МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ФІЗИКИ ТА МАТЕМАТИКИ
КАФЕДРА ІНФОРМАТИКИ, ПРОГРАМНОЇ ІНЖЕНЕРІЇ ТА ЕКОНОМІЧНОЇ КІБЕРНЕТИКИ

РОЗРОБЛЕННЯ СЕРВЕРНОЇ ЧАСТИНИ СЕРВІСУ МОНІТОРИНГУ ВІДВІДУВАНЬ

Кваліфікаційна робота (проєкт)

на здобуття ступеня вищої освіти "бакалавр"

Виконав: студент 4 курсу
Спеціальності 121 Інженерія програмного забезпечення
Освітньо-професійної програми
«Інженерія програмного забезпечення»
першого (бакалаврського) рівня освіти
Шишман Владислав Володимирович
Керівники кандидат педагогічних наук,
доцент Вінник Максим Олександрович
кандидат фізико-математичних наук,
доцент Єрмолаєв Вадим Анатолійович
Рецензент кандидат педагогічних наук,
доцент Єрмакова-Черченко Наталія Олександрівна

Херсон – 2020

ЗМІСТ

ВСТУП

Відвідування занять студентами значно впливає на загальні показники успішності та рівень здобутих знань протягом освітнього процесу.

Одним із можливих шляхів його підтримки на високому рівні є контроль. За даними дослідження, проведеного журналом Chickago, існування перевірки з боку ВНЗ збільшує візити студентів на заняття на 11-18% [1].

Зазвичай цей процес здійснюється вручну. Значна чисельність людей потребує великої кількості часу на його на здійснення. Кожен пропуск може мати причину, яку потрібно враховувати та можливі похибки. Полегшити контроль може автоматизація обліку відвідувань.

Об’єкт дослідження: облік відвідування занять студентами, архітектура серверної частини.

Предмет дослідження: шляхи автоматизації обліку відвідування занять студентами за допомогою інформаційних технологій.

Мета: розробити серверну частину сервісу моніторингу відвідувань.

Завдання:

1. Дослідити засоби створення серверної частини веб-додатків.
2. Розробити систему ролей для надання обмеженого доступу до даних.
3. Спроєктувати базу даних для зберігання інформації про відмітки.
4. Розробити API для взаємодії серверної і клієнтської частин.
5. Опублікувати отриманий проєкт в мережі інтернет.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Огляд процесу контролю відвідувань в університеті

Розробляєма система є альтернативою уже сформованої системи планової перевірки відвідуваності, тому доцільно буде її розглянути.

Для відмітки студентів потрібно знати дані про пару такі як: назва, час проведення, ім’я та фамілія вчителя, назви груп, які мають бути присутні, та імена і фамілії студентів що їх складають.

Вищий навчальний заклад(далі ВНЗ) в своїй основі має певну структуру [2]. Є колегіальне керівництво(ректорат), цей відділ вирішує основні питання щодо діяльності навчального закладу та здійснює управління вложеними структурами, навчальними підрозділами (факультетами). Кожен може вирішувати питання щодо своєї діяльності, має напрям існування та список спеціальностей за якими можуть навчатися студенти. На факультеті є керуючий орган (деканат) та науково-дослідні підвідділи (кафедри), рис. 1.1.

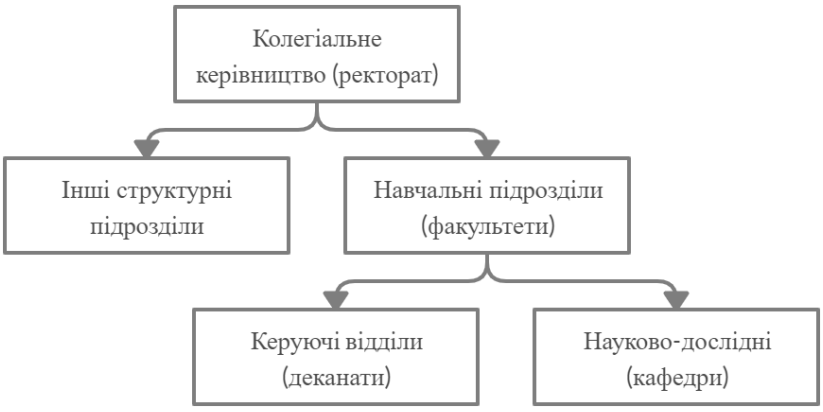

Рис. 1.1 Структура ВНЗ

Дані про пари створюються людьми на факультеті які формують розклад. Потім заняття розміщуються в ньому відповідно до дня тижня (лекційного або практичного) та номера пари. Тижні періодично чередуються.

Кожна пара викладається певним вчителем. На ній можуть бути присутні декілька груп з різних факультетів.

Викладач і студент зв’язані з кафедрою. Тільки для першого це місце роботи а для другого це місце навчання. Однак в прямому обліку відвідувань кафедра участі не бере.

Опираючись на вищенаведені твердження в спроєктованій системі мають бути:

- ролі студента, викладача, адміністратора.
- поняття пар зв’язує викладача, студентів, час проведення.

1.2. Огляд існуючих аналогів

В ході роботи ми дослідили основні системи, які використовуються на ринку, це: BioTime Edu, ClassDojo, GoGuardian Teacher, TimeTac.

BioTime Edu – це компонентна, напівавтоматична система моніторингу. Студент фіксує свою присутність завдяки відбитку пальця або біоментричного контролю. Після чого викладач може побачити автоматично згенеровані списки присутніх з датами приходу і виходу в веб версії панелі користувача або android додатку. Планування пар здійснюється візуально викладачем [3]. Окрім цього, вона має партнерство з сторонніми сервісами такими як UKBA та Tier 4. Завдяки чому також може виконувати додаткові функції: зберігати данні про картки visa, і повідомляти про закінчення дії паспорта.

Оскільки система є компонентною, то є необхідність в підключенні та налаштуванні складових пристроїв, рис. 1.2.

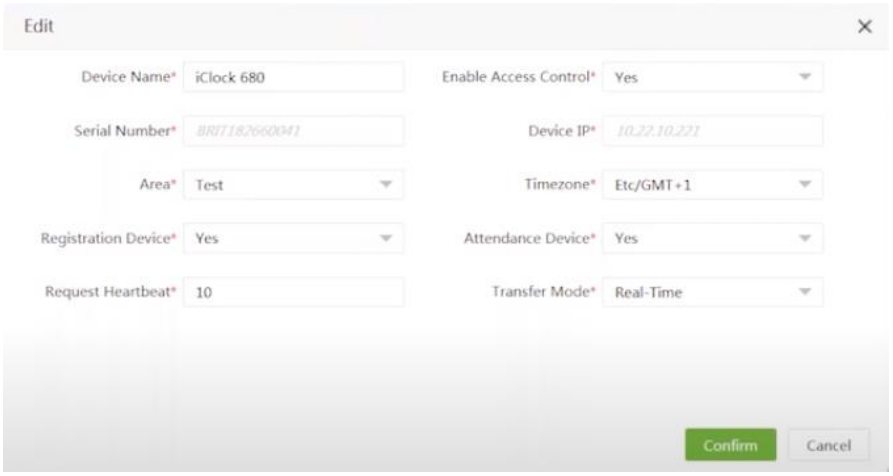

Рис. 1.2 Підключення сканеру відбитків пальців

TimeTac також є компонентною системою. Студент може зафіксувати свою присутність завдяки відбитку пальця або скориставшись RFID/NFC чіпом. Як і в випадку з BioTime Edu запис про відвідування в базу додається автоматично після взаємодії з тригером. Викладач може переглянути статистику зайшовши в веб версію, android додаток, або на спеціальному приймачу [4]. Особливість цієї системи полягає в тому, що вона відображає дані за принципом вкладок і таблиць та викладач може побачити інформацію в полях таблиці, яку вказав учень, рис. 1.3. За записами система може відобразити: статистичні дані, перерви, статус роботи.

