МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерної алгебри та дискретної математики

(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

на здобуття ступеня вищої освіти «бакалавр»

(освітньо-кваліфікаційний рівень)

на тему

«Система моніторингу успішності учнів»

«Система мониторинга успеваемости учащихся»

«Monitoring system of student progress»

Виконав: студент денної форми навчання

напряму підготовки 123 – Комп'ютерна інженерія

(шифр і назва напряму підготовки, спеціальності)

Краснопольський Андрій Олегович

(прізвище, ім'я, по-батькові)

Керівник канд. фіз.-мат. наук, доцент Варбанець С.П.

(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент доц. Федоровський С.В.

(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№___від «___»_____2021 р.

Завідувач кафедри

_____ Варбанець П.Д.

(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___

протокол №___від «___»_____2021 р.

Оцінка / /

(за національною шкалою, шкалою ECTS, бали)

Голова ЕК_____

_____ _Н.Ф.Казакова_

(підпис) (прізвище, ініціали)

Одеса - 2021

АНОТАЦІЯ

У дипломній роботі розробляється тема «Система моніторингу успішності учнів».

Мета даної дипломної роботи - розробка системи моніторингу успішності учнів учбового закладу. Дана прикладна програма дозволяє студентам взаємодіяти з даною системою (перегляд власних оцінок, перегляд власної відвідуваності одного чи усіх дисциплін і тд.).

Додатково буде проведено аналіз різних функцій, наприклад відвідуваності кожного студента та підрахунок його пропусків та визначення отримає він «письмо щастя» чи ні.

АНОТАЦИЯ

В данной дипломной работе разрабатывается проект на тему "Система мониторинга успеваемости учащихся".

Цель данной дипломной работы - разработка системы мониторинга успешности учеников учебного заведения. Данная прикладная программа позволяет студентам взаимодействовать с данной системой (просмотр собственных оценок, просмотр собственной посещаемости по одной или всех дисциплинах и тд.).

Дополнительно будет проведен анализ разных функций, например посещаемости каждого студента и подсчет его пропусков и определения получит он "письмо счастья" или нет.

ANNOTATION

In this thesis, a project is being developed on the topic "Monitoring system of student progress".

The purpose of this thesis is to develop a system for monitoring the success of students of an educational institution. This application program allows students to interact with this system (viewing their own grades, viewing their own attendance in one or all disciplines, etc.).

Additionally, an analysis of various functions will be carried out, for example, the attendance of each student and the calculation of his absences and determining whether he will receive a "letter of happiness" or not.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

Скорочення

API – Application Programming Interface (інтерфейс створення додатків);

MVC – Model – View – Control (патерн проектування);

ООП – об’єктно - орієнтоване програмування;

БД – база даних;

UML - Unified Modeling Language (уніфікована мова моделювання);

ПрО – предметна область;

ЗУБД – засіб управління базою даних;

Фреймворк - це програмна платформа, яка визначає структуру програмної системи; програмне забезпечення, що полегшує розробку і об'єднання різних компонентів великого програмного проекту;

Патерн проектування — це типовий спосіб вирішення певної проблеми, що часто зустрічається при проектуванні архітектури програм;

ВСТУП

У час інформаційного прогресу кожен навчальний заклад потрібен мати свою електронно-інформаційну систему моніторингу успішності студентів. Ця система повинна спрощувати роботу для викладачів та для учнів навчальних закладів. Однак насьогодні в нашій країні дуже мало учбових закладів, які мають подібну систему.

Завдяки цій системі кожен студент зможе контролювати кількість своїх пропусків та свою успішність по одній чи декількох дисциплін. У кожного студента буде свій власний email та пароль для входу у свій власний електронний кабінет, у якому він зможе все це контролювати та перевіряти.[10] Можливості даної системи мають стимулювати та стабілізувати роботи учнів під час навчального процесу.

Для викладачів ця система буде добрим інструментом тому, що завдяки цій системі вони зможуть перевіряти відвідуваність окремих студентів чи цілих груп та курсів на своїх заняттях. У кожного викладача буде свій власний електронний кабінет. Електрона система призвана полегшати роботу викладачів, вчасності під час пандемії, шляхом зменшення роботи з паперами та поступового переведення всієї роботи у електрону систему.

Проблема, яку має вирішує електрона система – покращення учбового процесу, шляхом спрощення контролю своєї успішності для студентів та зменшення паперової роботи для викладачів.

Мета даної дипломної роботи - розробка системи моніторингу успішності учнів учбового закладу.

Для досягнення поставленої мети необхідно виконати наступні завдання:

- Аналіз предметної області
- Формування вимог та проектування функціональної моделі
- Проектування бази даних та обрання технологій реалізації програми
- Забезпечення сортування доступу до системи

ПОСТАНОВКА ЗАДАЧІ

Мета дипломної роботи – створення електронної системи моніторингу успішності студентів. Ця система повинна вміти робити збір статистики та видавати її у різних запитах. Наприклад, успішність студента по одній конкретній дисципліні.

Дана система буде мати три типи користувачів:

1. Деканат – має можливість переглядати усі дані студентів: їх відвідуваність по різним дисциплінам;
2. Викладач – має можливість відмічати та перевіряти відвідуваність студентів по своїм дисциплінам.
3. Студент – має можливість переглядати свої оцінки по різним дисциплінам не заходячи до деканату.

Таким чином, кожен користувач даної системи має свою роль, базуючись на цій ролі функціональність, яку йому надає продукт, в даному випадку – сайт та мобільний додаток.

1. ТЕОРИТИЧНА ЧАСТИНА

Android Studio — інтегроване середовище розробки (IDE) для платформи Android, представлене 16 травня 2013 року на конференції Google I/O менеджером по продукції корпорації Google — Еллі Паверс (англ. Ellie Powers). 8 грудня 2014 року компанія Google випустила перший стабільний реліз Android Studio 1.0.

Flutter — це програмний каркас із відкритим кодом, для створення додатків для платформ Android та iOS, а також на вебі, розроблений компанією Google. Він є основним способом створення додатків для Google Fuchsia. Весь графічний інтерфейс Google Fuchsia створено за допомогою Flutter.

Архітектура Flutter відрізняється від інших програмних каркасів (React, Apache Cordova) тим, що він не використовує для побудови інтерфейсу мови HTML, CSS та Javascript, відповідно і вбудований рушій WebView. Використовується власний рушій для рендерингу[3].

Flutter використовує тільки одну мову програмування Dart.

Серед плюсів Flutter можно виділити:

- Кросплатформеність
- Перспективність і активний розвиток
- Власний графічний движок

Dart — мова програмування, яку розробляє компанія Google, позиціонуючи як мову структурованого програмування для Веб. Розробники вважали, що в довгостроковій перспективі Dart може стати прогресивною заміною JavaScript, котрий потерпає від наявних в даний час проблем з розширюваністю, продуктивністю і підтримкою розробки складних застосунків. Мова має схожий на Java синтаксис, не вимагає явного визначення типів і її можна використовувати для створення серверних та клієнтських застосунків[9].

API – забезпечує взаємодію між двома системами. «Веб-сервіс» - це веб-додаток, що надає ресурси в форматі, використовуваному іншими комп'ютерами. Веб-сервіси – це, в основному, запити та відповіді між клієнтами і серверами (комп'ютер запитує ресурс, а веб-сервіс відповідає на запит). API, що використовують протокол HTTP для передачі запитів в відповідей, розглядаються як «веб-сервіси». Між клієнтом і сервером API існує запит і відповідь. Клієнт і сервер можуть бути засновані на будь-якій мові, але HTTP – це протокол, який використовується для передачі повідомлення. Цей шаблон запиту і відповіді, по суті, є принципом роботи моделі API REST (рис.1).

