**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра комп'ютерної інженерії

До захисту допущено: «На правах рукопису»
Завідувач кафедри _____ Юрій Бойко
« _ » _____ 2023 р.

# КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

# «СИСТЕМА МОНІТОРИНГУ ТА ОБЛІКУ КОРИСТУВАЧІВ БЕЗДРОТОВИХ МЕРЕЖ НА БАЗІ КОНТРОЛЕРУ UNIFI»

**Виконав:**
студент 4-го курсу бакалаврату
денної форми навчання
спеціальності 123 Комп'ютерна інженерія
ОНП «_____»
Світлана Діденко _____

**Науковий керівник**:
кандидат технічних наук, асистент
Віталій Мар'яновський _____

**Рецензент:**
_____

Засвідчую, що у цій бакалаврській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____

Робота допущена до захисту в ЕК рішенням кафедри _____
від «__»_____ 2023 р., протокол № __.

Завідувач кафедри _____,
кандидат фізико-математичних наук, доцент
Бойко Юрій Володимирович

(підпис)

Київ – 2023

## РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра містить 60 сторінок, 17 ілюстрацій, 15 лістингів коду, 12 джерел посилань, 3 додатки.

Робота складається з трьох частин: теоретичної, практичної та демонстраційної.

У теоретичній частині розглядаються основні принципи роботи технології Wi-Fi та точок доступу. Розглядається метод REST API та його використання з метою збору інформації про під’єднаних користувачів, база даних MySQL з метою зберігання інформації для подальшої обробки, фреймворк Django, використаний для створення серверного застосунку та огляд існуючого рішення. У практичній частині описано процес створення серверної частини системи моніторингу користувачів та веб-інтерфейсу застосунку. У демонстраційній частині наведено результати роботи та порівняння з існуючим рішенням.

Ключові слова: Wi-Fi, WAP, Network Controller, REST API, MySQL, Django, Unifi, web-interface, clients, monitoring.

# ЗМІСТ

# ВСТУП

Процес розширення та вдосконалення внутрішньої мережі університету значно ускладнили забезпечення коректної роботи всіх засобів надання користувачам безперебійного доступу до мережі. Збільшення кількості пристроїв та мереж неминуче призводить до несправностей у роботі мережевого обладнання, що негативно відображається на ефективності роботи всіх підрозділів університету. Якщо з певної причини мережа перестає працювати й необхідні дані не можуть бути передані, обслуговування клієнтів під час простою припиняється. Це може створити труднощі та невдоволення клієнтів або навіть серйозні проблеми для всього підрозділу.

Інженерні організації зазвичай покладаються на окремі системи для моніторингу зовнішніх інтерфейсів своїх веб-сайтів, коду серверної програми, журналів і базової IT-інфраструктури. Ця залежність від інструментів точкових рішень ускладнює кореляцію між системами та визначення джерела будь-яких проблем, з якими стикається користувач. Крім того, більшість існуючих рішень перевантажені функціональними засобами, які рідко використовуються, що ускладнює та уповільнює роботу системних адміністраторів.

Одним із найефективніших засобів запобігання та своєчасного виявлення неправильної роботи мережевого обладнання може бути імплементація більш гнучкої системи моніторингу мереж та роботи Wi-Fi для використання в мережевому секторі інформаційно-обчислювального центру. Таким рішенням може стати серверний застосунок із веб-інтерфейсом, який дозволяє централізовано слідкувати за статистикою користувачів бездротових мереж факультетів. Застосунок має бути простим у розгортанні на сервері, мати інтуїтивно зрозумілий та простий інтерфейс, а також дозволяти доповнювати готове рішення необхідними інструментами в майбутньому, тобто, бути цілком автономним із погляду залежності від застосунків компаній, які представляють готові рішення.

# МЕТА РОБОТИ

Метою даної роботи є розробка та впровадження системи для забезпечення моніторингу та обліку користувачів, реалізація сповіщень про несправності в роботі мережі до створеного для даних цілей боту в Telegram, задля своєчасного реагування на проблеми та їх оперативного усунення. Застосунок повинен бути адаптованим під будь-яку організацію, яка використовує у якості готового рішення мережевий контролер Unifi. Основною ціллю роботи є забезпечення системи зручним веб-інтерфейсом та створення умов для гнучкого керування та оптимізації серверної частини програми залежно від потреб у подальшій розробці.

# 1 ТЕОРЕТИЧНІ ВІДОМОСТІ

## 1.1 Технологія Wi-Fi

Wi-Fi являє собою бездротову мережеву технологію, за допомогою якої пристрої, які підтримують цю технологію, можуть отримувати доступ до Інтернету. Це дозволяє підключеним пристроям обмінюватись інформацією між собою, утворюючи власну мережу. Підключення до Інтернету відбувається через бездротовий маршрутизатор або точку доступу [1].

Принцип роботи Wi-Fi полягає в розбитті радіонесучої хвилі на різні діапазони й розбитті кожного діапазону на кілька каналів. У цих незалежних діапазонах, 2.4 та 5 ГГц є найвідомішими діапазонами, у яких радіосигнал транслюється бездротовими мережевими пристроями. Для підключення до мережі, пристрою користувача необхідно знати ідентифікатор мережі, так клієнт може з’ясувати, чи можливе підключення до певної точки доступу. При потраплянні в зону дії двох точок доступа з ідентичними ідентифікаторами пристрій може вибирати між ними на основі даних про рівень сигналу.

## 1.2 Точки доступу

Точка доступу являє собою мережеве обладнання для підключення пристроїв, які мають доступ Wi-Fi до дротової мережі. Для того, щоб забезпечити приміщення середнього розміру доступом до Wi-Fi мережі, достатньо мати лише один бездротовий маршрутизатор із вбудованою точкою доступу. Якщо ж мова йде про обладнання великого приміщення, то доцільно використовувати додаткові точки доступу, які зведуть потребу в кабелях до мінімуму. Точки доступу підсилюють сигнал Wi-Fi, тому пристрій користувача може знаходитись далеко від маршрутизатора, але все одно мати доступ до мережі.


Рис.1.1 Сценарій підключення до мережі точки доступу Wi-Fi [2]

У даній роботі користувачі підключаються безпосередньо до точок доступу, з яких у свою чергу отримується більш детальна інформація про кожного користувача для подальшої обробки.

Сама по собі точка доступу являє собою бездротовий «подовжувач» мережі та не здатна забезпечити мережу DHCP-сервером для автоматичного розподілу IP-адрес, авторизувати користувачів при підключенні до мережі, контролювати та перерозподіляти пропускну здатність підключень. Для вирішення проблеми відсутності централізованого керування такими автономними системами існує рішення у вигляді контролеру безротової Wi-Fi мережі.

## 1.3 REST API

API (Application Programming Interface) — це набір інструкцій, які визначають як програмні системи можуть підключатись та взаємодіяти між собою. Наприклад, програма для моніторингу підключених користувачів містить API, який запитує MAC-адресу користувача, час під’єднання та ідентифікатор мережі. Отримавши цю інформацію, інтерфейс внутрішньо обробляє поля даних користувачів та повертає запитувану інформацію.

REST API — це API, який відповідає принципам дизайну архітектурному стилю REST, який у свою чергу використовується для забезпечення стандартів між комп’ютерними системами задля полегшення їх взаємодії.

Перевагою методу REST перед такими, як SOAP або XML-RPC є можливість розробки, використовуючи майже будь-яку мову програмування та підтримка різних форматів даних, що не накладає жорстких обмежень на розробників. Єдина вимога полягає в узгодженні із шістьма наступними принципами дизайну REST, також відомими, як архітектурні обмеження [3]:

1. Uniform interface. Усі запити API до одного й того ж ресурсу повинні виглядати однаково, незалежно від того, звідки надходить запит. Повинна бути гарантія, що один і той самий фрагмент даних належить лише одному уніфікованому ідентифікатору ресурсу (URI). Ресурс не повинен бути занадто великим і повинен містити всю інформацію, необхідну клієнту.
2. Client-server decoupling. Клієнтська і серверна програми мають бути повністю незалежними одна від одної. Єдина інформація, яку повинен знати клієнтський додаток — це URI запитуваного ресурсу, і він не може взаємодіяти із серверним додатком ніяким іншим чином. Аналогічно, серверна програма не повинна змінювати клієнтську програму, окрім як передавати запитувані дані через HTTP.
3. Statelessness. REST API не має стану, тому кожен запит повинен містити всю інформацію, необхідну для його обробки. Іншими словами, REST API не вимагає сеансу на стороні сервера. Серверна програма не може зберігати жодних даних, пов’язаних із запитом клієнта.
4. Cacheability. Задля підвищення продуктивності на стороні клієнта та збільшення масштабованості на стороні серверу, ресурси повинні кешуватися на стороні клієнта або сервера, якщо це можливо.
5. Layered system architecture. Помилковим може бути припущення, що клієнтська та серверна програми взаємодіють безпосередньо одна з одною. Насправді, в REST API запити та відповіді проходять через різні рівні, так як у комунікаційному колі може бути декілька посередників. REST API

повинен бути спроектований так, щоб ні клієнт, ні сервер не могли визначити, чи вони спілкуються з кінцевою програмою, чи з посередником.

6. Code on demand (optional). API REST зазвичай надсилають статичні ресурси, але в деяких випадках відповіді також можуть містити виконуваний код. У цих випадках код має виконуватися лише за вимогою.

Загально принцип роботи REST API можна описати наступною схемою:

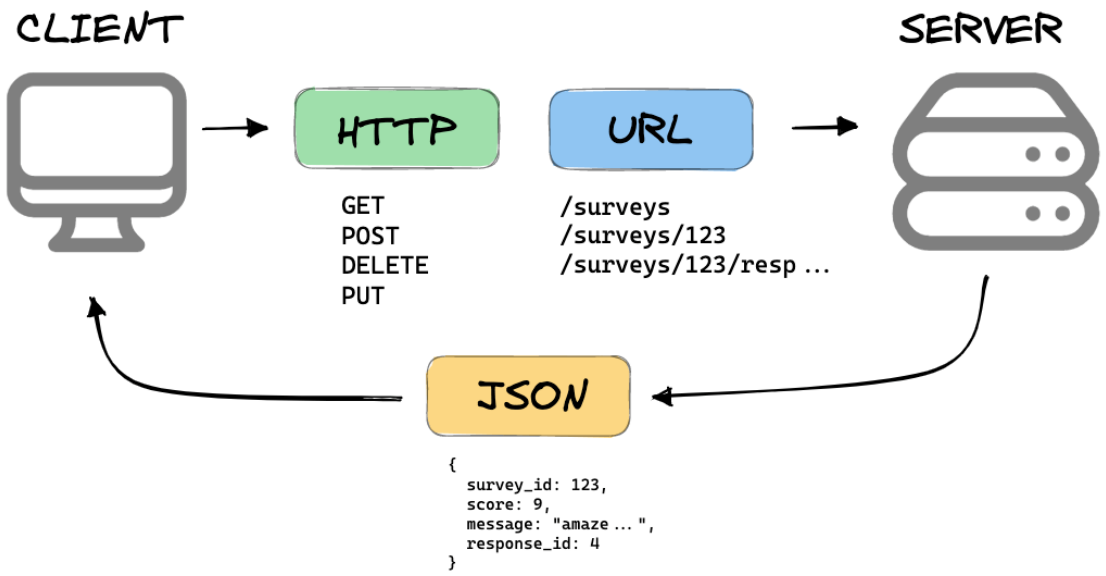


Рис.1.2 Принцип роботи REST API [4]

REST API використовують запити HTTP для створення, читання, оновлення та видалення записів, тобто виконання стандартних функцій бази даних. До запитів є певні вимоги, зокрема вони мають містити наступні компоненти:

- Унікальний ідентифікатор ресурсу

Сервер надає кожному ресурсу унікальний ідентифікатор ресурсу. У випадку служб REST сервер ідентифікує ресурси за допомогою універсального покажчика ресурсів (URL). URL-адреса вказує шлях до ресурсу. URL-адреса

аналогічна адресі веб-сайту, який вводиться в браузері для відвідування веб-сторінки.

