МІНІСТЕРСТВО ОСВІТИ НАУКИ УКРАЇНИ
ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ПРИВАТНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра інформаційних технологій

ДО ЗАХИСТУ ДОПУЩЕНА

Завідувач кафедри,
д.е.н., доц.
_____________ С.І. Левицький

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

РОЗРОБКА ВЕБ-ЗАСТОСУНКУ «ОСОБИСТИЙ ВЕТЕРИНАРНИЙ КАБІНЕТ ТВАРИНИ» НА БАЗІ ФРЕЙМВОРКІВ ANGULAR І SYMFONY

Виконав

ст. гр. ІПЗ – 218 _________________________ Б. Ю. Буров

Керівник

к.е.н., доц. _________________________ О. В. Шляга

Запоріжжя

2023

ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ПРИВАТНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри,

д.е.н., доц.

____________ С.І. Левицький

___.___.______ р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ

студенту гр. ІПЗ-218 ,

спеціальності 121 - «Інженерія програмного забезпечення»

Бурову Богдану Юрійовичу

1. Тема: Розробка веб-застосунку «Особистий ветеринарний кабінет тварини» на базі фреймворків Angular і Symfony

затверджена наказом № 02-10 від 27 січня 2023 р.

2. Термін здачі студентом закінченої роботи: 12 червня 2023 р.

3. Перелік питань, що підлягають розробці:

1. Розглянути питання актуальності розробки застосунку.

2. Провести огляд галузі та аналітику проблеми і її рішень загалом

3. Провести огляд та аналіз популярних аналогів, зробити висновки про вимоги до проекту

4. Розглянути методи та способи створення веб-застосунків та обрати направлення розробки

5. Провести огляд стеку технологій для розробки проекту та зробити вибір на підставі вимог

6. Розробити проектування архітектури веб-застосунку

7. Створити відповідний застосунок, спираючись на отримані дані

8. Проаналізувати отримані результати

9. Оформити звіт за результатами роботи

4. Календарний графік підготовки кваліфікаційної бакалаврської роботи.

№ етапу	Зміст	Терміни виконання	Готовність по графіку %, підпис керівника	Підпис керівника про повну готовність етапу, дата
1	Формулювання (корегування) теми кваліфікаційної бакалаврської роботи та збір практичного матеріалу за темою	16.01.23-11.02.23		
2	**I атестація** I розділ кваліфікаційної бакалаврської роботи	27.03.23-31.03.23		
3	**II атестація** II розділ кваліфікаційної бакалаврської роботи	24.04.23-28.04.23		
4	**III атестація** III розділ кваліфікаційної бакалаврської роботи, висновки та рекомендації, додатки, реферат	22.05.23-26.05.23		
5	Перевірка кваліфікаційної бакалаврської роботи на оригінальність	15.05.23-12.06.23		
6	Доопрацювання кваліфікаційної бакалаврської роботи, підготовка презентації, отримання відгуку керівника і рецензії	29.05.23-12.06.23		
7	**Попередній захист кваліфікаційної бакалаврської роботи**	**12.06.23-18.06.23**		
8	Подача кваліфікаційної бакалаврської роботи на кафедру	за 3 дні до захисту		
9	**Захист кваліфікаційної бакалаврської роботи**	**19.06.23-24.06.23**		

Дата видачі завдання: ___.___.______ р.

Керівник кваліфікаційної

бакалаврської роботи _____________ О В. Шляга

Завдання отримав до виконання _____________ Б. Ю. Буров

РЕФЕРАТ

Бакалаврська робота містить 75 сторінок, 3 таблиці, 39 рисунків, 19 лістингів, 15 бібліографічних посилань.

Метою розробки є розробка веб-застосунку «Особистий ветеринарний кабінет тварини» на базі фреймворків Angular і Symfony як зручної системи управління станом тварини для лікаря і користувача

Об’єктом дослідження є сучасний додаток для ведення ветеринарного кабінету.

Предметом дослідження є фреймворки для створення веб-застосунків.

Здійснено детальний огляд предметної області, сучасних аналогів, таких як: JetVetPass, Enote, Appointe. Мною виявлено, що наразі зовсім відсутній централізований, стандартний підхід до візитів, аналізів та інших послуг ветеринарного кабінету. Розроблено два додатки для покращення навичок планування розробки за допомогою різних засобів. Було покращено навички розробки мовою PHP, а також поглиблено знання у двох фреймворках – Angular та Symfony. Для розробки програм були обрані такі стеки: Backend (PHP 8.0, OpenServer, Symfony, IDE Visual Studio Code); Frontend (Typescript 4.8.2, Angular 15.0.0, IDE Webstorm).

Отриманий програмний продукт є для користувача простим у використанні та гнучким у налаштуванні.

JAVASCRIPT, ANGULAR, PHP,SYMFONY, VET CABINET

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ТЕРМІНІВ, ПОЗНАЧЕНЬ І СКОРОЧЕНЬ 7

ВСТУП 8

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 10

1.1 Актуальність теми 10

1.2 Принцип вибору аналогів 11

1.3 Можливі, але не відповідні аналоги 12

РОЗДІЛ 2 ЗАСОБИ ТА МЕТОДОЛОГІЇ РОЗРОБКИ ДОДАТКУ 15

2.1. Обґрунтування вибору мов програмування 15

2.1.1. Вибір мови програмування для веб-застосунку клієнтської сторонни 16

2.1.2. Вибір мови програмування для веб-застосунку серверної сторони 16

2.2 Фреймворк 18

2.2.1 Вибір фреймворку для веб-застосунку серверної сторони 20

2.2.2 Вибір фреймворку для веб-застосунку клієнтської сторони 22

2.2.3 Середовище розробки 30

2.2.3.1 Вибір середовища розробки серверної сторони 31

2.2.3.2 Вибір середовища розробки клієнтської сторони 32

РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ 35

3.1 Проектування кінцевого продукту 35

3.1.1. Роутінг та RestAPI 35

3.1.2. Аутентифікація 39

3.1.3. Структура серверної сторони 42

3.1.4. ORM Doctrine 45

3.1.5. Міграції 47

3.1.6. Структура бази даних..48
3.2 Клієнтська програма...50
3.2.1. Створення програми та структура...50
3.2.2. Роутинг и Lazy Loading...54
3.2.3. Захист роутингу та охоронці...57
3.2.4. Структура компонента...58
3.2.5. Директиви та ролі..61
3.2.6. Сервіси та бізнес логіка...63
3.2.7. Репозиторії та DI..67
3.3 Дизайн та UI..68
ВИСНОВКИ..73
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ...74

ПЕРЕЛІК УМОВНИХ ТЕРМІНІВ, ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

Скорочення	Повна назва	Пояснення/переклад
API	Application Programming Interface	Прикладний програмний інтерфейс
JS	JavaScript	Язык программирования
TS	TypeScript	Язык программирования
CLI	Command line interface	Интерфейс командной строки для разных действий
ORM	Object-relational mapping	Модель для связи с базой данных
DB	Data base	База данных
SQL	Structured Query Language	Язык запросов
CRUD	Create, Read, Update, Delete	4 базовые операции по работе с данными
DI	Dependency Injection	Система для вставки компонентов программы через базовый контейнер
URL	Uniform Resource Locator	Строка запроса

ВСТУП

За тваринами, як і за людьми теж потрібен догляд, особливо щодо здоров'я.

Тварини як і люди схильні до багатьох хвороб, і так само як і людям тваринам потрібні необхідні щеплення для убезпечення себе від багатьох смертельних хвороб.

У певний момент часу мені і самому довелося зіткнутися з відвідуванням ветериварних клінік, і через деякі обставини довелося відвідувати більше, ніж одну.

Проаналізувавши весь досвід, я усвідомив що існує величезна кількість незручностей при зберіганні та пошуку інформації про тварину, пошуку інформації про клініку та загалом у зручності цього всього.

Саме це і побудило мене на написання програми для централізованого зберігання та видачі інформації у зручному форматі.

Метою розробки є розробка веб-застосунку «Особистий ветеринарний кабінет тварини» на базі фреймворків Angular і Symfony як зручної системи управління станом тварини для лікаря і користувача

Об’єктом дослідження є сучасний додаток для ведення ветеринарного кабінету.

Предметом дослідження є фреймворки для створення веб-застосунків.

Здійснено детальний огляд предметної області, сучасних аналогів, таких як: JetVetPass, Enote, Appointe. Мною виявлено, що наразі зовсім відсутній централізований, стандартний підхід до візитів, аналізів та інших послуг ветеринарного кабінету.

Розроблено два додатки для покращення навичок планування розробки за допомогою різних засобів. Було покращено навички розробки мовою PHP, а також поглиблено знання у двох фреймворках – Angular та Symfony. Для розробки програм були обрані такі стеки: Backend (PHP 8.0, OpenServer, Symfony, IDE Visual Studio Code); Frontend (Typescript 4.8.2, Angular 15.0.0, IDE Webstorm).

Структура роботи включає вступ, три розділі, висновки, перелік використаних джерел …

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність теми

Тварини є невід'ємною частиною людського життя. Без них неможлива більшість процесів.

Спочатку тварини були дикими, і для їхнього «використання» знадобилося приручення та одомашнення

Одомашнення або доместикація - процес зміни диких тварин або рослин, при якому протягом багатьох поколінь вони утримуються людиною генетично ізольованими від їхньої дикої форми і піддаються штучному добору.

Тварини «служать» у різних сферах діяльності, пов'язаної з людським життям, починаючи від психотерапії, і закінчуючи пошуком людей при рятувальних операціях.

Також як і люди, тварини схильні до величезної кількості хвороб безліч з яких є смертельними. Так само як і люди, тваринам потрібна обов'язкова вакцинація. Домашнім тваринам також потрібний паспорт для вдалого перетину кордону. Майже всі європейські країни вимагають паспорт тварини, щоб вона могла перетнути кордон.

Рис 1.1 – Кількість котів у різних країнах

У певний момент життя мені довелося зіткнутися з обслуговуванням у різних ветеринарних клініках, і я зіткнувся з такими незручностями:

- Частково або повне відсутність інформації, та/або її неточність. Кілька разів я стикався що інформацію про ветеринарну клініку можна було знайти тільки через Google Maps, при цьому там був мінімальний обсяг інформації, і іноді навіть неактивний номер телефону, через що доводилося їхати, і шукати клініку. Через брак інформації було незрозуміло в якому годиннику працює клініка, і чи працює вона взагалі.
- Неповна інформація про послуги, що надаються клінікою. Доводилося обдзвонити безліч клінік, щоб знайти необхідну послугу, наприклад одна клініка надає послуги з УЗД, а інша не надає, але там є послуги лікаря онколога.
- Безліч паперової інформації, складність пошуку інформації. Найчастіше клініки надають інформацію у паперовому вигляді, що не є особливо зручним варіантом зберігання, оскільки папірці дуже легко втратити і складно знайти потрібний. Рідше інформація надається на Viber, що трохи зручніше, але більшість клінік надсилають інформацію не з корпоративних, підписаних облікових записів, а з особистих, які підписані просто за номером телефону. Всього один раз я зіткнувся з клінікою, яка працює через мобільний додаток, але його функціонал залишає бажати кращого, через сильне спрощення. Також досить складно контролювати візити, оскільки вони часто написані на візитівках, які дуже легко втратити.

1.2 Принцип вибору аналогів

Спочатку було обрано набагато більше аналогів, але з мірою їх більшого вивчення стало зрозуміло, що в них відсутній необхідний функціонал, такий як створення та перегляд даних про тварину, більшість аналогів являли собою лише калькулятори медикаментів або довідник хвороб із симптомами, але, не надаючи доступу для фахівців та зручного застосування для клієнтів.

Проаналізувавши потреби клієнтів та виходячи з мого досвіду було вирішено шукати аналоги за такими параметрами:

- Повне створення всіх даних про тварин
- Перегляд візитів до лікаря
- Перегляд аналізів
- Вартість
- Доступність
- Платежі

За основу для порівняння аналогів було знайдено 3 аналоги відповідно за характеристиками та можливостями.

Таблиця 1.1 – Порівняльна характеристика аналогів

	My program	JetVetPass	Enote	Appointer
Повне створення всіх даних про тварин	+- (тільки тварина)	+- (тільки тварина)	+	+
Перегляд візитів до лікаря	+	+	+	+
Перегляд аналізів	+	+	-	-
Вартість	+	-	-	-
Доступність	+	+*	+	+
Платежі	-	-	+	+
Перенесення даних	-	-	-	+

* - Існує можливість перегляду як онлайн версії, так і мобільного додатка

1.3 Можливі, але не відповідні аналоги

Всього існує безліч аналогів, але серед них виявилася велика кількість подібних, але не відповідних за функціоналом систем, з відсутністю якоїсь складової, або з зовсім іншою тематикою, але подібним напрямок.

Наприклад, додаток «Equine Drugs», опис звучить як: «Використовується більш ніж 5000 ветеринарами в усьому світі, «Equine drugs» є найповнішим ветеринарним формуляром, доступним для використання на конях.»

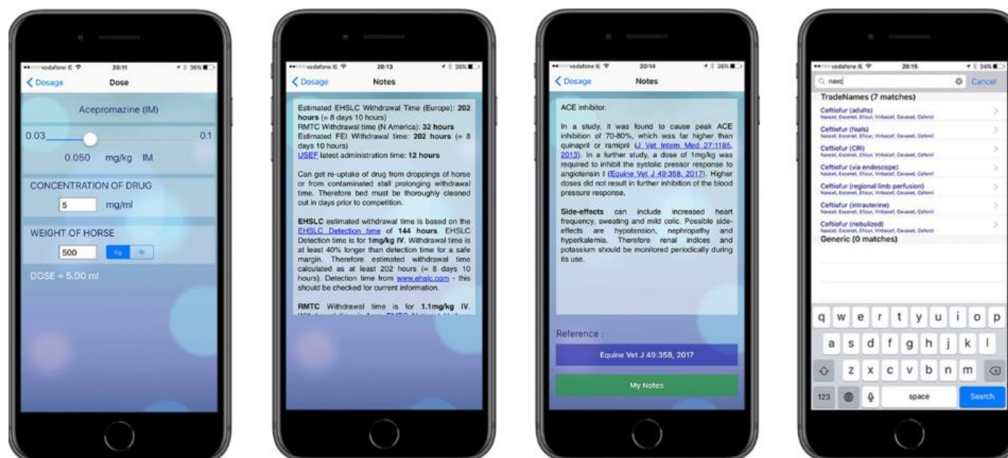

Рис 1.2 – Система Equine Drugs

З опису зрозуміло, що система підходить тільки для однієї тварини - коней, і не може розглядатися як аналог, тому що є вузькоспеціалізованою системою і не може покрити всі завдання, що необхідні від системи

Система Viralvet, являє собою систему для обміну досвіду та незвичайними випадками хвороби тварин, а саме опис звучить як:

«Діліться зображеннями та відео та обговорюйте цікаві та складні випадки з колегами-ветеринарами. Це весело, просто у використанні, освітнє, інтерактивне та абсолютно безкоштовно!

ViralVet — це перш за все дві речі — чудовий контент і захоплююче обговорення. Ми зробили це простим, щоб ви могли зосередитися на вивченні рідкісних та інтригуючих випадків, одночасно співпрацюючи з іншими над важкими діагнозами.»

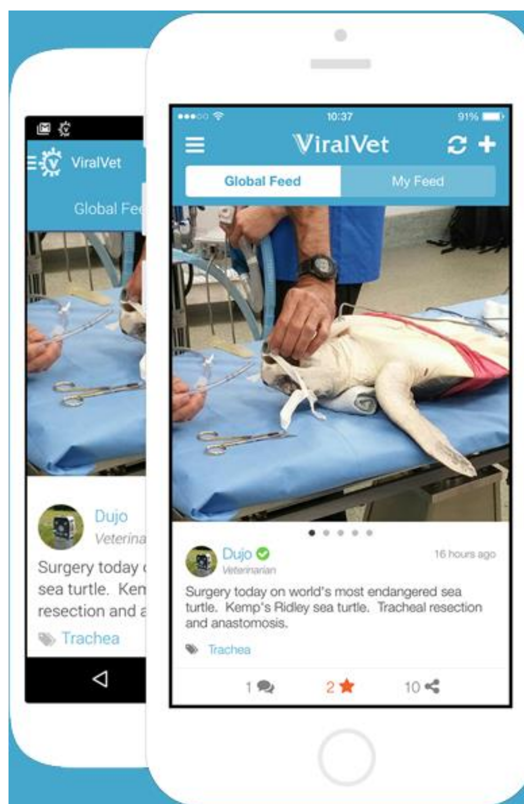

Рис 1.2 – Система Viralvet

Програма повністю задовольняє потребу в передачі інформації про тварин і хвороби, але абсолютно не підходить для поодиноких випадків і в цілому для зв'язку лікар-тварина-пацієнт.