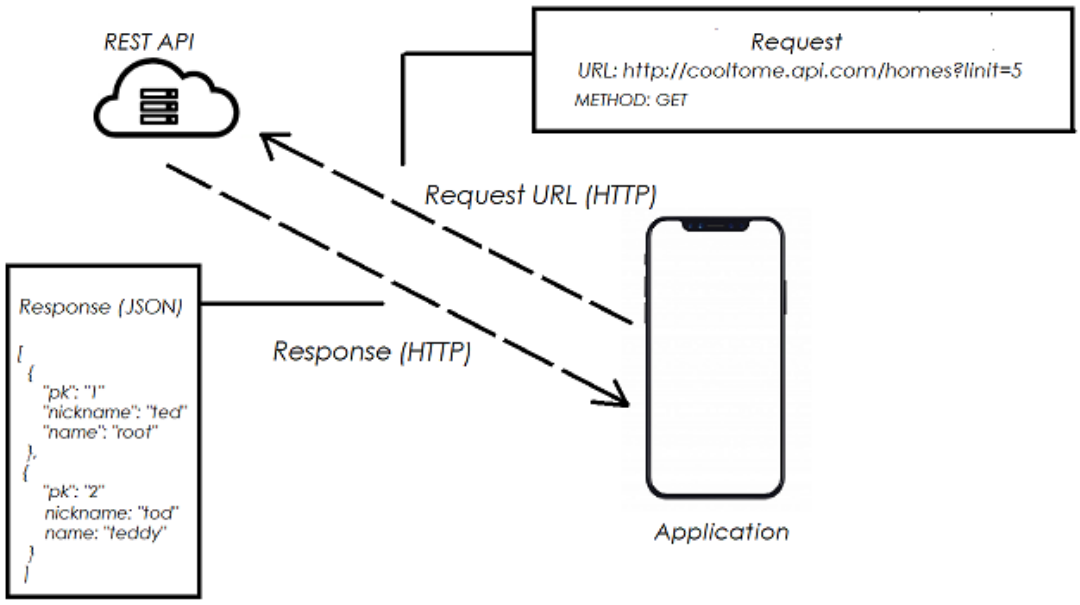

Рисунок 1 - Модель API REST

В якості ЗУБД був обраний PostgreSQL. PostgreSQL - об'єктно-реляційна ЗУБД, що дає їй переваги перед іншими SQL базами даних з відкритим вихідним кодом[2]. У цій ЗУБД існує можливість розширення функціоналу за рахунок використання збережених процедур, тригерів, уявлень, складових типів даних. Він має добре розроблену багату бібліотеку.

Valentina Studio, це інструмент для роботи з базами даних - один зі способів для перетворення даних в значущу інформацію, програма підтримує власну Valentina DB, а також MySQL, SQLite і Postgre.

В якості мови програмування був обраний PHP[4], в якості фреймворку обраний Laravel[8], який має ряд переваг:

- розвивається фреймворк має на увазі хорошу документацію, в порівнянні з іншими фреймворками або CMS, в більшості яких потрібно шукати додаткову інформацію;
- є можливість створювати масштабні проекти, незалежно від складності та напрямки, в тому числі і багаторівневі веб-сайти;
- надійна вбудований захист бази даних від SQL, CSRF, XSS;
- патерн MVC, який є зручним для розробки, а також при роботі навіть з великими базами даних.

2. АНАЛОГИ НА РИНКУ ІНФОРМАЦІЙНИХ СИСТЕМ

У процесі перегляду ринку інформаційних систем були знайдені декілька гарних аналогів нашій системі. У них були виявлені свої недоліки та свої переваги.

Кундолук – це електрона система, яка є ефективним та функціонально освітнім програмним рішенням, котре забезпечує повне охоплення електронних систем для учбовий закладів. Інтерактивна комунікаційна середа, у котрій адміністратори, викладачі та співробітники служби допомоги освіти зв’язані з батьками та учнями. Крім того, ця система може бути використана для оцінок, відвідуваності, складання оголошень, системи читання карт при вході до навчального закладу. Переваги: зручний інтерфейс, багато функціональний. Недоліки: вартість.

Посещаемость / Идентификация - підтримує додатком для основного - "Відвідуваність - робочий час", і дозволяє використовувати згенерований QR-код для ідентифікації співробітників і для запису їх приходу або відходу. Додаток працює тільки спільно з додатком «Відвідуваність - робочий час». Переваги: простий інтерфейс, безкоштовний. Недоліки: потрібен додаток для роботи «Відвідуваність – робочий час».

Monitask – це універсальна електрона система, котра існує для контроля відвідуваності. Її використовують за кордом у різних офісах, школах. дитсадках, тощо . Монітаск дозволяє проводити повноцінні проектні заходи - створювати команди, запрошувати нових членів, створювати нові проекти і додавати завдання для окремих співробітників. З Монітаск дуже просто організувати і управляти робочими процесами компанії - Вашим віддаленим співробітникам потрібно буде тільки вибрати проект і завдання, призначене ним, і натиснути старт. Переваги: зручний інтерфейс, багато функціональний, безкоштовний. Недоліки: підтримує мало користувачів.

3. ОБГРУНТУВАННЯ ПРОЕКТНОГО РІШЕННЯ

У сучасному суспільстві ,під час ізоляції, усі навчальні заклади перейшли на дистанційну форму навчання. У зв’язку з цим необхідність у електронній системі моніторингу успішності учнів зросла. Вибір пав на створення мобільного додатку тому, що телефон це те, що ми завжди носимо з собою та завдяки ньому ми маємо постійний вихід в інтернет.

Після аналізу схожих додатків та виявлення їх переваг та недоліків були зроблені висновки на основі котрих був розроблений дизайн системи. Дизайн цієї електронної системи був створений з метою досягнення двох пунктів:

- Простий інтерфейс
- Багатофункціональність

Головною різницею мого проекту від інших аналогів є багата функціональність. Наприклад, у цій системі ви маєте можливість перевіряти свої оцінки по різнім дисциплінам чи по всім разом. Також ви маєте можливість перевіряти свою відвідуваність по різним дисциплінам. Головною особистістю цього проекту можна вважати систему сповіщення студентів.

Перевагою цієї електронної системи для студента ,по-перше, є зручний та простий інтерфейс, по-друге, завдяки цьому додатку студенти більше не повинні зв’язуватися з викладачами чи приїжджати до деканату для того, щоб дізнатися про свої заборгованості[6].

Перевагою цієї електронної системи для викладача ,по-перше, є зменшення роботи із паперами, тобто не потрібно заповнювати списки та відзначати відвідуваності студентів на аркушах. По-друге, під час карантину усі мають дотримуватися соціальної дистанції та якщо мають згоду, то залишатися удома. Ця системи допоможе перевести частину навчального процесу до інтернету.

4. ПРОЕКТУВАННЯ СИСТЕМИ

4.1 Проектування інформаційної системи

Для реалізації інформаційної системи обрана трирівнева архітектура клієнт-сервер. У такій технології існує тенденція, заснована на великому використанні розподілених обчислень. Вона реалізується на основі моделі сервера додатків, де мережевий додаток розділене на дві і більше частин, в даному випадку – три частини, кожна з яких може виконуватися на окремому комп'ютері.

У даній архітектурі між клієнтом і сервером бази даних є проміжний рівень – сервер додатків. Він виступає у ролі сервера для користувача та клієнтом для системи бази даних. Сервер додатків аналізує вимоги користувача і формує запити до сервера БД.

Для реалізації даної структури був обраний MVC патерн(рис. 2), який підходить до даної архітектури найбільше. У схемі ми маємо усього три компоненти: модель, уявлення та контролер, завдяки цьому модифікація кожного компонента може здійснюватися незалежно.

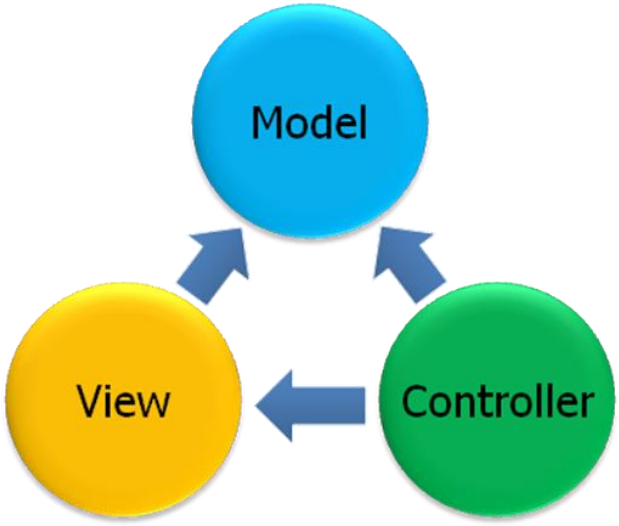

Рисунок 2 - Робота шаблону проектування MVC

Модель – це клас, що передає контролеру дані, запитувані користувачем. Далі отримані і оброблені дані відправляються у view (уявлення). Уявлення

передбачає собою інтерфейс користувача, на якому відображаються усі запитані дані з моделі. Контролер – це спеціальний клас, який отримує дані з запитів користувача і управляє ними. Коли виникає необхідність, контролер викликає відповідну модель для виконання поставленого завдання і відправки даних в БД.

