username)
end note
:Вернуть все личные записи пользователя;
stop
@enduml
```

Алгоритм метода `setNote(Note note)` изображений на рисунку 3.3.4.

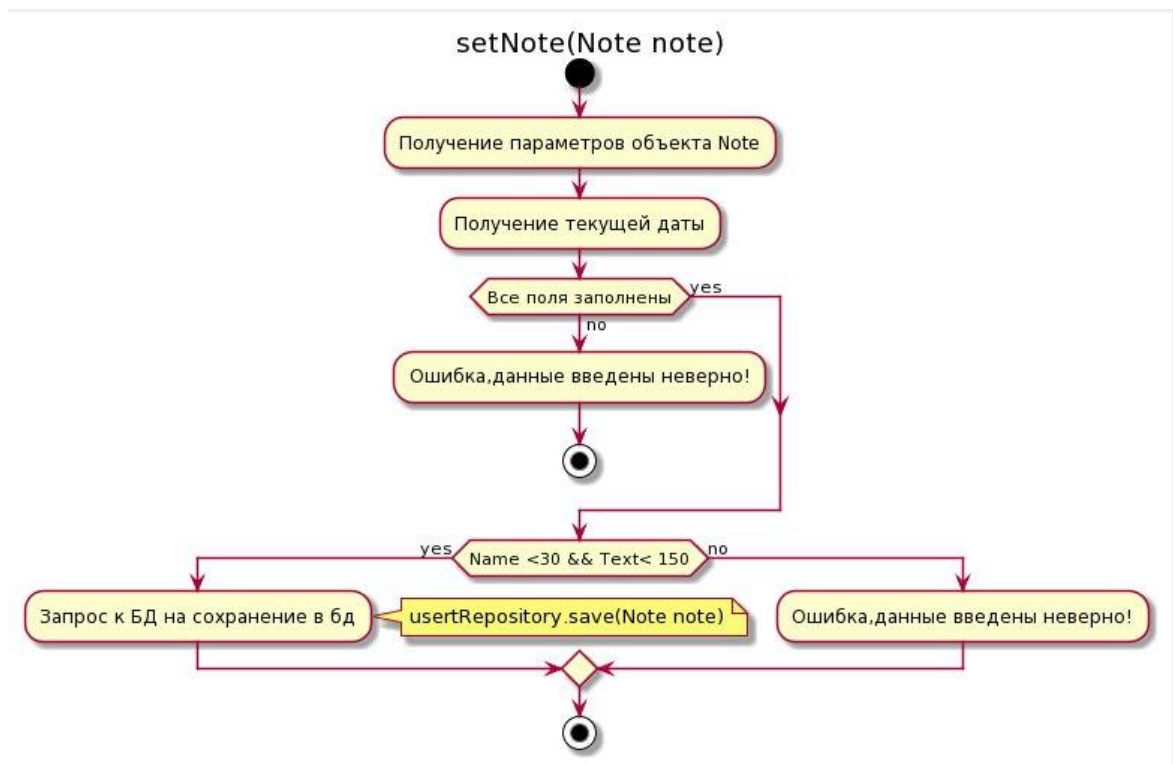


Рисунок 3.3.4 – Алгоритм метода setNote(Note note)

Код до метода setNote(Note note):

```

@startuml
start
title setNote(Note note)
:Получение параметров объекта Note;
:Получение текущей даты;
if (Все поля заполнены) then (yes)
else (no)
:Ошибка, данные введены неверно!;
stop
endif
if (Name <30 && Text < 150) then (yes)
:Запрос к БД на сохранение в бд;
note right
userRepository.save(Note note)
end note
else (no)
:Ошибка, данные введены неверно!;
endif
stop @enduml
  
```

Алгоритм метода getOneNoteForEdit(int id) изображений на рисунку 3.3.5

getOneNoteForEdit(int id)

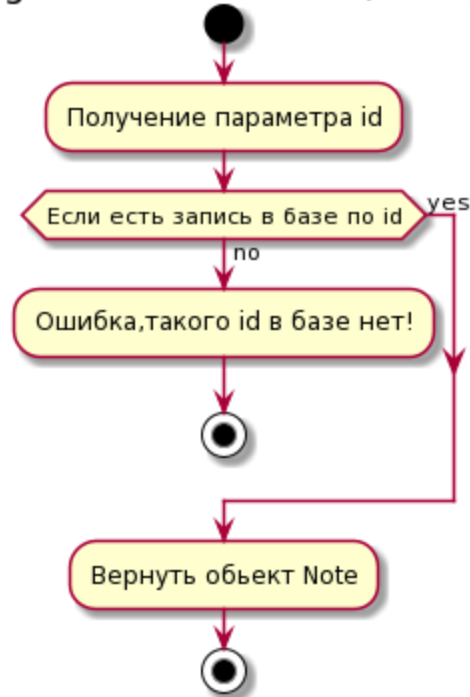


Рис. 3.3.5 - Алгоритм метода `getOneNoteForEdit(int id)`

Алгоритм метода `setPersonalDiary(PersonalDiary personalDiary)` изображений на рисунку 3.3.6.

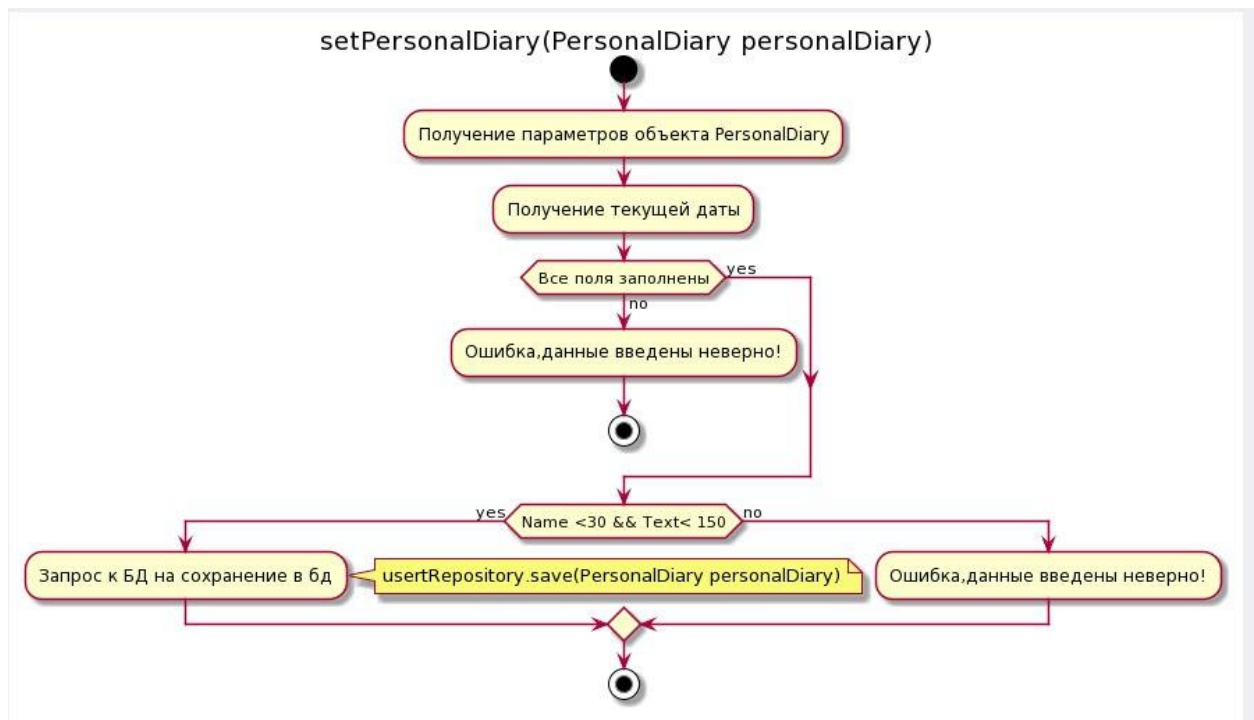


Рисунок 3.3.6 – Алгоритм метода `setPersonalDiary(PersonalDiary personalDiary)`

Код до метода `setPersonalDiary(PersonalDiary personalDiary)`:

```

@startuml
start
title setPersonalDiary(PersonalDiary personalDiary)
:Получение параметров объекта PersonalDiary;
:Получение текущей даты;
if (Все поля заполнены) then (yes)
else (no)
:Ошибка, данные введены неверно!;
stop
endif
if (Name < 30 && Text < 150) then (yes)
:Запрос к БД на сохранение в бд;
note right
userRepository.save(PersonalDiary personalDiary)
end note
else (no)
:Ошибка, данные введены неверно!;
endif
stop @enduml @enduml

```

Алгоритм метода setPasswordManager(PasswordManager passwordManager) изображений на рисунку 3.3.7.

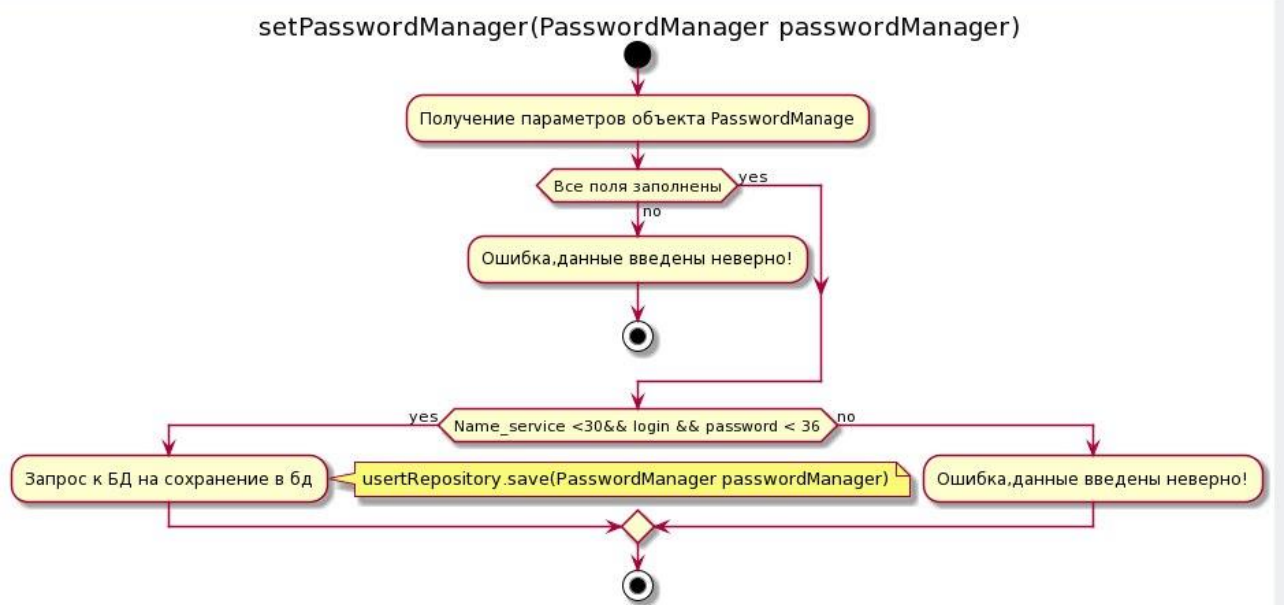


Рисунок 3.3.7 – Алгоритм метода setPasswordManager(PasswordManager passwordManager)

Код до метода setPasswordManager(PasswordManager passwordManager):

```

@startuml
start
title setPasswordManager(PasswordManager passwordManager)
:Получение параметров объекта PasswordManager;
if (Все поля заполнены) then (yes)
else (no)
:Ошибка, данные введены неверно!;
stop
endif
if (Name_service < 30 && login && password < 36) then (yes)
:Запрос к БД на сохранение в бд;
note right
userRepository.save(PasswordManager passwordManager)
end note
else (no)
:Ошибка, данные введены неверно!;
endif
stop
@enduml

```

Алгоритм методу DeleteNote(int id) изображений на рисунку 3.3.8.

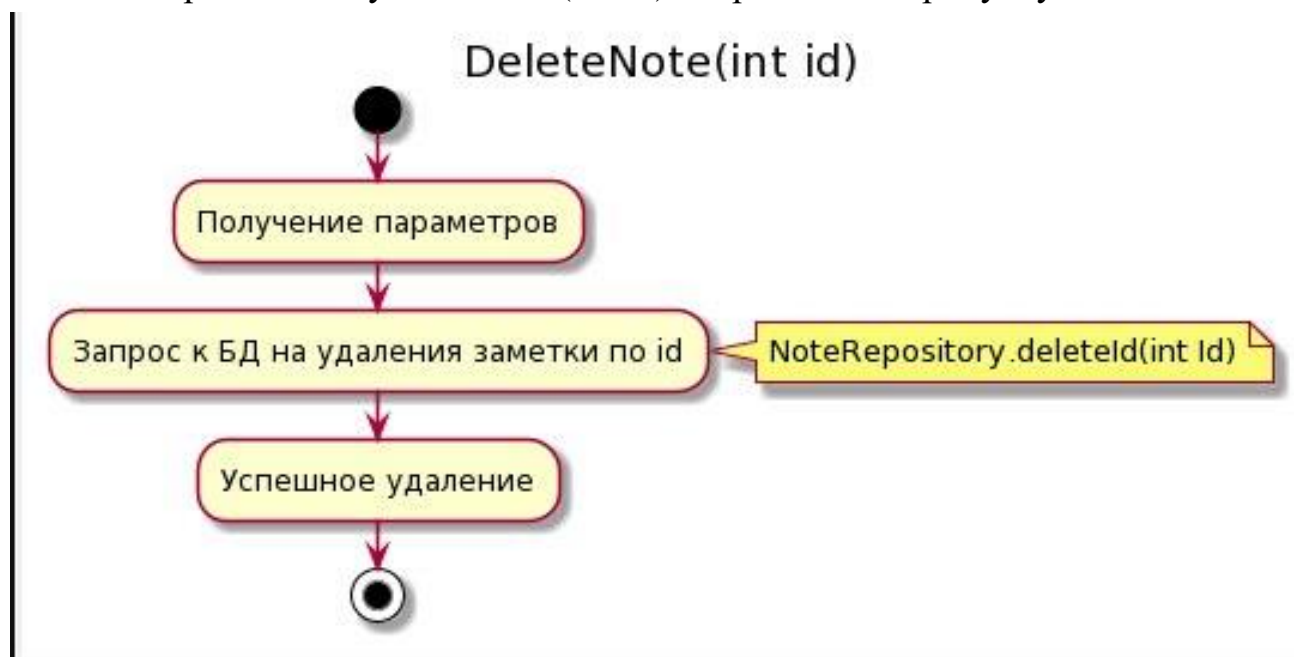


Рисунок 3.3.8 – Алгоритм методу DeleteNote(int id)

Код до метода DeleteNote(int id):

```
@startuml
start
title DeleteNote(int id)
:Получение параметров;
:Запрос к БД на удаления заметки по id;
note right
NoteRepository.deleteId(int Id)
end note
:Успешное удаление;
Stop @enduml@enduml
```

Алгоритм метода DeletePersonalDiary (int id) изображений на рисунке 3.3.9.



Рисунок 3.3.9 – Алгоритм метода DeletePersonalDiary (int id)

Код до метода DeletePersonalDiary (int id):

