Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Краснюка Михайла Андрійовича*
(ПІБ)

академічної групи *122-19-3*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Автоматизована система формування та контролю авторизації користувачів сервісу*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи				
розділів:				
спеціальний	*проф. Мороз Б.І.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)
М.О. Алексєєв
(підпис) (прізвище, ініціали)
« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу
бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-19-3* (група) *Краснюк М.А.* (прізвище та ініціали)

тема кваліфікаційної роботи *Автоматизована система формування та контролю авторизації користувачів сервісу*

затверджена наказом ректора НТУ «ДП» від *16.05.2023* № *350-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2023 р.***

Завдання видав _____ _____
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ _____
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *12,06.2023 р.*

РЕФЕРАТ

Пояснювальна записка: 65 с., 42 рис., 0 табл., 3 дод., 17 джерел.

Об'єкт розробки: автоматизована система формування та контролю авторизації користувачів сервісу.

Мета кваліфікаційної роботи: розробка ефективної автоматизованої системи формування та контролю авторизації користувачів, яка забезпечуватиме безпеку, надійність, зручність процесу авторизації та моніторингу. Головним завданням є створення програмного забезпечення, що дозволить ефективно ідентифікувати та аутентифікувати користувачів, контролювати їх доступ до різних ресурсів та застосунків.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформи для розробки, виконано проектування і розробка програми, описана робота програми, алгоритм і структура її функціонування, а також виклик та завантаження програми, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення полягає у створенні автоматизованої системи, що надає можливість електронного зберігання даних про ролі користувачів, реалізації авторизації користувачів у системі, ефективне обслуговування користувачів, стійкість системи, реалізація функціонала надання та позбавлення ролей.

Актуальність теми полягає у зростаючій потребі в надійній і безпечній авторизації користувачів у світі швидкого технологічного розвитку та зростання загроз кібербезпеці. Забезпечення безпеки, та ефективності авторизації є важливим завданням для багатьох організацій та підприємств.

Список ключових слів: КЛІЄНТ, СЕРВЕР, АВТОМАТИЗОВАНА СИСТЕМА, АВТОРИЗАЦІЯ, БАЗА ДАНИХ, ПРОЄКТУВАННЯ, КЕШ ПАМ’ЯТЬ, СЕРВІС, API, ФРЕЙМВОРК.

ABSTRACT

Explanatory Note: 65 p., 42 figs, 0 tabl., 3 apps., 17 sources.

Development Object: Automated system for user authentication and control in a service.

Objective of the Qualification Work: Development of an efficient automated system for user authentication and control that ensures security, reliability, convenience in the authentication and monitoring process. The main task is to create software that enables effective identification and authentication of users, control their access to various resources and applications.

The Introduction section provides an analysis of the problem's relevance and current state, specifies the objective of the qualification work and its application field, justifies the topic's significance, and defines the task's scope.

Chapter 1 analyzes the subject area, determines the task's relevance and purpose, formulates the problem statement, specifies requirements for software implementation, technologies, and tools.

Chapter 2 reviews existing solutions, selects development platforms, performs program design and development, describes the program's functionality, algorithm, and structure of its operation, as well as program invocation and loading, defines input and output data, and characterizes the parameters of technical devices.

The Economic section determines the labor intensity of the developed information system, calculates the cost of program development, and estimates the time required for its creation.

The practical significance lies in the creation of an automated system that allows electronic storage of user role data, implementation of user authorization in the system, efficient user service, system stability, and the functionality of role assignment and revocation.

The relevance of the topic is based on the increasing need for reliable and secure user authentication in a rapidly developing technological world with growing cybersecurity threats. Ensuring security and effectiveness in user authentication is an important task for many organizations and enterprises.

Keywords: CLIENT, SERVER, AUTOMATED SYSTEM, AUTHENTICATION, DATABASE, DESIGN, CACHE MEMORY, SERVICE, API, FRAMEWORK.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ОС – операційна система;

ЕОМ – електронно-обчислювальна машина;

API – це Application Programming Interface (з англ. інтерфейс для програмування застосунків);

ВСТУП

У сучасному цифровому світі, де онлайн-сервіси та додатки стають необхідними інструментами в повсякденному житті, питання безпеки та контролю доступу користувачів набувають особливої важливості. Автоматизована система формування та контролю авторизації користувачів сервісу є одним з ключових факторів, що забезпечують безпечне та надійне використання цих сервісів.