4.2 Інформаційне моделювання

Першим етапом проектування є створення схеми БД, яка включає в себе десять таблиць та формалізовані зв'язки між ними. Результатом цього етапу є схема БД, представлена у вигляді UML діаграми(рис. 3). На цій діаграмі відображені основні сутності ПрО, а також відповідні сутності та зв'язки[1]. Для створення схеми ми повинні виконати наступні завдання: визначати сутності, визначити зв'язки між сутностями, визначення атрибутів, завдання первинних і зовнішніх ключів.

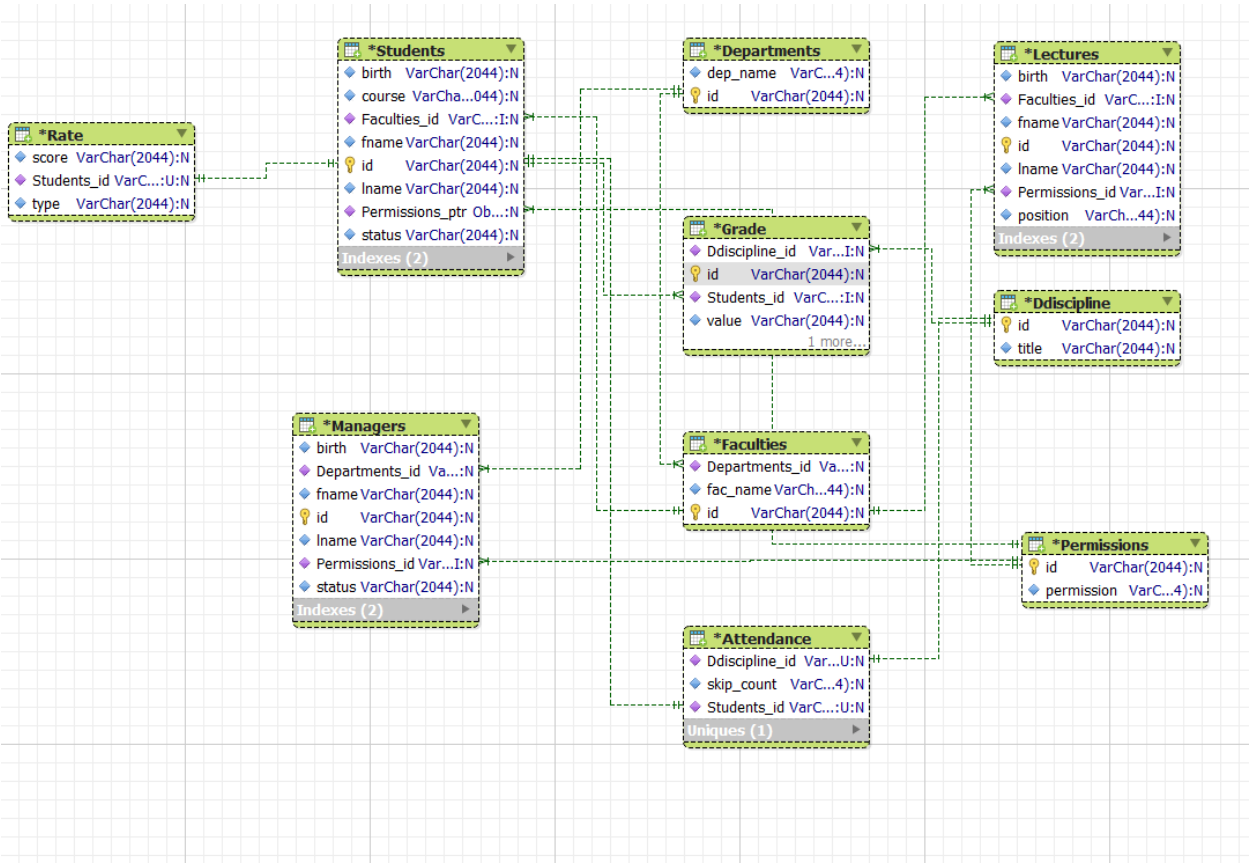

Рисунок 3 - UML діаграма

Для кожної сутності ми задали первинний ключ – унікальний ідентифікатор, який однозначно характеризує кожен екземпляр, а також унікальні ключі. На прикладі суті Student: id – первинний ключ.

Наступним кроком є формалізація зв’язків між сутностями. Зазначимо, якщо тип зв’язку між двома сутностями має вид «багато – багато», тоді використовується формалізація зв’язку – нова сутність. Ми можемо побачити таке між сутностями Student та Discipline, та у результаті з’являється нова сутність Attendance.

Існують інші допоміжні таблиці такі як migrations, oauth_access_tokens, oauth_auth_codes, oauth_clients, oauth_personal_access_clients, oauth_refresh_tokens, password_resets, users. Вони сформовані в цілях підтримки реєстрації та входу користувача до системи, а також підтримки API, необхідність якого виникла для подальшої роботи з мобільним додатком.

5. ПРАКТИЧНА ЧАСТИНА ПРОЕКТУ

5.1 Розробка та заповнення бази даних

В першу чергу ми розглянемо створення однієї з таблиць бази даних – Students[1].

```
public function up()
    {
       Schema::create('student',function(Blueprint $table) {
            $table->bigIncrements('id');
            $table->date('birth');
            $table->integer('course');
            $table->string('fname');
            $table->string('lname');
            $table->string('status');
            $table->unsignedInteger('Faculty_id');
            $table->unsignedInteger('Permission_id');
            $table->timestamps();
        });
    }
```

Створення таблиці відбувається за допомогою функції up(), яка користується об’єктом типу Schema для відтворення таблиці методом create. Метод create, у свою чергу, приймає два параметри: перший – назва для створення таблиці, а другий – функція, що створює атрибути таблиці за допомогою об’єкта Blueprint. Поле id є первинним ключем таблиці Student. Про те, що це первинний ключ каже запис bigIncrements. Запис буде являтися NOT NULL - означає, що при заповненні таблиці командою INSERT значення не може бути нульовим, або ж порожнім, відповідно, запис NULL означає, що поле може бути не заповнено і залишатися порожнім. Integer - рядок, який створює атрибут з числовим типом даних. String – це рядок, котрий створює атрибут з строковим типом даних. Date – рядок, який створює атрибут з типом даних date. Рядок unsignedInteger є зовнішнім ключем таблиці.

Всі інші таблиці створені з використанням відповідних запитів, детально наведено в Додатку А.

Заповнення таблиць робиться за допомогою команди Insert. Котру ми прописуємо у нашому ЗУБД[1]. Вона має такий формат: INSERT INTO назва_таблиці VALUES (значення_таблиці); Всі інші таблиці були заповненні по такому ж самому методу.

5.2 Розподіл ролей в базі даних

Дана інформаційна система буде мати три типи користувачів. Розглянемо детальніше їх права на таблиці бази даних:

1) Студент – має право на читання усіх сутностей даної системи;
2) Викладач – має право на читання таких сутностей: Студент, Дисципліна, Відвідуваність, Факультети, Департаменти та Рейтинг. Також має право на редагування та заповнення деяких з них: Оцінка, Відвідуваність;
3) Деканат – має право на читання усіх сутностей даної системи;

5.3 Програмна реалізація інформаційної системи

Так як основний компонент ООП - клас, реалізація програми представлена за допомогою класів[8]. Для реалізації інформаційної системи був створений ряд класів, які вирішують різні завдання в залежності від рівня, в якому вони знаходяться.