```
@startuml
start
title DeletePersonalDiary(int id)
:Получение параметров;
:Запрос к БД на удаления по id;
note right
DeletePersonalDiaryRepository.deleteId(int Id)
end note
:Успешное удаление; stop @enduml
```

Алгоритм метода DeletePasswordManager(int id) изображений на рисунке 3.3.23.

3.4 Проектування тестових наборів методів програмних класів

У відповідності із планами на розробки програмного продукту між всіма учасниками проектної команди необхідно розподілити розробку функцій (методів класів) та відповідних їм програмних модулів.

Для кожної функції (методу класу) необхідно створити тестові набори(таблиця 3.4.1),використовуючи будь-який метод чорного ящика та представити їх у вигляді таблиці.

Таблиця 3.4.1 – Тести мотодів

Назва функції	Номер тесту	Опис введених значень	Опис очікуваного результату
Authentication(string email, string password)	1	Email = dima@gmail.com Password = qwerty123	Перенаправлення до особистого кабінету
Authentication(string email, string password)	2	Email = dima@gmail.com Password = qwerty123iiv	Помилка, невірний введенні дані
Authentication(string email, string password)	3	Email = dimdsdsa@gmail.com Password = qwerty123	Помилка, невірний введенні дані
setNote(Note note)	4	Name= dsasd...n(n>100)	Ім'я не може бути більше 100
setNote(Note note)	5	Name= Text=	Поля не повинні бути пустими
setNote(Note note)	6	Name=Любимая песня Text=Цой	Успіх
SetDiary(Personal_diary personal_diary)	7	Name= dsasd...n(n>100)	Ім'я не може бути більше 100
SetDiary(Personal_diary personal_diary)	8	Name= Text=	Поля не повинні бути пустими
deleteDiary(int id)	9	Id=2	Видалено
deleteNote(int id)	10	Id=2	Видалено
getDiary()	11		Виведено всі дані

getNote()	12		Виведено всі дані
getOneNoteForEdit(int id)	12	Id=3	Виведено один нотаток для редагування

4 Конструювання програмного продукту

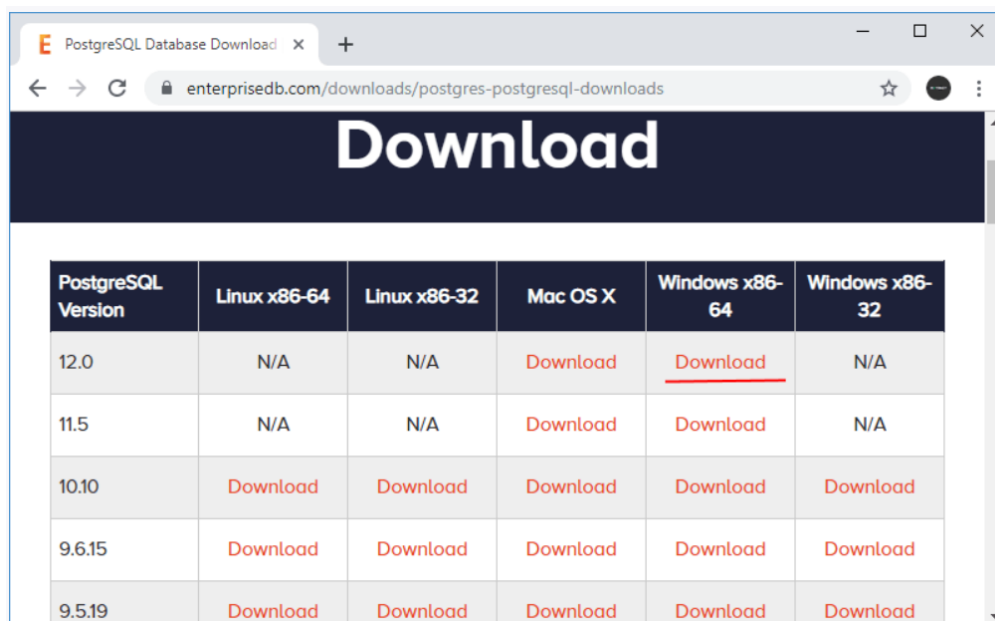
4.1. Особливості інсталяції та роботи з СУБД

PostgreSQL - вільна об'єктно-реляційна система управління базами даних.

З тих пір вийшло безліч версій postgresql. Поточною версією є версія 12.

PostgreSQL підтримується для всіх основних операційних систем - Windows, Linux, MacOS.

На сторінці <https://www.postgresql.org/download/> можна знайти посилання на завантаження різних дистрибутивів для різних операційних систем. Зокрема, для завантаження дистрибутива для Windows треба перейти на сторінку <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads> і вказати всі необхідні опції для завантаження: версію postgres і операційну систему.



PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
12.0	N/A	N/A	Download	<u>Download</u>	N/A
11.5	N/A	N/A	Download	Download	N/A
10.10	Download	Download	Download	Download	Download
9.6.15	Download	Download	Download	Download	Download
9.5.19	Download	Download	Download	Download	Download

Рис. 4.1.1 Посилання на завантаження різних дистрибутивів для різних операційних систем