Метою даної кваліфікаційної роботи є розробка ефективної автоматизованої системи формування та контролю авторизації користувачів, яка забезпечуватиме безпеку, надійність, зручність процесу авторизації та моніторингу. Головним завданням є створення програмного забезпечення, що дозволить ефективно ідентифікувати та аутентифікувати користувачів, контролювати їх доступ до різних ресурсів та застосунків, а також забезпечувати відповідність прав доступу вимогам, обмеженням та політикам безпеки.

Актуальність теми полягає у зростаючій потребі в надійній і безпечній авторизації користувачів у світі швидкого технологічного розвитку та зростання загроз кібербезпеці. Забезпечення безпеки авторизації є важливим завданням для багатьох організацій та підприємств, які надають свої послуги в Інтернеті. Відсутність ефективної системи авторизації може призвести до витоку конфіденційної інформації, порушення прав доступу та збитків, пов'язаних з крадіжкою ідентифікаційних даних.

Завдання даної кваліфікаційної роботи полягають у конкретизації та реалізації системи формування та контролю авторизації користувачів сервісу. Це охоплює розробку та впровадження алгоритмів ідентифікації та аутентифікації, реалізацію механізмів контролю доступу, створення інтерфейсів для користувачів та адміністраторів, а також тестування та оцінку ефективності системи.

У результаті виконання даної кваліфікаційної роботи очікується отримання працездатної автоматизованої системи формування та контролю авторизації користувачів, яка зможе застосовуватися у різних галузях, забезпечуючи надійний та безпечний доступ до сервісів та додатків для користувачів.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

У сучасному світі, з постійним розвитком інформаційних технологій та зростаючою кількістю онлайн сервісів, проблема авторизації користувачів набуває особливої ваги. Безпека і конфіденційність важливої інформації стають основними пріоритетами для багатьох організацій та інтернет-сервісів. Автоматизована система формування та контролю авторизації користувачів сервісу є важливим кроком для забезпечення безпеки даних та захисту узі авторизації користувачів вказує на наявність різноманітних методів та рішень. Переважна більшість сучасних онлайн-сервісів використовує системи авторизації на основі комбінації логіну та пароля, а також від несанкціонованого доступу.

Аналіз аналогів в гал можливість двофакторної аутентифікації. Зокрема, популярними методами є використання сесійних токенів, OAuth-протоколу та інших технологій для забезпечення безпеки під час авторизації користувачів.

Однак, існують певні технічні сперечання та прогалини знань у цій галузі. Наприклад, проблемою є використання слабких паролів або повторних паролів користувачами, що може стати лакмусовим тестом для зловмисників. Крім того, з'явилися нові виклики, пов'язані зі збільшенням кількості мобільних пристроїв та підвищенням рівня їх безпеки.

Не всі вимоги до виробів або розробок наукового, організаційного або іншого характеру вже реалізовані повністю. Серед нездійснених вимог можна відзначити покращення механізмів двофакторної аутентифікації, розвиток систем одноразових паролів, використання біометричних даних для авторизації та впровадження інших новаторських методів безпеки.

Об'єктом, в якому використовується автоматизована система формування та контролю авторизації користувачів, може бути будь-який інтернет-сервіс або додаток, що має обмежений доступ та вимагає ідентифікації користувачів. Це можуть бути електронні комерційні платформи, соціальні мережі, онлайн-банкінг, системи управління вмістом тощо. Застосування програми або програмного виробу у таких об'єктах дозволяє забезпечити безпеку і надійність процесу авторизації користувачів та захисту їхніх даних.

Загальні відомості з предметної галузі демонструють значну важливість автоматизованої системи формування та контролю авторизації користувачів сервісу у сучасному інформаційному середовищі. Необхідність безпеки та захисту інформації зумовлює постійну потребу у вдосконаленні технологій авторизації та застосуванні нових методів для розв'язання проблеми авторизації користувачів.

У малих інфраструктурах, які з часом вимушенні із-за збільшення кількості користувачів, або збільшення функціонала вимушені масштабуватися, як правило виникає проблема з централізованою системою авторизації користувачів у системі. Ця децентралізована система авторизації користувачів, по-перше, має великий обсяг, через децентралізації ускладнюється моніторинг системи авторизації, по-друге, при подальшому розширенні збільшується як навантаження на БД, бо кожен сервіс самостійно займається перевіркою, та визначенням ролей по логіну та паролю, так і збільшується площа ураження такої критичної системи для інфраструктури як авторизація. Саме ці проблеми є ключовими та будуть вирішені у визначеній кваліфікаційній роботі.