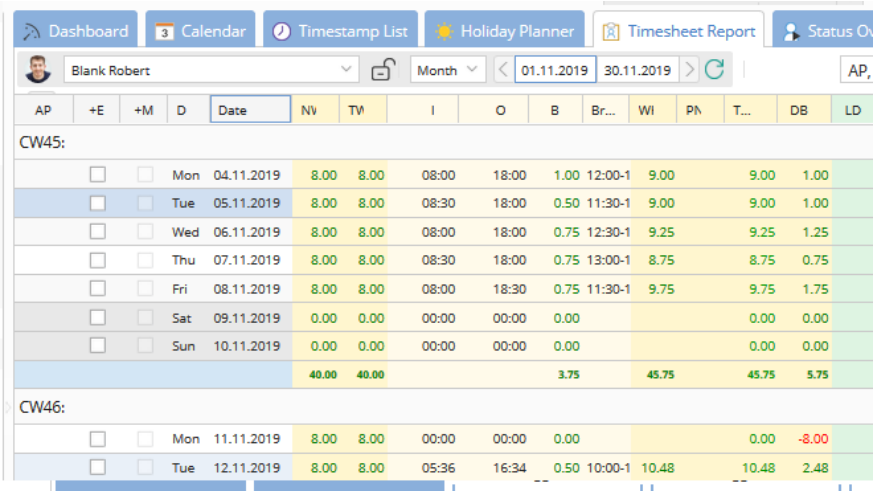

Рис. 1.3 Інтерфейс користувача TimeTac

GoGuardian Teacher є цілісною системою. Зайшовши в веб, десктоп чи мобільний додаток і приєднуючись до класу, студент фіксує свою присутність. Вчитель може сформувати загальний список студентів і позначити в них присутніх та відсутніх, налаштувати та ввімкнути трекінг дій учня, встановити час для виконання конкретного завдання. Окрім цього в системі існує можливість спільного навчання в режимі класа та є декілька ролей [5]. Вони відрізняються кількістю дозволених дій, рис. 1.4.

	Permissions		
Feature	Helper	Teacher	Owner
Edit Classroom Info	✗	✗	✓
Archive Classroom	✗	✗	✓
Add / Remove Teachers	✗	✗	✓
Add / Remove Students	✗	✓	✓
Start Sessions	✗	✓	✓
View Active Sessions	✓	✓	✓
End Sessions	✓	✓	✓
Send Commands	✓	✓	✓
Toggle Teacher Chat	✓	✓	✓
Apply / Change Scenes	✓	✓	✓

Рис. 1.4 Дозволи ролей в системі GoGuardian Teacher

Проаналізувавши наведені вище додатки, було виявлено, що в спроєктованій системі має бути можливість:

- створювати або інтегрувати попередньо створений розклад, як це зроблено в системах BioTime Edu та TimeTac.
- формувати та вносити корективи в списки студентів і дозволи, на кшталт того як є в системі GoGuardian Teacher.

1.3. Вибір мови програмування та фреймворку

Серверна частина може працювати на великій кількості різних технологій. Вони відрізняються мовою програмування, функціональною складовою, часом створення, підтримкою спільноти, складністю подальшої підтримки [6].

Для того щоб вибрати ту яка буде використовуватись в розробці було проведено дослідження по наведених вище критеріях.

У якості відправної точки ми розглянули таблицю з найбільш популярні мови програмування за індексом сайту PYPL. В ній стовпець Share характерезує на скільки мова є поширеною, Trend - на скільки збільшилася популярність в порівнянні з попереднім місяцем. Серед перелічених є 8 мов які можна використати для програмування серверної частини: Python, Java, Node(JS), C#, PHP, Kotlin, Go, Ruby, рис. 1.5.

Worldwide, May 2020 compared to a year ago:				
Rank	Change	Language	Share	Trend
1	undef	Python	31.17 %	+4.3 %
2	undef	Java	17.75 %	-2.4 %
3	undef	Javascript	7.99 %	-0.3 %
4	undef	C#	7.05 %	-0.2 %
5	undef	PHP	6.09 %	-1.0 %
12	up	Kotlin	1.55 %	+0.3 %
14	up	Go	1.26 %	+0.1 %
15	down	Ruby	1.19 %	-0.2 %

Рис. 1.5 Мови PYPL

Найбільшу популярність має Python, підтримку та розмір спільноти – Node JS. Це можна пояснити тим, що багато людей цікавляться Python, так як він вивчається в навчальних закладах, але поки що не так багато людей використовує його на роботі. Зараз спільнота Python продовжує зростати, в ближчому майбутньому вона імовірно буде на рівні з Node. Потім за зменшенням йдуть Java, C#, PHP і Ruby.

Важливу роль також грає підтримка мовами роботи з новими технологіями, мікросервісами. Є два варіанти інфраструктури для їх побудови: безсерверний та контейнерний.

Безсерверні рішення та мови, які їх підтримують:

- Amazon Lambda підтримує Python, Ruby, Java, C#, Go і Node.
- Google Cloud Functions підтримує Python, Go та Node.
- Microsoft Azure Functions підтримує Python, Java, C#, Node.
- Apache OpenWhisk Python, Ruby, Java, .NET, Go, Scala, PHP, Node.
- Kubeless підтримує Python, Node.js, Ruby, PHP, Golang, .NET.

Оскільки Kotlin працює на JVM то його можна використовувати там, де підтримується Java. Дані цих двох мов залежать одне від одного.

Мови, в порядку зростання кількості доступних контейнерів: C#, Kotlin, Ruby, Java, Python, Node, Go, PHP.

Для побудови мікросервісів найкращими варіантами є:

- Завдяки без серверним рішенням - Python та Node.
- Завдяки контейнерам - PHP та Go.

Найбільш швидкими мовами є Java, C# и Go, так як вони компілюються в високо оптимізований байткод. Продуктивність Kotlin така ж як Java, NodeJS на 40% < Java, PHP ~ в 2 рази < NodeJS, Ruby ~ на 10% > Python, Python ~ в 2 рази < PHP [6].

З точки зору можливостей інтеграції з іншими системами, найкращими мовами програмування є PHP та Java. Також хороший вибір - Python та Node. Більшість SDK створені саме для них. Java в значній мірі використовується в розробці корпоративних додатків, так як вона має гарну інтеграцію та є стабільною [7].

Отже, з точки зору майбутньої підтримки слід вибрати Java або Python.

Основним фреймворком Java є Spring. Він дозволяє будувати складні системи з великою кількістю залежностей та має стрімку криву вивчення.

Основними фреймворками Python є Flask і Django. Перший є мікрофреймворком. Другий має:

- чітку систему проєктування
- вбудований ORM для роботи з базами даних
- можливість захисту від основних типів загроз

Ці характеристики в проєкті є важливими, тому в якості фреймворку для роботи було обрано Django.

1.4. Вибір бази даних

База даних – це логічно структуровані сховища будь-яких типів даних. Кожна база даних слідує моделі яка задає певну структуру обробки даних [8].

СУБД – це додатки (або бібліотеки), що управляють базами даних різних форм, розмірів і типів.

За типом бази даних бувають:

- Реляційні – такі бази будуються на таблицях та відношеннях між ними. Кожна таблиця має атрибути з певним типом даних. Таблиці в реляційній БД мають пов’язані між собою ключами, які використовуються для ідентифікації конкретних стовбців або рядків. Таким чином досягається велика швидкість доступу до даних.
- Не реляційні – ключовою відмінністю від реляційної моделі є те що в них не має значних вимог до структури [9]. Такі бази даних дозволяють керувати не структурованими даними.

Оскільки в проєкті будуть структуровані дані був вибраних реляційний тип баз даних.

Клієнтів для керування базами даних зараз існую багато, але для порівняння ми вибрали основні три: SQLite, MySQL, PostgreSQL.

Переваги SQLite:

- Файлова система: вся база даних зберігається в одному файлі, що полегшує переміщення.
- Стандартизованість: SQLite використовує SQL; деякі функції опущені (RIGHT OUTER JOIN, FOR EACH STATEMENT), але є і деякі нові.
- Добре підходить для розробки і тестування

Недоліки SQLite:

- Не можна налаштувати базу щоб зробити її продуктивнішою

- Відсутність панелі управління користувача, однак цей факт частково компенсується встановленням додаткового програмного забезпечення, на кшталт DB Browser.

Переваги MySQL:

- Панель управління користувача.
- СУБД підтримує більшу частину функціоналу SQL.
- Має налаштування параметрів безпеки, можна встановити дозволи для окремого користувача, в версії 8.0 встановити шифрування пароля по алгоритму sha256 [10].
- Масштабованість, цю базу даних можливо розподілити між декількома серверами.

Недоліки MySQL:

- Обмеження функціональності для деяких типів операцій.
- Проміжок часу між виходом версій.

Переваги PostgreSQL:

- Повна SQL-сумісність.
- Можна програмно розширити за рахунок збережених процедур.
- Об’єктно-орієнтована СУБД, тобто вона добре суміщується з концепцією ООП.
- Стійкість, ведення WAL [11].

Недоліки PostgreSQL:

- Для кожного нового користувача створюється процес.
- Реплікація даних може зайняти значну частину ресурсів [12].

Опираючись на проведене вище дослідження для проєкту була вибрана база даних MySQL.