Роблячи висновки з усіх цих факторів видно, що підбір аналогів повинен складатися виключно з набору певних характеристик, при цьому необхідно ретельно вивчати всі можливості аналогів, щоб не втратити нічого

РОЗДІЛ 2 ЗАСОБИ ТА МЕТОДОЛОГІЇ РОЗРОБКИ ДОДАТКУ

2.1. Обґрунтування вибору мов програмування

Мова програмування — це комп’ютерна мова, яка використовується програмістами (розробниками) для спілкування з комп’ютерами. Це набір інструкцій, написаних будь-якою конкретною мовою (C, C++, Java, Python) для виконання конкретного завдання.

Мова програмування в основному використовується для розробки настільних додатків, веб-сайтів і мобільних додатків. Мови бувають:

- Мова програмування низького рівня. Мова низького рівня є машинно-залежною (0s і 1s) мовою програмування. Процесор запускає програми низького рівня безпосередньо без необхідності компілятора чи інтерпретатора, тому програми, написані на мові низького рівня, можуть виконуватися дуже швидко.
- Мова програмування високого рівня. Мова програмування високого рівня (HLL) призначена для розробки зручних програм і веб-сайтів. Ця мова програмування потребує компілятора або інтерпретатора для перекладу програми на машинну мову (виконання програми).

Головна перевага мови високого рівня полягає в тому, що її легко читати, писати та підтримувати. Тобто нижче поріг входження в мову

Мова програмування високого рівня включає Python, Java, JavaScript, PHP, C#, C++, Objective C, Cobol, Perl, Pascal, LISP, FORTRAN і мову програмування Swift.

Застосунки для дипломної роботи складається з двох програм, що працюють окремо, програми сервера та програми клієнта.

Виходячи з цього слід вибрати дві мови програмування та базу даних для реалізації готових продуктів.

Розглянемо дані мови.

2.1.1. Вибір мови програмування для веб-застосунку клієнтської сторонни

Для клієнтської сторони спочатку навіть не стояло питання вибору мови, оскільки він існує лише один - Javascript.

Єдиний можливий вибір може бути лише між мовою Javascript та TypeScript.

TypeScript — мова програмування, розроблена Microsoft у 2012 році та є надбудовою для мови JavaScript.

TypeScript – це розширення специфікації ECMAScript 5. Додані такі опції:

- Анотації типів та перевірка їх узгодження на етапі компіляції
- Виведення типів
- Класи
- Інтерфейси
- Перераховані типи (Enum)
- Узагальнене програмування (Generics)
- Модулі
- Скорочений синтаксис "стрілок" для анонімних функцій

2.1.2. Вибір мови програмування для веб-застосунку серверної сторони

Серед серверних мов вибір значно більше, ніж серед клієнтської частини. Вибір складався з мов:

- Python
- Java
- JavaScript
- Ruby
- C#
- PHP

Проаналізувавши всі мови було вирішено вибрати мову програмування PHP, виходячи з таких переваг:

- Швидкість мови
- Простота та швидкість розробки
- Простота налаштування сервера
- Особистий досвід роботи з цією мовою
- С-подібний синтаксис мови легко читати.
- Динамічна типізація

PHP — це мова сценаріїв загального призначення, призначена для веб-розробки. PHP був задуманий восени 1994 року Расмусом Лердорфом. Ранні неопубліковані версії використовувалися на його домашній сторінці, щоб відстежувати, хто переглядає його онлайн-резюме. Перша версія, яку використовували інші, була доступна десь на початку 1995 року та була відома як Personal Home Page Tools.

Код PHP зазвичай обробляється на веб-сервері інтерпретатором PHP, реалізованим у вигляді модуля, демона або виконуваного файлу Common Gateway Interface (CGI). На веб-сервері результат інтерпретованого та виконаного PHP-коду, який може бути будь-яким типом даних, як-от згенерований HTML або двійкові дані зображення, утворить повну або часткову відповідь HTTP. Існують різноманітні системи веб-шаблонів, системи керування веб-контентом і веб-фреймворки, які можна використовувати для оркестрування або сприяння генерації відповіді. Крім того, PHP можна використовувати для багатьох завдань програмування поза веб-контекстом, таких як автономні графічні програми та роботизоване керування безпілотниками. Код PHP також можна виконувати прямо з консолі.

Стандартний інтерпретатор PHP на базі Zend Engine є безкоштовним програмним забезпеченням, випущеним за ліцензією PHP. PHP був широко перенесений і може бути розгорнутий на більшості веб-серверів на різноманітних операційних системах і платформах.

Мова PHP розвивалася без офіційної письмової специфікації чи стандарту до 2014 року, причому оригінальна реалізація діяла як стандарт де-факто, якому інші реалізації прагнули слідувати. З 2014 року тривала робота над створенням офіційної специфікації PHP.

W3Techs повідомляє, що станом на січень 2023 року «PHP використовують 77,8% усіх веб-сайтів, мова серверного програмування яких ми знаємо». Він також повідомляє, що лише 8% користувачів PHP використовують версії 8.x, які наразі підтримуються. Більшість використовує непідтримуваний PHP 7, точніше 7.4, і навіть PHP 5 має 23% використання, також не підтримується оновленнями безпеки, відомо, що він має серйозні вразливості безпеки.

2.2 Фреймворк

Фреймворк — абстракція, Програмне забезпечення, яке є каркасом, основою для майбутніх проектів і містить у собі необхідний мінімум, який значно спрощує, покращує і робить більш продуктивним розробку, містить уже готовий набір методів і властивостей для зручності користувача. Завдяки гнучкості фреймворків можливе їх використання для великого спектру завдань та проектів, від звичайного прототипування, до створення складних, високонавантажених систем

Фреймворки можуть включати програми підтримки, компілятори, бібліотеки коду, набори інструментів та інтерфейси прикладного програмування (API), які об’єднують усі різні компоненти для забезпечення розробки проекту чи системи.

Фреймворк має певні особливості, за якими їх можна відрізнити від бібліотек:

- Інверсія керування: у фреймворку, на відміну від бібліотек або стандартних користувальницьких програм, загальний потік керування програмою диктується не викликачем, а фреймворком. Зазвичай цього досягають за допомогою шаблону методу шаблону.

- Поведінка за замовчуванням: це може бути забезпечено інваріантними методами шаблону методу шаблону в абстрактному класі, який надається фреймворком.
- Розширюваність: користувач може розширити базовий функціонал за допомогою наприклад успадкування від суперкласів. Також можливе наслідування функціоналу через інтерфейси, що надаються самим фремворком.
- Незмінюваний код фреймворку: код фреймворка, як правило, не повинен бути змінений, але приймає розширення, реалізовані користувачем. Іншими словами, користувачі можуть розширювати структуру, але не можуть змінювати її код.

Таким чином, фреймворк може покривати деякі "болючі" місця в мовах, дозволяючи не робити певних перевірок.

Також фреймворк багаторазово дозволяє прискорити швидкість розробки програми, за допомогою вже вбудованих методів і абстракцій, а також дозволяє уникнути будь-яких помилок, попереджаючи про них.

Також фреймворк може пропонувати або пропонує реалізацію певних функцій, таких як, наприклад, lifecycle hooks, який надає можливість відстеження життєвого циклу компонента/додатку.

Також хороший (саме хороший) фреймворк задає строгі стилі написання (code standart), строгу (найчастіше строгу) файлову страуктуру, що дозволяє стандартизувати підхід до розробки, тим самим прискоривши перехід від одного проекту до іншого.

Також сувора файлова система найчастіше продумана логічно, що дозволяє розбити сайт або програму на модулі, тим самим відкривши доступ для lazy loading.

Також в основному будь-який фреймворк є екосистемою, яка згодом «обростає» різними корисними бібліотеками, пакетами.

Також найчастіше фреймворки йдуть разом із CLI, що дозволяє швидко створювати, запускати, розгортати проект, створювати компоненти та тести

В основному через ці переваги і були розроблені фреймворки

Після визначення, що таке фреймворк, необхідно вибрати їх для написання серверної та клієнтської частини програми. Виходячи з перерахованих вище переваг – написання додатка «чистою» мовою програмування навіть не розглядається.

2.2.1 Вибір фреймворку для веб-застосунку серверної сторони

Для серверної сторони було 3 можливі фреймворки на вибір:

- Laravel
- Yii
- Symfony

Далі розглянемо кожен із них

Переваги Laravel:

- Гарна документація
- Інтеграція з поштовими сервісами
- Підтримує популярні версії кешу (Memcached, Redis, тощо)
- Command-line interface Artisan
- Контейнер IoC
- ORM (doctrine)

Недоліки Laravel:

- Складність оновлення проекту за версіями
- Деякі частини фреймворку застарілі та складно підтримувані (Legacy code)

Переваги Yii:

- Генератори CRUD.
- Безліч варіантів макетів і тем для унікального дизайну веб-сторінки.
- Захист від ін’єкцій SQL, XSS, CSRF, підробки файлів cookie.
- Підтримує популярні версії кешу (Memcached, Redis, тощо)

- Велике ком’юніті.
- Автентифікації на основі ролей і вбудованого контролю користувача.

Недоліки Yii:

- Відносно високий поріг входження
- Складність у вивченні
- Складність у створенні логічної структури, при певних помилках у проектуванні - додаток буде проблемно розширювати
- Необхідність знання патернів навіть для найменшої програми

Переваги Symfony:

- Захищеність
- Гнучкість
- Легка підтримка
- Найкраща архітектура серед фреймворків
- Найкраще підходить для корпоративних додатів (Enterprise)
- Відмінна офіційна документація

Недоліки Symfony:

- Проблеми з продуктивністю на старих версіях
- Високий поріг входу

Проаналізувавши всі три фреймворки, можна сказати, що для написання масштабованого програми додатка найбільш підходить Symfony.

Так, поріг входження в цей фреймворк можна назвати найвищим, зате він компенсується гнучкістю, наявністю всіх необхідних інструментів для розробки повного циклу програми, а так само легкою масштабованістю програми, що дозволить у майбутньому без особливих проблем нарощувати функціонал.

Виходячи з усіх перерахованих вище переваг був обраний фреймворк Symfony.

2.2.2 Вибір фреймворку для веб-застосунку клієнтської сторони

З клієнтської сторони (або сторона фронтенду) існує 4 фреймворки. Визнано вважати, що фреймворків всього 2 Angular і AngularJS, але при будь-якому запиті при пошуку фреймворку для фронтенду, відповідь видасть 3 найбільш популярних:

- Vue
- React
- Angular

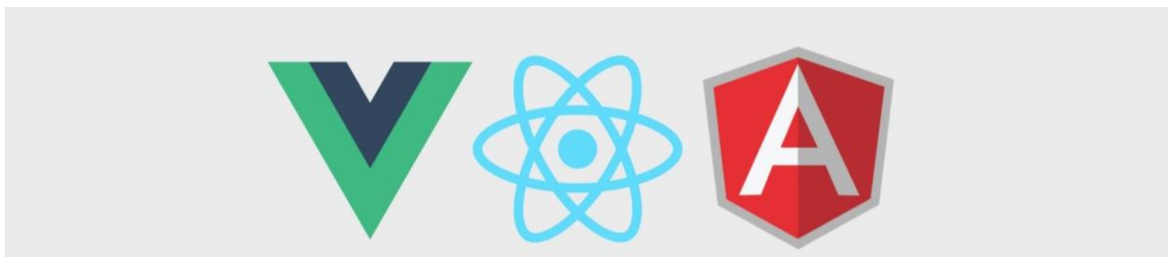

Рисунок 2.1 – Три найпопулярніші фронтенд фреймворку.
(Vue, React, Angular)

Vue (VueJS) – фреймворк, созданный Еван Ю, який працював у Google над проектами AngularJS. Він надихався фреймворком AngularJs, його легкістю та простотою, це послужило стимулом створення VueJs. Перший вихідний код був випущений у липні 2013 року, а VueJs — у лютому 2014 року.

Vue є MVVM фремворком, так само є реактивним, що дозволяє полегшити керування станом програми. Але у реактивності є інша сторона, у великих і складних програмах дуже легко потрапити в ситуацію, коли незрозуміло через що змінюється стан програми, тим самим реактивність з одного боку полегшує розробку, а з іншого боку ускладнює дебаггінг.

Переваги:

- Реактивність
- Маленький розмір фреймворку

- Швидкість роботи завдяки Virtual DOM
- Data binding (пов'язаність даних, двостороння зв'язність)
- Відносно низький поріг входу

Недоліки:

- Реактивність
- Відносно невелике комьюніті
- Невелика кількість великих проектів
- Через невелике ком'юніті складно, що взагалі неможливо знайти необхідне готове рішення, наприклад готовий слайдер або карусель

React це найпоширеніший JavaScript фремворк на поточний час. Хоча насправді фреймворком не вважається. Через відсутність будь-якої обв'язки з необхідних пакетів, таких як роутинг, вважається бібліотекою.

Переваги:

- Маленький розмір фреймворку
- Швидкість роботи завдяки Virtual DOM
- Найнижчий поріг входу
- Велике комьюніті
- Легко та швидко можна створити прототип програми
- Багато різноманітних бібліотек, інструкцій, плагінів та іншого, що покриває майже всі потреби при розробці
- Так як фреймворк не є реактивним, всі дані необхідно безпосередньо змінювати, оновлюючи стан компонента повністю, таким чином це легше відстежувати і у разі помилки легше виявити.

Недоліки:

- Найнижчий поріг входу, через це більшість розробників мають дуже низьку якість навичок у програмуванні
- Місцями абсолютно застаріла документація, наприклад, React переходить на функціональні компоненти, а в офіційній документації на головній сторінці все ще класові компоненти, які вважаються застарілими.

- Реакт не є реактивним, і стан крізь додатк можна контролювати тільки за допомогою Redux. А це тягне за собою величезну кількість коду, який при поганому плануванні дуже складно підтримувати, і який збільшується настільки стрімко, що навіть у не дуже великому додатку його стає складно підтримувати.
- JSX/TSX. Вся логіка компонента, стилі і шаблон може перебувати в одному файлі, що перетворює його в один великий шматок коду, що складно підтримується, який згодом стає все складніше підтримувати.

Цікавий факт, але кожна версія рамки називається на честь певного аніме твору (за винятком бігуна Blade, який є фільмом) [6].

```
28
29          default:
30              return "#000000";
31      }
32  };
33
    no usages   new *
34  export const GlassTypeWrapper = styled.span<{glassType: string}>`
35    font-weight: 700;
36    color: ${({ glassType }) => handleColor(glassType)};
37  `;
38
    3 usages
39  export const SearchForm = () => {
40    const { searchTerm, setSearchTerm } = useGlobalContext();
41
42    const handleChange = useCallback(
43      callback: (event: ChangeEvent<HTMLInputElement>) => {
44        if (setSearchTerm && typeof setSearchTerm === "function") {
45          setSearchTerm(event.currentTarget.value);
46        }
47      },
48      deps: [setSearchTerm]
49    );
50
51    return (
52      <>
53        <TextField
54          id="search-title"
55          label="Cocktail"
56          variant="outlined"
```

Рисунок 2.2 – TSX, Стилі, логіка та шаблон в одному компоненті

Реакт непоганий варіант для додатків, і так само для Enterprise рішень, але для підтримки великих проектів потрібен високий рівень навичок розробників, інакше проект має шанс розростись у величезний, не підтримує легаси код, в якому все складніше буде що-небудь додати і витрачатися величезна кількість часу на виправлення помилок.

AngularJS дуже часто плутають з Angular 2+, але це два абсолютно різні фреймворки.