Model – це клас, що представляє собою таблиці бази даних. Для кожної сутності в БД існує своя модель. Наприклад, модель Student – успадковує клас Model і має в собі декілька функцій, котрі описують відносини суті в БД. У цьому прикладі це функції grade(), attendance(), вони описують ставлення «один»-«багато» до ці їх сутностей. Це задається методом hasMany та має такий вид: $ this -> hasMany (Student:: class).

Controller – базовий клас, який взаємодіє з моделлю і представленням. На кожну основну задачу створюється свій контролер. Наприклад, розглянемо

контролер, що оброблює модель класу Student - StudentController. У ньому є функція __invoke (), яка дозволяє при ініціалізації контролера вже виконувати якісь логічні дії, і в підсумку повертатися файл уявлення оператором return. Повний вид такої функції – return view(‘students’,[‘students’=>\App\Student ::all()]), де в квадратних дужках можуть передаватися параметри, отримані в результаті деяких дій логіки контролера, а так само дані, отримані при взаємодії з Model.

Розглянемо реалізацію розмежування ролей з боку архітектури програми. Перед тим як приступити до розгляду, варто сказати, що спочатку доступ до системи належить ролі Гість[2]. У даної ролі є обмежені права, що було описано вище.

Існує базовий клас Auth, який представляє собою аутентифікацію користувача. За допомогою нього, можна отримати методом check() інформацію про те, вийшов користувач в систему, чи ні. Програмна реалізація:

```
if(\Auth::check()) {
    $username = session('username');
    $ds;
    $u;
    $pass;
    $rol = \Auth::user()->role_;
```

З уявлення коду зрозуміло, що якщо користувач увійшов в систему, то з сесії дістається його для користувача ім'я, далі не започатковано змінні для нового підключення до БД. Потім, дістається значення із сутності БД Користувач, поле role_, яке повідомляє яку назву ролі у поточного користувача.

Далі має бути умовна конструкція, яка вибирає ім'я і пароль для нового підключення до баз даних, під новою роллю. Після такої маніпуляції, в сесію під ключем 'role_panel' додається назва ролі.

5.4 Запити до бази даних для вирішення результатів

У попередньому розділі було згадано про виникнення необхідності реалізації чистих SQL запитів, і було запропоновано нове рішення. Ґрунтуючись на реалізованому рішенні, була побудована структура, яка є файлом, де запити описані в поданні публічних функцій. Для прикладу, візьмемо функцію, яка здійснює запит до БД для отримання відвідуваності студентів по окремій дисципліні.

```
function studet_attend_solo($dbh, $student_id, $disc_id,
$course_number){
    $result = $dbh->query("select * FROM
student_attend($student_id, $disc_id,$course_number );");
    $array = array();
    while($data = $result->fetch(PDO::FETCH_ASSOC)){
      array_push($array, $data);
    }
    return $array;
   }
```

З прикладу видно, що в параметр функції потрапляє підключення до БД. Радиться ще раз звернути увагу, що підключення відбувається вже під існуючими ролями. Далі, в деяку змінну $result потрапляє запит, який здійснюється за допомогою підключення до БД і методу query(), в параметр якого приходить чистий SQL запит.

Після таких викликів, створюється масив, в який будуть додаватися дані, отримані з описаної змінної $result.

Заздалегідь , такі функції будуть викликатися на рівні контролера, де вже буде відбуватися маніпуляція даними. Розглянемо один з таких прикладів - StudentController, призначений для перегляду відвідуваності студента.

Функціонал контролера досить великий, в ньому описана вагома частина логіки процесів. Функція, яка реалізує даний функціонал має наступну реалізацію:

```
public function student_attend1(Request $request){
        require_once __DIR__.'../../../../queries.php';
        $stud_id = $request->only('Student_id');
```

```
        $disc_id = $request->only('Discipline_id');
        $course_number = $request->only('Course_number');
       $std_att         =         studet_attend_solo($dbh,
$stud_id['Student_id'],          $disc_id['Discipline_id'],
$course_number['Course_number']);//создание  переменной    =
результат
        return response()->json([
           'student' => $std_att,
       ], 200);
```

Як видно, функція отримує в параметрах об'єкт типу Request, що дозволяє отримувати дані відправлені API-запитом[4]. Розберемо детальніше обробку даних, наприклад, зміна $stud_id отримує ідентифікаційний номер студента таким чином: $request -> only ('Student_id'), що означає із екземпляру об'єкта Request, тобто ми отримаємо Student_id і записуємо у відповідну зміну. Усі інші дані записуються у змінну тим самим методом. У разі успішно виконаної роботи можна відправити успішну відповідь на запит:

```
return response()->json([
   'student' => $std_att,
], 200);
```

Як видно відповідь відправляється у вигляді json-об'єкту з двома параметрами – тіло відповіді та код відповіді. У даному випадку код відповідає 200, тому що все виконано успішно.

Далі можемо розглянути декілька прикладів збережених процедур. Перша задача – вивести на екран рейтинг студентів певного курсу. Для вирішення даної задачі було написано такий запит:

```
CREATE  OR  REPLACE  FUNCTION  student_rating(  stud_id  int,
faculty_id int , course_number int)
RETURNS TABLE (course int ,student text,faculty VARCHAR(255),
rating INT)
AS $$
    SELECT s.course,
           (s.fname ||' '||s.lname) as student,
           f."Fac_name" as faculty,
            r.score
    FROM Student s
    INNER JOIN faculty f on s."Faculty_id" = f.id
```

```
    INNER JOIN rate r on s.id = r."Student_id"
    WHERE CASE stud_id WHEN 0 THEN TRUE ELSE stud_id = s.id
END
        AND
          CASE faculty_id WHEN 0 THEN TRUE ELSE faculty_id =
f.id  END
        AND
          CASE   course_number   WHEN   0   THEN   TRUE   ELSE
course_number = s.course  END
$$ LANGUAGE SQL;
```

Друга задача – вивести на екран усіх студентів певного факультету та їх статус. Для неї був прописан такий запит:

```
CREATE OR REPLACE FUNCTION students_faculty(  faculty_id int
)
RETURNS TABLE (course int ,student text,faculty VARCHAR(255),
stat varchar(255))
AS $$
    SELECT s.course,
           (s.fname ||' '||s.lname) as student,
           f."Fac_name" as faculty,
            s.status
    FROM Student s
    INNER JOIN faculty f on s."Faculty_id" = f.id
    WHERE  faculty_id = f.id
$$ LANGUAGE SQL;
```

Для завдачі: вивести на екран відвідуванність певного курса по певній дисциплині. Для її вирішеня був написан наступний запит:

```
CREATE  OR  REPLACE  FUNCTION  student_attend(  stud_id  int,
disc_id int , course_number int)
RETURNS   TABLE   (course   int   ,student   text,discipline
VARCHAR(255), skip_count INT)
AS $$
            SELECT s.course,
                   (s.fname ||' '||s.lname) as student,
                   d.title as discipline,
                   a."Skip_count"
            FROM attendance a
            INNER JOIN student s on a."Student_id" = s.id
            INNER  JOIN  discipline  d  on  a."Discipline_id"  =
d.id
```

```
            WHERE CASE stud_id WHEN 0 THEN TRUE ELSE stud_id
= s.id  END
                AND
                  CASE disc_id WHEN 0 THEN TRUE ELSE disc_id
= d.id  END
                AND
                  CASE course_number WHEN 0 THEN TRUE ELSE
course_number = s.course  END;

$$ LANGUAGE SQL;
```

Також був написан тригер[5] для того, щоб викладач не міг поставити оцінку більше 100 та менше за 60:

```
CREATE FUNCTION grade_check() RETURNS TRIGGER
AS $$
    BEGIN
        if(new.value < 60 OR new.value > 100)
        THEN
            RAISE EXCEPTION 'Ошибка: оценка выше 100 или
меньше 60';
        END IF;
        return new;
    END
$$ LANGUAGE 'plpgsql';

CREATE TRIGGER grade_trigger
    BEFORE INSERT OR UPDATE  ON  Grade
    FOR EACH ROW
    EXECUTE PROCEDURE grade_check();
```