- Метод

Як правило, розробники реалізують RESTful API за допомогою протоколу передачі гіпертексту (HTTP). Метод HTTP повідомляє серверу, що необхідно зробити з ресурсом. Нижче наведено чотири поширені методи HTTP:

GET

Клієнти використовують GET для доступу до ресурсів, розміщених на сервері за вказаною URL-адресою. Вони можуть кешувати запити GET і надсилати параметри в запиті RESTful API, щоби повідомити сервер про необхідність фільтрувати дані перед відправкою.

POST

Клієнти використовують POST для надсилання даних на сервер. При цьому вони включають запит на подання даних. Надсилання одного й того ж запиту POST кілька разів має побічний ефект — багаторазове створення одного й того ж ресурсу.

PUT

Клієнти використовують PUT для оновлення ресурсів, що існують на сервері. На відміну від POST, надсилання одного й того ж запиту PUT кілька разів дає той самий результат у веб-службі RESTful.

DELETE

Клієнти використовують запит DELETE для видалення ресурсу. Запит DELETE може змінити стан сервера. Однак, якщо користувач не має відповідної автентифікації, запит завершується помилкою.

- Заголовки HTTP

Заголовки запитів — це метадані, якими обмінюються клієнт та сервер.

- Дані

Запити REST API можуть містити дані для успішної роботи POST, PUT та інших методів HTTP.

- Параметри

Запити RESTful API можуть включати параметри, які надають серверу докладнішу інформацію про необхідні дії. Нижче наведено деякі типи параметрів:

- Установки шляху, які визначають деталі URL.
- Параметри запиту, які вимагають додаткову інформацію про ресурс.
- Параметри cookie, які швидко автентифікують клієнтів.

Відповідь серверу містить такі компоненти, як рядок стану, текст повідомлення та заголовки. Рядок стану являє собою тризначний код відповіді HTTP, який інформує про те, чи був успішно виконаний певний запит. Ці коди можна згрупувати в п’ять класів:

- 100–199: інформаційні
- 200–299: успішне виконання
- 300–399: перенаправлення
- 400–499: помилка зі сторони клієнта
- 500–599: помилка зі сторони сервера

Текст повідомлення або стан ресурсу в конкретний момент часу називають представленням ресурсу. Ця інформація надається клієнту практично в будь-якому форматі, включаючи JSON, HTML, XML, PHP або звичайний текст. Так, наприклад, якщо клієнт надсилає запит на отримання IP-адреси пристрою за MAC-адресою «de:a9:2c:17:28:3d», сервер повертає подання JSON у наступному форматі:

'{«mac»:«de:a9:2c:17:28:3d», «ip»:«13.156.98.135»}'

У даній роботі використовується саме JSON, зважаючи на його простоту для читання, розповсюдженість та незалежність від мови програмування.

Заголовки або метадані відповіді дають більш докладний контекст відповіді та включають таку інформацію, як назва сервера, кодування, дата та тип контенту.

## 1.4 Система управління базами даних MySQL

MySQL — одна з найбільш відомих технологій у сучасній монументальній системі даних. Не просто так її називають найпоширенішою базою даних та ефективно використовують повсюдно, незалежно від галузі.

Звертаючись до термінології, MySQL являє собою реляційну систему керування базами даних (RDBMS) з відкритим вихідним кодом, розроблену Oracle та використовує мову структурованих запитів (SQL) [5].

База даних — це структурований набір даних. Реляційна база даних — це цифрове сховище, яке збирає дані та упорядковує їх у відповідності з реляційною моделлю. У цій моделі таблиці складаються з рядків та стовпців, а зв’язки між елементами даних підтримують чітку логічну структуру. Іншими словами, реляційна система управління базами даних — це просто набір програмних інструментів, які використовуються для впровадження, керування та управління такою базою даних [5].

Однією з переваг MySQL перед іншими технологіями є широка сумісність з іншими технологіями та архітектурами, незважаючи на те, що MySQL часто асоціюється з інтернет-додатками або веб-сервісами. RDBSM працює на всіх основних обчислювальних платформах, включаючи операційні системи на базі Unix, тобто на безлічі дистрибутивів Linux, Mac OS та Windows. Клієнт-серверна архітектура дозволяє підтримувати різні серверні модулі та інтерфейси програмування, так само, як розгортання в розподілених або централізованих віртуальних середовищах. Широка сумісність MySQL з усіма цими системами та програмним забезпеченням робить її особливо практичним рішенням у більшості

ситуацій, включаючи безпосередньо мету даного проекту. У практичній частині роботи MySQL використовується для запису та фільтрації отриманих даних із контролеру.

Загальна структура запиту MySQL до бази даних виглядає наступним чином [6]:

**SELECT** ('стовпці * для вибору всіх стовпців; обов’язково')
**FROM** ('таблиця; обов’язково')
**WHERE** ('умова/фільтрація; необов’язково')
**GROUP BY** ('стовпець, по якому групуються дані; необов’язково')
**HAVING** ('умова/фільтрація на рівні згрупованих даних; необов’язково')
**ORDER BY** ('стовпець, по якому потрібно відсортувати дані; необов’язково')

## 1.5 Django Framework

Django — це високорівневий веб-фреймворк Python, який забезпечує швидку розробку безпечних веб-сайтів, які зручно підтримувати. Обладнаний великою кількістю вбудованих бібліотек, Django стає зручним інструментом для зосередження безпосередньо на написанні додатку без необхідності розробляти вже існуючі рішення. Він є безкоштовним та має відкритий вихідний код зі зрозумілою документацією. Django можна використовувати для створення майже будь-якого веб-сайту: від систем керування контентом до соціальних мереж та сайтів новин. Він може працювати з будь-яким клієнтським фреймворком та надавати вміст у потрібних розповсюджених форматах даних (включаючи HTML, RSS-канали, JSON та XML), також його можна розширити для використання інших компонентів, якщо це необхідно.

Основною перевагою Django перед іншими рішеннями та причиною його вибору для створення описаної в роботі програми, є безпека. Django допомагає розробникам уникати поширених помилок безпеки, надаючи структуру, розроблену для автоматичного захисту веб-сайту. Наприклад, Django забезпечує безпечний спосіб керування обліковими записами та паролями користувачів,

уникаючи розповсюдженої помилки розміщення інформації про сеанс у файлах cookie, де вони вразливі (натомість файли cookie містять лише ключ, а самі дані зберігаються в базі даних) або пряме збереження паролів, а не хеш-пароля.

Django написано мовою Python, яка є кроссплатформною. Це означає, що розробник не прив'язаний до жодної конкретної серверної платформи та може запускати свої програми на багатьох версіях Linux, Windows і macOS. Крім того, Django добре підтримується багатьма провайдерами веб-хостингу, які часто надають спеціальну інфраструктуру та документацію для розміщення сайтів на Django.

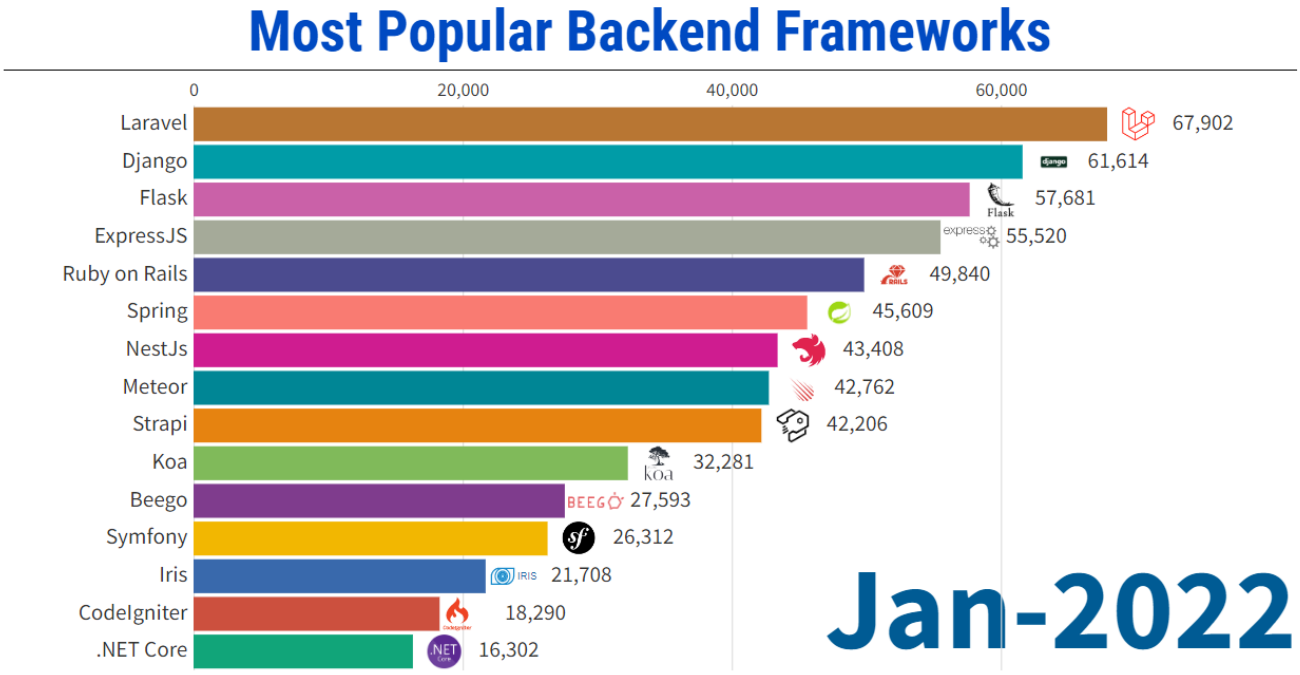


Рис.1.3 Графік популярності веб-фреймворків для серверної розробки станом на січень 2022 року [7]

Станом на січень 2022 року, Django посідає друге місце за кількістю репозиторіїв на GitHub, що можна спостерігати на графіку (див. рис. 1.3), та підтримує 84,466 веб-сайтів у 54 014 унікальних доменах [7].

На традиційному веб-сайті, керованому даними, веб-додаток очікує HTTP-запитів від веб-браузера (або іншого клієнта). Коли запит отримано, програма

обробляє потрібні дані на основі URL-адреси та іноді інформації в даних запитів POST або GET. Залежно від потреби, він може зчитувати або записувати інформацію з бази даних або виконувати інші завдання, необхідні для задовільнення запиту. Потім програма поверне відповідь веб-браузеру, часто динамічно створюючи HTML-сторінку для відображення браузером, вставляючи отримані дані в плейсхолдери в шаблоні HTML.

Веб-програми Django зазвичай групують код, який обробляє кожен із цих кроків, в окремі файли:

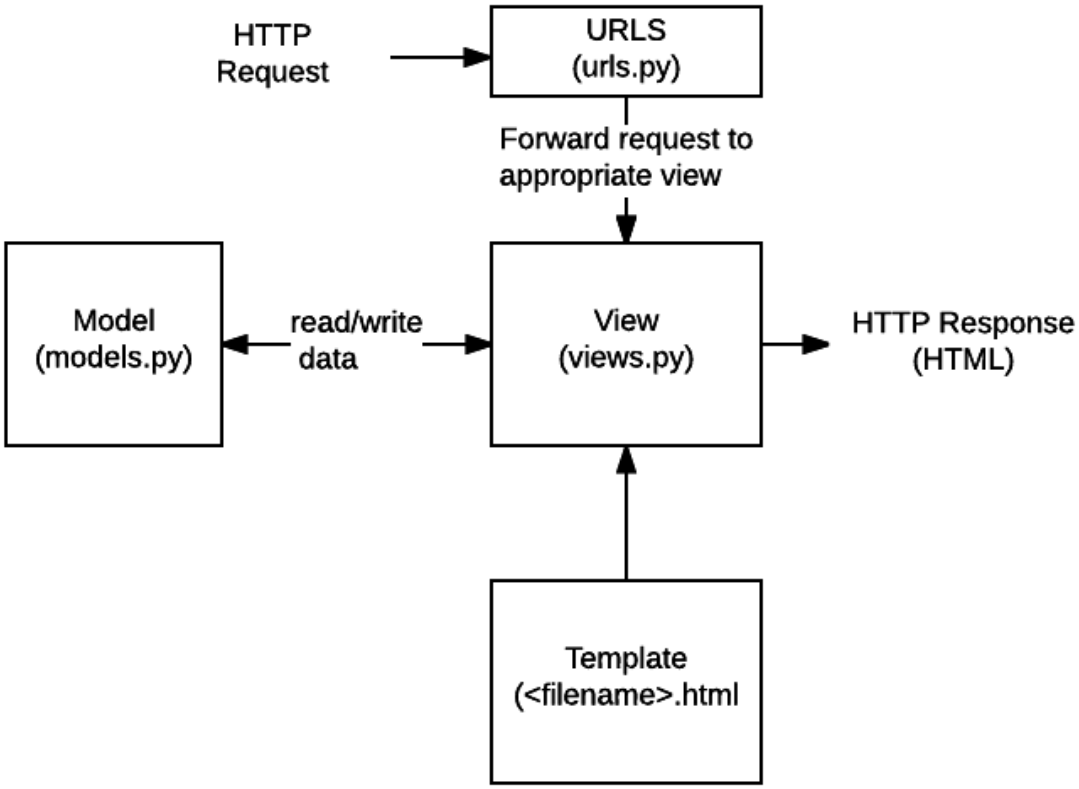


Рис.1.4 Блок-схема, яка візуалізує приклад процесу групування та обробки даних веб-програмою Django при спілкуванні клієнту та серверу [8]

- URLs: існує можливість обробляти запити з кожної окремої URL-адреси за допомогою однієї функції, але набагато зручніше написати окрему функцію перегляду для обробки кожного ресурсу. URL mapper використовується для перенаправлення HTTP-запитів до відповідного перегляду на основі URL-

адреси запиту та зіставлення певних шаблонів рядків або цифр, які з’являються в URL-адресі, і передавати їх у функцію перегляду як дані.