На даний час і вже як кілька років AngularJS вважається застарілим, на ньому майже ніколи не створюють нові проекти і він майже повністю замінили Angular 2+. Але при цьому існує безліч проектів на ньому, що в свою чергу створює попит на розробників на цій застарілій технології.2.3 Вибір середовищ розробки для веб-застосунку.

AngularJS — це безкоштовна веб-платформа з відкритим вихідним кодом на основі JavaScript для розробки односторінкових програм. Він підтримувався в основному Google і спільнотою окремих осіб і корпорацій. Він мав на меті спростити як розробку, так і тестування таких додатків, надаючи структуру для клієнтської архітектури model–view–controller (MVC) і model–view–viewmodel (MVVM), а також компоненти, які зазвичай використовуються у веб-додатках і прогресивних веб-додатки.

AngularJS використовувався як інтерфейс стека MEAN, який складався з бази даних MongoDB, фреймворку сервера веб-додатків Express.js, самого AngularJS (або Angular) і середовища виконання сервера Node.js.

З 1 січня 2022 року Google більше не оновлює AngularJS.

Недоліки: Застарілий

Так як фреймворк є застарілим - на ньому не рентабельно і небезпечно розгортати проект, найчастіше при пошуку фронтенд фреймворко AngularJS вже навіть не потрапляє до списку.

Angular 2+ прийшов на зміну AngularJS. Команда розробників вирішила повністю переглянути концепцію цього фреймворку і створити абсолютно новий.

Великі оновлення фремворку виходять кожні півроку, підвищуючи версію на одиницю. Найактуальніша версія – 14.

Angular є повноцінним фреймворком, який надає свою CLI, за допомогою якої можна створювати:

- Сам проект
- app-shell
- application
- class
- component
- config
- directive
- enum
- environments
- guard
- interceptor
- interface
- library
- module
- pipe
- resolver
- service
- service-worker
- web-worker

Таким чином, за допомогою CLI можна створювати всі можливі компоненти програми, з уже готовими наборами класів та успадкування інтерфейсів.

Переваги:

- Dependency injection
- CLI

- Модульність
- Власна архітектура, що дозволяє розділяти компоненти на шаблон, логіку та стилі, кожен з яких знаходиться в окремому файлі.
- Директиви
- Typescript
- Фрефмворк заточений під Enterprise програми
- Реактивність
- Data Binding
- Покриваність тестами «з коробки»
- Можливість створення бібліотек засобами Angular
- Існують вже готові рішення для різних ЦМС та платформ для управління продажами та персоналом

Недоліки:

- Високий поріг входу
- Реактивність
- Складність у навчанні
- Більшість ком'юніті фреймворку знаходиться в Індії, і якість їх гайдів та інструкцій залишає бажати кращого.
- Складність у навчанні
- При неправильному проектуванні та написанні логіки коду - фреймфорк працює дуже повільно і вимагає переписування для поліпшення продуктивності [9].

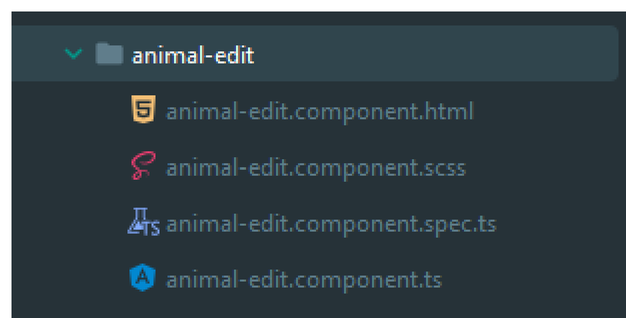

Рисунок 2.3 – Приклад розбиття компонента на файли

Я вважаю, що розбиття компонентів на підфайли є найкращим підходом. Проект має бути розбитий на модулі, модулі на підмодулі. Модулі повинні складатися з компонентів, а компоненти в свою чергу розбиті на невеликі файли.

Саме таким чином можна перевикористовувати компоненти, при цьому при їх зміні або неправильній роботі буде легко знайти помилку і буде завдано найменшої шкоди системі.

Під час створення нової програми за допомогою CLI Angular пропонує такі можливості:

- Надсилання анонімної статистики
- Можливість додавання Angular router
- Вибір використання препроцесора для CSS

```
$ ng new test-app
? Do you want to enforce stricter type checking and stricter bundle budgets in the workspace?
  This setting helps improve maintainability and catch bugs ahead of time.
  For more information, see https://angular.io/strict Yes
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? SCSS   [ https://sass-lang.com/documentation/syntax#scss    ]

CREATE test-app/angular.json (3721 bytes)
CREATE test-app/package.json (1198 bytes)
CREATE test-app/README.md (1016 bytes)
CREATE test-app/tsconfig.json (783 bytes)
CREATE test-app/tslint.json (3185 bytes)
CREATE test-app/.editorconfig (274 bytes)
CREATE test-app/.gitignore (631 bytes)
CREATE test-app/.browserslistrc (703 bytes)
CREATE test-app/karma.conf.js (1425 bytes)
CREATE test-app/tsconfig.app.json (287 bytes)
CREATE test-app/tsconfig.spec.json (333 bytes)
CREATE test-app/src/favicon.ico (948 bytes)
CREATE test-app/src/index.html (293 bytes)
```

Рисунок 2.4 – Консоль створення нового проекту Angular

Angular дуже зручний фреймворк для великих Enterprise рішень, який "з коробки" додає тести до компонентів та сервісів.

Однак виходячи з мого досвіду дуже ймовірний результат, що при неправильній архітектурі та підході до розробки – фреймворк працюватиме дуже повільно, запускатиметься пару хвилин і дуже навантажуватиме пристрій.

Також фреймворк написаний з урахуванням реактивності, саме бібліотеки RxJS.

RxJS — це бібліотека для реактивного програмування з використанням Observables, щоб полегшити створення асинхронного коду або коду на основі зворотного виклику. Цей проект є переписаним Reactive-Extensions/RxJS із кращою продуктивністю, кращою модульністю, кращими стеками викликів із можливістю налагодження, зберігаючи в основному зворотну сумісність із деякими критичними змінами, які зменшують поверхню API [7].

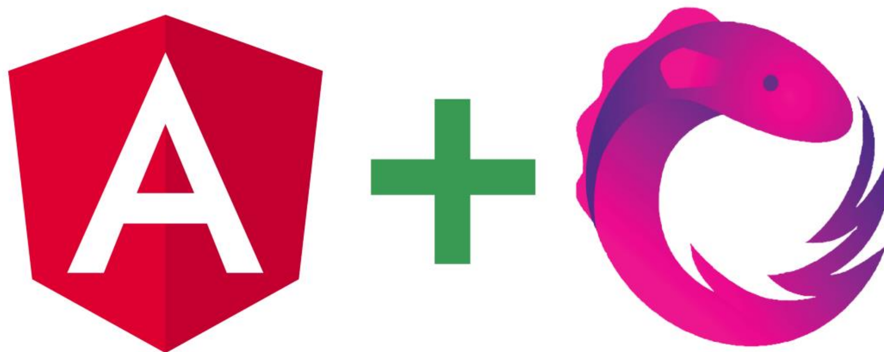

Рисунок 2.5 – Основна бібліотека в Angular - RxJs

Таким чином, для впевненого написання коду на Angular потрібно ще глибоке знання бібліотеки RxJS, а саме:

- Реактивність та асинхронність
- Operators
- Pipelines

Основні оператори RxJS [8]:

- Створення (of, from, fromEvent, interval)
- Перетворення (map, scan, buffer)
- Фільтрації (filter, take, skip, distinct)
- Опрацювання помилок (catchError, retry, onErrorResumeNext)
- Умови (skipUntil, skipWhile, takeUntil, takeWhile)
- Математичні (min, max, count)
- Утиліти (tap, delay)

2.2.3 Середовище розробки

Після вибору мов розробки необхідно вибрати середовища розробки для максимального зручного та швидкого процесу створення додатків.

Інтегроване середовище розробки (IDE) — це програма, яка допомагає програмістам ефективно розробляти програмний код. Він підвищує продуктивність розробників, поєднуючи такі можливості, як редагування програмного забезпечення, збірка, тестування та упаковка в зручній у використанні програмі. Розробники програмного забезпечення використовують IDE, щоб полегшити собі роботу.

Добре середовище розробки значно прискорює швидкість розробки програмного забезпечення. Хороше середовище розробки надає такі можливості:

- Автоматичне редагування коду та підскахування в режимі реального часу
- Підсвічування синтаксису, у тому числі помилок
- Швидке створення певних шаблонів, у тому числі можливість створення шаблонів користувача (Live templates)
- Доповнення коду в режимі реального часу, деякі середовища розробки також намагаються додавати штучний інтелект у такі процеси
- Засоби налагодження коду

Існують два типи середовищ розробки:

- Локальна, яка встановлюється на пристрій розробника
- Серверна, що знаходиться на віддаленому сервері

Серверний тип середовищ розробки (sandbox)[10] дозволяє розробникам обмінюватися своїми напрацюваннями, створювати шаблони додатків з помилками для можливості відтворення і просто використовувати їх як портфоліо.Основные факторы для выбора IDE это

- Зручність використання
- Продуктивність

- Підтримка мови/мов програмування
- Вартість

2.2.3.1 Вибір середовища розробки серверної сторони

При виборі середовища розробки спочатку вибір було зроблено у бік Jetbrains PHPStorm.

PHPStorm - PhpStorm являє собою інтелектуальний редактор для PHP, HTML і JavaScript з можливостями аналізу коду на льоту, запобігання помилок у сирцевому коді і автоматизованими засобами рефакторинга для PHP і JavaScript. Автодоповнення коду в PhpStorm підтримує специфікацію PHP 5.3/5.4/5.5/5.6/7.0/7.1 (сучасні і традиційні проекти), включаючи генератори, співпрограми, простори імен, замикання, типажі і синтаксис коротких масивів. Присутній повноцінний SQL-редактор з можливістю редагування отриманих результатів запитів.

PhpStorm розроблений на основі платформи IntelliJ IDEA, написаної на Java. Користувачі можуть розширити функціональність середовища розробки за рахунок установки плаґінів, розроблених для платформи IntelliJ, або написавши власні плаґіни.

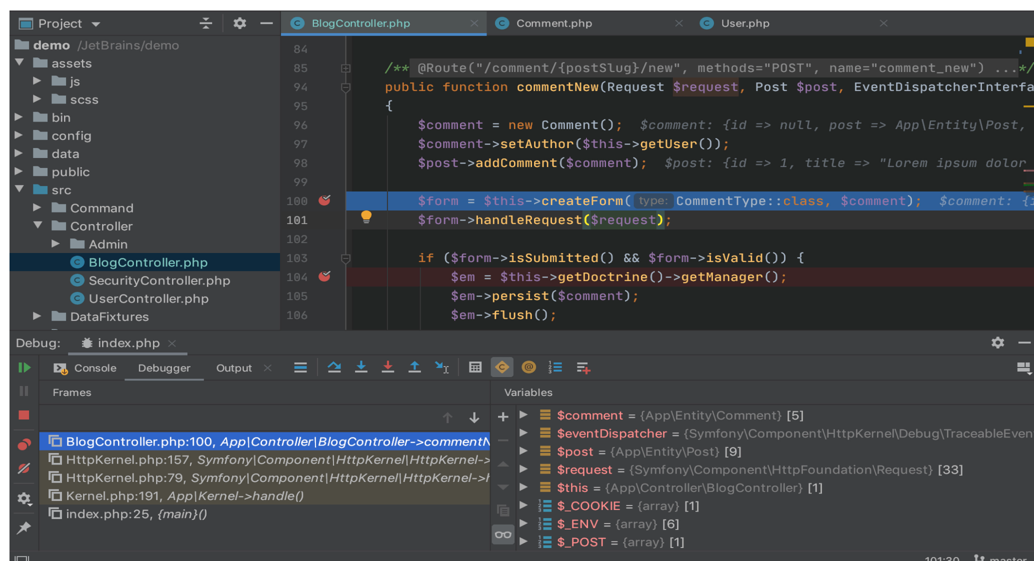

Рисунок 2.6 – Середовище розробки PHPStorm

У мене вже був досвід розробки у цьому середовищі розробки. На мою думку, це одна з найзручніших, якщо не найкраща IDE.

Єдиний і найголовніший мінус, через який не було обрано це середовище розробки – вартість у 99$ на рік. Тому для дипломної роботи було вирішено шукати безкоштовне середовище розробки.

Перепробувавши різні середовища розробки, я зупинив свій вибір на Visual Studio Code. Ця IDE є звичайним редактором, який дозволяє за допомогою плагінів налаштувати себе під різні мови розробки. З мінусів можу відзначити не таке повне доповнення коду, погану підтримку фреймворків і відносно незручний інтерфейс порівняно з PHPStorm.

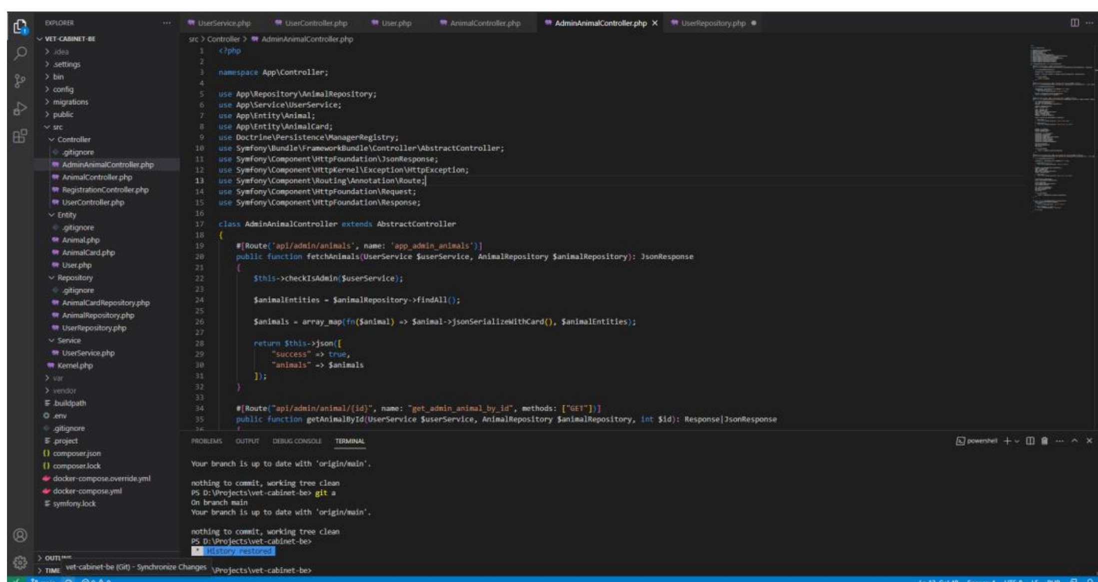

Рисунок 2.7 – Середовище розробки Visual Studio Code

2.2.3.2 Вибір середовища розробки клієнтської сторони

Для клієнтської боку вибір середовища розробки не стояв. Виходячи з мого досвіду найкращим середовищем розробки є WebStorm.

JetBrains WebStorm — інтегроване середовище розробки для JavaScript, HTML та CSS від компанії JetBrains, розроблена на основі платформи IntelliJ IDEA. WebStorm є спеціалізованою версією PhpStorm, пропонуючи підмножину

з його можливостей. WebStorm постачається з перед-установленим плагінами JavaScript (такими як для Node.js), котрі доступні для PhpStorm безкоштовно.

WebStorm підтримує мови JavaScript, CoffeeScript, TypeScript та Dart.

WebStorm забезпечує автодоповнення, аналіз коду на льоту, навігацію по коду, рефакторинг, зневадження та інтеграцію з системами управління версіями. Важливою перевагою інтегрованого середовища розробки WebStorm є робота з проектами (у тому числі, рефакторинг коду JavaScript, що міститься в різних файлах і теках проекту, а також вкладеного в HTML). Підтримується множинна вкладеність (коли в документ на HTML вкладений скрипт на Javascript, в який вкладено інший код HTML, всередині якого вкладений JavaScript) — в таких конструкціях підтримується коректний рефакторинг.