1.5. Принципи побудови RESTful архітектури

API це сукупність засобів необхідних для спілкування компонентів систем. Одним із підходів до архітектури API є REST [13]. Він дозволяє проєктувати підтримуванний API. Для цього в ньому заложено шість принципів:

- Client-server, полягає в відокремленні призначеного для користувача інтерфейсу від обробки даних. Тим самим, ми покращуємо можливості переносу функціоналу, та збільшення функцію.
- Stateless полягає в відсутність стану, в випадку раптового завершення зв’язку, API залишається функціональним, так як клієнт передає всю необхідну інформацію для обробки запиту.
- Cacheable, полягає в можливості кешування, кожен запит має в відповіді покажчик щодо повторного використання даних клієнтом.
- Uniform interface, полягає в існувані єдиного інтерфейсу між клієнтом та сервером, це спрощує і відокремлює їх архітектуру, що дозволяє кожній частині змінюватися окремо. Для отримання єдиного інтерфейсу існує 4 обмеження:
    - Identification of resources, ідентичність – кожен ресурс повинен бути ідентифікований завдяки стабільному ідентифікатору. В нашому випадку URL. Ресурсом являється все те чому можна дати ім’я(документ, зображення, і тд.)
    - Resources through representations – для виконання дій над ресурсами використовується подання. Воно відображає поточний або бажаний стан ресурсу.
    - Self-descriptive messages – кожне подання містить достатньо інформації для того щоб клієнт міг його обробити.
    - Hypermedia – статус ресурса передається через URL, його параметри, вміст заголовків або тіла запитів.

- Layered System, полягає в можливості розділу системи на шари. Клієнт повинен знати тільки про ту систему з якою спілкується.
- Code-On-Demand(опціонально), полягає в можливості завантаження коду на стороні клієнта.

Якщо указана служба, спроєктована з використанням цих положень то вона побудована на архітектурі Rest, а сервіс що її реалізує називається Restful.

1.6. Огляд технологій аутентифікації

В сучасному світі існує проблема ідентифікації користувача та підтримки сесії з ним, адже протокол HTTP не запам’ятовує статус. Таким чином неможливо прив’язати сесію до нього. Щоб вирішити цю проблему деякий час назад була придумана концепція Server Based [14].
В ній користувач відсилає на сервер свої логін та парол. Він генерує Cookie, зберігає його в базі даних і повертає на клієнт. При подальшому візиті користувача на сервіс, аутентифікація відбувається завдяки йому.

По мірі розвитку інтернету з ним почали з’являтися проблеми. Кожен раз, коли клієнт використовує Cookie, відбувається запит до місця зберігання даних і в випадку коли таку дію робить багато користувачів навантаження на сервер збільшуються [15]. Це робить підтримку та розширення інфраструктури складнішим.

Token Authentication полягає в тому що клієнт відправляє дані для аутентифікації на сервер. Він генерує унікальний токен і повертає. Потім, при кожному новому запиті клієнт відправляє його в заголовку запиту замість логіну і паролю. В Django Rest він кодується алгоритмом base64 [16, с. 121].

Він може бути stateless. За замовченням він stateful існує окрема таблиця що пов’язує токени з користувачами. Тому періодично потрібно

робити запит щоб виділити взаємозв’язок між ними. Можна зменшити кількість запитів, виконуючи кешування.

Json Web Token є покращеною версією попереднього. Він підтримує встановлення обмеження на час дії, генерацію унікальних кодів в залежності від клієнту [17].

Токен складається з трьох складових. Перша це header(навантаження), в ній міститься інформація про тип та алгоритм шифрування третьої частини. Друга частина це payload(корисне навантаження), вона містить поля які потрібно транспортувати: час дії, аудиторія, користувацькі поля і т.д. Третя називається signature(підпис). Вона отримується в результаті хешування об’єднання перших двох, алгоритмом описаним в навантаженні.

Такий підхід має декілька переваг в порівнянні з попередніми. Він є більш безпечним оскільки завдяки підпису валідність даних можна перевірити, є шифрування певним словом. В Django це SECRET_KEY. Для валідації не потрібно робити запит до бази даних. В випадку втрати даних або закінченні періоду дії його можна поновити завдяки refresh_token [18].

Орієнтуючись на вищепроведене дослідження, в системі буде використано останній спосіб підтримки сесії.

РОЗДІЛ 2
РОЗРОБКА СИСТЕМИ

2.1. Короткий опис ролей

На даний момент в системі існує три ролі, це студент, викладач та адміністратор. Основний сценарій використання системи - відмітка присутності на парі, яку здійснють студенти та підтверджує викладач. Адміністратор є одним для всієї системи, може бачити статистику їх відвідувань, переглядати списки користувачів, створювати, редагувати структуру університету: списки груп, факультетів та дані користувачів, рис 2.1.

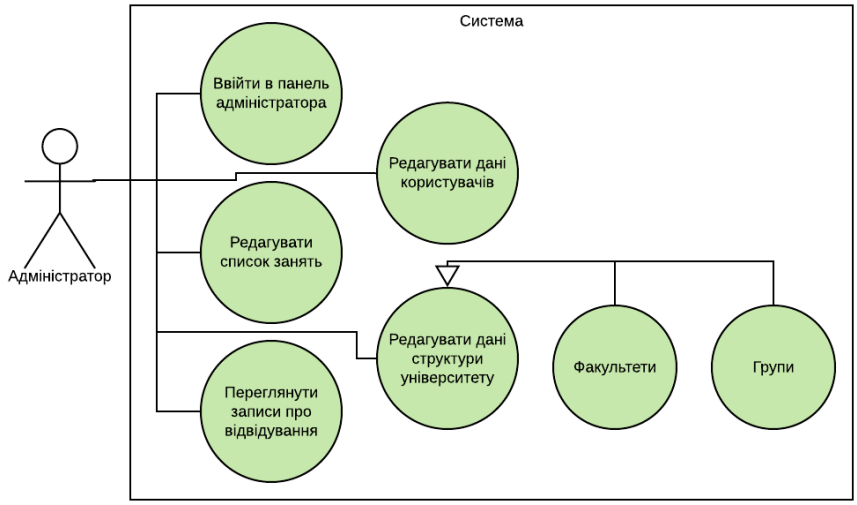

Рис. 2.1 Діаграма Use Case адміністратора

Більш детально ознайомившись з потребами майбутньої системи, було розглянуто ідею щодо створення ще однієї ролі – редактора.

Це один або декілька людей з факультету, які займаються створенням, оновленням розкладу, пар, зв’язуванням викладача, предмета і групп. Для створення розкладу редактор вибирає неділю тижня, А або Б, створює дні та за потреби додає пари, рис 2.2.

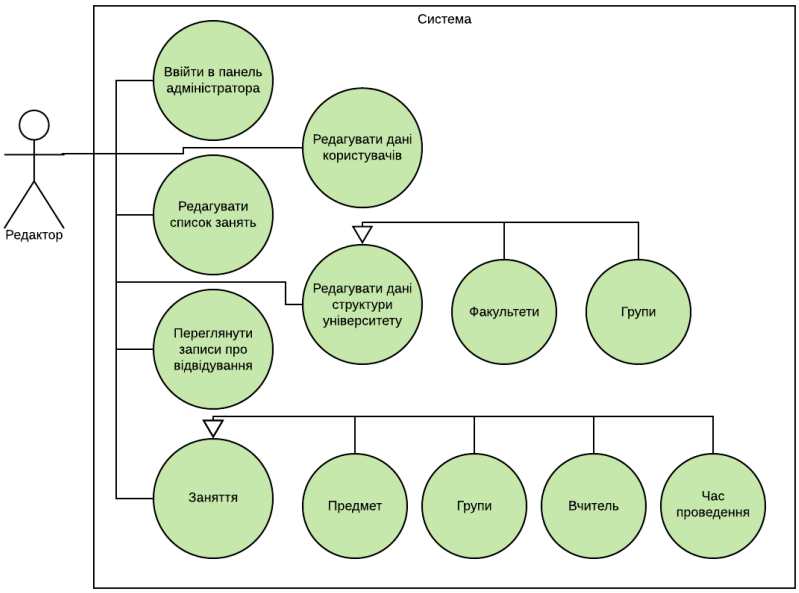

Рис. 2.2 Діаграма Use Case редактора

2.2. Проєктування бази даних

Цьому розділу була приділена значна увага. Якісна структура дозволяє зберігати інформацію цілісно, без надлишкових данных, що в свою чергу в майбутньому полегшить супровід та знизить загальне навантаження на систему.

Проєктування здійснюється в декілька етапів:

1. Виділення сутностей та їх основних полів.
2. Виділення відношень між таблицями.
3. Нормалізація даних, за необхідності.

На першому етапі відбувається виділення сутностей (об’єктів, які повинні зберігатися в таблицях бази даних) та полів (характеристик об’єктів) [19].