- View: представлення — це функція обробки запитів, яка отримує HTTP-запити та повертає HTTP-відповіді. Представлення отримують доступ до даних, необхідних для задоволення запитів через моделі, і делегують форматування відповіді шаблонам.
- Models: моделі — це об’єкти Python, які визначають структуру даних програми та надають механізми для керування (додавання, зміни, видалення) і запитів записів у базі даних.
- Templates: шаблон — це текстовий файл, який визначає структуру або макет файлу (наприклад, HTML-сторінки), з плейсхолдерами, які використовуються для представлення фактичного вмісту. Представлення може динамічно створювати HTML-сторінку за допомогою HTML-шаблону, заповнюючи її даними з моделі. Шаблон можна використовувати для визначення структури будь-якого типу файлу; це не обов’язково має бути HTML.

## 1.6 Контролери бездротових мереж

Wi-Fi контролери бездротових мереж розроблені для комерційного використання на підприємствах, в основному в мережах провайдерів. Архітектура мережі з використанням центрального контролеру дозволяє будувати масштабні бездротові мережі та забезпечує зручність їх моніторингу, адміністрування та експлуатації. У функції контролеру входить забезпечення автоматичного пошуку, централізованого налаштування точок доступу та оновлення програмного забезпечення обладнання. Окрім цього, контролер може виступати в ролі DHCP сервера для автоматичного призначення IP-адрес та проводити аналіз радіочастотного діапазону, регулювати потужність точок доступу та ширину каналу, на якому вони працюють.

## 1.7 Огляд існуючого рішення

Наразі в мережі університету існує вже готове рішення для централізованого управління та моніторингу бездротової мережі: мережевий контролер UniFi Network Management Controller, який пов’язує всі пристрої UniFi разом, надаючи веб-інтерфейс для їх налаштування. Для роботи мережі не обов’язково запускати програмне забезпечення контролера. Він потрібен лише для налаштування та моніторингу. Проте постійна робота програмного забезпечення має свої переваги, так як пристрої UniFi не мають великого обсягу пам’яті і їм потрібний контролер для реєстрації інформації про мережу. Постійна його робота також позбавляє від деяких проблем із налаштуванням, особливо у віддаленій мережі.

Однак, метою даної роботи є розробка системи моніторингу та обліку користувачів бездротової мережі, отже розглянуто буде функціонал існуючого контролеру саме для описаних цілей. Звичайно, програмне забезпечення, розроблене цілою компанією спеціалістів, які постійно удосконалюють та доповнюють його новим функціоналом не може порівнюватись із системою, розробленою в ході виконання даної роботи, проте мета цього розділу виділити певні недоліки існуючого рішення саме для потреб внутрішньої мережі університету.

Перше, на що необхідно звернути увагу — можливість спостерігати кількість підключених до бездротової мережі користувачів на даний момент:

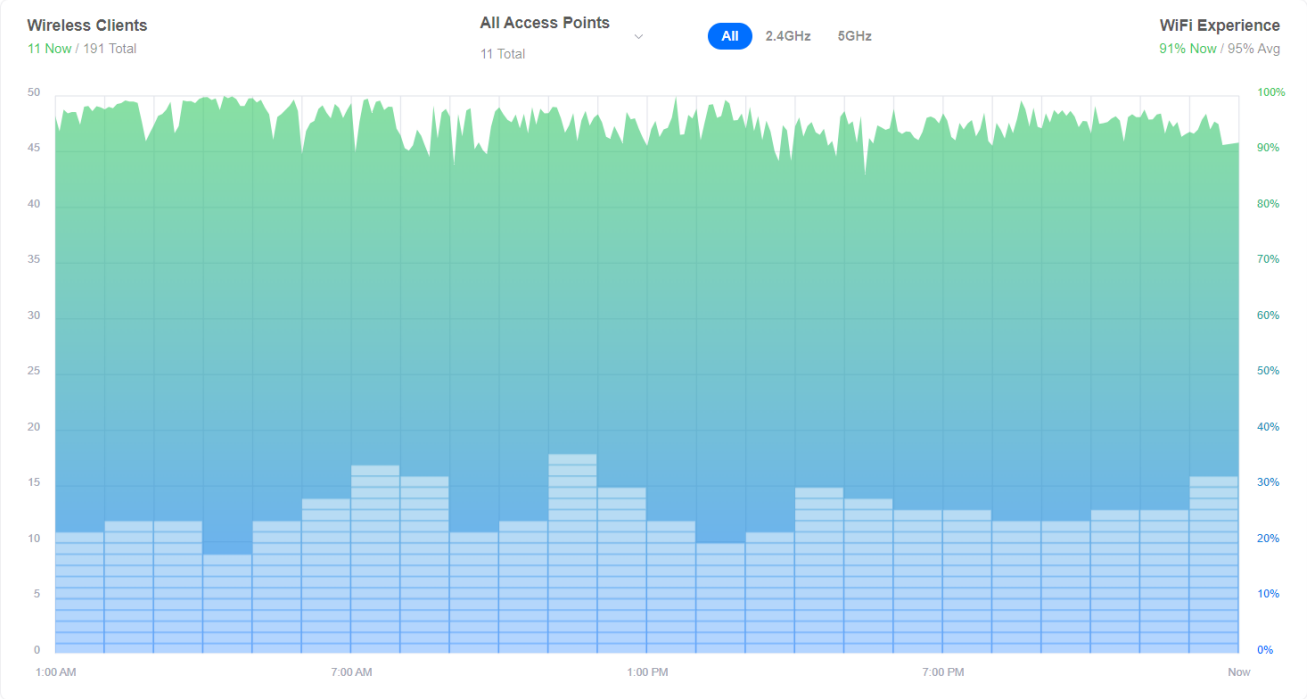


Рис.1.5 Графік використання бездротової мережі гуртожитку користувачами отриманий із веб-інтерфейсу контролеру UniFi

Безумовно, таку можливість контролер може представити. Як можна помітити в лівому верхньому кутку (див.рис.1.4), наразі до бездротової мережі під'єднано 11 користувачів. Також, з графіку можна спостерігати кількість підключених користувачів за останню добу та рівень продуктивності мережі за той самий період. Однак перегляд користувачів обмежений лише однією добою і переглянути їх кількість за попередні періоди часу неможливо. Це є першим та основним недоліком існуючого рішення: відсутність можливості запису даних про кількість користувачів до централізованої бази даних з метою подальшого її аналізу в разі необхідності, відсутність можливості збору статистики задля вдосконалення мережі.

Другим важливим моментом є можливість вибору факультету для моніторингу. Ця функція також реалізована та дозволяє обирати необхідний пункт із випадаючого списку:

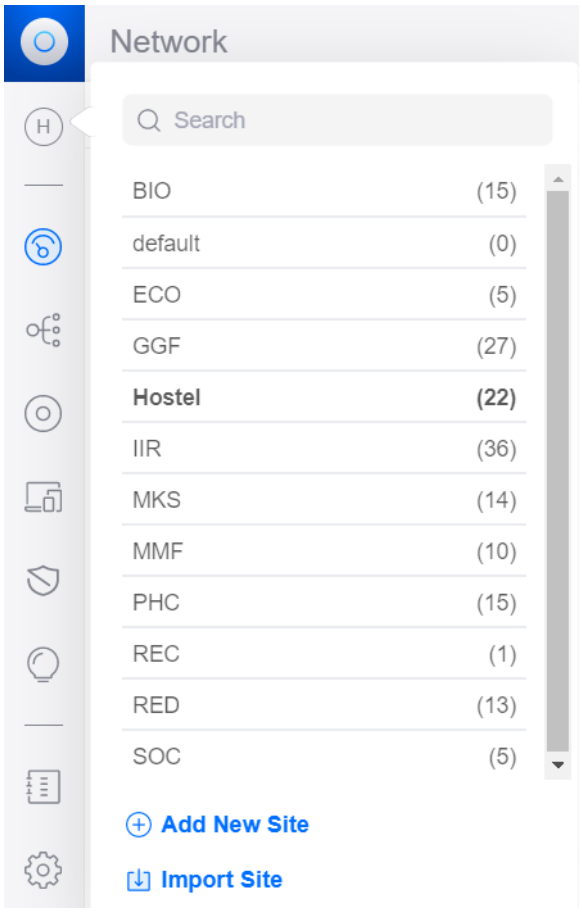


Рис.1.6 Демонстрація наявності функції вибору факультету для моніторингу

Як можна спостерігати, можливість переглянути користувачів по всьому університету відсутня, але необхідність такої функції є потрібною.

Веб-інтерфейс контролеру підтримує безліч додаткових функцій, таких як перегляд списку підключених пристроїв у даний момент, інформацію про MAC-адресу, IP-адресу, точку доступу кожного підключення та стан сигналу для кожного підключеного користувача.

Безумовно, існуюче рішення представляє мережевим адміністраторам низку можливостей для моніторингу мережі університету, але зважаючи на те, що компанія, яка розроблює та підтримує веб-інтерфейс контролеру орієнтована на величезну кількість користувачів продукту, вона не може задовольнити всіх індивідуальних потреб кожного споживача. Крім того, таке рішення зводить

нанівець можливість підключати додаткові функції та вдосконалювати інтерфейс власноруч у подальшому.