При установці запам'ятаємо пароль, так як він буде потрібно для підключення до сервера. Потім потрібно буде встановити порт, по якому буде запускатися сервер. Можна залишити порт за замовчуванням:

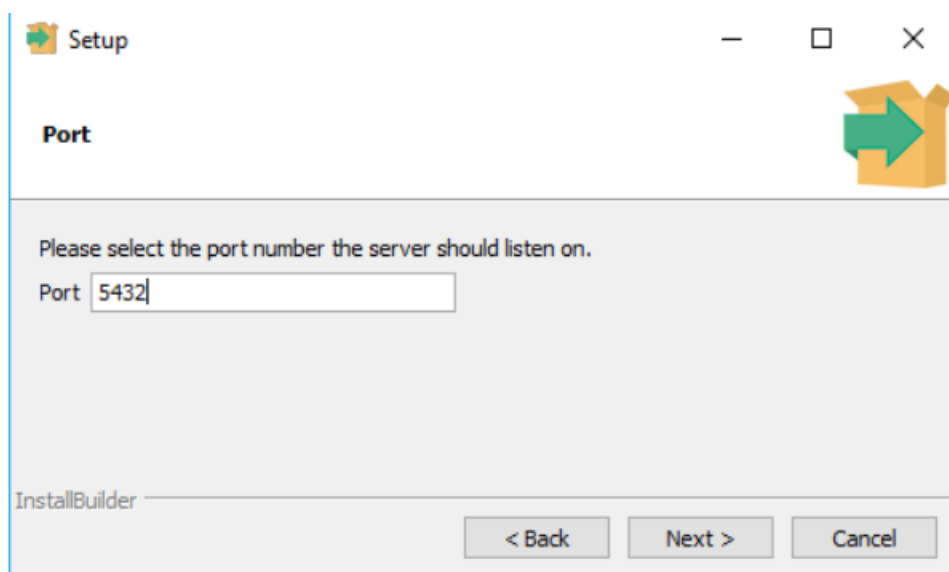


Рис. 4.1.2 Встановлення стандартного порту 5342

І після завершення установки ми побачимо наступне вікно, і для виходу натиснемо на кнопку Finish:

Таким чином, сервер PostgreSQL встановлений, і ми можемо починати з ним працювати.

Для спрощення адміністрування на сервері postgresql в базовий комплект установки входить такий інструмент як pgAdmin. Він являє графічний клієнт для роботи з сервером, через який ми в зручному вигляді можемо створювати, видаляти, змінювати бази даних і управляти ними. Так, на Windows після установки ми можемо знайти значок pgAdmin в меню Пуск і запустити його:

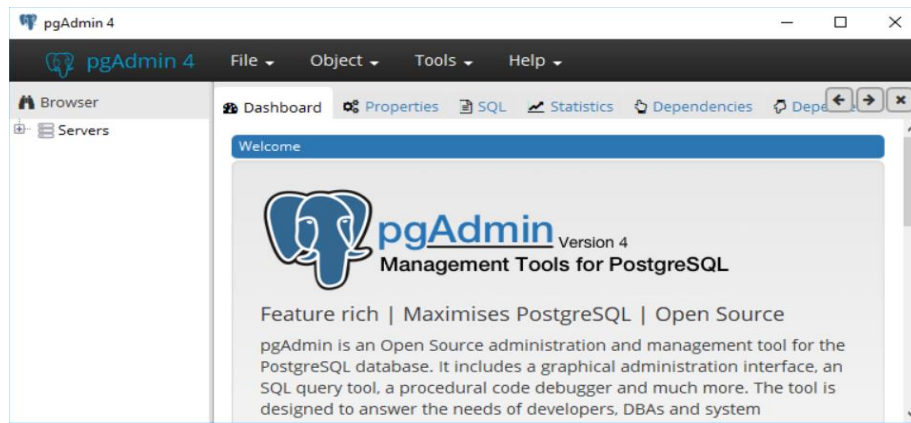
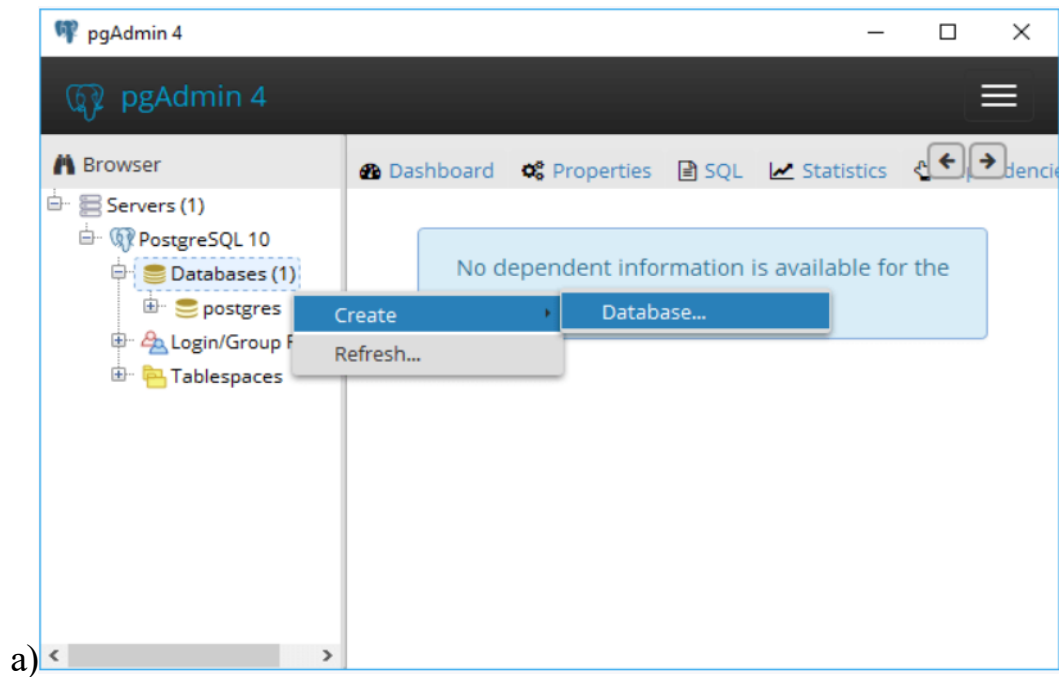


Рис. 4.1.3 Стартове вікно pgAdmin

4.2. Створення структур Даних

4.2.1. Створення бази даних.



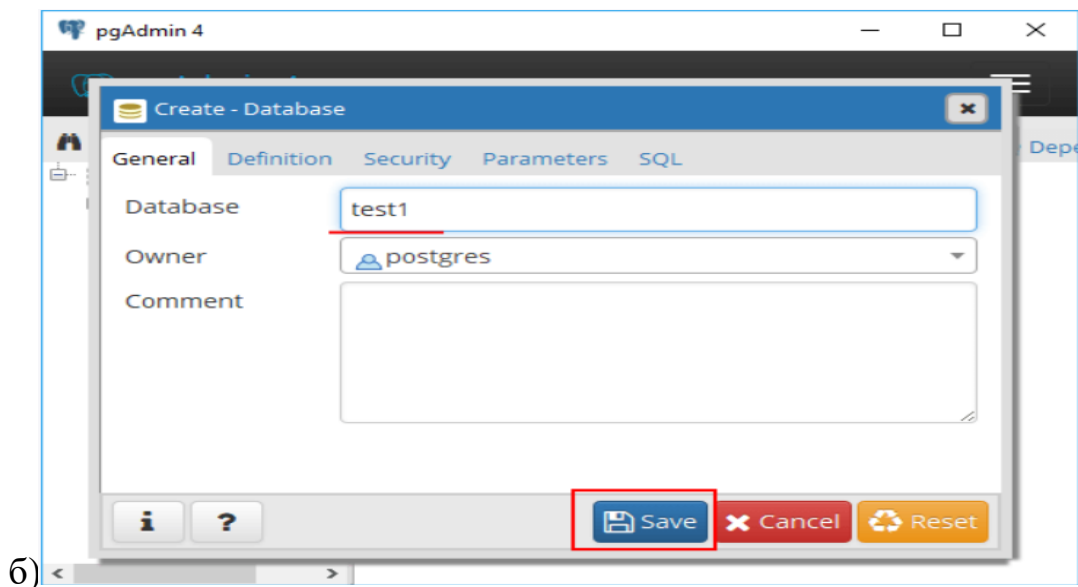


Рис.4.2.1.1 а) Створення БД; б) Введення необхідних параметрів БД

4.2.2. Створення таблиць.

За допомогою анотації `@entity` програма розуміє, що треба створити в базі даних таблиці з параметрами та зв'язками описаними у класі з анотацією `@entity`.

Таблиця 4.2.2.1. Опис анотації

Анотація	Опис
@NoArgsConstructor @AllArgsConstructor	Создание пустого конструктора, конструктора включающего все final поля, либо конструктора включающего все возможные поля
@Builder	Неализация паттерна bulder, <u>Singular</u> – используется для объектов в единственном экземпляре (добавления элемента в коллекции и т.п.)
@Data	Генерация всех служебных методов, заменяет сразу команды <code>@ToString</code> ,

	@EqualsAndHashCode, <u>Getter</u> , <u>Setter</u> , @RequiredArgsConstructorConstructo
@Entity	Эта аннотация указывает Hibernate, что данный класс является сущностью (entity bean). Такой класс должен иметь конструктор по-умолчанию (пустой конструктор).
@Table	С помощью этой аннотации мы говорим Hibernate, с какой именно таблицей необходимо связать (map) данный класс. Аннотация @Table имеет различные атрибуты, с помощью которых мы можем указать <i>имя таблицы, каталог, БД и уникальность столбцов в таблице БД</i> .
@Id	С помощью аннотации @Id мы указываем <i>первичный ключ (Primary Key)</i> данного класса.

Продовження таблиці 4.2.2.1 Опис анотації

@GeneratedValue	Эта аннотация используется вместе с аннотацией @Id и определяет такие параметры, как strategy и generator.
@Column	Аннотация @Column определяет к какому столбцу в таблице БД относится конкретное поле класса (аттрибут класса).

<p>Наиболее часто используемые атрибуты аннотации @Column такие:</p>	<ul style="list-style-type: none"> • name Указывает имя столбца в таблице • unique Определяет, должно ли быть данное значение уникальным • nullable Определяет, может ли данное поле быть NULL, или нет. • length Указывает, какой размер столбца (например количество символов, при использовании String).
--	---

Зразок коду, який відноситься до класу Node:

```
package pl.sda.finalProject.myOrganizer.entity;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.hibernate.validator.constraints.Length;
import org.hibernate.validator.constraints.NotEmpty;
import pl.sda.finalProject.myOrganizer.entity.MyUser;

import javax.persistence.*;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import java.time.LocalDate;

@Entity
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Note {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotEmpty
```

Продовження коду Node:

```
@Size(max = 100)

    private String name;

    private LocalDate creationDate;

    @Size(max = 4000)
    private String description;

    @ManyToOne
    private MyUser user;

}
```

4.3. Робота з IDEA

Наш програмний продукт використовує спосіб відображення сайтів засобами HTML, CSS, Java, Thymeleaf, Spring, Maven та СУБД PostgreSQL.

Для роботи в зв'язці з вищеперерахованими засобами найбільш підходить інтегроване середовище розробки програмного забезпечення IntelliJ IDEA .

Чому саме IntelliJ IDEA :

- Розумне автодоповнення, інструменти для аналізу якості коду, зручна навігація, розширені рефакторинг і форматування для Java, Groovy, Scala, HTML, CSS, JavaScript, CoffeeScript, ActionScript, LESS, XML і багатьох інших мов.
- Підтримка всіх популярних фреймворків і платформ, включаючи Java EE, Spring Framework, Grails, Play Framework, GWT, Struts, Node.js, AngularJS, Android, Flex, AIR Mobile і багатьох інших.
- Інтеграція з серверами Додатків, включаючи Tomcat, TomEE, GlassFish, JBoss, WebLogic, WebSphere, Geronimo, Resin, Jetty и Virgo.
- Інструменти для роботи з базами даних і SQL файлами, включаючи зручний клієнт і редактор для схеми бази даних.
- Фреймворки Spring
- Підтримка Thymeleaf 3.0

2.2. Створення програмної структури з урахуванням спеціалізованого Фреймворку

4.3.1 Spring MVC:

Фреймворк Spring MVC забезпечує архітектуру паттерна Model - View - Controller (Модель - Відображення - Контролер) за допомогою слабо пов'язаних

готових компонентів. Патерн MVC розділяє аспекти додатки (логіку введення, бізнес-логіку і логіку UI), забезпечуючи при цьому вільну зв'язок між ними.

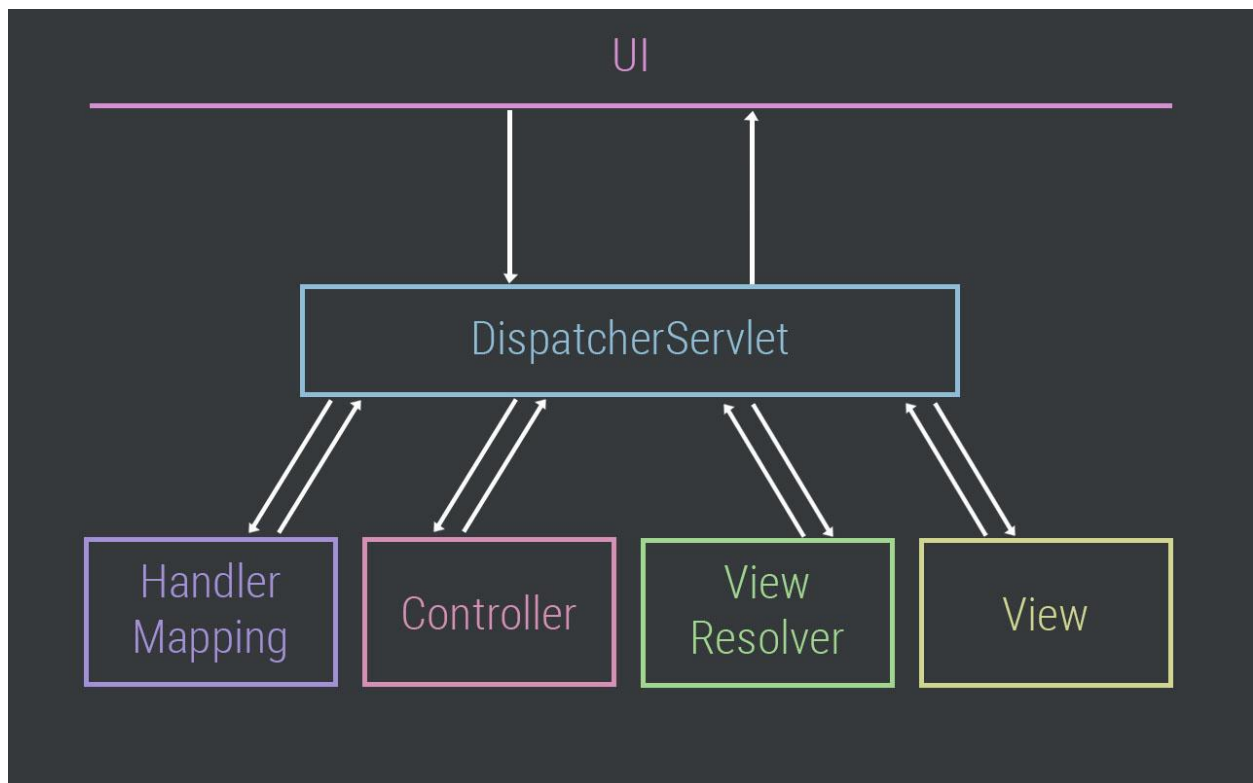


Рис.4.3.1.1 архітектуру паттерна MVC

1. **Model** (Модель) інкапсулює (об'єднює) дані програми, в цілому вони будуть складатися з РОЮ («Старих добрих Java-об'єктів», або бінів).
2. **View** (Отображення, Вид) відповідає за відображення даних Моделі, — зазвичай, генеруючи HTML, який ми бачимо в своєму браузері.
3. **Controller** (Контролер) обробляє запит користувача, створює відповідну Моделю і передає її для відображення в Вид.

Структура проекту

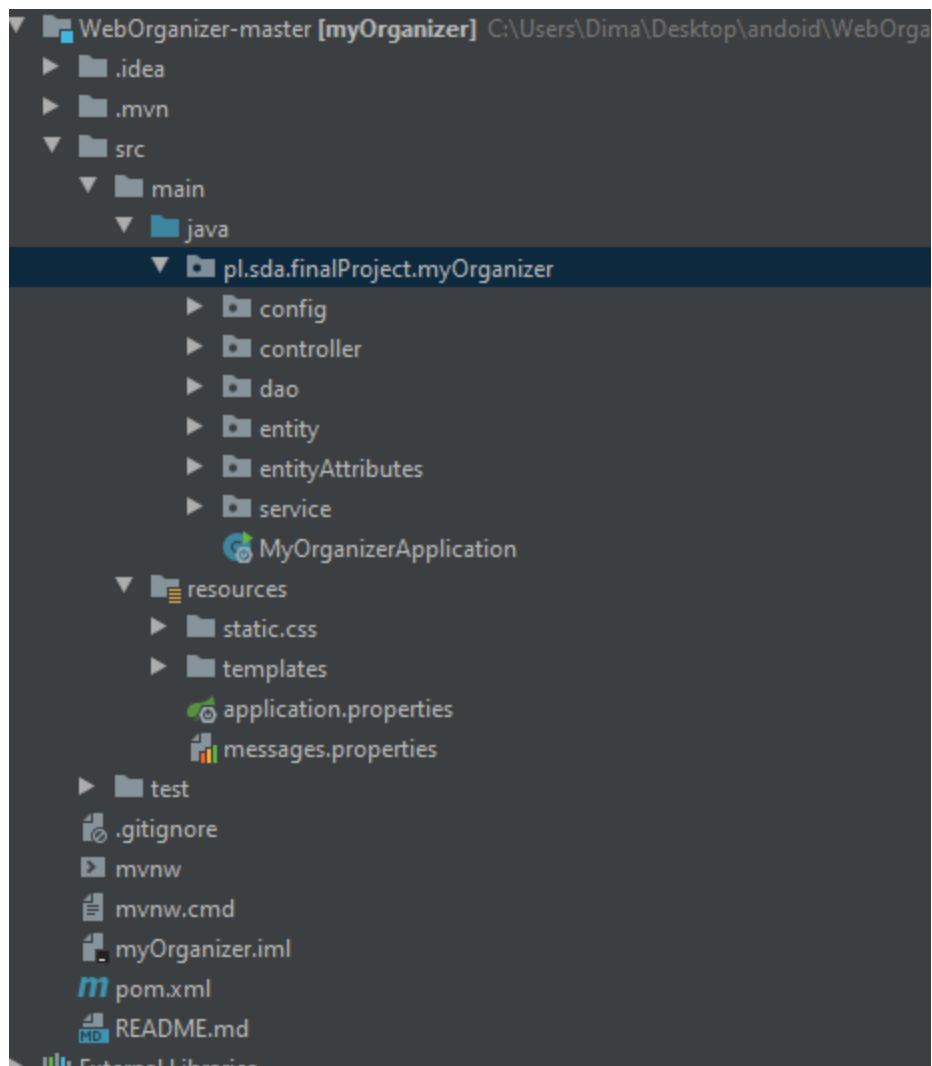


Рис.4.3.1.2 Структура проекту

Папка «dao», «entity» , «service» зберігає в собі класи , які потрібні для взаємодії з базою даних. Детальніше буде в наступних завданнях.

Entity – всі класи нашої моделі MVC.

Controller – всі класи контролерів MVC.

Templates – папка з html файлами (Вид MVC)

Файл application.properties - файл, де ми можемо вказати різні види властивостей (наприклад, параметри бази даних, чи параметри запуску серверу, чи параметри використання бібліотеки для авторизації та інше)

Maven автоматично завантажить бібліотеки залежностей Spring і помістить їх в локальний репозиторій Maven . Pom.xml – файл в якому, ми прописуємо потрібні бібліотеки.

Style.cc – папка з файлами css для стилізації вашої веб-сторінки.

4.3.2. Створення програмних класів

Модель Note:

```
package pl.sda.finalProject.myOrganizer.entity;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.hibernate.validator.constraints.Length;
import org.hibernate.validator.constraints.NotEmpty;
import pl.sda.finalProject.myOrganizer.entity.MyUser;

import javax.persistence.*;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import java.time.LocalDate;

@Entity
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Note {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotEmpty
    @Size(max = 100)
    private String name;

    private LocalDate creationDate;

    @Size(max = 4000)
    private String description;

    @ManyToOne
    private MyUser user;
}
```

Зразок коду контроллер Node:

```
@Controller
public class NoteController {

    @Autowired
    private NoteService noteService;
    @Autowired
    private IUserRepository userRepository;
    @Autowired
```


Продовження коду контроллеру Note:

```
private INoteRepository noteRepository;

@GetMapping("/organizer/notes")
public String showNotesPage(Model model, Principal principal) {
    Note newNote = new Note();
    MyUser activeUser = userRepository.findOne(principal.getName());
    model.addAttribute("newNote", newNote);
    model.addAttribute("notes", noteRepository.findByUserOrderByIdAsc(activeUser));
    return "notes";
}

@PostMapping("/organizer/notes")
public String addNote(@Valid @ModelAttribute("newNote") Note newNote, BindingResult
bindingResult,
    Principal principal) {
    MyUser activeUser = userRepository.findOne(principal.getName());
    newNote.setUser(activeUser);
    newNote.setCreationDate(LocalDate.now());
    if (bindingResult.hasErrors()) {
        return "notes";
    }
    noteService.addNote(newNote);
    return "redirect:/organizer/notes";
}

@GetMapping(path = "/organizer/notes/edit/{id}")
public String showEditForm(@PathVariable("id") Long id, Model model) {
    Note editNote = noteRepository.findOne(id);

    if (editNote == null) {
        return "noteNotFound";
    }
    model.addAttribute("editNote", editNote);

    return "editNote";
}

@PostMapping(path = "/organizer/notes/edit/{id}")
public String editNote(@PathVariable("id") Long id, @Valid @ModelAttribute("editNote") Note editNote,
    BindingResult bindingResult) {
    Note entity = noteRepository.findOne(id);

    if (entity == null) {
        return "noteNotFound";
    }
    if (bindingResult.hasErrors()) {
        return "organizer/notes/edit/{id}";
    }
    entity.setName(editNote.getName());
}
```

Продовження коду контролеру Note:

```

        entity.setDescription(editNote.getDescription());
        noteService.addNote(entity);
        return "redirect:/organizer/notes";
    }