Так як Webstorm створювався на основі PHPStorm, їх інтерфейси фактично ідентичні, що дозволяє швидко перемикатися між IDE з повним розумінням того як нова IDE працює, без втрати часу на навчання.

Також усі продукти Jetbrains дозволяють імпортувати/експортувати налаштування між середовищами розробки. Також можливо їх синхронізувати через сервер.

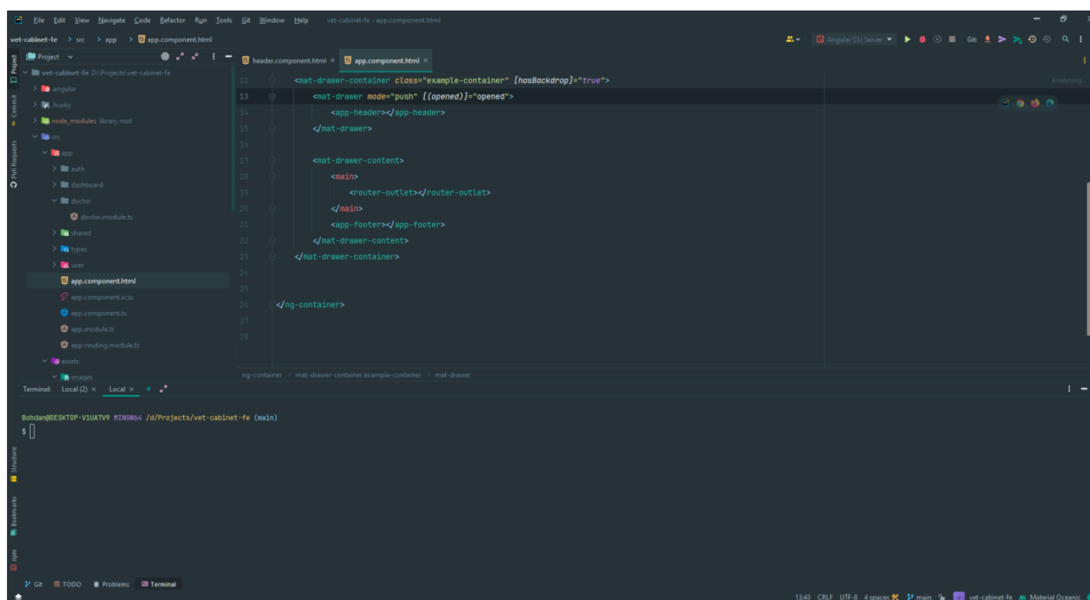

Рисунок 2.8 – Середовище розробки Webstorm

Вартість Webstorm на даний момент складає 69 $ на рік, завдяки тому, що я постійний клієнт і купую його вже декілька років, то зі знижкою виходить 41 $.[11]

Виходячи з перерахованих вище факторів і того, що IDE вже була куплена – було вирішено використовувати Webstorm.

Таким чином, для серверної сторони була обрана найбільш зручна і відповідна для умов програмного продукту мова програмування PHP, для клієнтської сторони особливого вибору не було, виходячи з особливостей фреймворку Angular.

Були розглянуті всі можливі фреймворки для розробки на клієнтській та серверній стороні, з яких було обрано два найбільш підходящі, Angular – для клієнтської сторони та Symfony для серверної.

Також були розглянуті можливі середовища розробки, і виходячи з їх зручності та фінансової складової, було обрано 2 середовища розробки, WebStorm – для клієнтської сторони та Visual Studio Code з доповненнями для серверної.

РОЗДІЛ 3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Проектування кінцевого продукту

3.1.1. Роутінг та RestAPI

Крім стандартних вимог до продукту, таких як:

- Стійкість до помилок
- Стабільність
- Розширюваність
- Тощо.

Є ще низка вимог:

- Два окремі програми, пов'язані з RestAPI, замість моноліту.
- Дизайн, заснований на Material UI (доступний npm пакет для Angular)

API, або інтерфейс програмування додатків, — це набір правил, які визначають, як програми або пристрої можуть підключатися та спілкуватися один з одним. REST API — це API, який відповідає принципам проектування REST або архітектурному стилю передачі репрезентативного стану.

RestAPI складається з 6 основних принципів:

- Уніфікований інтерфейс. Таким чином досягається взаємодія клієнт-сервера незалежно від програми або девайсу.
- Клієнт-сервер. Клієнт-сервер передбачає поділ завдань, що допомагає компонентам клієнта та сервера розвиватися незалежно. Відокремлюючи інтерфейс користувача (клієнт) від проблем зберігання даних (сервер), ми покращуємо переносимість інтерфейсу користувача між кількома платформами та покращуємо масштабованість за рахунок спрощення компонентів сервера. Поки клієнт і сервер розвиваються, ми маємо переконатися, що інтерфейс/контракт між клієнтом і сервером не порушується. Грунтуючись на цьому принципі можливо використання

одного і того ж інтерфейсу багатьма клієнтами, наприклад веб-додаток, андроїд додатком, тощо.

- Відсутність стану. Відсутність стану передбачає, що кожен запит з програми-клієнта повинен містити всю інформацію для успішної аутентифікації, таку як bearer-token. Сервер не зберігає будь-якої інформації на свій бік і всю інформацію повинен надати клієнт. Це дозволяє легко налагоджувати програми, адже можна просто використовувати токен користувача в сторонньому додатку, наприклад в Postman і відправляти з нього запити для перевірки сервера, навіть коли клієнтська сторона ще не почала розробку.

- Кешування. Запити можуть позначатися як кешовані, для зменшення використання трафіку користувача і для збільшення швидкості запитів. Не всі запити підлягають кешування. Крім серверного кешування ще існує кешування самим браузером, який можна обійти за допомогою додавання в запит поточного позначення часу.

- Багатошарова система. Система може складатися з безлічі шарів, що відповідає за свою сферу відповідальності. Наприклад у кожному запиті присуває сервіс аутентифікації, який перевіряє запит, і за його неповності - відвертає помилку ще до того, як запит дійде основний бізнес логіки. Також можна налаштувати додаткові проміжні шари, такі як ABC сервіс для кешування та обробки помилок, приховування адреси сервера, тощо.

- Код на вимогу (необов'язково). REST також дозволяє розширити функціональність клієнта шляхом завантаження та виконання коду у формі аплетів або сценаріїв.Завантажений код спрощує роботу клієнтів, зменшуючи кількість функцій, які необхідно попередньо впровадити. Сервери можуть надавати частину функцій, які надаються клієнту у вигляді коду, а клієнту потрібно лише виконати код. Одного разу мені потрібно було розробити додаток-систему, яка керувалася б з сервера за допомогою великих файлів налаштувань, таким чином можна було

створювати величезну кількість клієнтів, які працюють і виглядають по-різному, але мають однакову кодову базу, підлаштовуючи кожен з них під потрібного клієнта.

Фреймоврк Symfony має в собі декларативну систему роутингу, яка дозволяє гнучко її налаштувати під потрібні розробника. Також завдяки консольній команді php bin/console debug:router, консоль виведе всі зареєстровані шляхи серверного додатка в зручному форматі. Шляхи виглядають наступним чином:

Таблиця 3.1 – Система роутингу серверної програми

Назва	Методи	Шлях
api_login_check	GET/POST	/api/login
get_user	GET	/api/user
register	POST	/api/register
get_user_animals	GET	/api/animals
create_animal	POST	/api/animal
get_animal_by_id	GET	/api/animal/{id}
update_animal_by_id	POST	/api/animal/{id}
create_animal_analysis	POST	/api/animal/{id}/analysis
get_animal_analysis_by_id	GET	/api/animal/{id}/analysis
create_referral	POST	/api/referral/create
get_doctor_referrals	GET	/api/referral/by_animal/{id}
get_referral_by_animals	GET	/api/referral/by_doctor/{id}
create_admin_animal	POST	/api/admin/animal
update_admin_animal_by_id	POST	/api/admin/animal/{id}
get_admin_animal_by_id	GET	/api/admin/animal/{id}
app_admin_animals	GET	/api/admin/animal
get_admin_users	GET	/api/admin/users

- api_login_check - Шлях для входу користувача до системи
- get_user - Шлях для отримання даних про користувача по токену сесії
- register - Шлях для реєстрації як користувача, так і лікаря-адміністратора
- get_user_animals - Шлях для отримання всіх тварин поточного користувача по сесії.
- create_animal - Шлях для створення запису нової тварини.
- get_animal_by_id - Шлях для отримання єдиної сутності тварини створений для зменшення передачі даних і меншого обсягу зберігання даних на стороні клієнта. Клієнт може зайти на сторінку єдиної тварини і немає необхідності отримувати список усіх тварин.
- update_animal_by_id - Оновлення запису тварини.
- create_animal_analysis - Створення сутності аналізу для тварини.
- get_animal_analysis_by_id - Набуття сутності одиничного аналізу.
- create_referral - Створення направлення на візит до лікаря.
- get_doctor_referrals - Отримання візитів за певним лікарем, використовується лікарем для перегляду візитів власників тварин.
- get_referral_by_animals - Отримання візитів по певній тварині, використовується як користувачем для перегляду візитів, так і лікарем, для перегляду візитів виключно для однієї тварини
- create_admin_animal - Створення сутності тварини лікарем.
- update_admin_animal_by_id - Оновлення сутності тварини лікарем.
- get_admin_animal_by_id - Отримання сутності тварини певної тварини, відрізняється від get_animal_by_id тим, що адміністратор може отримати сутність будь-якої тварини, користувач може отримати запис виняткового того тварини, власником якого є.
- app_admin_animals - Отримання всіх тварин від бази даних

- get_admin_users – Отримання всіх користувачів програми, не безпечний шлях, але варто уточнити, що природно виходить мінімальна інформація, що виключає паролі.

Можна помітити, що шляхи

- update_admin_animal_by_id та get_admin_animal_by_id
- get_animal_analysis_by_id та create_animal_analysis
- get_animal_by_id та update_animal_by_id

Маю повністю однаковий шлях, і відрізняються виключно типом запиту (GET/POST), таким чином можна розділяти логіку програми, викликаючи різні методи контролера по тому самому шляху, в залежності від типу запиту.

Усі методи, за винятком входу та реєстрації повинні містити bearer-token.

3.1.2. Аутентифікація

Аутентифікація у додатку була виконана через додатковий пакет, а саме LexikJWTAuthenticationBundle. Як спосіб аутентифікації було обрано JWT token.

JSON Web Token (JWT) — це відкритий стандарт (RFC 7519) для безпечної передачі інформації між сторонами як об’єкт JSON.

Він компактний, читабельний і підписаний постачальником ідентифікаційної інформації (IdP) за допомогою закритого ключа/або пари відкритих ключів. Таким чином, цілісність і автентичність токена можуть бути перевірені іншими залученими сторонами.

Метою використання JWT є не приховування даних, а забезпечення автентичності даних. JWT підписаний і закодований, а не зашифрований.

JWT — це механізм автентифікації без стану на основі маркерів. Оскільки це сеанс без стану на стороні клієнта, сервер не повинен повністю покладатися на сховище (базу даних) для збереження інформації про сеанс.

Файл налаштувань для бібліотеки виглядає так:

Лістинг 3.1 – Файл налаштувань LexikJWTAuthenticationBundle

```
security:
    enable_authenticator_manager: true
    password_hashers:
        App\Entity\User: 'auto'
        Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:
            algorithm: 'auto'
            cost:      15
    providers:
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email
    firewalls:
        login:
            pattern: ^/api/login
            stateless: true
            json_login:
                check_path: /api/login
                username_path: email
                password_path: password
                success_handler: lexik_jwt_authentication.handler.authentication_success
                failure_handler: lexik_jwt_authentication.handler.authentication_failure

        api:
            pattern:   ^/api
            stateless: true
            jwt: ~
        dev:
            pattern: ^/(_(profiler|wdt)|css|images|js)/
            security: false
        main:
            lazy: true
            provider: app_user_provider

    access_control:
        - { path: ^/api/register, roles: PUBLIC_ACCESS  }
        - { path: ^/api/login, roles: PUBLIC_ACCESS  }
        - { path: ^/api,       roles: IS_AUTHENTICATED_FULLY }
```

Розглянемо окремі ділянки коду:

Лістинг 3.2 – Налаштування класу для автентифікації

```
    providers:
        app_user_provider:
            entity:
                class: App\Entity\User
                property: email
```

У цій ділянці коду налаштовується сутність, через яку проходитиме автентифікація та поле для входу як логіна

Лістинг 3.3 – Налаштування шляху та обробники помилок

```
    firewalls:
```

```
login:
    pattern: ^/api/login
    stateless: true
    json_login:
        check_path: /api/login
        username_path: email
        password_path: password
        success_handler: lexik_jwt_authentication.handler.authentication_success
        failure_handler: lexik_jwt_authentication.handler.authentication_failure
```

У цій ділянці коду налаштовуються шлях для запиту автентифікації. Також налаштовуються поле сутності входу. Бібліотека пропонує базові обробники помилок під назвою authentication_success та authentication_failure, далі надано два варіанти помилок та коректний процес аутентифікації:

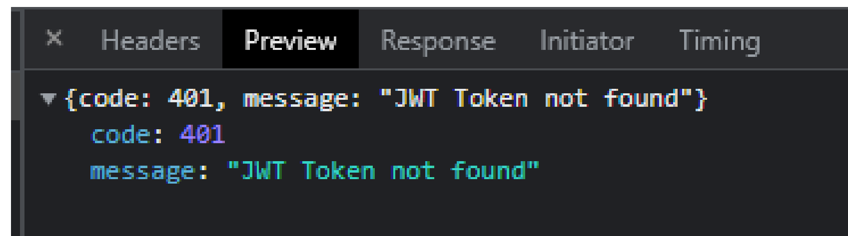

Рисунок 3.1 – Відсутність токена у запиті

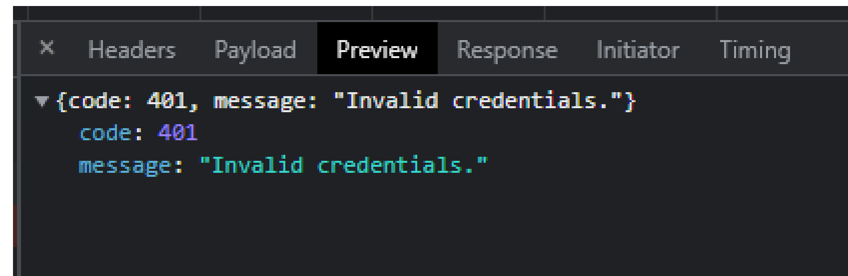

Рисунок 3.2 – Неправильні дані при вході

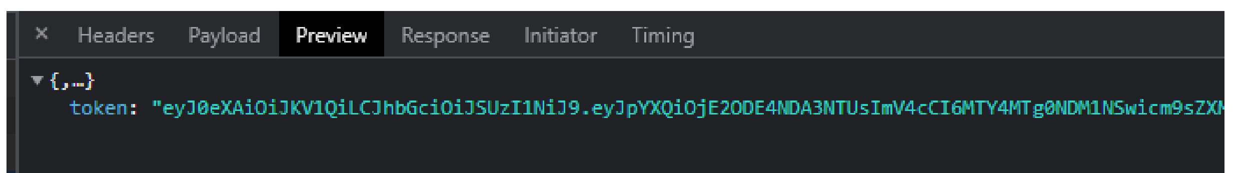

Рисунок 3.3 – Відповідь сервера при коректних даних аутентифікації

Бібліотека надає "з коробки" функціонал, який підходить більшості розробників.

Бібліотека також дозволяє налаштувати права доступу на певні шляхи, таким чином дозволивши або обмеживши доступ. Цю можливість можна використовуватиме захисту, чи дозволяти певні частини програми всім бажаючим користувачами, наприклад форму зворотний зв'язок чи чати.