# 2 ПРАКТИЧНА ЧАСТИНА

## 2.1 Постановка задачі та архітектура програми

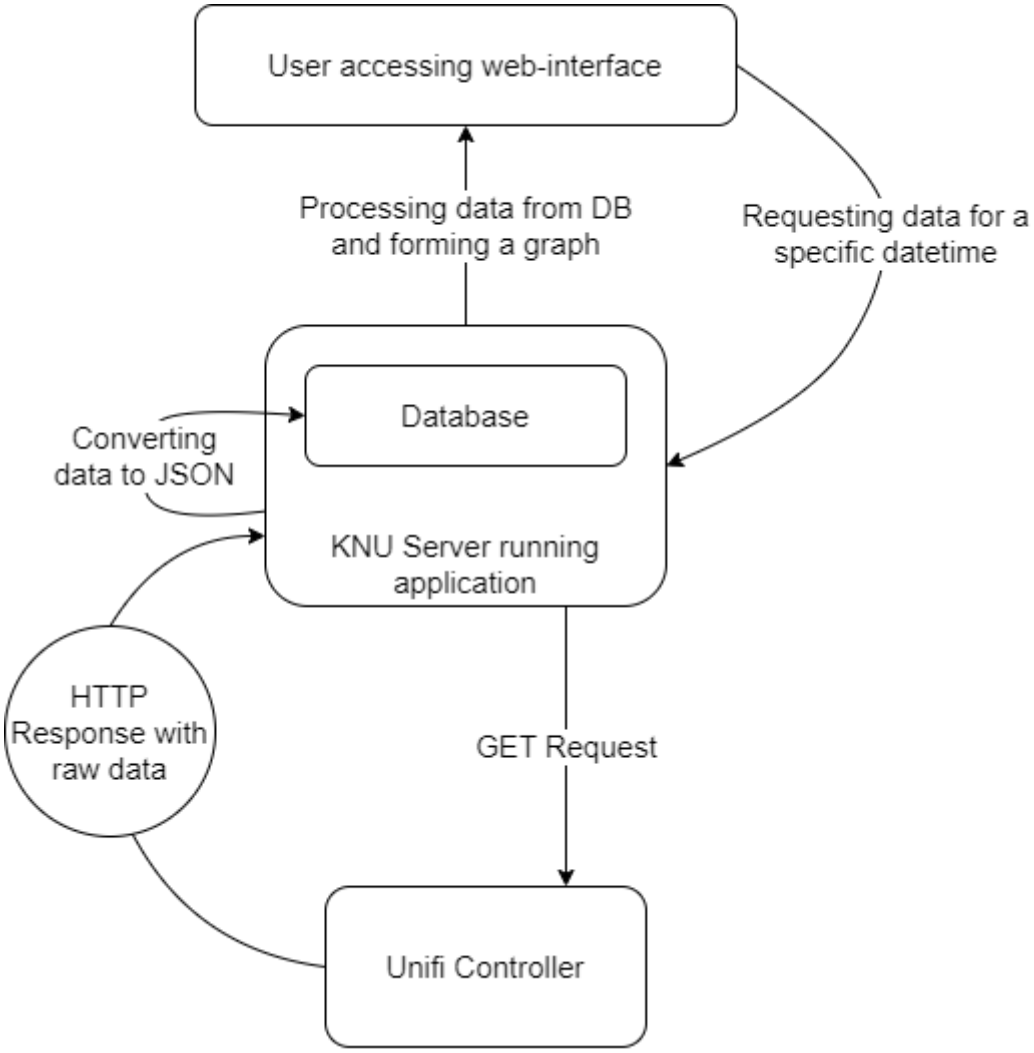


Рис.2.1 Архітектура програмного рішення

Першочергова задача полягає в розробці серверного застосунку, який, використовуючи метод REST API, буде звертатись до контролеру Unifi та виконувати наступну послідовність дій (див. рис. 2.1) :

1. Робити запит з метою отримання даних про підключених користувачів за допомогою HTTP запиту GET
2. Отримувати відповідь з необробленими даними

3. Обробляти отримані дані та конвертувати їх до формату JSON
4. Вибірково вносити дані про користувачів до бази даних на сервері

Наступним кроком є розробка зручного веб-інтерфейсу для того, щоб взаємодіяти через нього із сервером, а не напряму. Послідовність наступна:

1. Мережевий адміністратор на веб-сторінці обирає факультети для запису даних та інтервал запису даних
2. Програма на сервері починає відпрацьовувати з обраним інтервалом та збирати дані
3. На сторінці з графіком адміністратор обирає часовий проміжок для відображення даних, зібраних за бажаний період
4. Програма на сервері обробляє введену інформацію та виводить відповідний графік на сторінку браузеру

## 2.2 Програмне середовище

Для розробки описаного в пункті 2.1 застосунку було вирішено використовувати мову програмування Python. Причиною такого вибору стало те, що Python легко інтегрується з базами даних та широко використовується для обробки даних, а тому має велику кількість документації за даною темою. У якості фреймворку було обрано Django через велику кількість існуючих бібліотек, які суттєво спрощують створення програмного рішення та імплементації на Linux-подібних операційних системах. Для відрисовки графіку на сторінці веб-інтерфейсу обрано мову програмування JavaScript. Середовище для розробки, відповідно, PyCharm, а операційну систему для демонстраційного розгортання серверу — Ubuntu.

Під час виконання поставленої задачі було використано наступні основні пакети та віджети:

- django-chartjs — для створення графіку
- django rest framework — для створення web-API

- django-bootstrap-datepicker-plus — для створення вікна з вибором дати та часу
- httpcore — для відправки HTTP-запитів
- hyperframe — для парсингу HTTP-фреймів
- mysql-connector-python — для зв’язку із сервером MySQL
- python-telegram-bot — інтерфейс для API телеграм-боту
- pytz — для налаштування часового поясу
- requests — для використання HTTP
- urllib3 — для відкриття URL-адрес

## 2.2 Відправка запиту в правильне представлення (urls.py)

Зіставленням URL-адрес керують за допомогою змінної urlpatterns, яка є списком функцій path(). У наведеному нижче лістингу змінна urlpatterns визначає список співставлень між маршрутами та відповідними функціями перегляду. Код решти файлів зіставлення URL-адрес наведено в Додатку А.

```
urlpatterns = [
    path('', views.index, name='logger'),
    path('log', views.log, name='log'),
    path('stop', views.stop_log, name='log-stop')
]
```

Лістинг 2.1 Співставлення маршрутів та функцій перегляду [9]

Першим аргументом обох методів є маршрут, який буде співставлено. Метод path() використовує кутові дужки для визначення частин URL-адреси, які будуть захоплені та передані функції співставлення в якості іменованих аргументів. Другий аргумент — це ще одна функція, яка буде викликатись при співпадінні шаблону. У даному випадку views.index вказує, що викликається функція index() та може бути знайдена в модулі views (тобто у файлі views.py). Відповідно, функції log() та stop_log() також можна знайти в модулі views.

## 2.3 Обробка запиту (views.py)

Представлення — це «серце» веб-застосунку, вони отримують HTTP-запити від веб-клієнтів та повертають HTTP-відповіді [10]. У даному випадку асинхронна функція перегляду index(), викликана засобом співставлення URL-адрес у попередньому розділі, отримує в якості параметру request:

```
async def index(request):
..
if request.method == 'POST':
form = LoggerForm(request.POST)
if form.is_valid():
site_selector = form.cleaned_data['site_selector']
period = int(form.cleaned_data['period_selector'])
if period <= 0:
if site_selector == 'ALL':
await log_all(request)
await log(request, site_selector)
else:
    while True:
if stop_flag:
    break
            start_time = time.time()
if site_selector == 'ALL':
await log_all(request)
else:
            await log(request, site_selector)
elapsed_time = time.time() — start_time
if elapsed_time < period:
time.sleep(period — elapsed_time)
else:
```

```
form = LoggerForm()
return render(request, 'log.html', {'form': form})
```

Лістинг 2.2 Основна функція програми для формування запиту

При виклику даної функції виконується наступна клієнт-серверна взаємодія:

Адміністратор у веб-інтерфейсі програми обирає у відповідних полях факультет (або всі одразу), про користувачів якого він хоче почати збирати дані та інтервал, з яким цей збір даних буде відбуватись. Після цього функція передає введені адміністратором дані та, залежно від його вибору, викликає функцію log_all або log. Перша функція винесена окремо на випадок, якщо є потреба збирати дані про всі факультету одразу. Якщо виконані всі необхідні умови, функція index() рендерить та повертає html-шаблон, який містить у собі дані про запит, який далі буде відправлено до контролеру, як запит HTTP.

Основний метод запиту та безпосередньо сам запит до контролеру Unifi знаходиться в наступній функції:

```
async def log(request, site):
session = knu_auth()
data_url = KNU_URL +
f»/v2/api/site/{site}/clients/active?includeTrafficUsage=true»
data = json.loads(session.get(data_url, verify=False).text)
await networks_bulk(data)
return HttpResponse(request)
```

Лістинг 2.3 Запит до контролеру Unifi

Функція log, яка викликається у випадку правильно заповненої адміністратором форми, аутентифікує серверний застосунок на сайті контролеру та за постійним посиланням до контролеру KNU_URL (безпосередньо знаходиться у файлі settings.py) та доповненим до нього динамічним посиланням залежно від обраного факультету, збирає одне ціле посилання для надсилання

запиту GET. Отримані відповіддю від контролера дані про користувачів парсяться до формату JSON та в подальшому будуть записані до бази даних.

## 2.4 Визначення моделей даних (models.py)

Веб-застосунки Django керують даними та запитують їх через об’єкти Python, які називаються моделями. Моделі визначають структуру даних, що зберігаються, включаючи типи полів, максимальний розмір, значення за замовчуванням, параметри списку вибору, тощо. Після того, як було обрано базу даних, до неї більше не потрібно звертатись напряму, достатньо налаштувати структуру моделі, а Django буде автоматично виконувати всю роботу із запитами до бази даних:

```
class Device(models.Model):
id = models.UUIDField(primary_key=True, default=uuid.uuid4)
mac = models.GenericIPAddressField()
site_id = models.CharField(max_length=50)
ip = models.GenericIPAddressField(null=True)
logged_at = models.DateTimeField()
uptime = models.BigIntegerField()
```

Лістинг 2.4 Модель Django для об’єкту Device

У наведеному вище лістингу, всередині моделі визначені наступні поля для запису до бази даних:

- `id` — унікальний ідентифікатор типу UUID, який використовується в якості первинного ключа запису. За замовчуванням, при додаванні до бази даних запису про нового користувача, генерується новий UUID
- `mac` — поле типу `GenericIPAddressField`, призначене для зберігання MAC-адреси підключеного пристрою користувача
- `site_id` — поле типу `CharField` з максимальною довжиною 50 символів, яке призначене для зберігання ідентифікатору факультету, до якого підключено пристрій користувача

- `ip` — поле типу `GenericIPAddressField`, призначене для зберігання IP-адреси підключеного пристрою користувача (допускається значення `null`)
- `logged_at` — поле типу `DateTimeField`, призначене для зберігання дати та часу створеного запису
- `uptime` — поле типу `BigIntegerField`, призначене для зберігання часу з моменту підключення пристрою користувача до мережі в секундах

## 2.5 Наповнення бази даних

Після створення моделі даних необхідно забезпечити безпосередньо наповнення бази даних. Наведена далі функція знаходиться у файлі представлень та має наступний вигляд:

```
async def networks_bulk(networks):
..
bulk = [
Device(
mac=network.get('mac'),
site_id=site_dict.get(network['site_id']),
ip=network.get('ip', ''),
logged_at=timezone.now(),
uptime=network['uptime'],
)
    for network in networks
]
await database_sync_to_async(Device.objects.bulk_create)(bulk)
print('bulk inserted', len(bulk))
     ..
```

Лістинг 2.5 Механізм наповнення бази даних

Всередині асинхронної функції `networks_bulk` створюється список `bulk`, який містить об'єкти класу `Device` (див. лістинг 2.5). Далі відбувається асинхронне наповнення пакетної вставки записів `bulk` в базу даних та виведення на екран консолі повідомлення про кількість вставлених записів. Інакше кажучи, до бази даних записуються дані про підключені пристрої користувачів, шаблони яких визначені у файлі models.py, після кожної ітерації відпрацювання програми.

## 2.6 Налаштування міграцій бази даних

Міграції потрібні для розповсюдження змін, які вносяться в моделі (доповнення полів, видалення моделі тощо), у схему бази даних. Для взаємодії з міграціями та обробкою Django схеми бази даних будуть використані дві команди: «migrate» — відповідає за застосування та відміну міграції та «makemigrations» — відповідає за створення нових міграцій на основі змін, внесених у моделі. По суті, міграції являють собою систему контролю версій для схеми локальної бази даних. Файли міграції знаходяться в каталозі «migrations» та призначені для передачі та розповсюдження як частина його кодової бази.