5.5 Реалізація інтерфейсу мобільного додатку

Розглянемо декілька прикладів з клієнтської частини проекту. Проектування складається з віджетів, кожен з яких відповідає за свій набір функціоналу та має свої властивості. Віджет Expanded відповідає за блок, якому можна не задавати розмір, розширюється завдяки змісту внутрішніх спадків. Container поєднує в собі властивості: відображення, позиціонування та розмірності. Описана поведінка представлена наступним кодом:

```
Container(
      padding:    EdgeInsets.symmetric(horizontal:    15.0,
vertical: 0),
      child: Column(
        children: <Widget>[
```

```
              SizedBox(height: 20),
              TextFormField(
            cursorColor: Colors.white,
            style: TextStyle(color: Colors.white70),
            decoration: InputDecoration(
              icon:        Icon(Icons.person,        color:
Colors.white70),
              hintText: "Ваше имя",
              border:       UnderlineInputBorder(borderSide:
BorderSide(color: Colors.white70)),
              hintStyle: TextStyle(color: Colors.white70),
            ),
          ),
          SizedBox(height: 15.0),
          (TextFormField(
            cursorColor: Colors.white,
            obscureText: true,
            style: TextStyle(color: Colors.white70),
            decoration: InputDecoration(
              icon:        Icon(Icons.vpn_key,        color:
Colors.white70),
              hintText: "Дисциплина",
              border:       UnderlineInputBorder(borderSide:
BorderSide(color: Colors.white70)),
              hintStyle: TextStyle(color: Colors.white70),
            ),
          )),
          SizedBox(height: 15.0),
        ],
      ),
    );
```

Також був зроблений динамичний список List<> з типом данних AttendDiscipline , до котрого ми додаємо наши пропски та у результаті виводимо це на екран настпним кодом:

```
List<AttendDiscipline>             _studAttend            =
List<AttendDiscipline>();
  Future   <List<AttendDiscipline>>   fetchAttendDiscplines()
async{
    sharedPreferences                =                await
SharedPreferences.getInstance();
    String UId =  sharedPreferences.getString("user_id");
    Map data = {
      'user_id': UId.toString()
```

```
    };
    var response = await
http.post(Uri.parse("http://10.0.2.2:8000/api/student/stud_
dec"), body: data);
    var attendDisciplines = List<AttendDiscipline>();
    if(response.statusCode == 200) {
      Map<String, dynamic> map = json.decode(response.body);
      List< dynamic> disciplinesJson = map["student"];

      for(var disciplineJson in disciplinesJson){

attendDisciplines.add(AttendDiscipline.fromJsonDiscipline(d
isciplineJson));
      }
    }
    print(attendDisciplines);
    return attendDisciplines;
  }
```

5.6 Тестування реалізованної серверної частини

Для тестування реалізованого REST API було обрано програму Postman. У ролі прикладу на даному етапі роботи можна навести запит api/student/stud_dec, що відповідає за відвідуваність студентів. Всі роути серверної частини реалізовані в файлі api.php. Роут, відповідний до наведеного запиту, виглядає таким чином:

```
Route::group(['prefix' => 'student'], function () {
        Route::post('stud_dec',
'StudentsController@student_attend1');
```

Створюється група для запитів, яка об’єднується префіксом 'student' і може розширюватися зі зростанням функціональності відповідної до логічної структури. В роутах також видно яким контролером оброблюється інформація і яка функція відповідає саме за конкретну задачу.

Як ви можете побачити на рисунку 4, в параметрах body до запиту вказується ключ «Student_id», що відповідає за id студента, щодо якого необхідно дізнатися про відвідуваність.

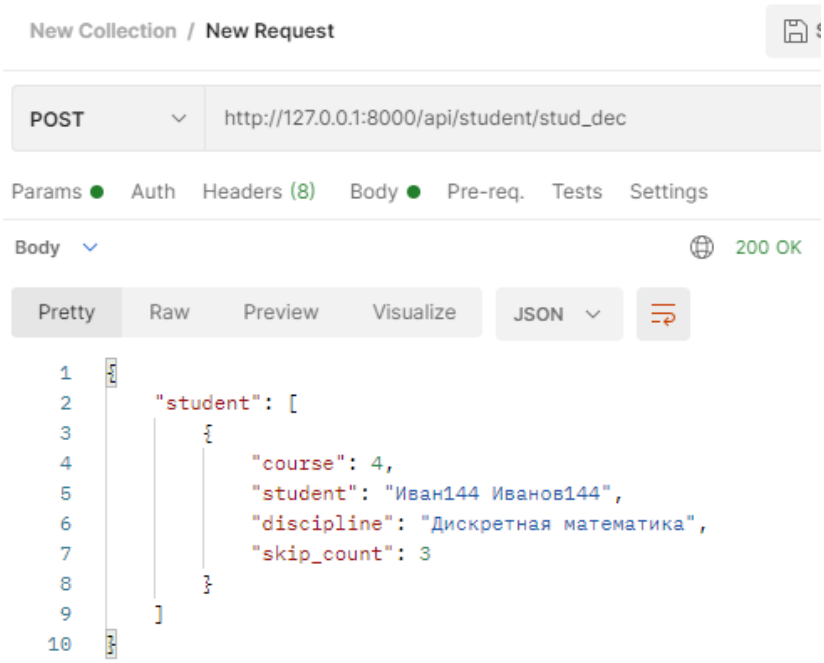

Рисунок 4 – Запит «відвідуваність певного студента по певній дисципліні»

В body response видно, що повернувся масив з ключем student, який видає його ім’я, курс студента, дисципліна та скільки у даного студента було пропусків. Згідно до цього, можна запевнити, що тестування запиту успішне, роут реалізований правильно і можна продовжити роботу далі.

5.7 Взаємодія клієнтської та серверної частини у мобільному додатку

Більшість дій між користувачем та програмою відбувається за допомогою кнопок, які дозволяють маніпулювати даними, які доступні користувачу. Уся стартова сторінка описана за допомогою кнопок. Розглянемо на прикладі кнопки, яка відповідає за задачу з відвідуваністю:

```
 RaisedButton(
                    onPressed: (){
                    setState(() {
//                    addPharmacy();

Navigator.of(context).pushAndRemoveUntil(MaterialPageRoute(
```

```
builder:    (BuildContext    context)    =>    AttendPage()),
(Route<dynamic> route) => false);
                  });},
                  color: Colors.white,
                  disabledColor: Colors.white70,
                  child:Column(
                      children: [
                        Padding(
                          padding:
EdgeInsets.only(left:100,right: 100),
                        ),
                        Text('Посещаемость     по     \nодной
дисциплине ', style: TextStyle(color: Color.fromRGBO(65, 120,
191, 0.8), fontWeight: FontWeight.bold,fontSize: 14)),]),
                  shape: RoundedRectangleBorder(
                      borderRadius:
BorderRadius.circular(10)),
                ),
```

Перед тим, як описати логіку блоку, яка вже демонструє вихідні дані, необхідно надати запит до серверу з відповідними вхідними даними. Даний етап здійснений за допомогою функції:

```
List<AttendDiscipline>             _studAttend             =
List<AttendDiscipline>();
  Future  <List<AttendDiscipline>>  fetchAttendDiscplines()
async{
    sharedPreferences                =                await
SharedPreferences.getInstance();
    String UId =  sharedPreferences.getString("user_id");
    Map data = {
      'user_id': UId.toString()
    };
    var             response             =             await
http.post(Uri.parse("http://10.0.2.2:8000/api/student/stud_
dec"), body: data);
    var attendDisciplines = List<AttendDiscipline>();
    if(response.statusCode == 200) {
      Map<String, dynamic> map = json.decode(response.body);
      List< dynamic> disciplinesJson = map["student"];

      for(var disciplineJson in disciplinesJson){

attendDisciplines.add(AttendDiscipline.fromJsonDiscipline(d
isciplineJson));
```

```
      }
    }
    print(attendDisciplines);
    return attendDisciplines;
  }
```

З наведеного класу видно, що сутність переймає описані атрибути сутності Student з бази даних, що й повертає API запит. Дана функція, що реалізує взаємодію між клієнтом та сервером, викликається при ініціалізації поточної сторінки таким чином:

```
void initState(){
    fetchAttendDiscplines().then((value) {
      setState((){
        _studAttend.addAll(value);
      });
    });
```

Для деяких сутностей нам небхідно було зробити форму для їх запиту, щоб пізніше вивести їх на екран разом з рельтатом роботи функції. Ви можете побачити цю форму на рисунку 5.