Лістинг 3.4 – Налаштування доступу для шляхів

```
access_control:
    - { path: ^/api/register, roles: PUBLIC_ACCESS  }
    - { path: ^/api/login, roles: PUBLIC_ACCESS  }
    - { path: ^/api,       roles: IS_AUTHENTICATED_FULLY }
```

Також вже в самій бізнес логіці можливе отримання сутності користувача з токена средставим Symfony, це можна реалізувати наступним способом:

```
public function getUserFromToken(): ?User {
    $token = $this->tokenStorage->getToken();
    if ($token instanceof TokenInterface) {
        return $token->getUser();
    }
    return null;
}
```

Рисунок 3.4 – Код отримання користувача з токену

3.1.3. Структура серверної сторони

Фреймворк Symfony задає строгу файлову структуру для програми, яка в основному виглядає так:

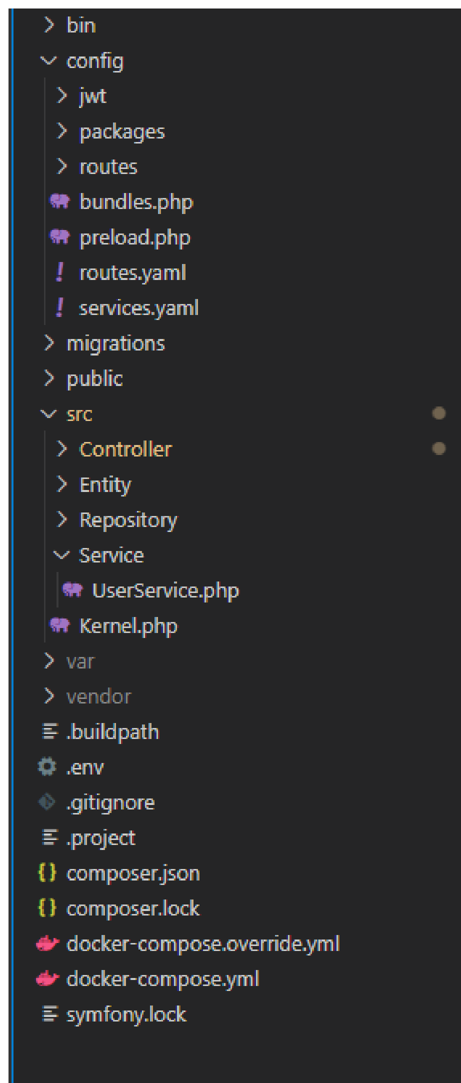

Рисунок 3.5 – Файлова структура серверної частини

Розглянемо кожну з папок:

- /config – папка налаштувань програми. Усі встановлені через комп'ютер бібліотеки додають файли конфігурації саме сюди.
- /migrations – папка з міграцією для бази даних, створених через ORM Doctrine.
- /public – містить index.html, точку входу для програми.
- /src/Controller – папка, що містить усі контролери програми. Всі нові контролери створюються через консольну команду symfony console make: controller CONTROLLER_NAME, яка створює контролер, додає його до папки та реєструє.

- /src/Entity – папка, що містить всі сутності програми, всі сутності створюються через консоль за допомогою ORM Doctrine і автоматично додаються до цієї папки.
- /src/Repository – папка, що містить всі репозиторії сутностей для проекту, створюється одночасно зі створенням сутності та містить набір базових функцій для операцій із сутністю та базою даних.
- /src/Service – папка, що містить сервіси з бізнес логікою для застосування.
- /.env – файл конфігурації всієї програми. Містить у собі різні налаштування, у тому числі секрети фрази для JWT аутентифікації, рядок підключення до бази диннх і так далі, найчастіше, також є можливість заміни файлу конфігурації за допомогою створення нових, для певних оточень, як наприклад .env.local, . env.$APP_ENV, .env.$APP_ENV.local, які будуть перезаписувати основний файл конфігурації локально, або на якомусь певному оточенні, наприклад DEV.
- /.composer.json – файл, що містить базові команди, список пакетів додатків та їх версії. Усі встановлені бібліотеки будуть автоматично додані сюди. Файл необхідний для встановлення тих самих бібліотек, необхідних версій при розгортанні проекту. Завдяки цьому немає необхідності зберігати всі бібліотеки Git, що дозволяє зберігати величезну кількість пам'яті.
- /.composer.lock – автоматично генерований файл після установок бібліотек. Містить всю інформацію про вже встановлені бібліотеки. Оновлюється під час оновлення та видалення бібліотек.

Завдяки тому, що Synfomy це досить суворий фреймворк, досить складно відійти від стандартів проектування. Також завдяки вбудованим устилітам у bin/console, майже всі необхідні файли створюються не вручну, а за допомогою CLI, автоматично реєструють, що спрощує роботу з ними та з Dependency Injection.

3.1.4. ORM Doctrine

Для роботи з базою даних можна використовувати мову запитів SQL, але для прискорення і підвищення зручності з БД також можна використовувати Object Relational Mapping.

Object Relational Mapping (ORM) — це техніка (шаблон проектування) доступу до реляційної бази даних з об’єктно-орієнтованої мови. Це не що інше, як сутності.

Щоб отримати доступ до бази даних в об’єктно-орієнтованому контексті, необхідний інтерфейс, що транслює логіку об’єкта, цей інтерфейс називається ORM. Він складається з об’єкта, який надає доступ до даних і зберігає бізнес-правила при собі.

Завдяки створеним сутності полегшується робота з БД через інтерфейси.

Наприклад створення сутності Animal виглядає так:

Лістинг 3.5 – Створення сутності Animal

```
$animal = new Animal();
$animal->setOwner($user);
$animal->setName($name);

$animalCard = new AnimalCard();
$animalCard->setType($type);
$animalCard->setAge($age);
$animalCard->setDescription($description);
$animalCard->setNotes($notes);
$animalCard->setAnimal($animal);
$animalCard->setGender($gender);
$animalCard->setDiagnoses($diagnoses);

$em->persist($animal);
$em->persist($animalCard);
$em->flush();
```

Кожна властивість сутності визначається через відповідний сетер класу:

Лістинг 3.6 – Getter та Setter сутності Animal

```
    public function getOwner(): ?User
    {
        return $this->owner;
```

```
    }

    public function setOwner(?User $owner): self
    {
        $this->owner = $owner;

        return $this;
    }
```

Оновлення сутності відбувається абсолютно ідентичним чином, за винятком того, що клас сутності не створюється, а дістається з БД за допомогою репозиторію:

```
$animalEntity = $animalRepository->findOneBy(
    array(
        "owner" => $user->getId(),
        "id" => $id
    )
);
```

Рисунок 3.6 – Пошук сутності через функції репозиторію

Також ORM полегшує розробку тим, що надає вбудовані функції створення моделей. У ORM Doctrine сутність створюється через консольну команду php bin/console make:entity.

При цьому консоль запитує розробника назву сутності і які поля сутність матиме.

Консоль запитує такі параметри сутності:

- Назва сутності
- Тип даних поля
- Довжина поля
- Чи є поле nullable
- Якщо поле це зв'язок з іншою сутністю, то запитується назва сутності та тип зв'язку: OneToOne, OneToMany, ManyToMany.

```
1  $ php bin/console make:entity
2
3  Class name of the entity to create or update:
4  > Product
5
6  New property name (press <return> to stop adding fields):
7  > name
8
9  Field type (enter ? to see all types) [string]:
10 > string
11
12 Field length [255]:
13 > 255
14
15 Can this field be null in the database (nullable) (yes/no) [no]:
16 > no
17
18 New property name (press <return> to stop adding fields):
19 > price
20
21 Field type (enter ? to see all types) [string]:
22 > integer
23
24 Can this field be null in the database (nullable) (yes/no) [no]:
25 > no
26
27 New property name (press <return> to stop adding fields):
28 >
29 (press enter again to finish)
```

Рисунок 3.7 – Процес створення нової сутності через CLI

3.1.5. Міграції

Після створення сутності ORM пропонує створити файл міграції, з повним описом, що відбудеться при його застосуванні або при скасуванні.

Відразу після створення сутності консоль підказує з командою php bin/console make:migration. Після введення цієї команди створюється файл у папці /migration з певною версією. Міграція виглядає приблизно так:

```
0 references | prototype
public function up(Schema $schema): void
{
    // this up() migration is auto-generated, please modify it to your needs
    $this->addSql('CREATE TABLE analys (id INT AUTO_INCREMENT NOT NULL, name VARCHAR(255) NOT NULL, animal INT NOT NULL, datetime VARCHAR(255) NOT NULL, indicators LONGTE
    $this->addSql('CREATE TABLE animal (id INT AUTO_INCREMENT NOT NULL, owner_id INT NOT NULL, name VARCHAR(255) NOT NULL, INDEX IDX_6AAB231F7E3C61F9 (owner_id), PRIMARY
    $this->addSql('CREATE TABLE animal_card (id INT AUTO_INCREMENT NOT NULL, animal_id INT NOT NULL, type VARCHAR(255) NOT NULL, gender VARCHAR(255) NOT NULL, age DOUBLE
    $this->addSql('CREATE TABLE refferal (id INT AUTO_INCREMENT NOT NULL, animal_id INT DEFAULT NULL, doctor INT NOT NULL, datetime VARCHAR(255) NOT NULL, notes LONGTEXT
    $this->addSql('CREATE TABLE user (id INT AUTO_INCREMENT NOT NULL, email VARCHAR(180) NOT NULL, roles JSON NOT NULL, password VARCHAR(255) NOT NULL, role VARCHAR(255)
    $this->addSql('ALTER TABLE animal ADD CONSTRAINT FK_6AAB231F7E3C61F9 FOREIGN KEY (owner_id) REFERENCES user (id)');
    $this->addSql('ALTER TABLE animal_card ADD CONSTRAINT FK_5139D9E78E962C16 FOREIGN KEY (animal_id) REFERENCES animal (id)');
    $this->addSql('ALTER TABLE refferal ADD CONSTRAINT FK_F96988F68E962C16 FOREIGN KEY (animal_id) REFERENCES animal (id)');
}

0 references | prototype
public function down(Schema $schema): void
{
    // this down() migration is auto-generated, please modify it to your needs
    $this->addSql('ALTER TABLE animal DROP FOREIGN KEY FK_6AAB231F7E3C61F9');
    $this->addSql('ALTER TABLE animal_card DROP FOREIGN KEY FK_5139D9E78E962C16');
    $this->addSql('ALTER TABLE refferal DROP FOREIGN KEY FK_F96988F68E962C16');
    $this->addSql('DROP TABLE analys');
    $this->addSql('DROP TABLE animal');
    $this->addSql('DROP TABLE animal_card');
    $this->addSql('DROP TABLE refferal');
    $this->addSql('DROP TABLE user');
}
}
```

Рисунок 3.8 – Файл міграцій

Після цього можна застосувати міграцію до бази даних за допомогою команди php bin/console doctrine: migrations: migrate. При успішному виконанні процедури БД буде оновлено відповідно до останньої версії міграції, а в саму БД, до таблиці doctrine_migration_versions буде додано записи:

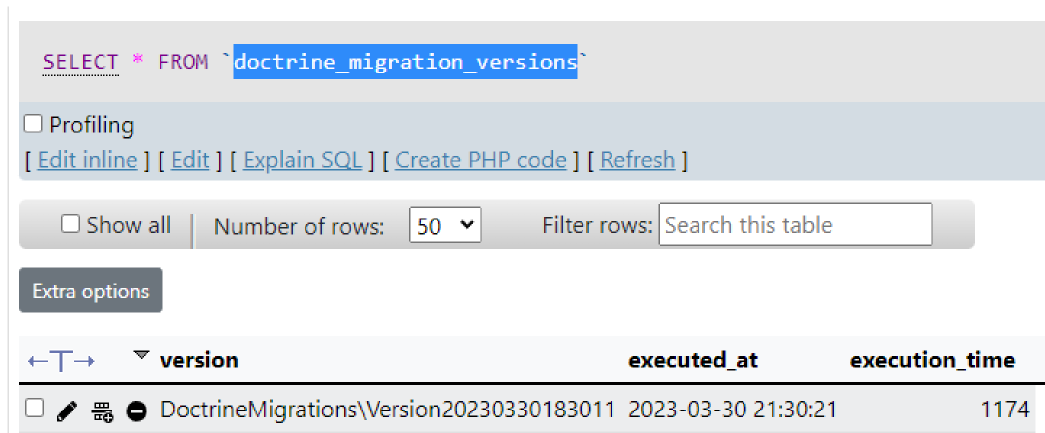

Рисунок 3.9 – Таблиця міграцій у базі даних

Такий підхід дозволяє містити базу даних консистентної у відповідності до моделей проекту. Також є можливість «відкочувати» міграції за будь-яких помилок у проектуванні бази даних, маючи своєрідні «зліпки», на кшталт системи контролю версій.

Також це дозволяє будь-якої миті створити базу даних на пристрої розробника, просто виконавши команду міграцій, і не перекидаючи базу даних як файл.

Також крім міграцій можна створювати seed файли для БД. Ці файли містять заготовки даних, таким чином можна зберегти свій час, просто розгорнувши вже підготовлений список користувачів, тварин, тощо.

3.1.6. Структура бази даних

База даних є серцем будь-якого проекту, сховищем даних. До проектування БД слід віднестися з особливою увагою, оскільки при неправильному

проектуванні, можливі майбутні помилки в бізнес-логіці та проблеми з продуктивністю, складності в розробці.

БД можна спроектувати за допомогою величезної кількості програм, і навіть замальовкою на аркуші. Під час проектування різних таблиць, я вирішив використовувати маркерну дошку, розписуючи сутності і що мені потрібно, щоб вони складалися з.ї

Рисунок 3.10 – Структура БД на маркерній дошці

База даних складається із 6 таблиць, дна з яких є службовою для міграцій. При проектуванні бази на дошці були не до кінця продумані зв'язки, іноді під час створення таблиці додавалися додаткові поля, які не були враховані спочатку, фінальна версія БД виглядає наступним чином:

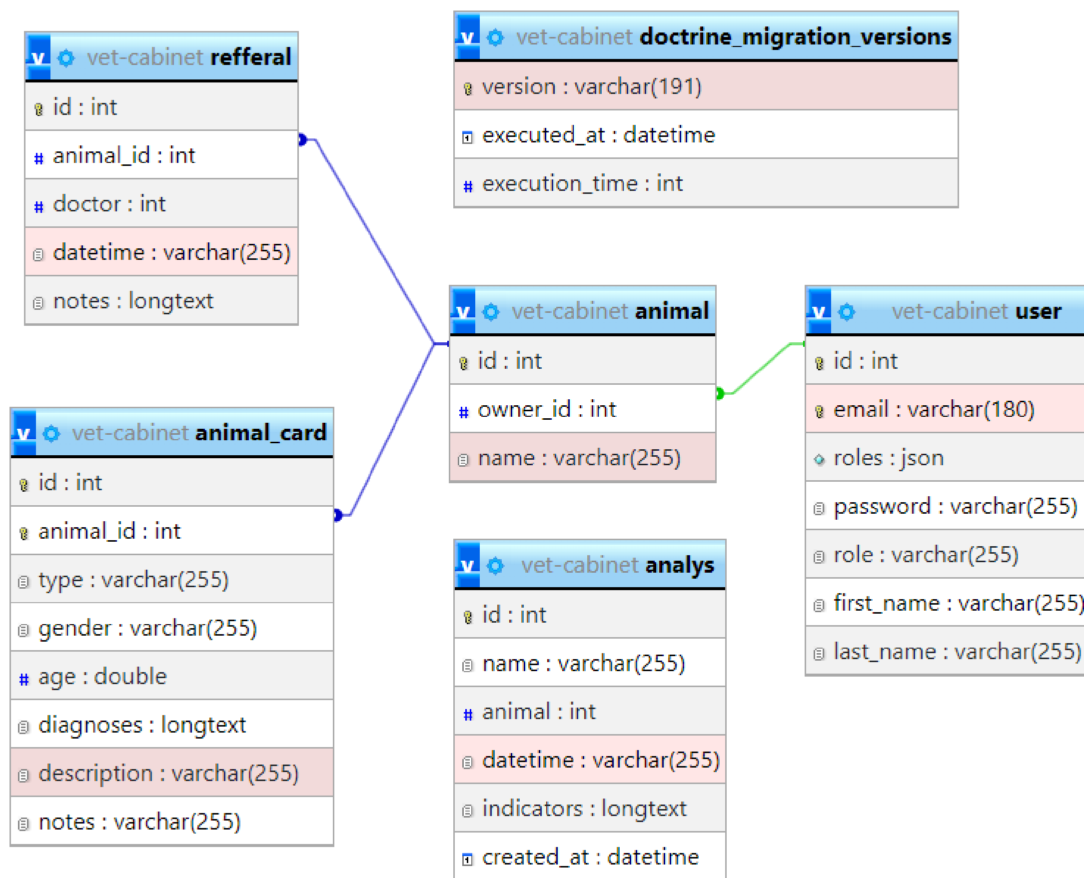

Рисунок 3.11 – Фінальна структура бази даних

3.2 Клієнтська програма

3.2.1. Створення програми та структура

Angular пропонує свою CLI для створення нових проектів, і це єдиний спосіб, яким можна створити проект останньої версії.