Для цього існує два підходи:

- Визхідний, грунтується на огляді інформації, наприклад, історії дій учасників системи та поступовому виділенні з неї важливих полів, які потім групуються в сутності.
- Низхідний є оберненим до попереднього. Він дозволяє побудувати базу даних значної за обсягом системи, загальні

моменти роботи якої можна визначити.

В роботі ми використовували їх обидва. В першому розділі роботи, було вияснено, що в системі в значній мірі будуть використовуватися дані про пару. Їх можна брати з зовнішнього сервісу через API. В такому випадку потрібно створювати часткову дублюючу структуру, виконувати періодичні запити для підтримання цілісності даних. Система буде значно залежити від зовнішнього інтерфейсу, буде мікросервісом.

Так як на момент розробки API сервісу з розкладом не існувало і потрібно було розв’язати вищеописані проблеми, ми прийняли рішення: зберігати дані там де вони використовуються, зробити систему монолітною. Для цього потрібно:

- Створити таблиці пар та зв`язаних елементів: групи, факультету, дня, тижня.
- Створити роль редактора для оновлення наведених атрибутів.

Отже існує декілька ролей: студент, викладач, адміністратор та редактор.

Так як вони мають деяку кількість відмінних полів, то була ідея створити одну таблицю для зберігання спільних даних та окремі таблиці притаманні кожному і зв’язувати їх відношенням один до одного. Однак, оскільки число атрибутів було невелике і це значно ускладнювало логіку, було прийнято рішення зберігати їх разом.

В результаті створені таблиці користувача (user), групи (studygroup), предмета (subject), пари (studyclass), факультету (faculty), дня (day), тижня (week), відмітки про присутність (mark) і з’єднуючі таблиці відповідальних за розклад (faculty_responsibles), допоміжна таблиця (studyclass_studygroup), рис. 2.3.

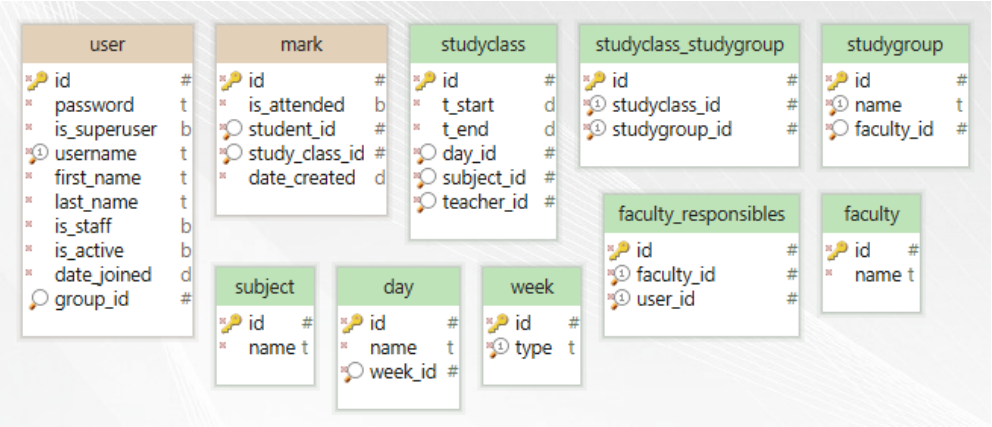

Рис. 2.3 Основні таблиці

В базі даних кожна таблиця має значення, але деякі з них задіяні більше: user, studyclass, mark.

В таблиці user є 6 основних атрибутів:

- id – первинний ключ
- username – email адреса
- password - пароль
- first_name – ім’я
- last_name – прізвище
- group_id – це поле є зовнішнім ключем, який необхідний для подальшої побудови зв’язку з таблицею studygroup.

Таблиця studyclass має поля початку і закінчення пари та зовнішні ключі на таблиці дня, предмета, викладача. Для зв’язку з групами існує окрема допоміжна таблиця studyclass_studygroup.

Таблиця mark має поля необхідні для обліку відвідувань та зовнішні ключі на таблиці студента та класу. Вона є єдиною в системі, яка має дату.

Другим етапом є виділення відношень між таблицями. Вони бувають трьох типів. В системі ми будемо використовувати два з них: один до багатьох та багато до багатьох.

В типі зв’язку один до багатьох один рядок із таблиці А (головна) співвідноситься з декількома рядками з таблиці В (залежна). Наприклад, користувач може мати одну групу, а вона в свою чергу складається з

багатьох користувачів. Більша частина системи побудована на цьому типі, таблиці(user-mark, week-day, subject-studyclass, і тд.)

В типі зв’язку багато до багатьох кожен рядок таблиці А співвідноситься з кожним рядком таблиці В. Прямий зв'язок між таблицями встановити не можна. Тому потрібно створити додаткову таблицю, яка буде залежною для об’єднуючих таблиць. Наприклад, на парі може бути декілька груп, а у кожної групи може бути декілька пар. В системі, за таким принципом побудовані зв’язки чотирьох таблиць (studyclass-studygroup, faculty-responsibles).

При зміні даних в звязних таблицях потрібно дотримуватись такого аспекту як посилальна цілістність. Вона потрібна для того, щоб недопустити виникнення аномалій, коли зміна даних однієї таблиці руйнує логіку іншої.

Вони є трьох видів: видалення, вставки та оновлення. Аномалія видалення виникає у випадку вилучення з основної таблиці рядка. При цьому зовнішній ключ продовжує посилатися на видалений рядок. Аномалія вставки відбувається при додаванні рядка в залежну таблицю. Зовнішній ключ при цьому не відповідає первинному ключу ні одному з рядків головної таблиці. Аномалія оновлення заключається в тому, що декілька рядків однієї таблиці містять однакові дані. При зміні даних в одному рядку відбувається протиріччя з даними іншого.

Для попередження виникнення аномалії видалення є два можливих сценарії:

- якщо рядок залежної таблиці обов’язково має посилатися на запис із головної, то для даних всановлюється каскадне видалення.
- якщо він допускає відсутність зв’язку, то при вилученні, зовнішній ключ починає вказувати на NULL. Це значення має бути дозволено в налаштуваннях стовпця.

Для запобігання появи аномалії вставки потрібно дозволити зовнішньому ключу вказувати на NULL в випадку, якщо немає вказівки на дані в головній таблиці.

Для вирішення проблеми оновлення застосовується нормалізація даних, яка буде розглянута наступним кроком.

Третім етапом є нормалізація даних. Це процес ділення даних по окремим таблицям. Він передбачає застосування нормальних форм. Нормальні форми – це певні правила проєктування, нині їх існує 7 штук, кожна наступна (окрім першої) має умову дотримання попередньої.

Вони дозволяють:

- позбавитися від надмірності - коли в базі значно дублюються данні в декількох таблицях.
- досягти цілісності – зміни в одній таблиці не мають приводити до значних змін в залежній від неї таблиці.

Чим більше їх можна вжити не порушуючи зручності користування, тим краще. Розглянемо спроєктовану вище базу. Вона знаходиться в перших трьох нормальних формах [20].

В першій нормальній формі, так як:

- Кожна таблиця немає повторюваних рядків
- Всі атрибути прості типи даних
- Всі значення визначені

В другій нормальній формі, так як:

- Таблиці знаходиться в першій нормальній формі
- Кожна має первинний ключ
- Неключові атрибути залежать від нього повністю, композитних первинних ключів в системі немає.

В третій нормальній формі, так як:

- Таблиці повинні приведені до другої нормальної форми
- Неключові атрибути залежать тільки від первинного ключа. Наприклад, таблиця studyclass має поля t_start, t_end, day_id,

subject_id, teacher_id. За значенням кожного з них не можна остаточно визначити значення іншого. Атрибути subject_id і teacher_id не є зв’язними, адже викладач може вести декілька предметів і предмету можуть навчати декілька різних викладачів.

Після завершення проєктування ми з’єднали утиліту з локальною базою даних MySql, таким чином структура була перенесена, рис 2.4.