Дана програма передбачає загалом 7 файлів міграції бази даних, кожен із яких залежить від попереднього:

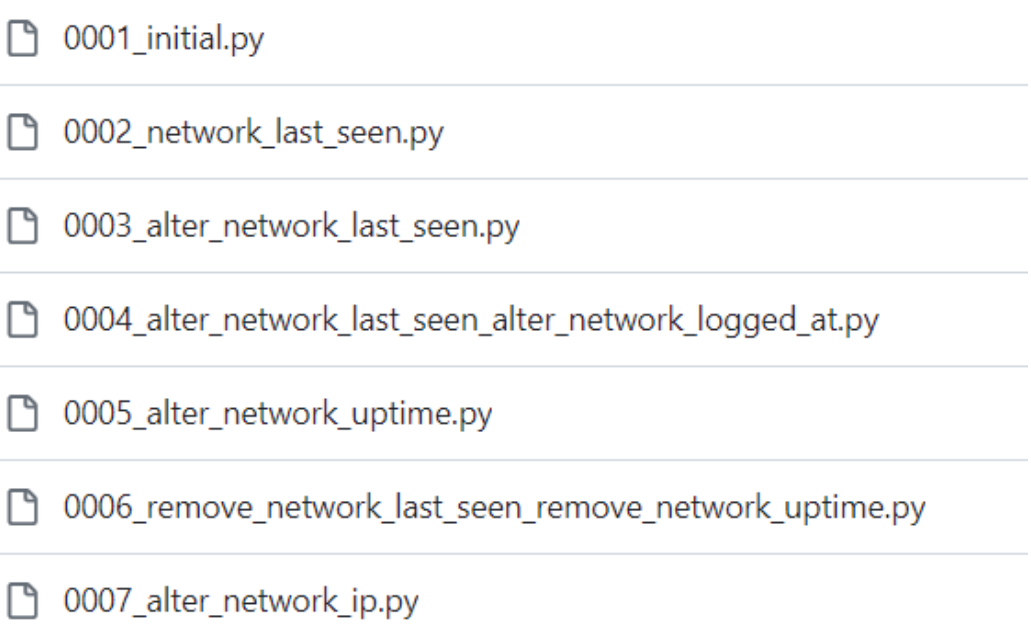

Рис.2.1 Список файлів міграції бази даних

Розглянуто буде початковий файл міграції, тобто «0001_initial.py», (код решти файлів наведено в Додатку Б):

```
class Migration(migrations.Migration):

operations = [

migrations.CreateModel(

name='Network',

fields=[

 ('id', models.UUIDField(default=uuid.uuid4, primary_key=True,
serialize=False)),

('mac', models.GenericIPAddressField()),

('network_name', models.CharField(max_length=50)),

('ip', models.GenericIPAddressField()),

('logged_at', models.DateTimeField()),

('uptime', models.IntegerField()),

('raw_data', models.JSONField()),

],

      ),

]
```

Лістинг 2.6 Початковий файл міграції бази даних

У наведеному файлі міграції створюється модель «Network» із відповідною таблицею в базі даних. Значення полів таблиці було описано в пункті 2.4. Власне, після виконання початкової міграції до бази даних уже можна звертатись за допомогою створеної моделі, такий метод створення таблиць дозволяє уникнути взаємодії із синтаксисом SQL.

## 2.7 Налаштування та обробник подій для Telegram-боту

Розроблена програма забезпечує можливість моніторити збої в роботі обладнання за допомогою відправки повідомлень у Telegram-бот. Наразі існує

механізм виявлення пристроїв користувачів, які не отримують IP-адресу за певний час при підключенні до мережі, що може допомогти виявити помилки в роботі DHCP-серверу.

Перше, що потрібно реалізувати — створити екземпляр Telegram-бота із використанням унікального токену, який можна отримати при створенні пустого боту безпосередньо в самому Telegram:

```
..
def bot_instance():
bot = ApplicationBuilder().token(TELEGRAM_TOKEN).build()
return bot
```

Лістинг 2.7 Створення екземпляру Telegram-боту

Другий крок — запис інформації про ініціалізацію боту адміністратором до бази даних та відправлення відповідного повідомлення про успішний старт:

```
async def handle_start(update: Update, context:
ContextTypes.DEFAULT_TYPE):
user = update.effective_user
await asyncio.sleep(1)
created = await
database_sync_to_async(TelegramUser.objects.get_or_create)(username=
user.username, chat_id=user.id)
if created:
await context.bot.send_message(
chat_id=user.id,
text=«unifi knu\nbot started\n»
)
```

Лістинг 2.8 Ініціалізація боту та створення запису про Telegram-користувача

Запис до бази даних про користувачів Telegram-ботом є необхідною мірою для спостереження за тим, кому приходять повідомлення про помилки. За

необхідністю, можна припинити сесію в боті будь-якого Telegram-користувача вручну, видаливши запис із бази даних, або прописати умови, які будуть дозволяти лише певним користувачам ініціалізацію боту.

Третій крок — задати відпрацювання функції відправки повідомлення, зміст якого описано в лістингу 2.10, та забезпечити логування можливих помилок:

```
async def handle_err_notify(chat_id, message, bot):
try:
await bot.send_message(chat_id=chat_id, text=message)
except Exception as e:
logging.error(f«Error sending message to chat {chat_id}: {e}")


bot = bot_instance()
handler = CommandHandler('start', handle_start)
bot.add_handler(handler)
def main():
bot.run_polling()
```

Лістинг 2.9 Механізм обробки ботом змісту повідомлення та запуск

Так, крім описаної функції, у коді реалізовано створення обробника команди «start», яку натискає користувач Telegram-боту для початку роботи з ним. Після зазначених вище дій, бот запускається в режимі очікування нових повідомлень.

Останній крок — задати умови внесення користувачів, які не отримують IP-адресу в окремий список та створити шаблон для виводу повідомлення. Для цього до функції з лістингу 2.5 було додано пустий список та прописано наступні умови:

```
async def networks_bulk(networks):
..
ip_not_found = []
```

```
..
for network in networks:
if network.get('type') == 'WIRED' or network.get('is_wired' ==
True):
# Skip saving in the database for WIRED devices
continue
 ip = network.get('ip', '')
if (ip == '' or None) and network['uptime'] > 120:
ip_not_found.append(network)


device = Device(
mac=network.get('mac'),
site_id=site_dict.get(network['site_id']),
ip=ip,
   logged_at=timezone.now(),
uptime=network['uptime'],
)
    bulk.append(device)
print('Bulk inserted:', len(bulk))
if bulk:
await database_sync_to_async(Device.objects.bulk_create)(bulk)
if ip_not_found:
message = «No IP address found for the following devices:\n\n»
for device in ip_not_found:
site_id = device.get('site_id', '-')
mac = device.get('mac', '-')
ip = device.get('ip', '-')
uptime = device.get('uptime', '-')
message += f«Site ID: {site_id}\nMAC: {mac}\nIP: {ip}\nUptime:
{uptime} seconds\n\n»
```

```
await send_message_tg(message)
```

Лістинг 2.10 Механізм створення повідомлення та його зміст

Так, якщо пристрій користувача не отримує IP-адресу впродовж більш, ніж двох хвилин (для визначення використовується поле «uptime»), запис про нього заноситься до створеного списку. Далі відбувається ітерація по кожному входженню в списку та витягується інформація про факультет, MAC-адресу, IP-адресу та час роботи пристрою. При цьому відкидаються користувачі, які в даний момент підключені не до бездротової мережі. Формується повідомлення, яке в подальшому буде відправлено до телеграм боту. Отримана інформація додається до шаблону повідомлення та відправляється в обробник подій із лістингу 2.9.

## 2.8 Формування та відображення графіку

У попередніх розділах було описано реалізацію серверної частини програми, тобто логіку роботи програми. Тепер задача полягає в створенні веб-інтерфейсу для керування застосунком без необхідності вводити команди в термінал вручну.

Відрисовка сторінки з вибором факультетів для логування являє собою файл розмітки HTML, який є доволі об’ємним та знаходиться в Додатку В.

Основна частина веб-інтерфейсу, з якою найчастіше буде відбуватись взаємодія мережевого адміністратора — сторінка з графіком підключених користувачів. Реалізація складається з двох частин: логічної та відображення. Спершу буде розглянута логічна частина:

```
class BarChartJSONView(View):
def get(self, request, *args, **kwargs):
start_date_str = request.GET.get('start')
end_date_str = request.GET.get('end')


start_date = datetime.fromisoformat(start_date_str)
```

```
end_date = datetime.fromisoformat(end_date_str)
queryset = Device.objects
.filter(logged_at__range=[start_date, end_date])
.annotate(altered_logged_at=TruncMinute('logged_at'))
.values('altered_logged_at', 'site_id')
.annotate(mac_count=Count('mac', distinct=True))
.distinct()
.order_by('altered_logged_at')
```

Лістинг 2.11 Механізм формування часових рамок та фільтрація унікальних входжень до графіку

Тут відбувається отримання початкових та кінцевих дати та часу, які обирає адміністратор у віджеті з календарем. Далі реалізована фільтрація об’єктів по діапазону часу запису до бази даних та видалення MAC-адрес, запис яких відбувається повторно за обраний період часу. Це необхідно для того, щоб у разі підключення пристрою користувача до мережі декілька разів за один проміжок часу, при формуванні графіку враховувалось лише підключення, дійсне на момент відпрацювання програми.

```
dataset_label = []
for obj in queryset:
if obj['site_id'] not in dataset_label:
dataset_label.append(obj['site_id'])
labels = []
for obj in queryset:
if obj['altered_logged_at'].strftime('%d.%m%H:%M') not in labels:
labels.append(obj['altered_logged_at'].strftime('%d.%m%H:%M'))


datasets_dict = {label: [0] * len(labels) for label in
dataset_label}
```

```
for obj in queryset:
label = obj['site_id']
data_index =
labels.index(obj['altered_logged_at'].strftime('%d.%m%H:%M'))
datasets_dict[label][data_index] = obj['mac_count']
..
chart_data = {
'labels': labels,
'datasets': datasets
}
    return JsonResponse(chart_data)
chart = BarChartJSONView.as_view()
```

Лістинг 2.12 Формування лейблів для кожного датасету на графіку

Код вище формує список унікальних «site_id», тобто, факультетів, значення дати та часу, коли було зроблено запис до бази даних та список унікальних входжень MAC-адрес на момент відпрацювання програми. Далі створюється словник із лейблами та наборами даних та повертається JSON-відповідь із даними для графіку. Представлення в кінці створюється для використання даних в інших маршрутах. Лейбли необхідні для того, щоби при наведенні курсором на стовпчик графіку, можна було побачити час, назву факультету (або підрозділу) та кількість підключених пристроїв користувачів.

Друга частина реалізації графіку необхідна для, безпосередньо, відображення на сторінці веб-інтерфейсу і виконана з використанням мови програмування JavaScript:

```
{% block scripts%}
<script type=«text/javascript»
src=«https://cdn.jsdelivr.net/npm/chart.js/dist/chart.umd.min.js»></
script>
<script>
```

```
const barChart = new Chart(document.getElementById(«line-chart»), {
type: 'line',
data: {
labels: [],
datasets: []
},
..
```

Лістинг 2.13 Початок налаштування параметрів графіку

За основу було взято зовнішній скрипт Chart.js [10]. Після його підключення, було створено об’єкт графіку з типом «line», тобто, лінійна діаграма, ініціалізовано масиви лейблів та набору даних графіку, які будуть заповнені даними, отриманими в коді лістингу 2.12.

У цьому ж файлі необхідно реалізувати ще два скрипти. Перший скрипт буде обробляти подію зміни значень полів вводу діапазону дати та часу (DateTime):

```
<script>
function filterData() {
const startDateTime =
document.getElementById('startDateTime').value;
const endDateTime = document.getElementById('endDateTime').value;
const url = `/graph/bar-
json/?start=${startDateTime}&end=${endDateTime}`


fetch(url)
.then(response => response.json())
.then(data => {
barChart.data.labels = data.labels;
barChart.data.datasets = data.datasets;
barChart.update();
```

```
}).catch(error => console.error(error));
}
</script>
```

Лістинг 2.14 Скрипт обробки зміни значень DateTime та функція оновлення графіку

Даний скрипт після отримання значень початкової та кінцевої DateTime формує URL-адресу для запиту графіку. Функція «fetch» виконує HTTP-запит за вказаною адресою та перетворює отриману відповідь у формат JSON. На основі отриманих даних оновлюються лейбли на графіку та набори даних.

Другий скрипт описує налаштування формату дати та часу до відповідного часового поясу та конвертацію поточної дати до формату ISO (див. лістинг 2.15):

```
<script>
const dateParams = {
timeZone: 'Europe/Kyiv',
}
const lastHour = new Date(Date.now().toLocaleString() – 3,60 млн);
const lastHourString = lastHour.toLocaleString('en-US', dateParams);
const lastHourISOstring = new
Date(lastHourString).toISOString().slice(0, 16);

const now = new Date();
const nowString = now.toLocaleString('en-US', dateParams);
const nowISOstring = new Date(nowString).toISOString().slice(0, 16);

document.getElementById(«startDateTime»).defaultValue =
lastHourISOstring;
document.getElementById(«endDateTime»).defaultValue = nowISOstring;
</script>
```

Лістинг 2.15 Налаштування дати та часу до потрібного часового поясу

Це необхідно для коректного відображення зазначених параметрів на графіку та в календарі вибору потрібних дати та часу.