    @GetMapping(path = "/organizer/notes/delete/{id}")
    public String deleteNote(@PathVariable("id") Long id) {
        if (noteRepository.findOne(id) == null) {
            return "noteNotFound";
        }
        noteRepository.delete(id);
        return "redirect:/organizer/notes";
    }
}

```

Таблиця 4.3.2.1. Опис анотації:

Анотація	Опис
@Controller	@Controller - (Слой представления) Аннотация для маркировки java класса, как класса контроллера. Данный класс представляет собой компонент, похожий на обычный сервлет (HttpServlet) (работающий с объектами HttpServletRequest и HttpServletResponse), но с расширенными возможностями от Spring Framework.

Продовження таблиці 4.3.2.1. Опис анотації:

@Autowired	Аннотация @Autowired обеспечивает контроль над тем, где и как автосвязывание должны быть осуществлено. Мы можем использовать @Autowired как для методов, так и для конструкторов
@GetMapping	Аннотация @GetMapping говорит — метод list() должен быть вызван, когда

	кто-то вызывает метод GET на пути /notes. Имя пути, очевидно, берётся из параметра @RequestMapping на классе.
@PostMapping	Аннотация @PostMapping без спецификации пути, говорит нам, что метод create() обслуживает <i>POST</i> запросы по пути /notes.

4.4. Особливості розробки алгоритмів методів програмних класів

DAO (Data Access Object) - це шар об'єктів які забезпечують доступ до даних. Зазвичай для реалізації DAO використовується EntityManager і з його допомогою ми працюємо з нашою БД, але в нашому випадку це система не підійде, так як ми вивчаємо Spring Data нам потрібно використовувати її засоби інакше нема чого він нам.

Spring Data надає набір готових реалізацій для створення DAO але Spring вважали за краще цей шар називати не DAO, а Repository.

Приклад створення NodeRepository:

```
package pl.sda.finalProject.myOrganizer.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import pl.sda.finalProject.myOrganizer.entity.MyUser;
import pl.sda.finalProject.myOrganizer.entity.Note;

import java.util.List;

@Repository
public interface INoteRepository extends JpaRepository<Note, Long> {

    List<Note> findByUserOrderByByIdAsc(MyUser user);
    void deleteAllByUser(MyUser user);
}
```

JpaRepository - це інтерфейс фреймворка Spring Data надає набір стандартних методів JPA для роботи з БД.

Service - це Java клас, який надає з себе основну (Бізнес-Логіку). В основному сервіс використовує готові DAO / Repositories або ж інші сервіси, для того щоб надати кінцеві дані для призначеного для користувача інтерфейсу.

Приклад створення NoteService:

```
package pl.sda.finalProject.myOrganizer.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import pl.sda.finalProject.myOrganizer.dao.INoteRepository;
import pl.sda.finalProject.myOrganizer.dao.IUserRepository;
import pl.sda.finalProject.myOrganizer.entity.MyUser;
import pl.sda.finalProject.myOrganizer.entity.Note;

import java.security.Principal;
import java.time.LocalDate;
import java.util.List;

@Service
public class NoteService {

    @Autowired
    private INoteRepository noteRepository;

    public void addNote(Note note) {
        note.setCreationDate(LocalDate.now());
        noteRepository.save(note);
    }
}
```

4.5. Використання спеціалізованих програмних бібліотек

За допомогою конфігураційного файлу pom.xml встановлюються бібліотеки. Все що потрібно це вказати пакети, які безпосередньо потрібні. Можна легко знайти пакети, використовуючи Google, і пошук "maven репозиторій".

Приклад файлу pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>pl.sda.finalProject</groupId>
    <artifactId>myOrganizer</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>
```

Продовження файлу pom.xml:

```
<name>myOrganizer</name>
<description>Demo project for Spring Boot</description>
// Все приложения Spring Boot конфигурируются от spring-boot-starter-parent
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.10.RELEASE</version>
    <relativePath/>
</parent>
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
</properties>
<dependencies>
/* Оскільки ми створюємо REST API, то необхідно в якості залежно використовувати spring-boot-
starter-web, яка неявно визначає всі інші залежності, такі як spring-core, spring-web, spring-webmvc,
servlet api, і бібліотеку jackson-databind */
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
    </dependency>
//Підключення бібліотеки для роботи с БД postgresql
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>
//Підключення бібліотеки Lombok
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
//Підключення бібліотеки Security
    <dependency>
```

Продовження файлу pom.xml:

```
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-test</artifactId>
        <scope>test</scope>
    </dependency>
//Підключення бібліотеки Hibernate
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-java8</artifactId>
        <version>5.1.0.Final</version>
    </dependency>
//Підключення бібліотеки Security
    <dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-springsecurity4</artifactId>
        <version>3.0.2.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>1.4.196</version>
        <scope>test</scope>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>
```

4.6. Модульне тестування програмних класів

Опишіть процес реального модульного тестування, використовуючи проект тестових наборів, описаних в лабораторній роботі No 8.

Якщо при тестуванні використовуються спеціалізовані програми, тоді опишіть процес їх використання.

1. Теоретичні відомості

Специфікація метода Authentication (string email, string password):

1) email

2) password

Таблиця 4.6.1 Тестові набори 1

Authentication(string email, string password)	1	Email = dima@gmail.com Password = qwerty123	Перенаправлення до особистого кабінету
Authentication(string email, string password)	2	Email = dima@gmail.com Password = qwerty123iiv	Помилка, невірний введенні дані
Authentication(string email, string password)	3	Email = dimdsdsa@gmail.com Password = qwerty123	Помилка, невірний введенні дані

```

package pl.sda.finalProject.myOrganizer.config;

import ...

@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private DataSource dataSource;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.jdbcAuthentication()
            .dataSource(dataSource)
            .usersByUsernameQuery("select email as principal, password as credentials, true from my_user where email=?")
            .authoritiesByUsernameQuery("select email as principal, user_role as role from my_user where email=?")
            .passwordEncoder(passwordEncoder());
    }

    @Bean
    public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }

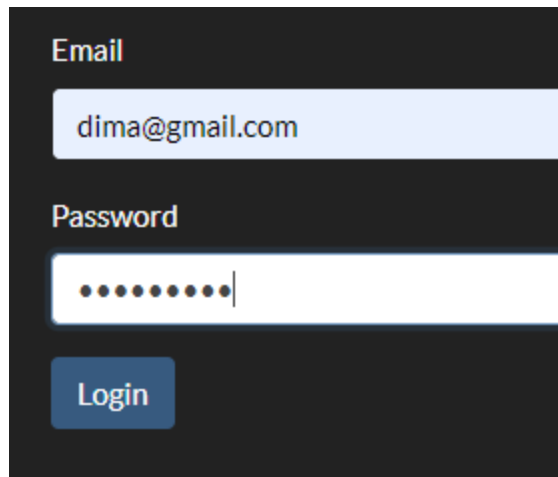
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests().antMatchers("/organizer", "/organizer/register")
            .permitAll().antMatchers("/organizer/users").hasRole("ADMIN")
            .antMatchers("/organizer/notes", "/organizer/tasks",
                "/organizer/profile", "/organizer/events")
            .hasAnyRole("USER", "ADMIN").and().formLogin()
            .passwordParameter("password")
            .usernameParameter("username")
            .loginPage("/organizer/login")
            .loginProcessingUrl("/loginProcessing")
            .failureUrl("/login-failure")
            .defaultSuccessUrl("/organizer").and().logout()
            .logoutUrl("/logout").logoutSuccessUrl("/organizer/login").and()
            .exceptionHandling().accessDeniedPage("/access-denied").and()
            .httpBasic().disable();
    }
}

```

Рис. 4.6.1 – Конфігураційний клас для авторизації

Після конструювання програмного модуля було проведено модульне мануальне тестування відповідно до тестами №1-3.

Процес виконання тесту №1(Рис 4.6.2-4.6.3), завершився отриманням наступних результатів:



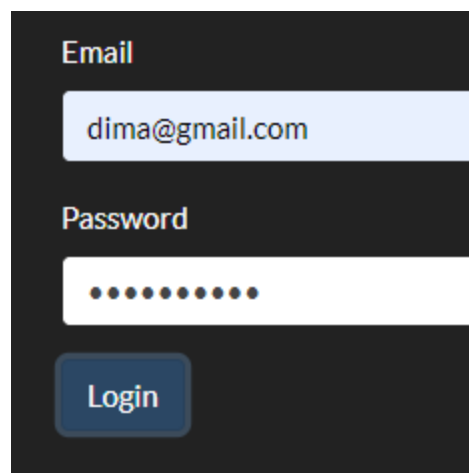
A login form on a dark background. It has two input fields: 'Email' containing 'dima@gmail.com' and 'Password' with masked characters. Below the fields is a blue 'Login' button.

Рис 4.6.2. – введення даних тесту №1



Рис 4.6.3. – успіх авторизації

Процес виконання тесту №2(Рис.4.6.4-4.6.5), завершився отриманням наступних результатів:



A login form on a dark background, identical to the one in Figure 4.6.2. It contains the email 'dima@gmail.com' and a masked password, with a blue 'Login' button below.