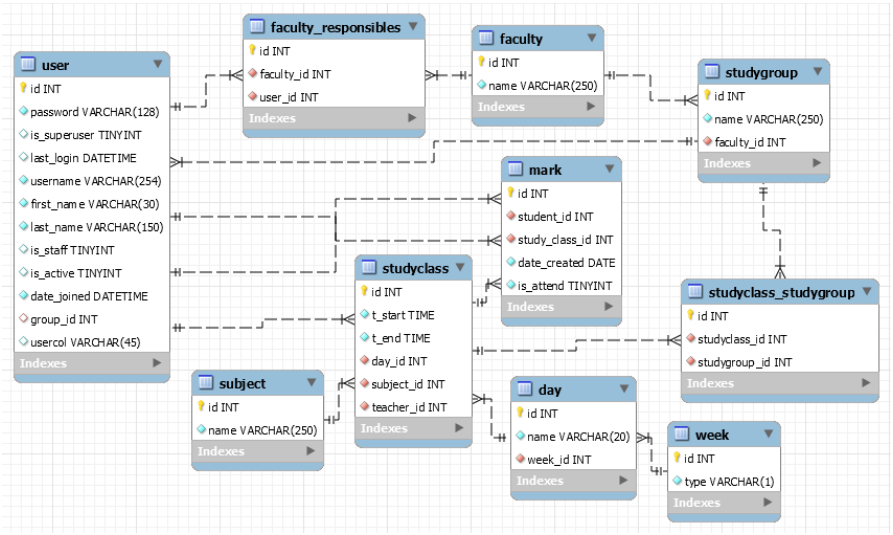

Рис. 2.4 ERD діаграма бази даних

2.3. Перенесення моделі бази даних в Django ORM

ORM (Object-relational mapping) – це прошарок фреймворку для зв’язку бази даних і серверної частини, написаної на Django. Він працює не залежно від платформи [21]. Для роботи з даними вони повинні бути описані всередині моделі. Їх можна буде перевірити, обробити, відобразити в панелі адміністратора, серіалізувати та передати як відповідь по API. Всередині структури проєкта ORM описується в файлах models.py.

Моделей, створених користувачем, може бути досить багато. Тому доцільно використати інструмент для генерації відповідних моделей та потім відредагувати отриманий результат. Для цього потрібно виконати етапи:

1. Встановити модуль зв’язку з базою даних MySql.

2. Вказати налаштування підключення в файлі settings.py.
3. Створити додаток, в якому буде знаходитися код моделі.
4. Виконати з’єднання та провести обернену міграцію.

На першому етапі потрібно встановити бібліотеку mysqlclient. При цьому можна скористатися стандартною інструкцією, в ній використовується команда pip і зовнішній репозиторій. Однак, встановлення таким способом не завжди працює. В нашому випадку довелося завантажувати компільований файл бібліотеки і встановлювали його локально.

На другому етапі потрібно замінити стандартні налаштування бази даних на відповідні для MySQL. При заміні важливо вказати налаштування sql_mode, charset. Перший необхідний для зміни режиму працювання бази на STRICT_TRANS_TABLES в випадку текстування [22]. Другий потрібен щоб задати кодировку, з якою буде працювати ORM та будуть зберігатися дані на сервері. Попередня версія MySQL, 5.5.3 використовувала кодування utf8mb3, що дозволяло зберігати строки в діапазоні до 171 символа (3 байта), потім його було замінено utf8mb4, який може зберігати більші діапазони [23]. Ці та інші налаштування ще зустрічаються на серверах. Тому встановлення конкретного кодування є необхідним.

На третьому етапі потрібно створити додаток всередині проєкта, в який потім буде перенесено код необхідних нам моделей.

На четвертому етапі використовується інструмент inspectdb для побудови моделей на основі існуючої бази [24]. Для цього виконується команда python manage.py inspectdb > models.py. В результаті формується файл з моделями. З нього ми виділяємо потрібні і переносимо в models.py додатку, створеного на попередньому етапі.

2.4. Редагування панелі адміністратора

Для функціонування даної програми важлива панель адміністратора. Завдяки їй відбувається взаємодія редактора і адміністратора з системою. Однією з найбільш об’ємних сторінок є User. Її описує клас UserAdmin. Він має атрибути:

- list_display описує колонки, які відображаються зверху сторінки: email, ім’я, прізвище, група аутентифікації(роль).
- list_filter містить кортеж полів для фільтрації в правому блоці сторінки: дата, назви ролей. Для дати використовується клас DateRangeFilter зовнішньої бібліотеки rangefilter.
- search_fields містить кортеж полів що доступні для пошуку: email, ім’я, прізвище.
- ordering вказує на поле, яке використовується для упорядкування, id.
- list_per_page позначає після якої кількості рядків повинна початися пагінація.

Після них написаний функціонал для отримання даних колонки групи аутентифікації. Так як користувач може мати одночасно декілька ролей, то в базі даних таблиці мають тип зв’язку багато до багатьох, який складно показати. Одним із способів є застосування команди '\n'.join([p.name for p in obj.groups.all()]). Він відображає інформацію, але робить запит до сховища для кожного рядка, що уповільнює відображення [25].

Альтернативою є використання методів get_queryset й prefetch_related. Ідея полягає в тому, щоб викликати перший метод у класа предка, objects. Тим самим отримати QuerySet даних всієї моделі. До якого застосувати другий, для того щоб з’єднати відношення та повернути. Після цього написати метод для отримання даних зв’язної таблиць та використати в якості поля list_display [26]. Для того щоб була

можливість їх сортувати методу вказується атрибут admin_order_field = 'groups'.

Отриманий код дозволяє значно зменшити витрати на звернення. Кількість запитів до - була 16, опісля – 6, рис 2.5.

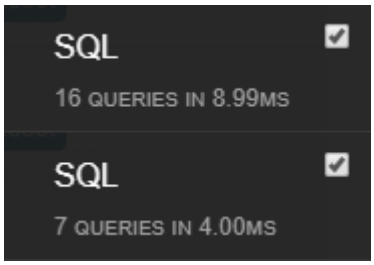

Рис. 2.5 get_form()

Нижче перевизначено метод get_form. Django йде в комплекті з системою аутентифікації. За замовченням вона включає: користувачів, групи та дозволи. Після створення модель отримує чотири види дозволів застосовуваних до всіх її полів: створити, видалити, оновити та переглянути. В такому випадку можлива ситуація в якій користувач має можливість оновлювати поля інших та може змінити власні і стати адміністратором. Для того, щоб цьому запобігти потрібно вимкнути відображення деяких полів, рис. 2.6.

```
def get_form(self, request, obj=None, **kwargs):
    form = super().get_form(request, obj, **kwargs)
    is_superuser = request.user.is_superuser
    if not is_superuser:
        disabled_fields = (
            'is_superuser',
            'user_permissions',
            'groups'
        )
        for f in disabled_fields:
            form.base_fields[f].disabled = True
    return form
```

Рис. 2.6 get_form()

Окрім безпосередньої зміни моделі, зміну даних можна зробити опосередковано через спеціальні дії на сторінці управління [27]. За замовчення такі права має кожен хто має доступ в відповідної сторінки.

Дії можуть бути різними, наприклад зміна статусу, відмітка присутності списку людей. Для ілюстрації можна взяти приклад видалення декількох користувачів, рис 2.7.

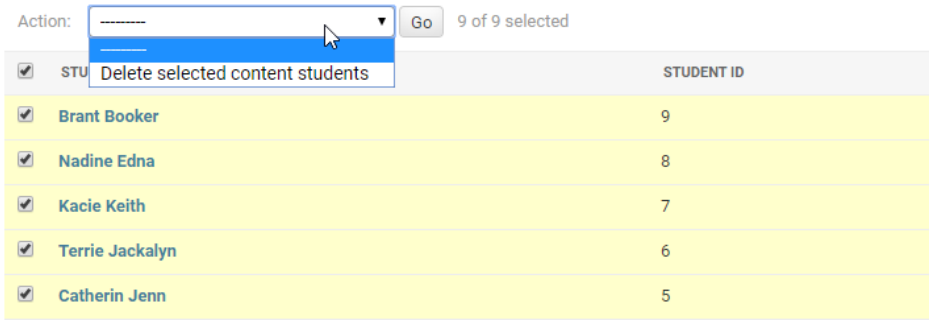

Рис. 2.7 Панель адміністратора

Для того щоб цьому запобігти в функцію яка виконує дію додається перевірка на рівень доступу, рис 2.8.

```
13      def get_actions(self, request):
14          actions = super().get_actions(request)
15          if not request.user.has_perm('auth.change_user'):
16              del actions['delete_users']
17          return actions
```

Рис. 2.8 Функція get_actions()

Загалом управління користувачами і дозволами має свої нюанси але Django дозволяє їх враховувати.

2.5. Підтримка сесії

В Django Rest Framework за замовченням не має JWT автентифікації. Для його роботи потрібно встановити зовнішню бібліотеку. Їх існує декілька pyjwt, simplejwt, djoser. Ми обрали другу, так як вона розвивається і дозволяє змінювати параметри дії токену. Після її встановлення було зроблено декілька кроків:

1. В файл налаштувань проєкту, словнику REST_FRAMEWORK змінено значення ключа, пов’язаного з автентифікацію та потрібне для роботи JWT, 'DEFAULT_AUTHENTICATION_CLASSES': ['rest_framework_simplejwt.authentication.JWTAuthentication',]

2. В тому ж файлі додано словник SIMPLE_JWT з двома парами, що зберігають час життя токенів, 'ACCESS_TOKEN_LIFETIME': timedelta (minutes=5) та 'REFRESH_TOKEN_LIFETIME': timedelta (days=3),
3. Створено додаток api, а в ньому - файл urls.py.
4. В urls.py імпортовано представлення із пакету simplejwt. Під ними в urlpatterns описані шляхи: авторизації (вказує на клас TokenObtainPairView) та оновлення (вказує на клас TokenRefreshView).