Такої суворої структури файлів як у Symfony у Angular немає. Кожен розробник може створювати зручну для нього файлову структуру. З одного боку це плюс, тому що дає повну свободу, але з іншого мінус, тому що відсутня стандарт і найчастіше структура не масштабується, а іноді і в корені невірною.

Я у свою чергу за кілька років розробки на цій технології прийшов до наступної файлової структури:

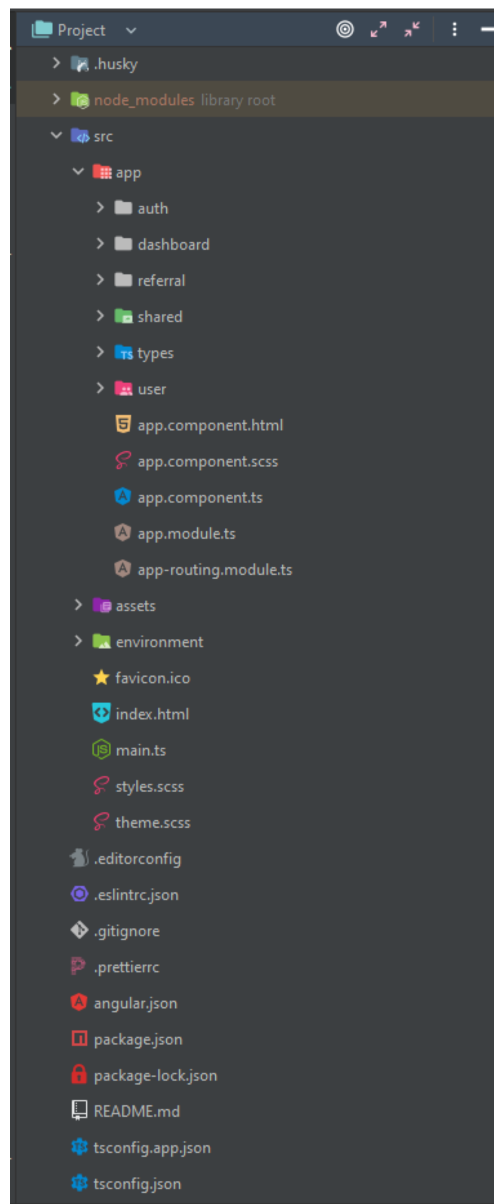

Рисунок 3.12 – Файлова структура програми клієнта

Поточна файлова система дозволяє домогтися простого lazy loading, збити програму на компоненти, а також у разі потреби підміняти їх на інші за допомогою Dependency Injection. Розглянемо кожен пункт файлової структури:

- /.husky – Папка одноименного пакета husky, устанавливается отдельно. Позволяет перед каждым коммитом проводить проверку проекта с помощью ESLint, и не позволяет записать нерабочий или неправильный код в систему контроля версий.
- /src/app – Основная папка проекта где находится главный модуль всего приложения

- /src/app/auth, /src/app/dashboard, /src/app/referral – Модулі програми, які будуються залежно від роутингу, містять у собі файл модуля і всі компоненти, які можуть бути в нього інкапсульовані. Кожен окремий шлях програми, допустимо якщо додатися сторінка landing, повинна лежати в окремому модулі, таким чином її можна вантажити окремо, реалізувавши lazy loading.
- /src/app/shared – Окремий модуль з усіма загальними компонентами, сервісами, директивами і так далі.

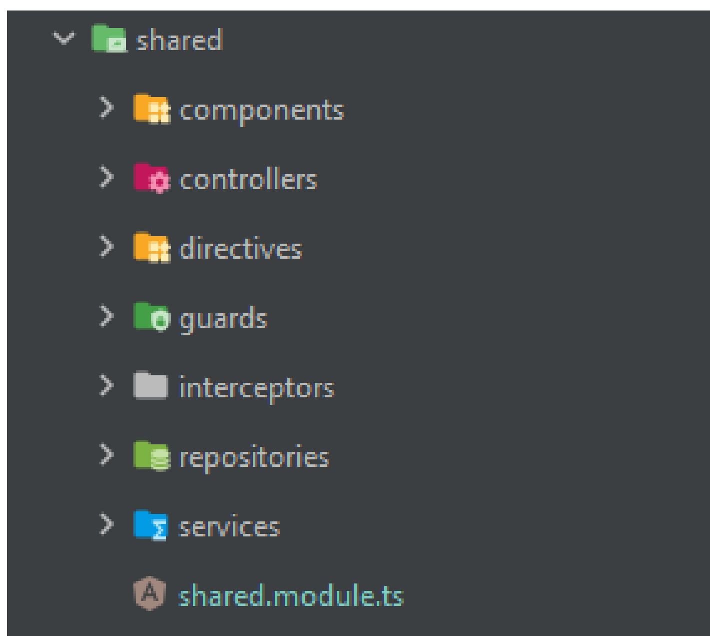

Рисунок 3.13 – Файлова структура папки shared

Всі файли, які використовуються крізь усі програми повинні бути додані саме в цей модуль, тому що якщо додати їх в один модуль, а використовувати в іншому – порушиться система імпортів і модулі будуть взаємопов'язані, що спричинить завантаження зайвих даних

- /src/app/types – Містить у собі інтерфейси та enums, так як typescript є строго типізованим мовою, всі змінні та дані мають бути описані. Ця папка містить усі спільні інтерфейси. Також можна створювати інтерфейси безпосередньо в контролерах і сервісах, але це порожній тон, тому що збільшує

кількість коду, і при необхідності використовувати цей інтерфейс в іншому місці все одно тягне за собою його переміщення в спільну папку.

- /node_modules – Файл з усіма бібліотеками та файлами фремворку. Ні в якому разі не повинен потрапляти в систему контролю версій, тому що займає величезне кількість місць і може досягати пари гігабайт. Усі необхідні пакети повинні встановлюватись з package.json файла.
- src/assets – Папка з усіма ассетами для програми. Папка містить різні файли стилів, шрифтів, зображення і може містити налаштування тем.
- src/environment – Папка з оточенням файлу. Працює ідентично до файлів оточення в Symfony. Також можна створювати файли для різних версій програми.
- /index.html – Вхідний файл програми містить базову розмітку сторінки і блок, в який рендерується програма.
- /main.ts – Вхідний файл для самого фремворку, саме цей файл запускає AppModule із папки src/app.
- /angular.json – Файл налаштування фреймворку. Дозволяє налаштувати точку входу, додати файли стилів, які завантажуватимуться без прямих імпортів, перевизначити кореневу папку та загалом налаштувати проект під свої потреби.
- /package.json – Идентичный файл настроек и библиотек, отвечает точно за тоже самое что и composer.json в Symfony
- /README.md – Файл, який служить для записування всієї інформації про проект для передачі іншим розробникам. Найчастіше містить інструкцію з розгортання та налагодження проекту, особливостях і в цілому, як і чому проект працює в цілому. Хороший розробник завжди записує особливості проекту саме сюди, і цей файл – перше місце куди варто дивитися при виникненні якихось проблем.

3.2.2. Роутинг и Lazy Loading

Angular, як і всі frontend фреймворки є Single Page Application, тобто вся програма – це по суті одна сторінка.

Але так як на одній сторінці неможливо розмістити всю інформацію, необхідно приховувати певні частини сайту, і в цілому це буде незручно - було створено Angular-router, бібліотека, яка йде в комплекті з самим фреймворком і служить для побудови повноцінної навігації.

Весь роутинг є декларативним і будується всередині компонента <router-outlet></router-outlet>.

Роутінг починається з файлу app-routing.module.ts, і в поточному додатку виглядає так:

Лістинг 3.7 – Файл app-routing.module.ts

```
const routes: Routes = [
    {
        path: 'auth',
        loadChildren: () => import('./auth/auth.module').then(m => m.AuthModule),
        canActivate: [NonAuthGuard]
    },
    {
        path: "dashboard/visits",
        loadChildren: () => import("./referral/referral.module").then(m => m.ReferralModule),
        canActivate: [AuthGuard]
    },
    {
        path: "dashboard",
        loadChildren: () => import("./dashboard/dashboard.module").then(m => m.DashboardModule),
        canActivate: [AuthGuard]
    },
]


@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule]
})
export class AppRoutingModule {
}
```

Всі шляхи додатку повинні успадковувати інтерфейс Routes. У прикладі вище показаний головний навігаційний файл, який реєструє модулі в системі роутингу.

Лістинг 3.8 – Lazy Loading

```
loadChildren: () => import("./referral/referral.module").then(m => m.ReferralModule)
```

Ділянка коду з loadChildren бере в себе модуль, в якому повинен утримуватися вже його власний роутинг, імена на цьому принципі ґрунтується lazy loading, і якщо нам припустимо не потрібен шлях dashboard, то ці файли взагалі не завантажуватимуться. Під час складання проекту вже сам компілятор зрозуміє, на які файли необхідно розбити проект. Сам проект після складання виглядає так:

Lazy Chunk Files	Names	Raw Size	Estimated Transfer Size
321.f0eecef1e527b8c2.js	dashboard-dashboard-module	79.14 kB	14.02 kB
883.5a21c38b05ae645d.js	auth-auth-module	20.79 kB	4.17 kB
422.b45fd8822ed1fce5.js	referral-referral-module	8.34 kB	2.63 kB

Рисунок 3.14 – Project bundles

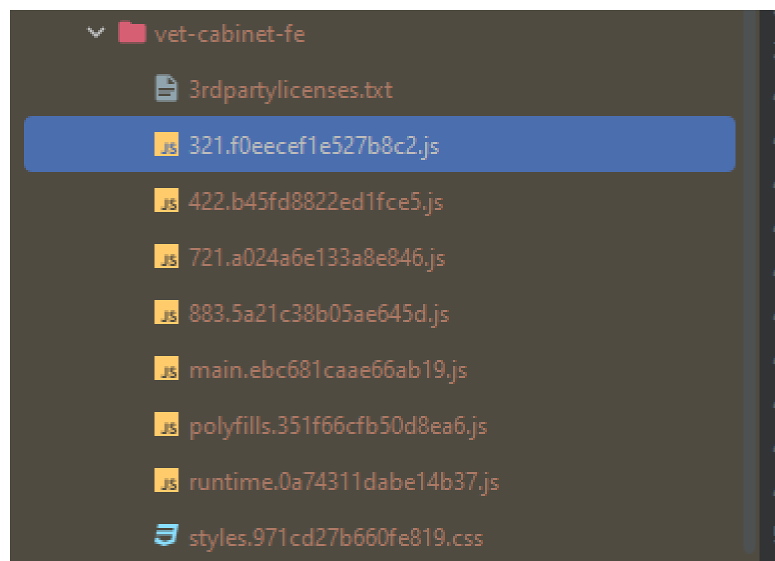

Рисунок 3.15 – Скомпилированный проект

З виведення в консолі і структури файлів видно, що кожен модуль зібрався в окремий файл, і коли як користувач я зайду на сторінку автентифікації, модуль dashboard і referral не буде завантажуватися, що збереже трафік і наш код від завантаження того, чим не можна користуватися.

Сами модули приложения и роутинг внутри них немного отличаются

Лістинг 3.9 – Модуль auth

```
const routes: Routes = [
    {
        path: "", component: AuthComponent, children: [
            {path: 'login', component: LoginComponent},
            {path: 'registration', component: RegisterComponent},
        ]
    }

];

@NgModule({
    declarations: [
        AuthComponent,
        LoginComponent,
        RegisterComponent
    ],
    imports: [
        CommonModule,
        RouterModule.forChild(routes),
        FormsModule,
        ReactiveFormsModule,
        MatFormFieldModule,
        MatInputModule,
        MatCardModule,
        MatButtonModule,
        MatProgressBarModule,
        MatRadioModule,
    ]
})
export class AuthModule {
}
```

А саме розрізняються пара місць:

- Component: LoginComponent – Усередині роутингу вже визначаються безпосередньо компоненти, а не модулі, що завантажує вже саму бізнес-логіку та файли шаблонів.Declarations – Усі компоненти, які будуть використовуватися всередині модуля та/або роутингу – повинні бути зареєстровані тут, інакше фреймворк не знайде потрібний компонент та видасть помилку.RouterModule.forChild(routes) – Відрізняється від головного роутингового файлу тим, що тепер викликається метод forChild() замість forRoot(). Всі модулі, крім головного, повинні реєструвати роутинг через метод forChild, використання методу forRoot більше одного разу призведе до помилки, оскільки це точка входу для всього роутингу.

Також через роутинг можна передавати різні параметри, наприклад:

- Лістинг 3.10 – Шлях із параметром

```
        {
      path: "edit/:id",
      component: AnimalEditComponent,
    },
```

Параметри через роутинг повинні бути визначені, починаючи з двокрапки, і після цього вони можуть бути отримані всередині компонента за допомогою ActivatedRoute модуля.

Лістинг 3.11 – Отримання параметра з id з URL рядка

```
this.route.snapshot.queryParamMap.get("id")
```

Таким образом передаются разные данные, по типу номера животного, необходимой секции формы обратной связи и так далее.

3.2.3. Захист роутингу та охоронці

У коді з минулого розділу можна помітити ділянку коду, яка відповідає за захист шляху.

Лістинг 3.12 – Захист шляху

```
canActivate: [AuthGuard]
```

Angular дозволяє "з коробки" захищати шляхи сайту за допомогою захисників. Розглянемо приклад із нашого проекту:

Лістинг 3.13 – Auth Guard

```
import {Injectable} from "@angular/core";
import {CanActivate, CanActivateChild, Router, UrlTree} from "@angular/router";
import {UserService} from "../services/user.service";
import {catchError, EMPTY, map, Observable} from "rxjs";

@Injectable()
export class AuthGuard implements CanActivate, CanActivateChild {
    constructor(private userService: UserService,
                public router: Router) {
    }
```

```
    canActivate(): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean | UrlTree
{
        if (this.userService.authorized) {
            return true;
        }
        return this.userService.fetchUser()
            .pipe(
                catchError(this.onError.bind(this)),
                map(() => true)
            );
    }

    canActivateChild(): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> | boolean |
UrlTree {
        if (!this.userService.authorized) {
            void this.router.navigate(['auth/login']);
            return false
        }
        return true;
    }

    private onError() {
        void this.router.navigate(['auth/login']);
        return EMPTY;
    }
}
```

Загалом логіка захисників дуже проста. Вони повинні обмежувати доступ за якоюсь умовою і робити перенаправлення в собі. Код вище перевіряє, чи існує сесія користувача, і якщо відсутня – перенаправляє на сторінку входу, повністю блокуючи використання заборонених частин сайту. Так само можна обмежити доступ, наприклад за ролі користувача, тощо. Найбільше в цьому цікавить успадкування від інтерфейсів:

- canActivate – Дозволяє чи забороняє використання певного шляху.
- canActivateChild – Дозволяє або забороняє використання всіх дочірніх шляхів, але не забороняє використання шляху, в якому вказується.

3.2.4. Структура компонента

При створенні компонента через командний рядок, створюється папка з файлами компонента і при цьому компонент реєструється в найближчому знайденому модулі або в модулі вище файлової системи.