Рис 4.6.4. – введення даних тесту №2

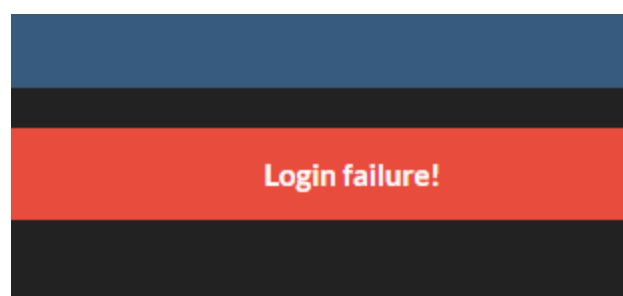
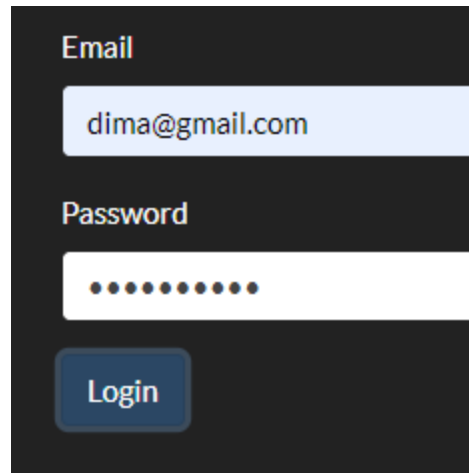


Рис 4.6.5. – помилка авторизації

Заповнення полів авторизації с угодою тест-кейсу №3(4.6.6-4.6.7):



The image shows a login interface on a dark background. It has two input fields: 'Email' containing 'dima@gmail.com' and 'Password' with masked characters. Below the fields is a blue 'Login' button.

Рис 4.6.6. – введення даних тесту №3

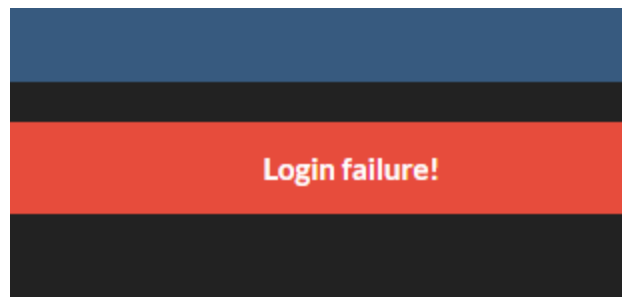


Рис 4.6.7. – помилка авторизації

2. Теоретичні відомості

Специфікація методу SetNote(Note note):

- 1) Date
- 2) Time
- 3) Name

Таблиця 4.6.2. Тестові набори

setNote(Note note)	4	Name=dsasd...n(n>100)	Ім'я не може бути більше 100
setNote(Note note)	5	Name= Text=	Поля не повинні бути пустими
setNote(Note note)	6	Name=Любимая песня Text=Цой	Успіх

```

package pl.sda.finalProject.myOrganizer.controller;

import ...

@Controller
public class NoteController {

    @Autowired
    private NoteService noteService;
    @Autowired
    private IUserRepository userRepository;
    @Autowired
    private INoteRepository noteRepository;

    @GetMapping("/organizer/notes")
    public String showNotesPage(Model model, Principal principal) {
        Note newNote = new Note();
        MyUser activeUser = userRepository.findOne(principal.getName());
        model.addAttribute("newNote", newNote);
        model.addAttribute("notes", noteRepository.findByUserOrderByIdAsc(activeUser));
        return "notes";
    }

    @PostMapping("/organizer/notes")
    public String addNote(@Valid @ModelAttribute("newNote") Note newNote, BindingResult bindingResult,
        Principal principal) {
        MyUser activeUser = userRepository.findOne(principal.getName());
        newNote.setUser(activeUser);
        newNote.setCreationDate(LocalDate.now());
        if (bindingResult.hasErrors()) {
            return "notes";
        }
        noteService.addNote(newNote);
        return "redirect:/organizer/notes";
    }
}

```

Рис.4.6.8. Перша частина коду контролеру NoteController

```

@GetMapping(path = "/organizer/notes/edit/{id}")
public String showEditForm(@PathVariable("id") Long id, Model model) {
    Note editNote = noteRepository.findOne(id);

    if (editNote == null) {
        return "noteNotFound";
    }
    model.addAttribute("editNote", editNote);

    return "editNote";
}

@PostMapping(path = "/organizer/notes/edit/{id}")
public String editNote(@PathVariable("id") Long id, @Valid @ModelAttribute("editNote") Note editNote,
    BindingResult bindingResult) {
    Note entity = noteRepository.findOne(id);

    if (entity == null) {
        return "noteNotFound";
    }
    if (bindingResult.hasErrors()) {
        return "organizer/notes/edit/{id}";
    }
    entity.setName(editNote.getName());
    entity.setDescription(editNote.getDescription());
    noteService.addNote(entity);
    return "redirect:/organizer/notes";
}

@GetMapping(path = "/organizer/notes/delete/{id}")
public String deleteNote(@PathVariable("id") Long id) {
    if (noteRepository.findOne(id) == null) {
        return "noteNotFound";
    }
    noteRepository.delete(id);
    return "redirect:/organizer/notes";
}

```

Рис.4.6.9. Друга частина коду контролеру NoteController

Після конструювання програмного модуля було проведено модульне мануальне тестування тесту 4(Рис. 4.6.10).

Рис.4.6.10.Помилка введення імені(тест 4)

Процес виконання тесту №5, завершився отриманням наступних результатів (Рис. 4.6.11):

Рис.4.6.11.Помилка введення полів(тест 5)

Процес виконання тесту №5, завершився отриманням наступних результатів (Рис. 4.6.12):

Your notes				
#	Note name	Creation date	Open/Edit	Delete
1	Любимая музыка	2020-12-11	Open/Edit	Delete

Рис.4.6.12.Успіх додавання замітки(тест 6)

3. Теоретичні відомості

Спецификация (Рис.4.6.13)функции deleteNote(int id):

- 1) Name
- 2) Description

Таблиця 4.6.3. Тестові набори

deleteDiary(int id)	9	Id=2	Видалено
---------------------	---	------	----------

```

@GetMapping(path = "/organizer/notes/delete/{id}")
public String deleteNote(@PathVariable("id") Long id) {
    if (noteRepository.findOne(id) == null) {
        return "noteNotFound";
    }
    noteRepository.delete(id);
    return "redirect:/organizer/notes";
}

```

Рис.4.6.13. Обробка запиту на видалення замітки

Після конструювання програмного модуля було проведено модульне мануальне тестування відповідно до тесту №9(рис. 4.6.14-4.6.15)

Your notes				
#	Note name	Creation date	Open/Edit	Delete
1	Любимая музыка	2020-12-11	<input type="button" value="Open/Edit"/>	<input type="button" value="Delete"/>

Рис.4.6.14 – Вивід заміток

Your notes				
#	Note name	Creation date	Open/Edit	Delete

Рис.4.6.15 – Вивід заміток після видалення

4. Теоретические ведомости

Спецификация(Рис.4.6.16) методу getOneNoteForEdit(int id):

Таблиця 4.6.4. Тестові набори

getOneNoteForEdit(int id)	12	Id=3	Виведено один нотаток для редагування
---------------------------	----	------	---------------------------------------

```

@GetMapping(path = "/organizer/notes/edit/{id}")
public String showEditForm(@PathVariable("id") Long id, Model model) {
    Note editNote = noteRepository.findOne(id);

    if (editNote == null) {
        return "noteNotFound";
    }
    model.addAttribute("editNote", editNote);

    return "editNote";
}

```

Рис. 4.6.16. Обробка запиту на виведення уже існуючої замітки для редагування

Після конструювання програмного модуля було проведено модульне мануальне тестування відповідно до тестами №12(Рис. 4.6.17-4.6.18).