Основу безпеки JWT складає два токени access та refresh. Перший використовується для отримання доступу до ресурсу. Він багаторазовий, його можна використовувати недовго, за замовченням п’ять хвилин. Другий потрібен для запиту нової пари токенів. На відміну від попереднього він одноразовий, але має значно довший термін життя, за замовченням один день.

Користувач здійснює вхід в додаток за допомогою логіна і пароля (авторизація). Вони відправляються на сервер, який у відповідь повертає пару токенів. Додаток їх зберігає і при кожному наступному запиті відправляє access токен в заголовку. Після закінчення дії access токену застосовується refresh для (оновлення) і отримується нова пара.

У випадку, якщо хтось отримав обидва токени і не скористався refresh токеном, то після закінчення дії вхід буде заборонено. Якщо він скористався, то ті записи що у нас перестають працювати. Потім ми заходимо в додаток й авторизуємся. При цьому іншій людині вхід буде заборонено. Для виконання цієї умови необхідно щоб refresh токен зберігався в базі даних на сервері і видалявся після використання. В simplejwt для цього існує модель OutstandingingToken. Часто в проєктах до токену також додається суміш даних по певному алгоритму.

Таким чином access і refresh токени допомагають підтримувати сесію та захищають користувача.

2.6. Робота з API

Робота API залежить від трьох компонентів:

- Серіалізатор виконує зв'язок з базою даних через модель. Він дозволяє робити перетворення між нею і форматом json, який розуміє клієнт.
- Вид містить функції за допомогою яких можна взаємодіяти з серіалізатором, виконувати операції читання, запису, оновлення і видалення з бази.
- Маршрутизатор має список ідентифікаторів, URL. По ним доступна взаємодія з API.

Розглянемо принцип роботи реєстрації студента та вчителя.

Першим етапом було створення серіалізатору. Для цього в папці додатку authentication було створено файл serializers.py. В ньому створено клас CreateUserSerializer, який ми наслідуємо від ModelSerializer. Всередині описані поля, які варто перевіряти та написаний класс Meta. Він вказує з якою моделлю ми працюємо та поля що мають бути взяті, рис. 2.9.

```
class CreateUserSerializer(serializers.ModelSerializer):

    email = serializers.EmailField(source='username')
    group_id = serializers.IntegerField(required=False)
    role = serializers.CharField()

    class Meta:
        model = User

        fields = ['email', 'first_name', 'last_name', 'password', 'group_id', 'role']
```

Рис. 2.9 Клас CreateUserSerializer

Другим етапом було створення виду в файлі views.py. В ньому написаний клас CreateUserAPIView, який наслідується від існуючого CreateAPIView. Всередині ми створюємо змінну serializer_class, на створений раніше серіалайзер та переопріділяємо метод create. Спочатку в ній перевіряється валідність типу отриманих даних, потім отримується список ролей які є в системі. Після цього в блоці try звіряється роль, яка

була передана користувачем і дані користувача зберігаються в змінній user. Якщо це student, то виконується перевірка по наявності ідентифікатора групи, рис. 2.10.

```
def create(self, request):
    serializer = self.serializer_class(data=request.data)
    serializer.is_valid(raise_exception=True)
    data = serializer.validated_data

    role = Group.objects.filter(name=data.pop('role').lower()).last()

    r_status = status.HTTP_200_OK
    detail = {'detail': 'success'}

    try:
        if role.name.lower() == 'student':
            group = general_models.StudyGroup.objects.get(id=data.pop('group_id'))
            user = User.objects.create_user(**data)
            group.students.add(user)
        else:
            user = User.objects.create_user(**data)
    except IntegrityError:
        r_status = status.HTTP_400_BAD_REQUEST
        detail = {'email': ['Is allready']}
    except ObjectDoesNotExist:
        r_status = status.HTTP_400_BAD_REQUEST
        detail = {'group_id': ['Is not correct']}
```

Рис. 2.10 Метод create, початок

Якщо все добре, то користувач заноситься в базу даних, створюється унікальний код, формується посилання для підтвердження і ініціюється відправка на email, функцією send_mail. В кінці цього процесу користувачу повертається статус код (r_status) і повідомлення (detail), рис. 2.11.

```
user.groups.add(role)
user.is_active = False
user.save()

token = uuid.uuid4().hex.replace('-', '')
ConfirmationToken.objects.create(user=user, token=token)
link = VERIFY_URL + '?key={0}'.format(token)

send_mail(
    'Email verification',
    'Будь-ласка, підтвердьте вашу електронну пошту. Для цього, перейдіть за посиланням {0}'.format(link),
    'postmaster@sandbox6f392605fbf14a02932e5ca1dfe4fe39.mailgun.org',
    [user.username]
)

return response.Response(data=detail, status=r_status)
```

Рис. 2.11 Метод create, кінець

Третім етапом є додавання посилання на описане view в файл urls.py, додатку api.

Четвертим етапом було проведено перевірку через програму Postman, для цього вказаний URL і введені поля в тілі запиту, рис. 2.12.

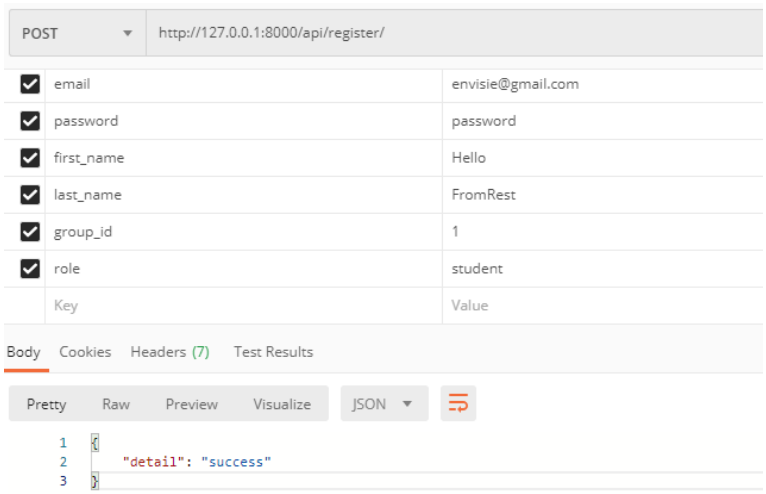

Рис. 2.12 Відправка запиту через програму Postman

Опісля користувач отримує повідомлення на email в якому є посилання для підтвердження реєстрації. Перехід по ньому викликає функцію email_verification. Вона шукає профіль в зв'язній таблиці user, по токену з URL. В випадку знаходження активує його.

Уваги заслуговую представлення ListStudyClassAPIView. Він потрібен для того, щоб повернути користувачу інформацію про його заняття на найближчі декілька днів: день, тиждень, предмет, вчитель, час початку та закінчення.

Визначення дня проходить в декілька кроків. Для підрахунку дат використовується бібліотека datetime. Спочатку вказаний масив day(день) що містить 8 елементів, кожен з яких відповідає певному дню тижня. Останній відноситься до понеділка наступного.

Потім в змінні c_day(сьогодні), p_day(вчора), n_day(завтра), записуються індекси відповідних днів. Далі написана логіка визначення тижнів (А або Б). Вони чергуються й тому індекс теперішнього дня буде дорівнювати залишку від ділення на два. Тиждень що був учора і буде завтра визначається по восьмому дню. Якщо минулий день з індексом нуль або наступний день з індексом вісім, то відповідні p_week, n_week будуть дорівнювати нулю(А) інакше одному(Б).

Потім в залежності від ролі(student, teacher) створюється змінна серіалізатору та моделі. Якщо це студент, то MarkStudyClassSerializer, таблиця user.group. Всередині нього описано поле mark, клас Meta з полями моделі StudyClass(заняття) та метод get_mark, який повертає id відмітки, відмітку й створює запис в випадку його відсутності. Якщо це викладач, то GroupStudyClassSerializer, таблиця user. Всередині він написаний схоже, однак в методі get_mark він викликає MarkGroupSerializer для кожного студента в групі.

Наприкінці дані об’єднуються в структуру з декількох рівнів і повертаються клієнту.

Інші API побудовані схожим чином.

2.7. Логування змін

Логування змін полягає в записі історії подій системи. Завдяки чому можна вирішувати неполадки та знаходити потенційні місця покращення роботи компонентів.

В Django за ці дії відповідає модуль logging. Його будова складається з п’яти частин. Вона зберігаються в однойменному словнику, файлу settings.py проєкту.