Повний код відрисовки графіку на сторінці веб-інтерфейсу наведено в Додатку Г.

## 2.9 Встановлення та запуск серверної програми в терміналі Linux

Після програмної реалізації основної логіки роботи програми та оформлення веб-інтерфейсу, необхідно встановити та запустити застосунок на сервері для подальших маніпуляцій з обробкою та виводом даних:

Перший крок — вибір платформи для запуску: у даній роботі для демонстрації функціоналу розробленого застосунку було обрано ОС Ubuntu, але його запуск можливий як на більшості Linux-подібних операційних системах, так і на Windows та MacOS. Далі необхідно переконатись у тому, що для запуску використовується версія Python 3.9 або 3.10 тому, що може виникнути конфлікт деяких бібліотек.

Другий крок — створення віртуального середовища для управління залежностями та ізоляції проекту:

«cd django-unifi-motitoring/"

«python3 -m venv venv»

«source venv/bin/activate»

Наведені команди вводяться у Linux-термінал та, відповідно, створюють у кореневій директорії проекту віртуальне середовище з подальшим його запуском.

Третій крок — встановлення серверу MySQL для створення бази даних, де будуть зберігатись необхідні дані для обробки та подальшої візуалізації [11]. Обов’язковою умовою є виконання «mysql-secure-installation». Також необхідно

виконати встановлення всіх потрібних модулів, частково описаних у розділі 2.2: «pip3 install -r requirements.txt».

Четвертий крок — створення файлу.env, який дозволяє налаштовувати окремі змінні існуючого середовища (зірочками позначені значення, які необхідно задавати залежно від подальших персональних налаштувань):

«django-unifi-monitoring/.env»:

DJANGO_SETTINGS_MODULE = unifi.settings

APP_URL = '*server url*'

APP_DEBUG = True

APP_KEY = 'django-insecure-%bn3(i3bl#^(ajfzmimc)(*n3b7*+^i)yn$61=9uwk+4wfr#k1'

DB_NAME = *database name*

DB_USER = *database username*

DB_USER_PASSWORD = *database user password*

DB_HOST = *database host*

DB_DB_PORT = 3306

KNU_USERNAME = *user authentication login for https://unifi.noc.knu.ua:8443/*

KNU_PASSWORD = *user authentication password for https://unifi.noc.knu.ua:8443/*

TELEGRAM_TOKEN = *token for created telegram bot*

Останній крок — безпосередньо запуск відпрацювання серверної програми так само командами через Linux-термінал:

«python3 manage.py makemigrations»

«python3 manage.py migrate»

«python3 manage.py runserver»

Та запуск телеграм-боту:

«python3 manage.py bot start»

Після проведених маніпуляцій програма запущена та готова до систематичного відпрацювання та отримання подальших запитів від користувача веб-інтерфейсом, запису користувачів до створеної раніше бази даних, відображення графіків за запитом мережевого адміністратора та надсилання сповіщень про помилки до налаштованого телеграм-боту.

Повний код та усі файли, необхідні для запуску програми знаходяться за посиланням на репозиторій GitHub [12].

# 3 РЕЗУЛЬТАТИ ВИКОРИСТАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

## 3.1 Структура бази даних

В ході відпрацювання серверної програми, користувачі, підключені до мережі університету, записуються до певної таблиці бази даних для подальшої обробки інформації задля виводу статистики у вигляді графіку на веб-інтерфейс. Сама структура бази даних виглядає наступним чином:

```
mysql> describe logger_device;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| id        | char(32)    | NO   | PRI | NULL    |       |
| mac       | char(39)    | NO   |     | NULL    |       |
| site_id   | varchar(50) | NO   |     | NULL    |       |
| ip        | char(39)    | YES  |     | NULL    |       |
| logged_at | datetime(6) | NO   |     | NULL    |       |
| uptime    | bigint      | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
6 rows in set (0,02 sec)
```

Рис.3.1 Структура таблиці з інформацією про підключених користувачів

«id» — унікальний ідентифікатор входження до таблиці бази даних;
«mac» — MAC-адреса користувача, підключеного до мережі;
«site_id» — назва факультету або підрозділу університетської мережі, до якої підключений користувач;
«ip» — IP-адреса користувача;
«logged_at» — дата та час у який було зроблено запис до таблиці;
«uptime» — час у секундах, який користувач був підключений до мережі на момент відпрацювання серверної програми.

Заповнена таблиця після декількох ітерацій відпрацювання програми виглядає наступним чином:

| ffad7135edfe489397e25f39c8d57c54 | 28:16:7f:f8:80:d5 | MMF | 10.12.81.96 | 2023-05-31 12:30:14.147429 | 705 |
|---|---|---|---|---|---|
| ffafb4d4a0f24dd7b7eb4715c0dbb79d | 12:9a:16:d0:50:de | Hostel | 10.230.27.48 | 2023-05-11 10:48:56.357736 | 91 |
| ffb227e554a64c13bb1a37f6178c1130 | f4:f5:24:f9:21:98 | MMF | 10.12.83.170 | 2023-05-31 16:45:16.834809 | 223 |
| ffb79a9036944fd8a5086b60816819d2 | f0:7d:68:f0:6d:d2 | MKS | NULL | 2023-05-27 15:45:06.725313 | 5289962 |
| ffbf40bbc17b4b14ae1fb7adbdcdd9b9 | 0e:be:d4:26:2c:da | MMF | 10.12.82.36 | 2023-05-26 12:22:21.255856 | 507 |
| ffc43a09649945968ad2779655811da2 | 2c:3b:70:3a:f0:eb | MMF | 10.12.65.134 | 2023-05-26 12:37:17.842916 | 3902 |
| ffc68c8e4abc4405b6d76ed653ec409a | e0:1f:88:3c:67:48 | IIR | NULL | 2023-05-10 15:18:44.450466 | 227 |
| ffc8c17177144478a6ed413e115004f8 | 9e:7b:82:3d:68:60 | Hostel | 10.230.20.57 | 2023-05-27 19:45:09.508834 | 632 |
| ffcfc93afd8a465abc3e0dc8d641a88f | 24:11:45:b1:3c:da | PHC | 10.44.80.141 | 2023-05-26 13:00:12.899565 | 21 |
| ffd88f2622bc474e9470b96add7280e7 | 0e:be:d4:26:2c:da | MMF | 10.12.82.36 | 2023-05-11 09:04:00.688532 | 574 |
| ffea8748476b4eb0bf407a0b9ce03742 | f0:79:59:41:dc:7f | MKS | NULL | 2023-05-31 11:30:07.622322 | 282 |
| ffeb70300cc9415eb93b3c741c51f609 | 5a:44:4d:1e:5f:16 | Hostel | 10.230.43.254 | 2023-05-27 14:15:06.893515 | 2098 |
| ffec08561d034aa7a24526a1e71269b1 | 1a:37:e9:96:d4:d1 | MKS | 10.23.78.202 | 2023-05-31 08:30:05.452520 | 1353 |
| ffec49c2e73648a691f5c29013460341 | 92:2f:bc:f4:29:82 | MKS | 10.23.76.82 | 2023-05-31 12:45:05.542779 | 126 |
| fff1d0f47c6e4432b8396c07082751eb | ce:0c:66:c1:a5:f7 | MKS | 10.23.76.158 | 2023-05-31 09:45:05.980739 | 2688 |
| fff4d153919b48d7b6771fc23a87d4d6 | fe:12:a6:b6:30:51 | MMF | 10.12.73.106 | 2023-06-01 12:45:13.038446 | 2037 |
| fff86ab0a8c147fe909da5682d5983d5 | 42:ff:60:4a:e3:39 | RED | 10.39.66.106 | 2023-06-01 17:30:05.874668 | 3043 |
| fffe02643e564483820357340d811c2e | 8e:21:8f:f5:75:04 | IIR | 10.46.72.32 | 2023-05-31 09:15:05.769352 | 3645 |

15584 rows in set (0,14 sec)

Рис.3.2 Приклад заповнення таблиці бази даних

Так, на рис.3.2 можна побачити, що сортування рядків відбувається за первинним ключем, тобто полем «id». Також варто відмітити, що в деяких записів у полі «ip» присутнє значення NULL — це означає, що пристрій не отримав IP-адреси на момент відпрацювання програми.

Окрім запису інформації про користувачів університетською мережею, у базі даних міститься таблиця про користувачів телеграм-боту, до якого надходять сповіщення про певні несправності в роботі мережі:

```
mysql> describe tgbot_telegramuser;
+----------+-----------------+------+-----+---------+----------------+
| Field    | Type            | Null | Key | Default | Extra          |
+----------+-----------------+------+-----+---------+----------------+
| id       | bigint          | NO   | PRI | NULL    | auto_increment |
| username | varchar(100)    | NO   |     | NULL    |                |
| chat_id  | bigint unsigned | NO   | UNI | NULL    |                |
+----------+-----------------+------+-----+---------+----------------+
3 rows in set (0,00 sec)
```

Рис.3.3 Структура таблиці з інформацією про користувачів телеграм-боту

«id» — унікальний ідентифікатор входження до таблиці бази даних; «username» — юзернейм акаунту користувача телеграму;

«chat_id» — унікальний системний ідентифікатор акаунту, який створюється в момент запуску телеграм-боту.

Облік таких користувачів необхідний для коректної відправки повідомлень та запобігання отримання інформації сторонніми особами, які можуть потенційно отримати доступ до персональних даних користувачів мережі університету.

## 3.2 Веб-інтерфейс користувача

При переході на головну сторінку веб-інтерфейсу застосунку, можна побачити два випадаючі списки та верхнє меню з посиланням (див. рис. 3.4), яке при натисканні перенаправляє адміністратора на авторизаційну сторінку веб-інтерфейсу мережевого контролеру Unifi.

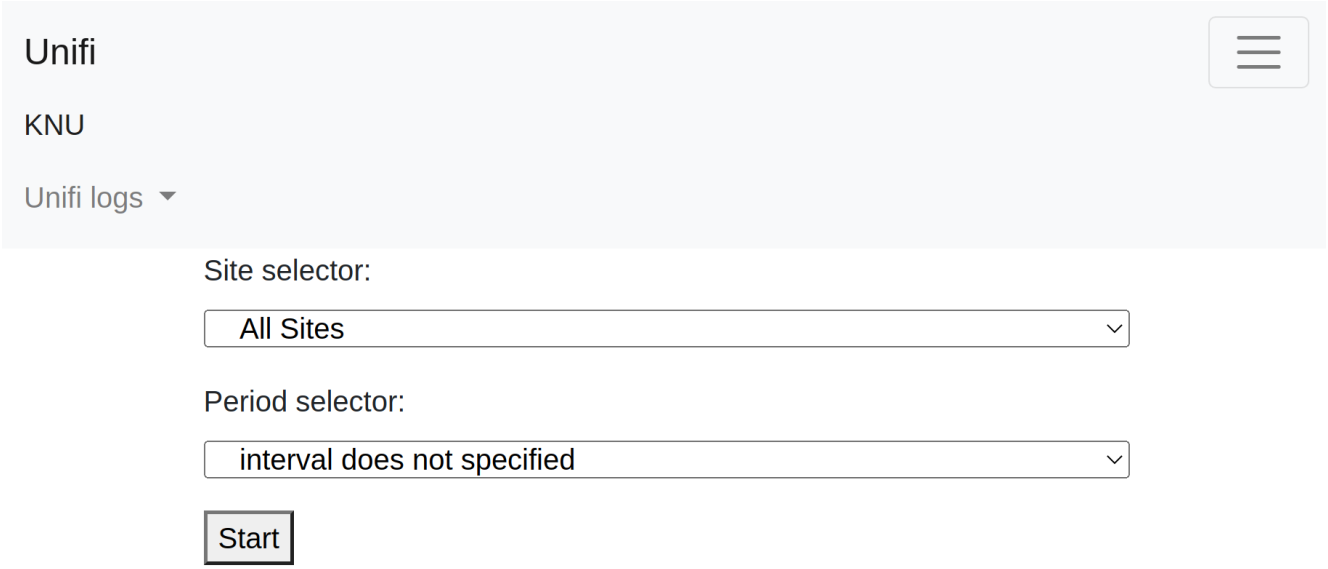


Рис.3.4 Головна сторінка веб-інтерфейсу

На головній сторінці реалізовано дві функції: вибір одного або декількох підрозділів (також реалізована функція вибору всіх підрозділів одразу) для запису користувачів (див. рис.3.5) та вибір інтервалу відпрацювання серверної програми (див. рис.3.6).