Your notes				
#	Note name	Creation date	Open/Edit	Delete
1	Любимая музыка	2020-12-11	Open/Edit	Delete

Рис. 4.6.17 – Виведення заміток

Note name:

Любимая музыка

Цой

Edit Note

Back to your notes

Рис. 4.6.18 – Виведення інформації для редагування після натискання на «Open/Edit»

5 Розгортання та валідація програмного продукту

5.1 Інструкція з встановлення програмного продукту

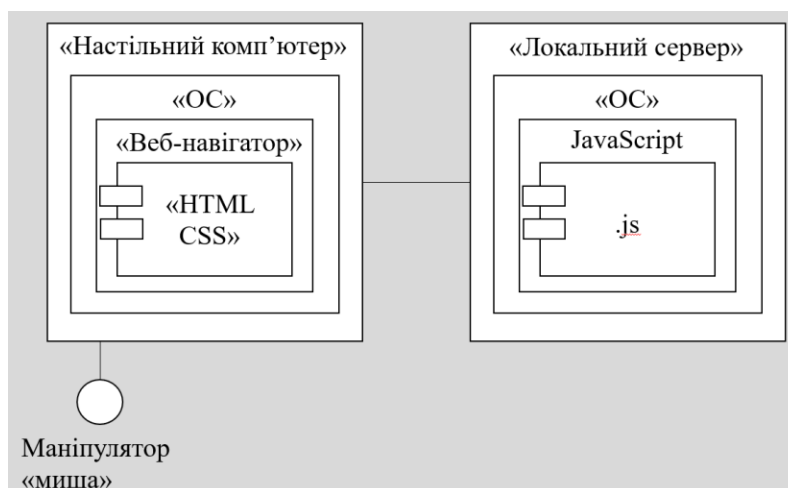


Рис.5.1.1. Приклад UML-діаграми розгортання ПП

5.2 Інструкція з використання програмного продукту

1) Реєстрація користувача

ПП надає можливість користувачу ролі «гість» вести параметри реєстрації (імя користувача, email та його пароль), як показано на (Рис.5.2.1).

The registration form is displayed on a dark background. It consists of three input fields, each with a label above it: 'Name', 'Email', and 'Password'. Each field has a placeholder text 'Enter name...', 'Enter email...', and 'Enter password...' respectively. Below the input fields is a blue button labeled 'Register'.

Рис.5.2.1. Екранна форма реєстрації користувача

Для реєстрації користувач повинен ввести значення, як показано на (Рис.5.2.2).

При цьому необхідно пам'ятати про обмеження значення пароллю: не менше 5 символів, не більше 20 символів у логін та собака при заповненні електронної пошти.

A screenshot of a registration form on a dark background. It contains three input fields: 'Name' with the text 'DSD', 'Email' with the text 'cool1@gmail.com', and 'Password' with ten dots. Below the fields is a blue 'Register' button.

Рис. 5.2.2. Екранної форма заповнення даних реєстрації користувача

Після етапу реєстрації користувачу необхідно буде увести свої дані у форму логіну (Рис.5.2.3).

A screenshot of a login form on a dark background. It contains two input fields: 'Email' with the text 'cool1@gmail.com' and 'Password' with ten dots. Below the fields is a blue 'Login' button.

Рис. Рис.5.2.3. Екранної форма заповнення даних логіну користувача

Далі користувач зможе вільно користуватися програмним продуктом(Рис.5.2.4).

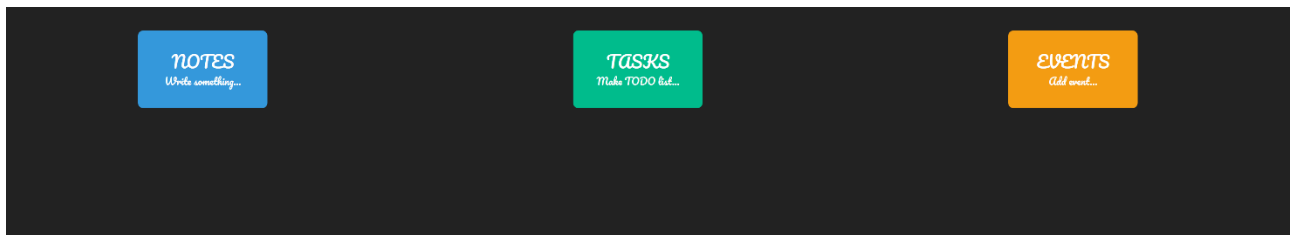


Рис.5.2.4. Екранної форма заповнення даних логіну користувача

Наприклад він може додати замітку, вікно інтерфейсу якої показано на (Рис.5.2.5.).

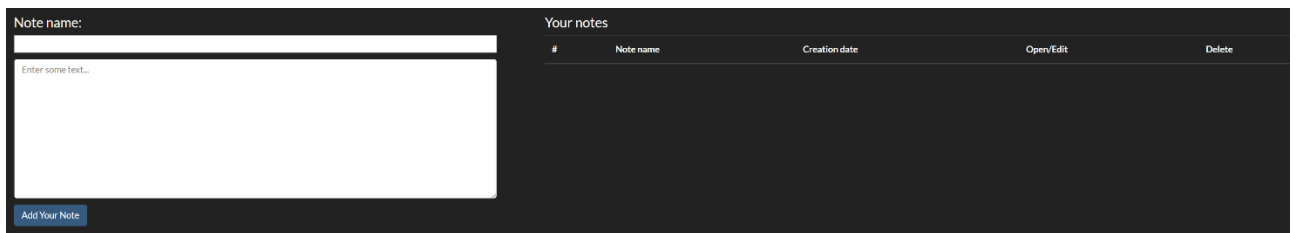


Рис.5.2.5. Інтерфейс заміток

Або додати та подивитись задачі, вікно інтерфейсу яких показано на (рис. 5.2.6)

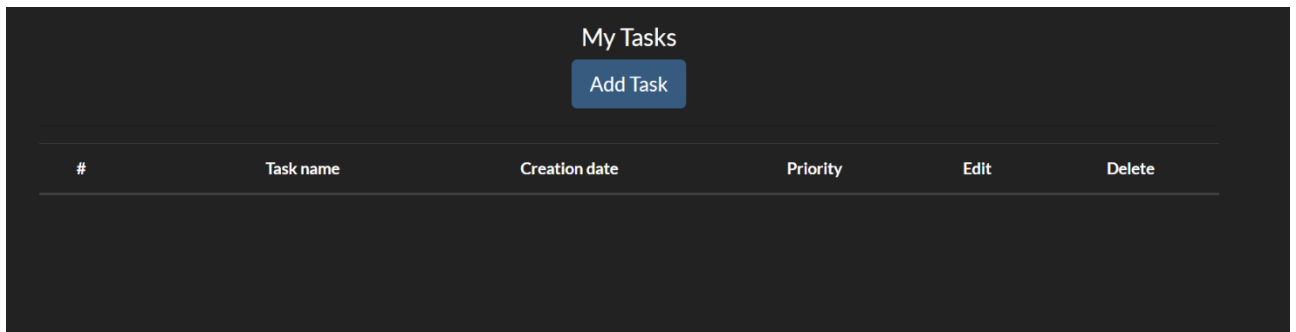


Рис.5.2.6. Інтерфейс задач

Чи на останок запланувати подію, вікно інтерфейсу якої показано на (рис. 5.2.7)

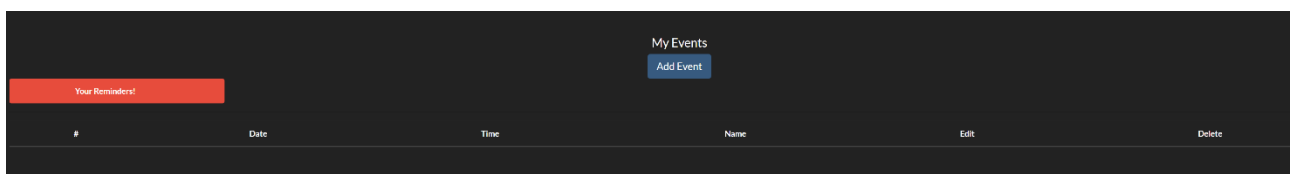


Рис.5.2.7. Інтерфейс подій

5.3 Результати валідації програмного продукту

Нашим завданням було допомогти користувачам структурувати свій день, а також не залишати поза голови плани та ідеї. Я вважаю, що з цією метою наша команда впоралася на усі сто.

1) Завдяки простому і зрозумілому інтерфейсу користувач зможе з легкістю додавати замітки / завдання / події.

2) Весь функціонал програмного продукту був протестований, тому з ним не виникне проблем.

Висновки

В результаті створення програмного продукту була досягнута наступна мета його споживача: «визначення мети з підпункту 1.2.2.2».

Доказом цього є наступні факти:

1) Завдяки простому і зрозумілому інтерфейсу користувач зможе з легкістю додавати замітки / завдання / події;

2) Весь функціонал програмного продукту був протестований, тому з ним не виникне проблем;

В процесі створення програмного продукту виникли такі труднощі (організаційні, проблеми відсутності досвіду, знань, потрібних в різних етапах):

1) Брак знань мови програмування;

2) Мала кількість досвіду в створенні програмних продуктів;

3) Незвичний досвід роботи в команді.

Через вищеописані непередбачені труднощі, а також через обмежений час на створення програмного продукту, залишилися нереалізованими такі прецеденти або їх окремі кроки роботи:

1) Плани зі створення розділу збереження паролів;

Зазначені недоробки планується реалізувати в майбутніх курсових роботах з урахуванням тем дисциплін наступних семестрів.