Поле 'version': 1 вказує на те, що словник є конфігурацією формата dictConfig. На разі вона єдина. Атрибут 'disabled_existing_loggers' має за замовченням значення True [28]. Коли воно таке, то існуючі логери Django буде вимкнуто. Для того, щоб цього не допустити встановлено значення False.

Logger є точкою зв’язку бібліотеки з фреймворком. Через нього можна виконувати запити для збереження записів. Має певний рівень важливості (log level). Якщо він більший або дорівнює значенню атрибута повідомлення, то запит передається далі. У нас є три логера: 'django', 'django.request' та 'general_admin'. Останній має три хендлери: 'console', 'file', 'mail'. В ньому є параметри 'level': INFO та propagate: False. Вказані

через точку шляхи створюють ієрархію. Так request відноситься до django. Це важливо, адже дані від приємника можуть передаватися до предка, в такому випадку його хендлер отримає всі запити. За це відповідає атрибут propagate.

Handler це обробник. Він визначає, що потрібно зробити з повідомленнями. Має таке ж обмеження що і попередній. Logger може мати їх декілька. Таким чином існують різні форми нотифікації в залежності від допуску. Наприклад, в нас є хедлери 'console', 'file', 'mail'. В першого обмеження 'INFO', в другого 'WARNING', третього 'ERROR'. До них прийшли дані з спільного логера 'general_admin' з рівнем 'WARNING'. Результатом буде вивід на консоль та збереження в файл.

Filter використовується для того, щоб додатково визначити або змінити ті записи, які передаються. Наприклад, можна написати фільтр, який в певних випадках буде змінювати WARNING на INFO чи приймати помилки лише з конкретного ресурсу. Його можна встановити на двох описаних вище компонентах. В нас він використовується в хендлерах 'console' та 'mail' для того, щоб перевірити чи ввімкнений режим відладки.

Formatter це модуль, який необхідний для того, щоб представити об’єкти записів (Log Record) в вигляді рядків. В ньому вказуються атрибути, які будуть конвертуватися та стиль. Його можна перевизначити та додати іншу логіку. В проєкті в ньому описані short та descriptive шаблони.

Потім можна розмістити виклик логеру в потрібних місцях проєкту. В нашому випадку це admin.py, додатку general. Запит warning визивається після збереження даних моделі Mark, рис 2.13.

```
# method for updating
def save_model(self, request, obj, form, change):
    super(MarkAdmin, self)\
        .save_model(request, obj, form, change)
    logger.warning('message: "the mark with id: {0}, '
                   'has been updated."'.format(obj.id))
```

Рис. 2.13 Метод save_model

2.8. Реалізація відправки повідомлень на email

Відправка повідомлень використовується в сервісі при реєстрації, зміні пароля студента та викладача на випадок якщо користувач забуде свій пароль.

Спочатку була зроблена локальна відправка з зберіганням повідомлень в папку sent_emails [29]. Для цього в файлі налаштувань проєкту була підключена бібліотека та додаток який відповідає за відправлення.

Для того щоб відправка відбулася необхідно щоб email користувача було зареєстровано в системі. Якщо це так, то після POST запиту по API на сервері створиться папка sent_emails і в ній з’явиться повідомлення в форматі .txt. Воно містить інформацію в залежності від викликаємої функції, якщо це реєстрації, це url адрес підтвердження.

Після перевірки роботи в локальній мережі, ми зробили відправку повідомлень на реальну email адресу.

Для того щоб це реалізувати повинен бути сервер на якому буде працювати SMTP протокол.

Сервер з SMTP протоколом має:

- Бути постійно ввімкненим для того щоб отримувати листи в чергу для відправки і відправляти [30].
- Мати систему аутентифікацікації
- Мати виділений адрес, оскільки сервер отримувача перевіряє адреса з яких приходять повідомлення

Побудова такого серверу досить складна, тому було прийнято рішення використати сторонній ресурс.

В якості сервісу відправки повідомлень був вибраний Mailgun, на ньому вже вирішені вищеперелічені потреби, також він має відкритий порт TLC для захисту повідомлення на транспортному рівні.

Після реєстрації та аутентифікації сервіс дає дані для доступу по API: hostname, username, password. Ці данні зберігаються в змінних середовища і використовуються в файлі settings.py проєкту.

Таким чином користувач має можливість отримати повідомлення на електронну пошту, рис. 2.14.

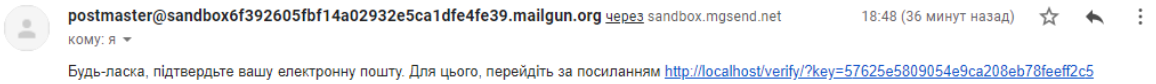

Рис. 2.14 Повідомлення при реєстрації

2.9. Розгортання проєкту в мережі інтернет

Для того щоб розроблений програмний продукт працював на сервері необхідно перенести: базу даних MySQL, серверну частину Django.

Для перенесення бази даних зроблені кроки:

1. На сервері Heroku було створено застосунок до якого було підключено додаток ClearDB MySQL.
2. Після підключення на сервері створюється панель управління базою, а також в налаштуваннях пишеться url адрес для зв’язку з нею.
3. В цьому адресу є дані аутентифікації, які потрібно розділити на компоненти, рис. 2.15.

```
1  mysql://
2  bd339b5fb2d6fd
3  :
4  ea6df310
5  @
6  eu-cdbr-west-02.cleardb.net  //hostname
7  /heroku_1da69ea70a47e37?reconnect=true
8
9
```

Рис. 2.15 Компоненти url адреси для звязку з базою

Другий рядок – ім’я користувача

Четвертий – пароль

Шостий – назва серверу для підключення

Сьомий – після знаку ‘/’ і до знаку ‘?’ написана назва бази даних

В url адресі не вказано порт для доступу, за замовченням він 3306

4. Перевірено підключення в MySQL Workbench.
5. В розробленому додатку дані підключення до серверу переносяться в змінні середовища. Вони використовуютьсь в файлі конфігурацій, settings.py. Такий крок необхідний щоб можна було вести подальшу розробку з локальною базою даних і при необхідності переключитись на ту, яка зберігається на сервері.

Для перенесення серверної частини зроблені наступні кроки:

1. В віртуальному середовищі проєкту встановлено пакет gunicorn. Він необхідний для робити з WSGI сервером в Unix системі heroku.
2. Встановлено whitenoise. Цей пакет потрібний для того, щоб фреймворк керував власними статичними файлами.
3. В папці проєкту було створено Procfile та файл requirements.txt. В Procfile знаходиться рядок, рис 2.16. Він дає команду серверу на запис даних статистики. Це потрібно, щоб при підключені бачити помилки в роботі якщо вони з’являться. Виконується один раз при завантаженні на сервер.

```
1  web: gunicorn daybook.wsgi --log-file -
```

Рис. 2.16 Вміст Procfile

4. Завантажено та встановлено heroku-cli, це консольний інструмент необхідний для роботи з сервером heroku по технології ssh.
5. В папці проєкту проініціалізовано систему контроля версій git.
6. На сайті компанії Bitbucket створено приватний репозиторій та завантажено проєкт.
7. Через heroku-cli створено додаток на сервері Heroku.
8. Зв’язано хостинг та віддалений репозиторій.

9. Почато розгортання командою в heroku-cli, сервер налаштував WSGI та встановив необхідні пакети із раніше створеного requirements.txt, дії відображаються в консолі.
10. Записано змінні середовища в панель управління проєктом.
11. В консолі введено команду для створення структури бази даних.
12. Перевірено роботу додатку.

ВИСНОВКИ

В ході бакалаврської роботи була розроблена серверна частина сервісу моніторингу відвідувань. Для цього були виконані наступні завдання:

1. Досліджено засоби створення серверної частини веб-додатків: мови програмування, проєктування бази даних, API, RESTful архітектура, підтримка сесії. Було визначено що для роботи системи буде доцільно застосувати Django, MySQL, JWT.
2. Розроблено систему ролей. Вона включає чотири ролі: студент, викладач, адміністратор і редактор. Для них визначені дозволи та середовище. Дві з них можуть працювати з системою через API, інші використовуючи панель адміністратора.
3. Спроєктовано базу даних для зберігання інформації про відмітки та дисципліни. Вона є нормалізованою, цілісною, без надмірності. За потреби її можна розширити для зберігання більшого числа даних.
4. Розроблено API для доступу до серверної частини. Для розробки використовувалися критерії описані в підході REST та Django Rest Framework. Правильність роботи API була перевірена через програму Postman.
5. Опублікувано отриманий проєкт в мережі інтернет. Розгортання було проведено на серверах Heroku. Вирішені проблеми з переносом бази даних та взаємодією додатку і сервера.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Preventable Failure [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: https://www.attendanceworks.org/wp-content/uploads/2017/09/Preventable-Failure-CCSR-April-2014.pdf.