Рисунок 5 – Форма для запиту даних

З наведеного коду ми можемо побачити, що ми отримуємо відвідуванність певного студента. Наступним кроком необхідно показати це користувачу:

```
ListView.builder(
          itemCount: _studAttend.length,
          itemBuilder: (context, i) {
           ......
           title: Text(_studAttend[i].dateandtime),
           trailing: GestureDetector(
           child:Icon(Icons.menu_open, color:Colors.black87)
                       onTap: () {
                         setState(() {
                           print('OPEN MENU');
                           ......
                         });
                       },
                     ),
            subtitle: Text(_studAttend [i]. dateandtime,
                           ......
        height: heightStudAttend
    )]));
```

Змінна _studAttend переймає всі характеристики з класу AttendPage та містить у собі масив отриманих даних. Після всіх описаних дій користувач отримує відповідь від серверу, яка після обробки даних продемонстрована на рисунку 5.

Рисунок 5 – Відвідуваність певного студента по певній дисципліні

6. КЕРІВНИЦТВО КОРИСТУВАЧА

При запуску нашого мобільного додатку в першу чергу ви потрапляєте на сторінку авторизації, якщо ви маєте аккаунт, або на сторінку реєстрації , якщо ви не маєте аккаунту(рис.6-9). Також ,якщо ви ввели невірні дані , то вам вискочить повідомлення про помилку у набраних даних.

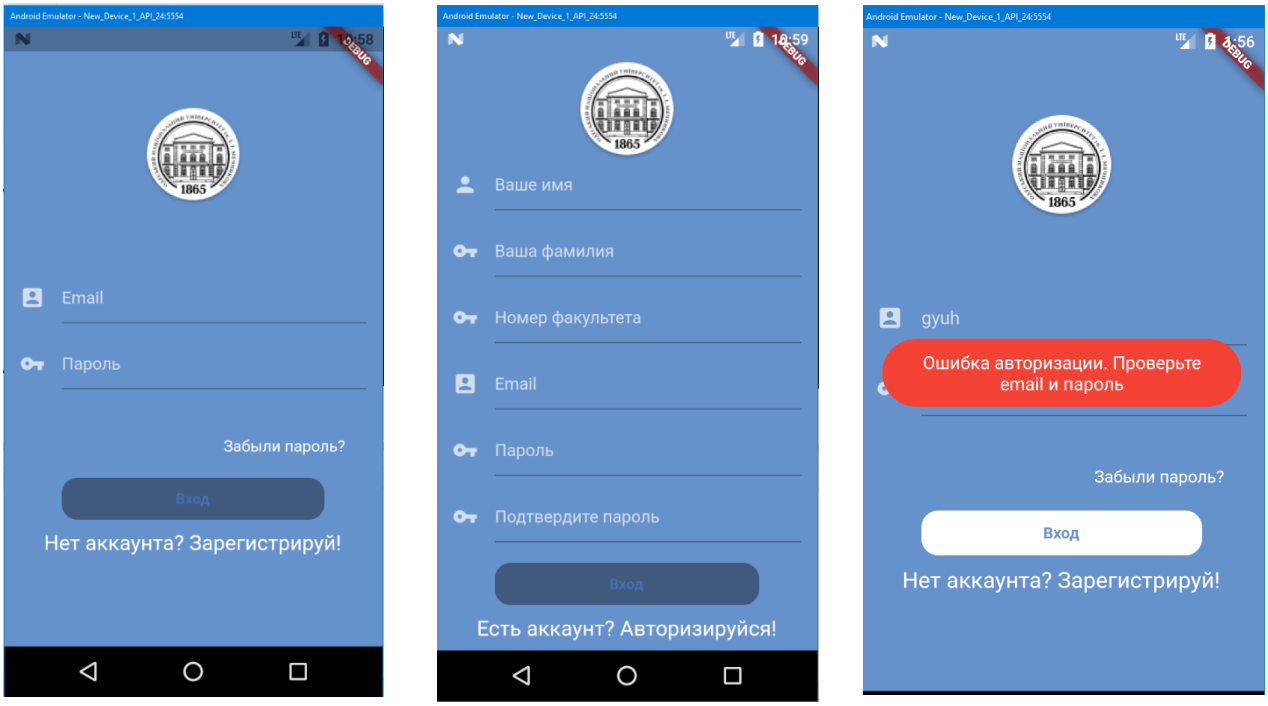

Рисунок 6-9 – Сторінка авторизації та реєстрації користувача

Як вже було зазначено система передбачає 3 типи користувачів: деканат, студент, викладач. На рис.10-11 представлено інтерфейс користувачів студенту та викладача. На ньому зображено ряд функцій які ви можете викликати натиском на відповідну кнопку. Серед ці їх функцій ви можете побачити інформаційну сторінку, перевірку відвідуваності, перевірку оцінок та кнопку вихід зі свого аккаунту.

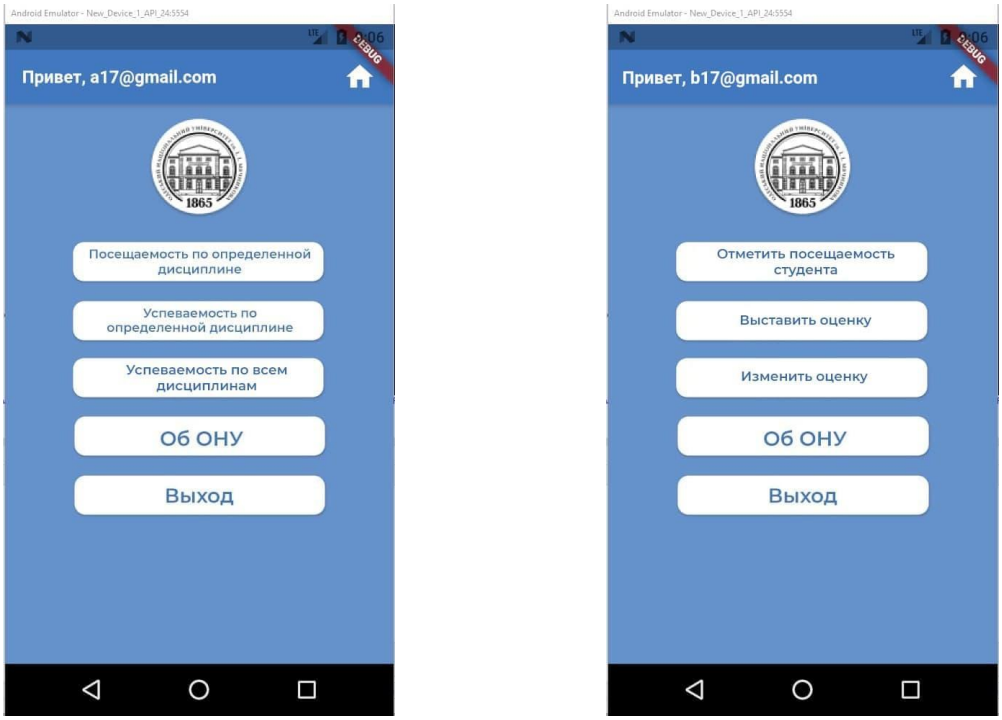

Рисунок 10-11 – Інтерфейс користувачів

Розглянемо одну з кнопок - це інформаційна сторінка, як показано на рис.12. На цій сторінці нам показується інформація о нашому університету, його історію.