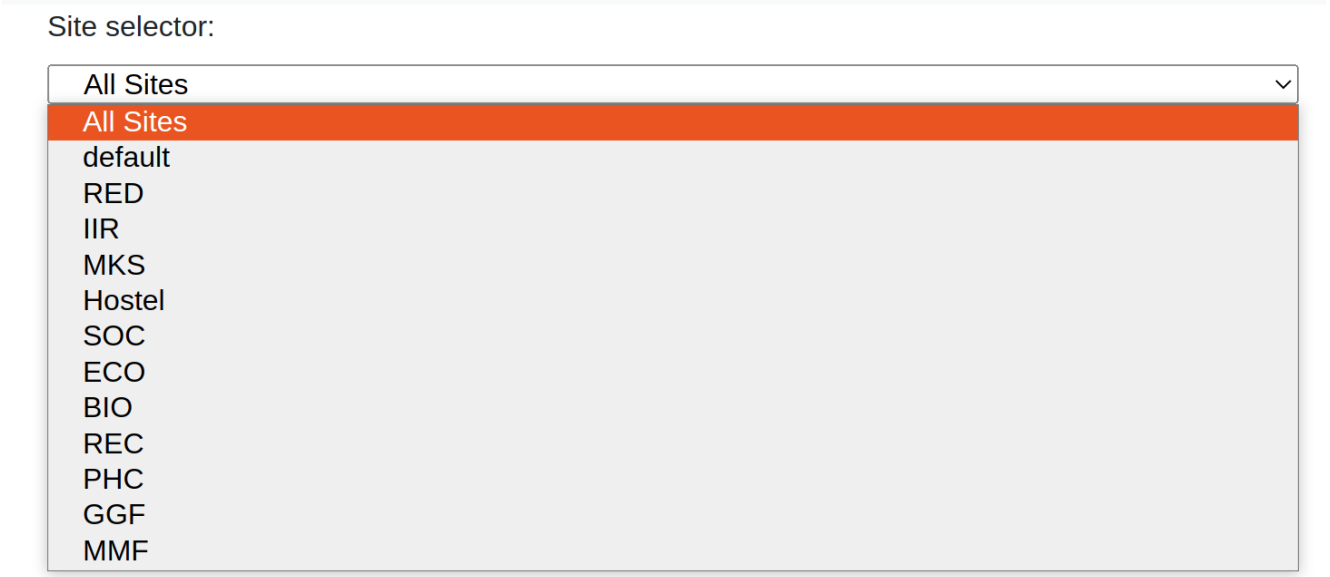


Рис.3.5 Випадаючий список із функцією вибору підрозділів


Рис.3.6 Випадаючий список із функцією вибору інтервалу відпрацювання серверного застосунку

При переході на вкладку Graph мережевий адміністратор отримує доступ до веб-сторінки з графіком та двома віджетами з вибором початкових та кінцевих дати та часу. На рис.3.7 можна побачити статистику користувачів на кожному підрозділі за два дні та загальну статистику по всьому університету.

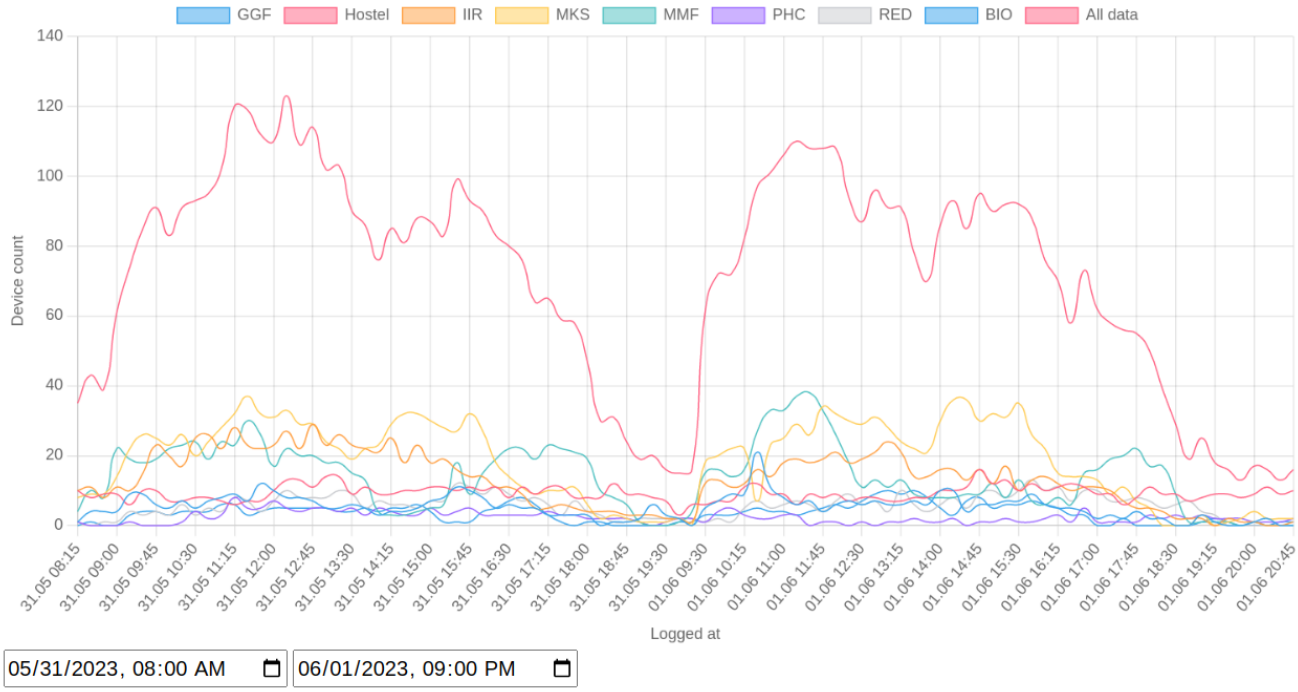


Рис.3.7 Статистика користувачів по кожному підрозділу та загальна по всьому університету

Як видно з графіку, зверху знаходяться позначення з кольоровими чекбоксами, кожен із яких відповідає певному підрозділу та загальній статистиці «All data». Чекбокси є динамічними та можуть бути відключені за потребою відобразити лише певні підрозділи. Так, наприклад, на Рис.3.8 можна побачити кореляцію загальної кількості підключених користувачів у мережі університету від кількості підключених користувачів на механіко-математичному факультеті. Також при наведенні на ключові точки на графіку з’являється лейбл із датою, часом та точною кількістю підключених користувачів для зручності орієнтування за графіком.

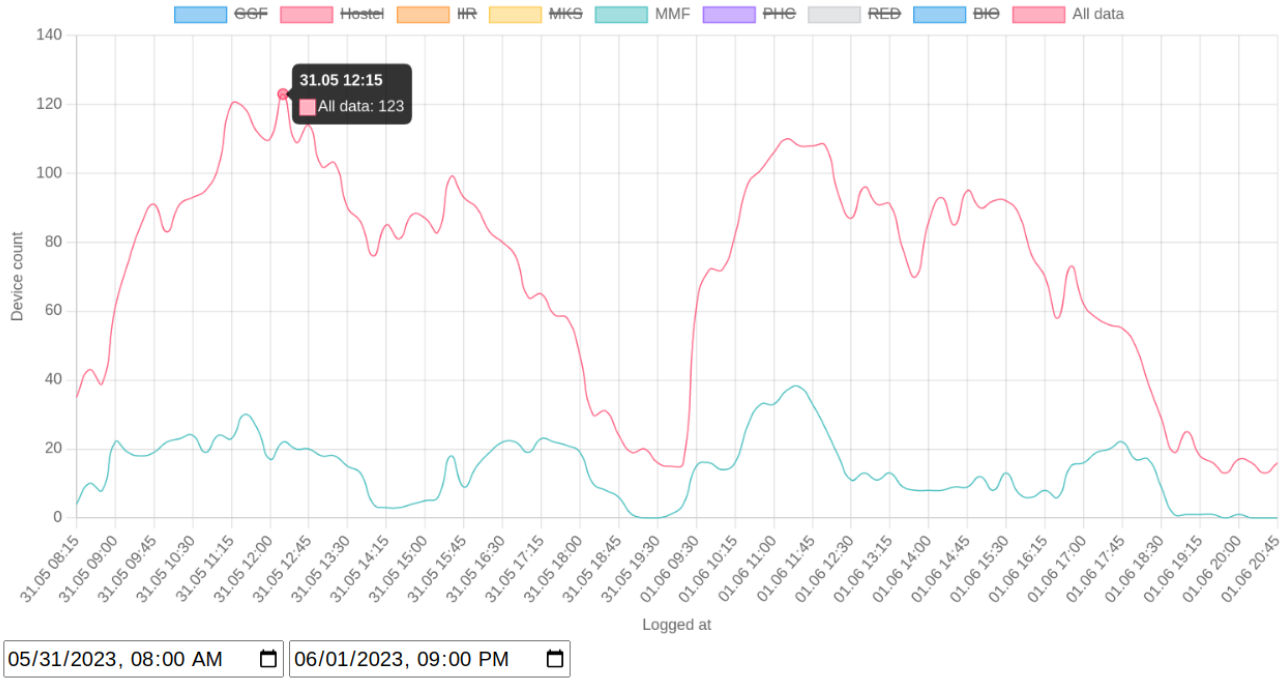


Рис.3.8 Відображення кореляції загальної кількості підключених користувачів у мережі університету від кількості підключених користувачів на механіко-математичному факультеті

Таким чином, можна виводити статистику за будь-який проміжок часу та обирати будь-які підрозділи, так як графік масштабується по осі ординат та робить зручним та наглядним порівняння обраних параметрів.

## 3.3 Система реагування на виявлені аномалії

Окрім динамічного виводу графіку у веб-інтерфейсі, розроблена функція реагування на виявлені аномалії в роботі DHCP-серверу. Реалізація полягає в детектуванні користувачів, які підключаються до мережі та не отримують IP-адресу більше, ніж дві хвилини та відправленні повідомлень про аномалії до телеграм-боту із вказанням підрозділу, MAC-адреси та точного часу в секундах, які користувач не отримував IP-адресу на момент відпрацювання серверної програми (див. рис.3.9).