2. Структурні підрозділи вищого навчального закладу [Електронний ресурс] – Режим доступу до ресурсу: https://pidruchniki.com/86521/menedzhment/strukturni_pidrozdili_vischogo_navchalnogo_zakladu.

3. BioTime Edu [Електронний ресурс] – Режим доступу до ресурсу: https://www.biotime.ru/advantages/.

4. TimeTac Time and Attendance System for All Industries [Електронний ресурс] – Режим доступу до ресурсу: https://www.timetac.com/en/time-attendance-system/.

5. Add Classroom Owners, Teachers, and Helpers [Електронний ресурс] – Режим доступу до ресурсу: https://help.goguardian.com/hc/en-us/articles/115005712763-Add-Classroom-Owners-Teachers-and-Helpers.

6. Which backend language to learn in 2020? [Електронний ресурс] – Режим доступу до ресурсу: https://mxx.news/backend-language-to-learn-in-2020-comparing-community-job-market-cloud-support-and-performance/

7. Why Java is the most preferred programming language for building end-to-end enterprise solutions? [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/webbasedevelopment/why-java-is-the-most-preferred-programming-language-for-building-end-to-end-enterprise-solutions-731027246f2a.

8. Основні відомості про бази даних [Електронний ресурс] – Режим доступу до ресурсу: https://support.office.com/uk-ua/article/%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%96-%D0%B2%D1%96%D0%B4%D0%BE%D0%BC%D0%BE%D1%81%D1%82%D1%96-%D0%BF%D1%80%D0%BE-%D0%B1%D0%B0%D0%B7%D0%B8-%D0%B4

%D0%B0%D0%BD%D0%B8%D1%85-a849ac16-07c7-4a31-9948-3c8c94a7c204.

9. Difference between SQL and NoSQL [Електронний ресурс] - https://www.geeksforgeeks.org/difference-between-sql-and-nosql/

10. MySQL 8.0 Reference Manual [Електронний ресурс] – Режим доступу до ресурсу: https://dev.mysql.com/doc/refman/8.0/en/sha256-pluggable-authentication.html.

11. ValInfo [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/9.0/wal-intro.html.

12. MySQL vs PostgreSQL -- Choose the Right Database for Your Project [Електронний ресурс] – Режим доступу до ресурсу: https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres.

13. REST Architectural Constraints [Електронний ресурс] – Режим доступу до ресурсу: https://restfulapi.net/rest-architectural-constraints/#uniform-interface.

14. Session vs Token Based Authentication [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@sherryhsu/session-vs-token-based-authentication-11a6c5ac45e4.

15. How does server-based authentication affect scalability? [Електронний ресурс] – Режим доступу до ресурсу: https://www.quora.com/How-does-server-based-authentication-affect-scalability.

16. Vincent W. S. Django for APIs: Build web APIs with Python & Django / William S. Vincent., 2018.

17. JSON Web Token [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/JSON_Web_Token.

18. Зачем нужен Refresh Token, если есть Access Token? [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/en/company/voximplant/blog/323160/.

19. Основи проєктування баз даних [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sql/tutorial/1.1.php.

20. Нормалізація баз даних [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F_%D0%B1%D0%B0%D0%B7_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85.

21. Отримуємо максимум від Django ORM [Електронний ресурс] – Режим доступу до ресурсу: https://codeguida.com/post/1952.

22. 5.1.10 Server SQL Modes [Електронний ресурс] – Режим доступу до ресурсу: https://dev.mysql.com/doc/refman/5.7/en/sql-mode.html.

23. 10.9.2 The utf8mb3 Character Set (3-Byte UTF-8 Unicode Encoding) [Електронний ресурс] – Режим доступу до ресурсу: https://dev.mysql.com/doc/refman/8.0/en/charset-unicode-utf8mb3.html.

24. inspectdb [Електронний ресурс] – Режим доступу до ресурсу: https://docs.djangoproject.com/en/3.0/ref/django-admin/#django-admin-inspectdb.

25. many-to-many in list display django [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/questions/18108521/many-to-many-in-list-display-django.

26. get-list-display-in-django-admin-to-display-the-many-end-of-a-many-to-one-rela [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/questions/38827608/get-list-display-in-django-admin-to-display-the-many-end-of-a-many-to-one-rela.

27. Logging [Електронний ресурс] – Режим доступу до ресурсу: https://docs.djangoproject.com/en/3.0/topics/logging/.

28. What You Need to Know to Manage Users in Django Admin [Електронний ресурс] - https://realpython.com/manage-users-in-django-admin/

29. Sending Email [Електронний ресурс] - https://docs.djangoproject.com/en/3.0/topics/email/

30. email - Why do I need an SMTP server? [Електронний ресурс] - https://superuser.com/questions/1006079/why-do-i-need-an-smtp-server

ДОДАТКИ

Додаток А

КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ
ЗДОБУВАЧА ВИЩОЇ ОСВІТИХЕРСОНСЬКОГО
ДЕРЖАВНОГО УНІВЕРСИТЕТУ

Я, Шишлов Владислав Володимирович,
учасник(ця) освітнього процесу Херсонського державного університету, **УСВІДОМЛЮЮ**, що академічна доброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

ЗАЯВЛЯЮ, що у своїй освітній і науковій діяльності **ЗОБОВ'ЯЗУЮСЯ**:
– дотримуватися:
- вимог законодавства України та внутрішніх нормативних документів університету, зокрема Статуту Університету;
- принципів та правил академічної доброчесності;
- нульової толерантності до академічного плагіату;
- моральних норм та правил етичної поведінки;
- толерантного ставлення до інших;
- дотримуватися високого рівня культури спілкування;

– надавати згоду на:
- безпосередню перевірку курсових, кваліфікаційних робіт тощо на ознаки наявності академічного плагіату за допомогою спеціалізованих програмних продуктів;
- оброблення, збереження й розміщення кваліфікаційних робіт у відкритому доступі в інституційному репозитарії;
- використання робіт для перевірки на ознаки наявності академічного плагіату в інших роботах виключно з метою виявлення можливих ознак академічного плагіату;

– самостійно виконувати навчальні завдання, завдання поточного й підсумкового контролю результатів навчання;
– надавати достовірну інформацію щодо результатів власної навчальної (наукової, творчої) діяльності, використаних методик досліджень та джерел інформації;
– не використовувати результати досліджень інших авторів без використання покликань на їхню роботу;
– своєю діяльністю сприяти збереженню та примноженню традиційуніверситету, формуванню його позитивного іміджу;
– не чинити правопорушень і не сприяти їхньому скоєнню іншими особами;
– підтримувати атмосферу довіри, взаємної відповідальності та співпраці в освітньому середовищі;
– поважати честь, гідність та особисту недоторканність особи, незважаючи на її стать, вік, матеріальний стан, соціальне становище, расову належність, релігійні йполітичні переконання;
– не дискримінувати людей на підставі академічного статусу, а також за національною, расовою, статевою чи іншою належністю;
– відповідально ставитися до своїх обов'язків, вчасно та сумлінно виконувати необхідні навчальні та науково-дослідницькі завдання;
– запобігати виникненню у своїй діяльності конфлікту інтересів, зокрема не використовувати службових і родинних зв'язків з метою отримання нечесної переваги в навчальній, науковій і трудовій діяльності;
– не брати участів будь-якій діяльності, пов'язаній із обманом, нечесністю, списуванням, фабрикацією;
– не підроблювати документи;
– не поширювати неправдиву та компрометуючуінформацію про інших здобувачів вищої освіти, викладачів і співробітників;
– не отримувати і не пропонувати винагород за несправедливе отримання будь-яких переваг абоздійснення впливу на зміну отриманої академічної оцінки;
– не залякувати й не проявляти агресії та насильства проти інших, сексуальнідомагання;
– не завдавати шкоди матеріальним цінностям, матеріально-технічній базіуніверситету та особистій власності інших студентів та/або працівників;
– не використовувати без дозволу ректорату (деканату) символіки університету в заходах, не пов'язаних з діяльністюуніверситету;
– не здійснювати і не заохочувати будь-яких спроб, спрямованих на те, щоб за допомогою нечесних і негідних методів досягати власних корисних цілей;
– не завдавати загрози власному здоров'ю або безпеці іншим студентам та/або працівникам.

УСВІДОМЛЮЮ, що відповідно до чинного законодавства у разі недотримання Кодексу академічної доброчесностібуду нести академічну та/або інші види відповідальностей до мене можуть бути застосовані заходи дисциплінарного характеру за порушення принципів академічної доброчесності.

20.04.20 (дата) _____ (підпис) Шишлов Владислав (ім'я, прізвище)