Створюється чотири файли, один з яких - файл тестів, який можна пропустити при створенні компонента. Структура виглядає так:

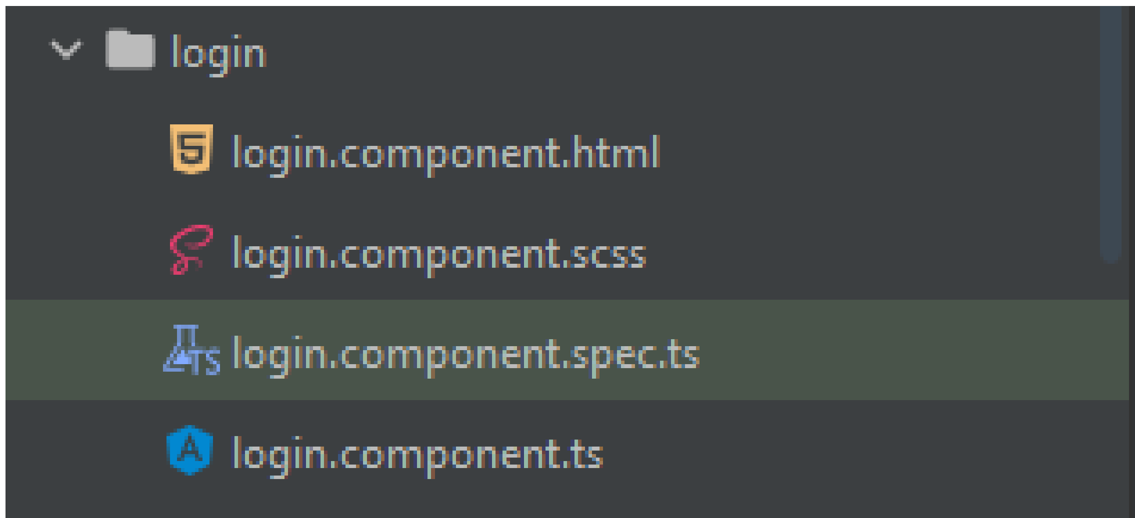

Рисунок 3.15 – Базовая структура компонента

- login.component.html – файл шаблону містить всю візуальну частину.
- login.component.scss – файл стилів містить всі css стилі для поточного компонента. Файл повністю інкапсульований і стилі, що написані в ньому, не впливатимуть на інші компоненти, навіть якщо в них збігаються імена класів.
- login.component.spec.ts – Файл тестів.
- login.component.ts – файл контролера для шаблону містить всю логіку компонента. В ідеалі повинен містити мінімальний набір логіки для роботи лише з візуалом та формами, а вся бізнес логіка має знаходитись у сервісах.

Файл шаблону виглядає як звичайний HTML, за винятком того, що часто складається з різних компонентів і не стандартних тегів.

```
<mat-card class="auth-card">
  <mat-card-content>
    <h2>Вход в Аккаунт</h2>
    <form [formGroup]="form" (ngSubmit)="onFormSubmit()">
      <mat-form-field appearance="outline">
        <mat-label>Почта</mat-label>
        <input matInput type="email" placeholder="Почта" formControlName="email">
        <mat-error *ngIf="isControlHasError(form, controlName: 'email', validationType: 'required')">
          Обязательное поле
        </mat-error>
        <mat-error *ngIf="isControlHasError(form, controlName: 'email', validationType: 'email')">
          Неправильная почта
        </mat-error>
      </mat-form-field>
      <mat-form-field appearance="outline">
        <mat-label>Пароль</mat-label>
        <input matInput type="password" placeholder="Пароль" formControlName="password">
        <mat-error *ngIf="isControlHasError(form, controlName: 'password', validationType: 'required')">
          Обязательное поле
        </mat-error>
      </mat-form-field>

      <button [disabled]="isLoading" mat-raised-button type="submit" color="primary">Вход</button>
    </form>
  </mat-card-content>

  <mat-card-footer *ngIf="isLoading">
    <mat-progress-bar mode="indeterminate"></mat-progress-bar>
  </mat-card-footer>
</mat-card>
```

Рисунок 3.16 – Файл шаблона

Контролер є звичайним typescript класом. Але він сам обернутий в декоратор @Component, який додає всю шаблонну функцію для розуміння фреймворку, а саме:

- selector – Назва компонента, яким він буде викликатися в шаблонах.
- templateUrl – Шлях до шаблону, який відповідає контролер.
- stylesUrls – Шлях до стилів компонента.
- changeDetection – Стратегія виявлення змін.

За останнім пунктом, якщо розглянути в кратці, то є дві стратегії, Default і OnPush. OnPush Більш продуктивна, не відстежує мутацію даних і всі дані повинні бути змінені повністю, що є правильним підходом продуктивності.

```
9   @Component({
10      selector: 'app-login',
11      templateUrl: './login.component.html',
12      styleUrls: ['./login.component.scss'],
13      changeDetection: ChangeDetectionStrategy.OnPush
14  })
15  export class LoginComponent extends AbstractController {
16      public form: FormGroup;
17      public isLoading = false;
18
19      constructor(private fb: FormBuilder,
20                  private authService: AuthService,
21                  private userService: UserService,
22                  private cdr: ChangeDetectorRef) {
23          super();
24          this.form = this.initForm();
25      }
26
27
28      public onFormSubmit() {
29          if (this.checkFormIsInvalid(this.form)) {
30              return;
31          }
32
33          const credentials: ILoginPayload = this.form.value;
34
35          this.isLoading = true;
36          this.authService.login(credentials)
37              .pipe(finalize(this.onComplete.bind(this)))
38              .subscribe(this.onSuccess.bind(this));
39      }
```

Рисунок 3.17 – Файл контролера

3.2.5. Директиви та ролі

У додатку є дві ролі для користувачів - адміністратор (або лікар) і звичайний користувач-клієнт. Це зумовлено тим, що користувач повинен мати обмеження на доступ до створення і видалення певних записів, наприклад створення візитів або аналізів, оскільки їх повинен створювати безпосередньо лікар.

У додатку було вирішено використовувати самі шаблони для обох ролей, але обмежувати певні блоки, залежно від ролі.

У Angular для цього є директиви. Усього є два види директив:

- Структурні – змінюють структуру DOM дерева.
- Атрибути - змінюють зовнішній вигляд або стандартну поведінку DOM-дерева.

Перший вид директиви повністю підходить для наших потреб. У Angular вже є дві структурні директиви, ngIf і ngFor.

Лістинг 3.14 – Частина компонента з директивами ngFor та ngIf

```
<mat-form-field appearance="outline">
```

```
        <mat-label>Animal</mat-label>
        <input type="text"
               placeholder="Animal"
               aria-label="Animal"
               matInput
               formControlName="animal"
               [matAutocomplete]="autoAnimal">
        <mat-error *ngIf="isControlHasError(form, 'animal', 'required')">
            Обязательное поле
        </mat-error>
        <mat-autocomplete #autoAnimal="matAutocomplete" [displayWith]="displayFnAnimal">
            <mat-option *ngFor="let animal of animals" [value]="animal">
                {{displayFnAnimal(animal)}}
            </mat-option>
        </mat-autocomplete>
         </mat-form-field>
```

- ngFor – Дозволяє проходити за списком, повністю ідентичний за логікою з foreach або for циклами.
- ngIf – Повністю ідентичний з оператором if, приховує весь вміст блоку разом з ним, якщо умова є false.

ngIf повністю підходить для створення ролей, але існує низка певних недоліків:

- Необхідно в кожний компонент включати сервіс, що зберігає логіку користувача.
- Сервіси не бажано оголошувати публічними і не бажано у шаблоні використовувати їх методи, таким чином кожен компонент повинен буде містити метод перевірки ролі, що суперечить принципу DRY (don't repeate yourself) і дуже збільшує кількість марного коду.

Єдиним вірним рішення для цієї проблеми є написання власної директиви, яка буде інкапсулювати і перевіряти доступ користувача, обмежуючи або дозволяючи доступ, код директиви наведений нижче:

Лістинг 3.15 – Директива role

```
<div *role="'admin'">content here</div>

@Directive({
  selector: '[role]'
})
export class RoleDirective implements OnInit, OnDestroy{
    private subscription: Empty<Subscription>;

    @Input() role?: string = UserRole.Admin;
```

```
    constructor(
        private templateRef: TemplateRef<unknown>,
        private userService: UserService,
        private viewContainer: ViewContainerRef){
    }

    ngOnDestroy(): void {
        this.subscription?.unsubscribe()
    }

    ngOnInit(): void {
        this.subscribeForUser();
    }

    private subscribeForUser() {
        this.subscription = this.userService.$getUser().subscribe((user) => {
            if (user?.role !== this.role) {
                this.viewContainer.clear();
            } else {
                this.viewContainer.createEmbeddedView(this.templateRef);
            }
        })
    }
}
```

Логіка директиви дуже проста. У методі ngOnInit відбувається підписка на сервіс з логікою користувача та перевіряється, чи ідентична роль користувача з роллю, яка передається в директиву за допомогою @Input() role. При знищенні шаблону та знищенні директиви викликається медом ngOnDestroy, який у свою чергу відписується від користувальницького сервісу, таким чином запобігаючи витоку пам'яті та ненавмисній логіці програми.

Метод this.viewContainer.clear() повністю відчищає весь компонент, на який накладена директива, таким чином не просто ховаючи компоненти, як це можна зробити за допомогою css, а повністю знищує всі можливі компоненти та розмітку всередині себе. Метод this.viewContainer.createEmbeddedView(this.templateRef) навпаки показує всі начинки компонента, на який накладена директива, повністю перерендерюючи себе.

3.2.6. Сервіси та бізнес логіка

Вся бізнес логіка та запити зберігаються в сервісах. Це єдино правильний підхід, що дозволяє відокремити UI та бізнес логіку. Всі сервіси знаходяться в

папці shared, тому що можуть бути використані в будь-якому місці програми, не тільки в компонентах, директивах, а й інших сервісах. Але включати один сервіс в інший слід з особливою обережністю, адже якщо наприклад в UserService включити AdminService, а в AdminService включити UserService - вийде циклічне посилання, яке призведе до попередження, яке іноді досить складно вирішити і іноді призводить до створення третього сервісу, який делегує логіку між першими двома.

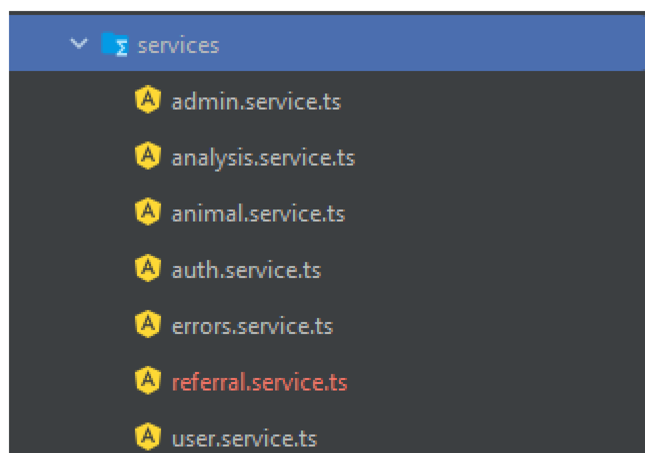

Рисунок 3.18 – Всі послуги програми клієнта

Сервіси в Angular є звичайними класами, за винятком одного моменту всі сервіси повинні маркуватися декоратором @Injectable.

Лістинг 3.16 – Декоратор @Injectable

```
@Injectable({
    providedIn: 'root'
})
export class UserService {}
```

Тим самим ми говоримо фреймфорку, що цей клас може бути включений в будь-яке місце програми і повинен бути зареєстрований.

Після цього сервіси вже можуть бути включені в будь-який клас за допомогою Dependency Injection, який Angular надає «з коробки».

```
constructor(private fb: FormBuilder,
            private authService: AuthService,
            private userService: UserService,
            private cdr: ChangeDetectorRef) {
    super();
    this.form = this.initForm();
}
```

Рисунок 3.19 – Приклад увімкнення сервісу за допомогою DI

Сервіси представляють з меб дата класи, які містять всі дині додатки та запити. Взагалі Angular є реактивним та виконує запит лише якщо на нього є спостерігач. Таким чином, на запит можна підписатися тільки в компоненті, бо надалі можна буде відписатися від нього і запобігти витоку пам'яті. Наприклад клас з логікою для тварин виглядає так:

Лістинг 3.17 – Animal Service

```
@Injectable({
    providedIn: 'root',
})
export class AnimalService {
    constructor(@Inject(AnimalRepositoryAbstract) private animalRepository:
AnimalRepositoryAbstract) {
    }
    private _animals: BehaviorSubject<IAnimal[]> = new BehaviorSubject<IAnimal[]>([]);

    public get animals(): BehaviorSubject<IAnimal[]> {
        return this._animals;
    }

    public fetchAnimals(): Observable<IAnimalsResponse> {
        return this.animalRepository.fetchAnimalsList();
    }

    public createAnimal(payload: IAnimalCreatePayload): Observable<IAnimalResponse> {
        return this.animalRepository.createAnimal(payload);
    }

    public setAnimals(animals: IAnimal[] | IAnimal): void {
        if (Array.isArray(animals)) {
            this._animals.next(animals);
            return;
        }

        const filtered = this.animals.value.filter(e => e.id !== animals.id);
        this._animals.next([...filtered, animals]);
    }

    public fetchAnimalById(animalId: number): Observable<IAnimalResponse> {
        return this.animalRepository.fetchAnimalById(animalId)
            .pipe(tap((response) => this.setAnimals(response.animal)))
    }
```

```
        public getAnimalById(animalId: Empty<number>): Empty<IAnimal> {
            return this._animals.value.find((e) => e.id === animalId);
        }

        public updateAnimalById(animalId: number, payload: IAnimalUpdateEntity):
Observable<IAnimalResponse> {
            return this.animalRepository.updateAnimal(animalId, payload)
                .pipe(tap((response) => this.setAnimals(response.animal)))
        }
    }
```

Розглянемо деякі частини класу:

Лістинг 3.18 – Оголошення спостерігається змінної

```
private _animals: BehaviorSubject<IAnimal[]> = new BehaviorSubject<IAnimal[]>([]);
```

Цей шматок коду оголошує змінну _animals, яка спостерігається. На неї можна підписатися, і як тільки вміст оновиться через метод next() – всі підписані на неї слухачі одразу отримають нові дані та частини програми перемальовуються.

Лістинг 3.19 – Оголошення спостерігається змінної

```
public fetchAnimalById(animalId: number): Observable<IAnimalResponse> {
        return this.animalRepository.fetchAnimalById(animalId)
            .pipe(tap((response) => this.setAnimals(response.animal)))
    }
```

У цій ділянці коду відбувається запит за допомогою репозиторію запитів.

Але найбільше цікавить метод pipe та оператор tap. RxJS бібліотека, на якій побудований Angular надає величезну кількість операторів для роботи з потоками та tap один з них. При успішному запиті він дублює дані з підписки і дозволяє його використання всередині сервісу, таким чином сервіс сам оновлює свої дані.

Усього існує два підходи для пробивання даних крізь усі програми – shared services та NgRX. Другий спосіб є бібліотекою для реалізації дата класів за підходом Redux. Я вирішив використовувати перший підхід, тому що він простіше та ідеально підходить для дрібних та середніх додатків.

Усі сервіси у додатку зареєстровані як singleton. Таким чином, наприклад, всі userservice в будь-якій частині програми є одним і тим же екземпляром класу.

3.2.7. Репозиторії та DI

Для спілкування із сервером було обрано підхід із репозиторіями. Основний підхід полягає в тому, щоб створити додатковий шар логіки між сервісами та API з певним інтерфейсом. Таким чином вирішується проблема якщо в майбутньому, наприклад, потрібно буде перенести сервер з PHP на приклад Firebase.