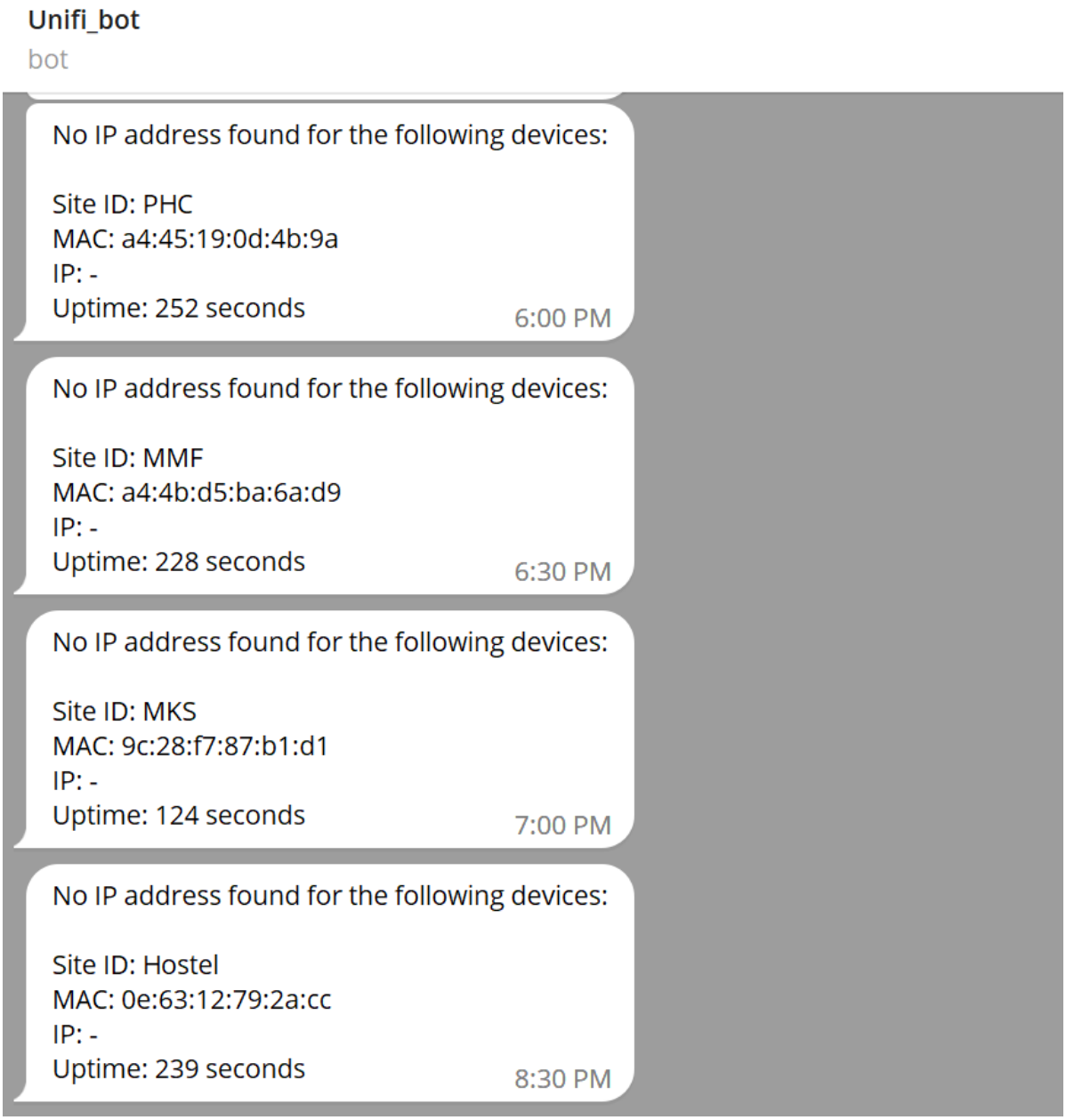


Рис.3.9 Приклад повідомлень про виявлені аномалії в роботі DHCP-серверу

## 3.4 Подальші вдосконалення розробленої системи

На даний момент планується імплементація розробленої системи до мережевого сектору, тому функціонал обмежений поточними потребами, які не може задовольнити існуюче рішення, на основі якого було створено описаний у даній роботі застосунок. У майбутньому планується розширення функціоналу, зокрема автоматизацію процесу більш детального виявлення помилок у роботі мережі та мережевого обладнання:

- замала кількість працюючих точок доступу;
- статистично аномально низька кількість підключених користувачів на певному підрозділі;
- окремі графіки підключень користувачів до кожної точки доступу в межах підрозділу;
- статистично аномально низька якість сигналу точки доступу.

Окрім вдосконалення та розширення функціональної складової розробленої системи, у подальшому можливе її імплементація в систему моніторингу Cacti, яка так само використовується для відслідковування стану мережі університету.

# ВИСНОВКИ

Аналіз принципів роботи існуючої системи моніторингу мережі університету дозволив зробити висновок, що функціонал веб-інтерфейсу використовуваного мережевого контролеру повною мірою не задовольняє потреби мережевого сектору та потребує доповнення додатковими функціями, які б спрощували виявлення та усунення несправностей у роботі мережі та мережевого обладнання.

Використовуючи API мережевого контролера Unifi задля отримання необхідної інформації про користувачів із її подальшими обліком та обробкою, розроблено механізм відслідковування кількості підключених користувачів до кожного підрозділу університету.

Розроблена система надає можливість мережевому інженеру не лише відслідковувати статистику поточного стану мережі, але й мати доступ до історії за будь-який проміжок часу та за потребою коригувати вивід даних у інтуїтивно зрозумілому зручному веб-інтерфейсі, зводячи до мінімуму потребу взаємодіяти з консольними командами. Крім того, створений застосунок передбачає можливість порівняння кількості користувачів серед різних підрозділів та з загальною кількістю по всьому університету, що може допомогти у ефективному розподілі точок доступу для запобігання перевантаження мережі та у подальшому може бути застосовано для автоматичного збору статистики та аналізу на рахунок аномалій у поведінці мереж та мережевого обладнання.

Продемонстровано принцип роботи системи виявлення аномалій на прикладі проблеми отримання IP-адреси кінцевими пристроями. Використовуючи сповіщення у телеграм-боті, які можуть бути переглянуті в реальному часі, досягається оперативне виявлення проблеми та можливість здійснити необхідні кроки для забезпечення своєчасного реагування на неї. Такий підхід допомагає як підтриманню безперебійної роботи мережі та підключених до неї пристроїв, так і запобіганню подальшим негативним наслідкам.

## ПЕРЕЛІК ПОСИЛАНЬ


1. Технологія Wi-Fi [Електронний ресурс]: «Cisco Systems, Inc.» Режим доступу URL: https://www.cisco.com/c/en/us/products/wireless/what-is-wifi.html (дата звернення: 17.05.2023).
2. Підключення пристроїв до точки доступу [Електронний ресурс]: Режим доступу URL: https://community.fs.com/blog/wireless-access-point-vs-router-what-are-the-differences.html (дата звернення: 17.05.2023).
3. Архітектурні обмеження REST [Електронний ресурс]: Режим доступу URL: https://www.ibm.com/topics/rest-apis (дата звернення: 18.05.2023).
4. Принцип роботи REST API [Електронний ресурс]: Режим доступу URL: https://mannhowie.com/rest-api (дата звернення: 18.05.2023).
5. Технологія MySQL [Електронний ресурс]: Режим доступу URL: https://www.mysql.com/ (дата звернення: 18.05.2023).
6. MySQL Reference Manual [Електронний ресурс]: Режим доступу URL: https://dev.mysql.com/doc/refman/8.0/en/ (дата звернення: 18.05.2023).
7. Backend frameworks popularity [Електронний ресурс]: Режим доступу URL: https://statisticsanddata.org/data/most-popular-backend-frameworks-2012–2022/ (дата звернення: 18.05.2023).
8. Django main functions [Електронний ресурс]: Режим доступу URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction (дата звернення: 18.05.2023).
9. Django URLs configuration [Електронний ресурс]: Режим доступу URL: https://docs.djangoproject.com/en/4.1/topics/http/urls/ (дата звернення: 22.05.2023).
10. Зовнішній скрипт Chart.js [Електронний ресурс]: Режим доступу URL: https://cdn.jsdelivr.net/npm/chart.js/dist/chart.umd.min.js (дата звернення: 22.05.2023).

11. Linux MySQL installation [Електронний ресурс]: Режим доступу URL: https://www.scaleway.com/en/docs/tutorials/setup-mysql/ (дата звернення: 22.05.2023).
12. Репозиторій на GitHub [Електронний ресурс]: Режим доступу URL: https://github.com/Lunnij/django-unifi-monitoring (дата звернення: 08.06.2023).

# ДОДАТОК А. ФАЙЛИ ЗІСТАВЛЕННЯ URL-АДРЕС

graph/urls.py

```
from django.urls import path
from django.views.generic import TemplateView
from.forms import chart

urlpatterns = [
path('', TemplateView.as_view(template_name='graph.html'),
name='index'),
path('bar-json/', chart, name='bar_json'),
]
```

unifi/urls.py

```
from django.contrib import admin
from django.urls import path, include
from. import views

urlpatterns = [
path('', views.index),
path('logger/', include('logger.urls')),
path('graph/', include('graph.urls')),
]
```

# ДОДАТОК Б. ФАЙЛИ МІГРАЦІЇ

0002_network_last_seen.py:

```
from django.db import migrations, models


class Migration(migrations.Migration):
dependencies = [
('logger', '0001_initial'),
]
operations = [
migrations.AddField(
model_name='network',
name='last_seen',
field=models.DateField(null=True),
),
  ]
```

0003_alter_network_last_seen.py:

```
from django.db import migrations, models


class Migration(migrations.Migration):
dependencies = [
('logger', '0002_network_last_seen'),
]
operations = [
migrations.AlterField(
model_name='network',
name='last_seen',
field=models.DateTimeField(null=True),
),
```

```
  ]
```

0004_alter_network_last_seen_alter_network_logged_at.py:

```
from django.db import migrations, models

class Migration(migrations.Migration):
dependencies = [
('logger', '0003_alter_network_last_seen'),
]
operations = [
migrations.AlterField(
model_name='network',
name='last_seen',
field=models.BigIntegerField(default=0),
),
   migrations.AlterField(
model_name='network',
name='logged_at',
field=models.DateTimeField(auto_now_add=True),
),
  ]
```

0005_alter_network_uptime.py:

```
from django.db import migrations, models

class Migration(migrations.Migration):
dependencies = [
('logger', '0004_alter_network_last_seen_alter_network_logged_at'),
]
operations = [
migrations.AlterField(
```

```
model_name='network',
name='uptime',
field=models.BigIntegerField(),
),
  ]
```

0006_remove_network_last_seen_remove_network_uptime.py:

```
from django.db import migrations

class Migration(migrations.Migration):
dependencies = [
('logger', '0005_alter_network_uptime'),
]
operations = [
migrations.RemoveField(
model_name='network',
name='last_seen',
),
   migrations.RemoveField(
model_name='network',
name='uptime',
),
  ]
```

0007_alter_network_ip.py:

```
from django.db import migrations, models

class Migration(migrations.Migration):
dependencies = [
('logger', '0006_remove_network_last_seen_remove_network_uptime'),
]
```

```
operations = [
migrations.AlterField(
model_name='network',
name='ip',
field=models.GenericIPAddressField(null=True),
),
  ]
```

# ДОДАТОК В. ФАЙЛ РОЗМІТКИ HTML

index.html:

```
<!doctype html>

<html lang=«en»>


<head>

<meta charset=«utf-8»>

<meta name=«viewport» content=«width=device-width, initial-scale=1, shrink-to-fit=no»>

<link rel=«stylesheet» href=«https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css»

integrity=«sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm» crossorigin=«anonymous»>


 {% block meta%}{% endblock%}


<title>Unifi KNU</title>

</head>


<body>

<!-- navbar -->

<nav class=«navbar navbar-expand-lg navbar-light bg-light»>

<a class=«navbar-brand» href="#">Unifi</a>

<button class=«navbar-toggler» type=«button» data-toggle=«collapse» data-target="#navbarSupportedContent»

aria-controls=«navbarSupportedContent» aria-expanded=«false» aria-label=«Toggle navigation»>

<span class=«navbar-toggler-icon»></span>

</button>
```

```
<div class=«collapse navbar-collapse» id=«navbarSupportedContent»>

<ul class=«navbar-nav mr-auto»>

<li class=«nav-item active»>

<a class=«nav-link» href=«https://unifi.noc.knu.ua:8443»>KNU <span
class=«sr-only»>(current)</span></a>

</li>


     <li class=«nav-item dropdown»>

<a class=«nav-link dropdown-toggle» href="#" id=«navbarDropdown»
role=«button»

data-toggle=«dropdown» aria-haspopup=«true» aria-expanded=«false»>

Unifi logs

       </a>

      <div class=«dropdown-menu» aria-labelledby=«navbarDropdown»>

<a class=«dropdown-item»
href=«http://localhost:8000/logger»>Logger</a>

<a class=«dropdown-item»
href=«http://localhost:8000/graph»>Graph</a>

</div>

           </li>

</ul>

    </div>

</nav>


<!-- main content implented from our templates -->

<div class=«container»>

{% block content%}{% endblock%}

</div>



<!-- Optional JavaScript -->
```

```
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
<script src=«https://code.jquery.com/jquery-3.2.1.slim.min.js»
integrity=«sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN»
crossorigin=«anonymous»></script>
<script src=«https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js»
integrity=«sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q»
crossorigin=«anonymous»></script>
<script src=«https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js»
integrity=«sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl»
crossorigin=«anonymous»></script>

 <!-- additional scripts of templates -->
{% block scripts%}{% endblock%}


</body>
</html>
```