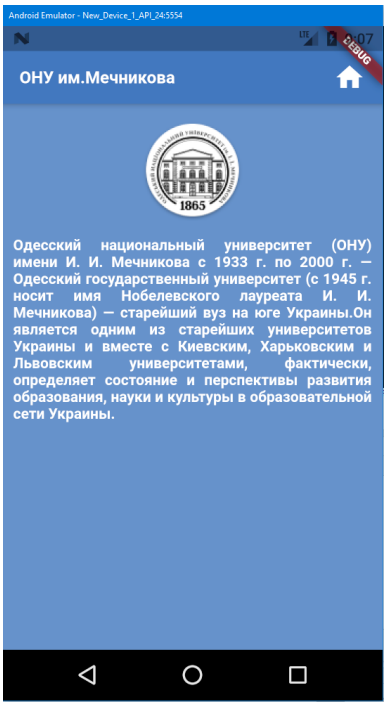

Рисунок 12 – Інформаційна сторінка

Далі можна продемонструвати роботу з відвідуваністю студента по всім дисциплінам. Весь процес зображено на рис.13-14. На першому рисунку ви можете побачити форму для запиту даних, а саме фамілії та імені. На другому ви можете побачити відвідуваність по усім дисциплінам.

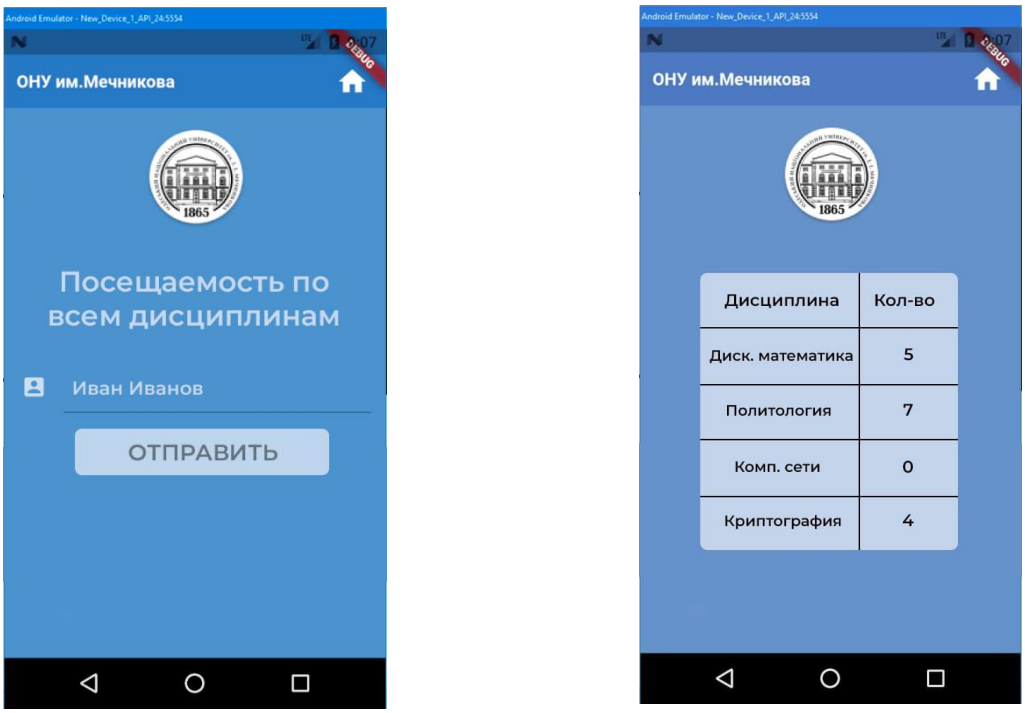

Рисунок 13-14 - Відвідуваність студента по всім дисциплінам

При досягненні певної кількості пропусків, у студента буде з’являтися повідомлення про те, що він ризикує попасти до списку на відчислення, як на рисунку 15.

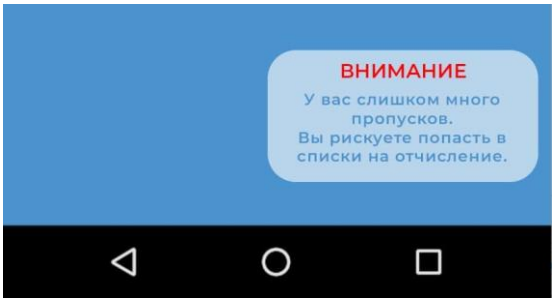

Рисунок 15 - Повідомлення

ВИСНОВОК

У результаті розробки електронно-інформаційної системи були досягненні усі поставлені перед проектом цілі, було виконано увесь перелік завдань, які вирішуються в поставленій ПрО. Була спроектована та побудована база даних, що використовуються у інформаційній системі, та були визначені вимоги для збереження даних. Для проекту було побудовано простий та багатий на функціонал інтерфейс.

Для цього проекту була використана трирівнева архітектура клієнт-сервер. Шаблоном проектування якої є патерн MVC.

У результаті роботи був створений мобільний додаток, завдяки якому користувачі зможуть легше получати доступ до даних та заповнювати їх, якщо є така можливість. Ця можливість реалізована шляхом створення ролей та їх можливостей (привілеїв).

Дана система має можливості для легкої масштабізації завдяки тому, що розроблена система створювалися на трирівневій архітектурі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Заповнення таблиць БД. [Електронний ресурс] – Режим доступу: https://postgrespro.ru/docs/postgrespro/9.5/dml-insert
2. Визначення прав доступу БД. [Електронний ресурс] – Режим доступу: https://postgrespro.ru/docs/postgresql/9.6/sql-grant
3. Коротко о головному. Flutter. [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/430918/
4. Керівництво по PHP. Розширення для роботи з базами даних. Рівні абстракції PDO. [Електронний ресурс] – Режим доступу: https://www.php.net/manual/ru/pdo.connections.php
5. Створення тригерів для БД. [Електронний ресурс] – Режим доступу: https://metanit.com/sql/sqlserver/12.1.php
6. Руководство к своду знаний по управлению проектами. Руководство PMBOK. 6-е издание , 2018. – 792 с.
7. Градусова Т.К., Жукова Т.А. Педагогические технологии и оценочные средства для проведения текущего и промежуточного контроля успеваемости и итоговой аттестации студентов. – Кемерово, 2013. – 100 с.
8. Документація фреймворка Laravel. [Електронний ресурс] – Режим доступу: https://laravel.com/docs/5.5
9. Документація Dart. [Електронний ресурс] – Режим доступу: https://flutter.su/docs
10. Неумоева-Колчеданцева Е.В. Психолого-педагогическое взаимодействие участников образовательного процесса. – 2016. – 159 с.

ДОДАТОК А

ЗАПИТИ НА СТВОРЕННЯ ТАБЛИЦЬ БАЗИ ДАНИХ

// Створення таблиці ‘Student’

```
class CreateStudentTable extends Migration
{
    public function up()
    {
        Schema::create('student',    function    (Blueprint
$table) {
            $table->bigIncrements('id');
            $table->date('birth');
            $table->integer('course');
            $table->string('fname');
            $table->string('lname');
            $table->string('status');
            $table->unsignedInteger('Faculty_id');
            $table->unsignedInteger('Permission_id');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('student');
    }
}
```

// Створення таблиці ‘Grade’

```
class CreateGradeTable extends Migration
{
    public function up()
    {
        Schema::create('grade', function (Blueprint $table)
{
            $table->bigIncrements('id');
            $table->integer('value');
            $table->unsignedInteger('discipline_id');
            $table->unsignedInteger('Student_id');
            $table->timestamps();
        });
    public function down()
    {
        Schema::dropIfExists('grade');
    }
}
```

// Створення таблиці ‘Faculty’

```
class CreateFacultyTable extends Migration
{
    public function up()
    {
        Schema::create('faculty',    function    (Blueprint
$table) {
            $table->bigIncrements('id');
            $table->string('Fac_name');
            $table->unsignedInteger('Department_id');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('faculty');
    }
}
```

// Створення таблиці ‘Discipline’

```
class CreateDisciplineTable extends Migration
{
    public function up()
    {
        Schema::create('discipline',   function   (Blueprint
$table) {
            $table->bigIncrements('id');
            $table->string('title');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('discipline');
    }
}
```

// Створення таблиці ‘Attendance’

```
class CreateAttendanceTable extends Migration
{
    public function up()
    {
        Schema::create('attendance',   function   (Blueprint
$table) {
            $table->bigIncrements('id');
            $table->unsignedInteger('Student_id');
```

```
            $table->unsignedInteger('Discipline_id');
            $table->integer('Skip_count');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('attendance');
    }
}
```