Для цього створюється абстрактний клас репозиторію, який виглядає приблизно так:

```
export abstract class AnimalRepositoryAbstract {
    public abstract fetchAnimalsList(): Observable<IAnimalsResponse>;

    public abstract createAnimal(payload: IAnimalCreatePayload): Observable<IAnimalResponse>;

    public abstract fetchAnimalById(animalId: number): Observable<IAnimalResponse>;

    public abstract updateAnimal(animalId: number, data: IAnimalUpdateEntity): Observable<IAnimalResponse>;
}
```

Рисунок 3.20 – Абстрактний клас репозиторію для запитів по тваринам

Після цього вже засобами DI в модулі необхідно зареєструвати сервіс по інтерфейсу класу і класу, що найвикористовується:

```
providers: [
    {
        provide: AnimalRepositoryAbstract,
        useClass: AnimalRepository
    },
    {
        provide: AnalysisRepositoryAbstract,
        useClass: AnalysisRepository
    },
    {
        provide: ReferralRepositoryAbstract,
        useClass: ReferralRepository
    },
],
```

Рисунок 3.21 – Реєстрація репозиторію в DI

Тим самим вказавши, який саме клас повинен використовуватися при вбудовуванні репозиторію в сервіс, компонент і будь-яке місце в проекті:

```
port class AnimalService {
  constructor(@Inject(AnimalRepositoryAbstract) private animalRepository: AnimalRepositoryAbstract) {
  }
```

Рисунок 3.22 – Вставка репозиторію у сервіс по токену

Цей підхід дозволяє зберігати всі імпорти в одній частині програми (модулі), і при необхідності просто підмінити класи, що виконуються, по тому ж самому інтерфейсу. У результаті це дозволить перемістити логіку на інший сервер або підхід, взагалі не торкнувшись бізнес логіки приождения і не зламавши його.

3.3 Дизайн та UI

Дизайн програми був заснований на підході Material Design. За основу було взято бібліотеку компонентів Angular Material.

Angular Material — це бібліотека компонентів інтерфейсу користувача (UI), яку розробники можуть використовувати у своїх проектах Angular для прискорення розробки елегантних і узгоджених інтерфейсів користувача. Angular Material пропонує багаторазові та красиві компоненти інтерфейсу користувача, такі як картки, вхідні дані, таблиці даних, інструменти вибору дати та багато іншого.

Кожен компонент готовий до роботи зі стилем за замовчуванням, який відповідає специфікації Material Design. Тим не менш, ви можете легко налаштувати зовнішній вигляд компонентів Angular Material. Список доступних компонентів Angular Material продовжує зростати з кожною ітерацією бібліотеки.

Усі модальні вікна сторінки зроблені у стандартному виконанні – невелике вікно посередині сторінки, яке має розмір 600 або менше пікселів. Наприклад форма входу виглядає так:

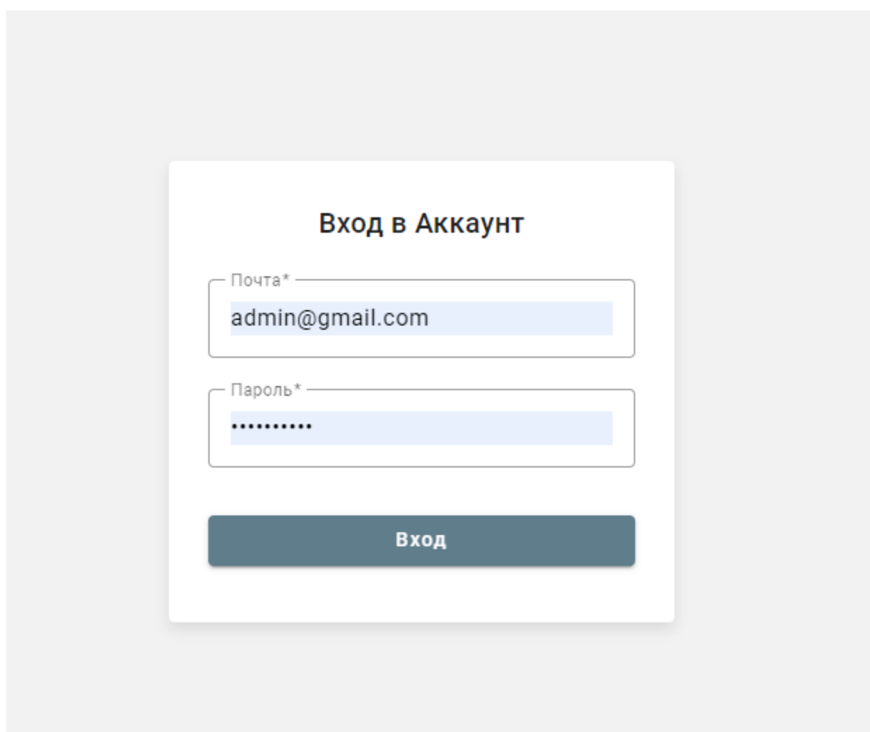

Рисунок 3.23 – Форма входу

Меню навігації було зроблено за допомогою компонента mat-drawer, що дозволило зробити його зручним, одразу з набором всіх мета тегів та анімованим. Меню містить два стани: залогіненого користувача та стан для входу:

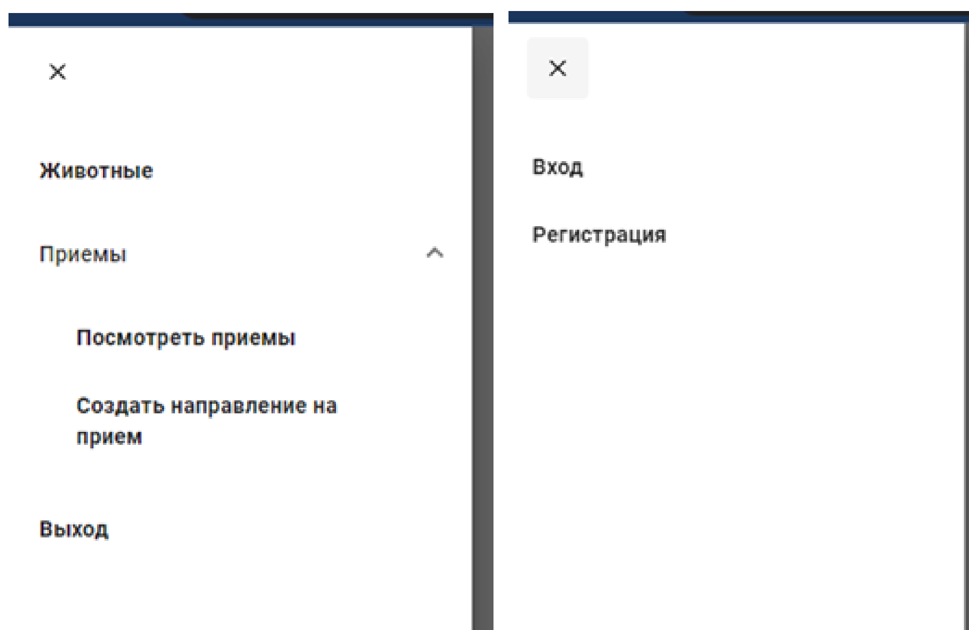

Рисунок 3.24 – Меню для зареєстрованого користувача та меню входу

Я спробував виведення тварин списком, але підхід не сподобався, і було вирішено реалізувати список тварин картками, використовуючи компонент mat-card.

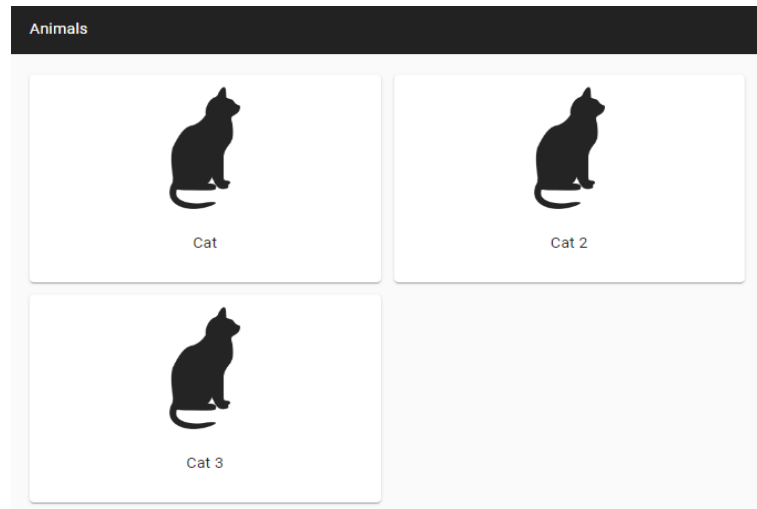

Рисунок 3.25 – Картки тварин, компонент mat-card

Також окрему увагу варто звернути на сторінку аналізів, яка не тільки показує аналіз, але також дозволяє за натисканням на іконку друку викликати стандартний для браузера режим window.print(), який дозволить відправити сторінку на принтер, або зберегти її в pdf форматі.

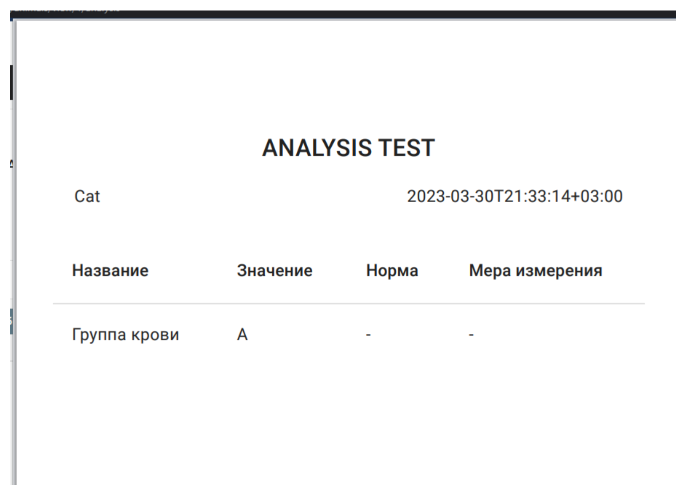

Рисунок 3.26 – Приклад друкованого аналізу

Але проект також складається з величезної кількості форм для введення даних. Всі поля повинні повністю валідуватися і не два користувачеві створити неправильний набір даних перед відправкою на сервер. Для цього Angular існують реактивні форми, які дозволяють додавати валідацію.

Лістинг 3.20 – Оголошення форми

```
protected initForm(): FormGroup {
    return this.fb.group({
        name: ["", [Validators.required]],
        animal: [undefined, Validators.required],
        indicators: this.fb.array([])
    });
}
```

Всі форми у додатку створені через formBuilder і є об'єктом з інтерфейсом FormGroup. Також до кожного елементу форми можна додати валідатори, як у прикладі вище доданий валідатор Validators.required, який вимагає від поля, щоб воно було обов'язковим. У прикладі нижче показані помилки, які виводяться користувачеві, якщо він натисне кнопку «створити» не заповнивши або заповнивши поля невірними даними:

Рисунок 3.27 – Валідація форми

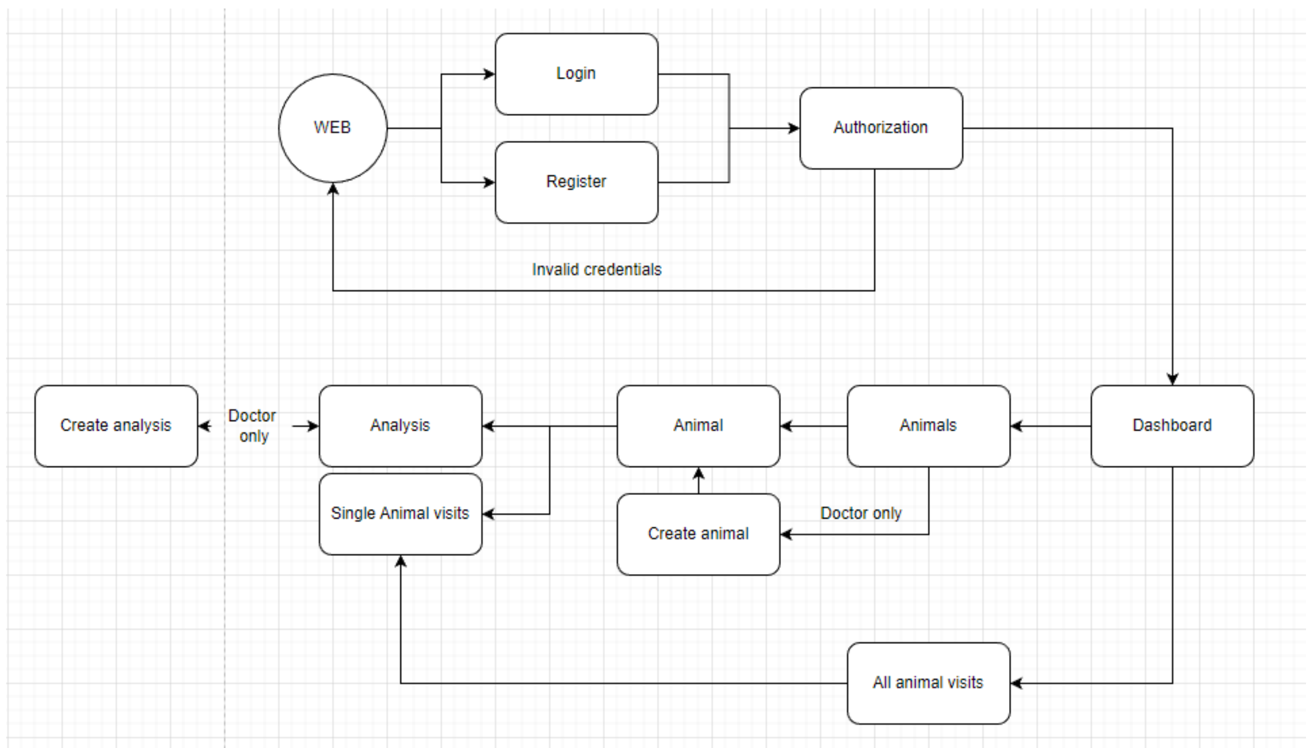

Рисунок 3.28 – Можливі шляхи використання програми користувачем

ВИСНОВКИ

Під час розробки двох додатків було покращено навички планування розробки за допомогою різних засобів.

Було покращено навички розробки мовою PHP, а також поглиблено знання у двох фреймворках – Angular та Symfony.

Завдяки проведеному аналізу аналогів тепер можливо використовувати цей підхід при подальшій розробці під час планування функціоналу додатків, оцінці рентабельності та подання зразкової реалізації компонентів ще до моменту їх реалізації.

Для розробки програм були вибрані такі стеки:

- Backend: PHP 8.0, OpenServer, Symfony, IDE Visual Studio Code.
- Frontend: Typescript 4.8.2, Angular 15.0.0, IDE Webstorm.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Метод аналізу ієрархій — Вікіпедія [Електронний ресурс] / Режим доступу www. URL: https://uk.wikipedia.org/wiki/Метод_аналізу_ієрархій
2. Метод анализа иерархий (метод Т. Л. Саати) [Електронний ресурс] / Режим доступу www. URL: https://edu.tltsu.ru/sites/sites_content/site216/html/media67140/lec1_is-2_2020%20(1).pdf
3. Алгоритм метода анализа иерархий [Електронний ресурс] / Режим доступу www. URL: https://studfile.net/preview/3348062/page:3/
4. Jet Vet - application and program for veterinarians and veterinarians [Електронний ресурс] / Режим доступу www. URL:https://jet.vet/en/
5. Топ 10: Программы для ветеринарных клиник [Електронний ресурс] / Режим доступу www. URL: https://www.livemedical.ru/tools/vet/
6. ENOTE® - программа для ветеринарных клиник и аптек [Електронний ресурс] / Режим доступу www. URL: https://enote.cloud/
7. Best 30 Mobile Apps for Veterinarians [Електронний ресурс] / Режим доступу www. URL: https://vetintegrations.com/insights/best-mobile-apps-for-veterinarians/
8. Software Engineering Requirements Analysis [Стаття] Режим доступу www. URL: https://www.javatpoint.com/software-engineering-requirement-analysis
9. Анализ требований / Хабр [Стаття] Режим доступу www. URL: https://habr.com/ru/post/340956/
10. Разработка требований к программному обеспечению. Стереотипное 3-е издание. Карл Вигерс Джой Битти. [Текст] навчальний посібник / Карл Вигерс, Джой Битти , 2014 - 736 с
11. Украина оказалась второй в мире по количеству котов на человека — Платформа — "Новини" [Електронний ресурс] / Режим доступу www. URL: https://platfor.ma/magazine/text-sq/news/kotoemkost-strany/

12. Angular Material UI component library [Електронний ресурс] / Режим доступу www. URL: https://material.angular.io/
13. The web development framework for building the future [Електронний ресурс] / Режим доступу www. URL: https://angular.io/
14. 9 User Flow Examples [Електронний ресурс] / Режим доступу www. URL: https://alvarotrigo.com/blog/user-flow-examples/
15.