// Створення таблиці ‘Manager’

```
class CreateManagerTable extends Migration
{
    public function up()
    {
        Schema::create('manager',    function    (Blueprint
$table) {
            $table->bigIncrements('id');
            $table->unsignedInteger('Department_id');
            $table->unsignedInteger('Permission_id');
            $table->date('birth');
            $table->string('fname');
            $table->string('lname');
            $table->string('status');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('manager');
    }
}
```

// Створення таблиці ‘Lecture’

```
class CreateLectureTable extends Migration
{
    public function up()
    {
        Schema::create('lecture',    function    (Blueprint
$table) {
            $table->bigIncrements('id');
            $table->unsignedInteger('Department_id');
            $table->unsignedInteger('Permission_id');
            $table->date('birth');
            $table->string('fname');
```

```
            $table->string('lname');
            $table->string('position');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('lecture');
    }
}
```

// Створення таблиці ‘Rate’

```
class CreateRateTable extends Migration
{
    public function up()
    {
        Schema::create('rate', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->integer('score');
            $table->string('type');
            $table->unsignedInteger('Student_id');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('rate');
    }
}
```

// Створення таблиці ‘Department’

```
class CreateDepartmentTable extends Migration
{
    public function up()
    {
        Schema::create('department',   function   (Blueprint
$table) {
            $table->bigIncrements('id');
            $table->string('dep_name');
            $table->string('address');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('department');
    }
}
```

// Створення таблиці ‘Permission’

```
class CreatePermissionTable extends Migration
{
    public function up()
    {
        Schema::create('permission',   function   (Blueprint
$table) {
            $table->bigIncrements('id');
            $table->string('Permission');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('permission');
    }
}
```

ДОДАТОК Б

ВИХІДНИЙ КОД ОСНВНИХ КЛАСІВ

Клас, який відповідає за стартову сторінку з описанням функціоналу усіх кнопок у меню, їх адресацію, описання та графічну складову. Також у цьому класі існує описання шапки нашого додатку.

```
class HomePage extends StatefulWidget{

  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {

  var menuHeigth = 0.0;
  var menuTextName = '';
  var auditId;
  var menuTextTime = '';
  var iconMenu = true;

  SharedPreferences sharedPreferences;

  List<AttendDiscipline>            _studAttend            =
List<AttendDiscipline>();
  Future  <List<AttendDiscipline>>  fetchAttendDiscplines()
async{
    sharedPreferences                 =                 await
SharedPreferences.getInstance();
    String UId =  sharedPreferences.getString("user_id");
    Map data = {
      'user_id': UId.toString()
    };
    var             response             =             await
http.post(Uri.parse("http://10.0.2.2:8000/api/student/stud_
dec"), body: data);
    var attendDisciplines = List<AttendDiscipline>();
    if(response.statusCode == 200) {
      Map<String, dynamic> map = json.decode(response.body);
      List< dynamic> disciplinesJson = map["student"];

      for(var disciplineJson in disciplinesJson){

attendDisciplines.add(AttendDiscipline.fromJsonDiscipline(d
isciplineJson));
      }
    }
    print(attendDisciplines);
    return attendDisciplines;
```

```
  }

  @override
  void initState(){

  }

  Widget build(BuildContext context) {
    return Container(
        color: Color.fromRGBO(65, 120, 191, 0.8) ,
        child: Column(
            children: <Widget>[
              Padding(
                padding: EdgeInsets.only(top: 10),
                child: Container(
                  child: Align(
                      child: Image.asset('images/onu.png',)

                  ),
                ),
              ),

              Padding(
                padding:    EdgeInsets.only(top:    10,left:
25,right: 25),
                child: RaisedButton(
                  onPressed: (){
                  setState(() {
//                  addPharmacy();

Navigator.of(context).pushAndRemoveUntil(MaterialPageRoute(
builder:    (BuildContext    context)    =>    AttendPage()),
(Route<dynamic> route) => false);
                  });},
                  color: Colors.white,
                  disabledColor: Colors.white70,
                  child:Column(
                      children: [
                        Padding(
                          padding:
EdgeInsets.only(left:100,right: 100),
                        ),
                        Text('Посещаемость     по     \nодной
дисциплине ', style: TextStyle(color: Color.fromRGBO(65, 120,
191, 0.8), fontWeight: FontWeight.bold,fontSize: 14)),]),
                  shape: RoundedRectangleBorder(
                      borderRadius:
BorderRadius.circular(10)),
                ),
              ),
              Padding(
```

```
                padding:    EdgeInsets.only(top:    10,left:
1,right: 1),
                child: RaisedButton(
                  onPressed: (){

Navigator.of(context).pushAndRemoveUntil(MaterialPageRoute(
builder:    (BuildContext    context)    =>    InfoPage()),
(Route<dynamic> route) => false);
                  },

                  color: Colors.white,
                  disabledColor: Colors.white70,
                  child: Column(
                    children: [
                      Padding(
                        padding:
EdgeInsets.only(left:100,right: 100),
                      ),
                      Text('Об ОНУ', style: TextStyle(color:
Color.fromRGBO(65,     120,     191,     0.8),     fontWeight:
FontWeight.bold, fontSize: 17)),]),
                        shape: RoundedRectangleBorder(
                      borderRadius:
BorderRadius.circular(10)),

                ),
              ),
              Padding(
                padding:    EdgeInsets.only(top:    10,left:
25,right: 25),
                child: RaisedButton(
                  onPressed: (){

Navigator.of(context).pushAndRemoveUntil(MaterialPageRoute(
builder:    (BuildContext    context)    =>    LoginPage()),
(Route<dynamic> route) => false);
                  },
                  color: Colors.white,
                  disabledColor: Colors.white70,
                  child:Column(
                    children: [
                      Padding(
                        padding:
EdgeInsets.only(left:100,right: 100),
                      ),
                  Text('Выход',    style:    TextStyle(color:
Color.fromRGBO(65,     120,     191,     0.8),     fontWeight:
FontWeight.bold,fontSize: 17)),]),
                  shape: RoundedRectangleBorder(
                      borderRadius:
BorderRadius.circular(10)),
                ),
```

```
                )
            ]
        )
    );
  }
```

Клас StudentController і його основний метод student_attend1, який підраховує кількість пропусків певного студента з певній дисципліні.

```
class StudentsController extends Controller
{
    //
    public function student_attend1(Request $request){
        require_once __DIR__.'../../../../queries.php';
        $stud_id = $request->only('Student_id');
         $disc_id = $request->only('Discipline_id');
         $course_number = $request->only('Course_number');
        $std_att          =          studet_attend_solo($dbh,
$stud_id['Student_id'],           $disc_id['Discipline_id'],
$course_number['Course_number']);//создание  переменной    =
результат
         return response()->json([
            'student' => $std_att,
        ], 200);
    }
}
```

RegisterController – це клас, який дозволяє новим користувачам реєструватися у системі. У ньому прописані усі необхідні поля, які потрібно заповнити для реєстрації в системі.

```
class RegisterController extends Controller
{

    use RegistersUsers;

    protected $redirectTo = '/home';

    public function __construct()
    {

        $this->middleware('guest');

    }

    protected function validator(array $data)
    {
        return Validator::make($data, [
```

```
            'fname' => ['required', 'string', 'max:255'],
            'lname' => ['required', 'string', 'max:255'],
            'email'  =>  ['required',  'string',  'email',
'max:255', 'unique:users'],
            'Faculty_id' => ['required', 'integer'],
            'birth' => ['required', 'date'],
            'password'  =>  ['required',  'string',  'min:8',
'confirmed'],
        ]);
    }

    protected function create(array $data)
    {

        $student = new Student();
        $student->status = 'учится';
        $student->fname = $data['fname'];
        $student->lname = $data['lname'];
        $student->birth = $data['birth'];
        $student->course = 1;
        $student->Faculty_id = $data['Faculty_id'];
        $student->Permission_id = 1;
        // $employee->news = 'news';
        $student->save();

        return User::create([
            'fname' => $data['fname'],
            'lname' => $data['lname'],
            'email' => $data['email'],
            'birth' => $data['birth'],
            'Faculty_id' => $data['Faculty_id'],
            'password' => Hash::make($data['password']),
            'role' => 2
        ]);

    }
}
```