1.2. Призначення розробки та галузь застосування.

Основна термінологія, що містить ключові слова:

- Автоматизована система: комплекс програмних компонентів, який забезпечує автоматизацію процесу формування та контролю авторизації користувачів.

- Формування авторизації: процес створення та налагодження механізмів, що дозволяють користувачам отримувати доступ до сервісу з використанням ідентифікаційних даних.

- Контроль авторизації: моніторинг та перевірка відповідності прав доступу користувачів у системі вимогам та обмеженням.

Причини виникнення необхідності розробки програмного забезпечення для автоматизації процесу формування та контролю авторизації користувачів сервісу полягають у зростаючій потребі в безпеці та захисті даних, а також у підвищенні ризику несанкціонованого доступу до інформації. Традиційні методи авторизації, які базуються на логіні та паролі, виявляються недостатньо ефективними у забезпеченні високого рівня безпеки. Тому розробка автоматизованої системи, що спрощує та забезпечує надійну авторизацію користувачів, є важливою для багатьох галузей.

Галузі, в яких може застосовуватися розроблена система, включають, але не обмежуються:

- Електронна комерція: система може забезпечити безпеку та захист особистих даних користувачів під час онлайн-покупок та оплати товарів і послуг.

- Фінансові установи: система може використовуватися для забезпечення безпечного доступу до банківських рахунків та фінансових операцій.

- Соціальні мережі: система може допомогти впровадити ефективний процес авторизації користувачів у соціальних мережах, що дозволить запобігти несанкціонованому доступу до особистих профілів.

- Онлайн-сервіси зберігання даних: система може забезпечувати безпеку під час доступу до важливих даних, які зберігаються у хмарних сервісах.

Застосування розробленої автоматизованої системи формування та контролю авторизації користувачів у цих галузях дозволить підвищити рівень безпеки та забезпечити надійну ідентифікацію користувачів у процесі доступу до різноманітних сервісів та додатків. Також сервіс буде розроблено для IT-інфраструктури за для того, щоб централізувати авторизацію у системі та маніпуляцію з правами доступу, а також створення нових прав у випадку розширення. Сервіси інфраструктури замість децентралізованої роботи з правами будуть звертатись до через протокол HTTP до автоматизованої системи формування та контролю авторизації та будуть отримувати список певних прав користувача за його логіном та паролем що був попередньо захешований алгоритмом sha. Таким чином, за принципом мінімізації поверхні атаки, буде створено вузьке місце в інфраструктурі, що допоможе суттєво знизити ризик кібератаки на систему авторизації та полегшити її моніторинг завдяки доданому функціоналу адміністрування.

1.3. Підстави для розробки.

Підставою для розробки кваліфікаційної роботи бакалавра на тему «автоматизована система формування та контролю авторизації користувачів сервісу» є наказ по Національному технічному університету «Дніпровська політехніка» від __.__. 2023р. № ____-_.

1.4. Постановка завдання.

Метою даної кваліфікаційної роботи є розробка централізованої автоматизованої системи формування та контролю авторизації користувачів сервісу. Система має на меті спростити процес авторизації користувачів і забезпечити безпеку доступу до сервісу.

Для досягнення поставленої мети необхідно виконати наступні етапи:

- Дослідити існуючі методи та підходи до автоматизації процесу авторизації користувачів у веб-сервісах та додатках.
- Спроектувати структуру та архітектуру автоматизованої системи формування та контролю авторизації користувачів.
- Визначитись з API сервісу та з вихідною інформацією.
- Розробити програмне забезпечення, яке буде виконувати функції формування та контролю авторизації користувачів.
- Розробити таблиці у БД, де буде зберігатись дані про користувачів, їх ролі та інша інформація.
- Реалізувати функціонал для реєстрації нових користувачів, включаючи збереження їхніх особистих даних та інформації для авторизації.
- Розробити механізми для перевірки автентичності користувачів під час авторизації, використовуючи захищені протоколи та шифрування даних.
- Розробити систему управління правами доступу користувачів, яка дозволить надавати різні рівні доступу в залежності від їхніх ролей та повноважень.
- Забезпечити механізми для збереження та захисту конфіденційної інформації користувачів, включаючи паролі та інші облікові дані.
- Реалізувати механізми журналювання та моніторингу активності користувачів, що дозволять виявляти потенційні загрози безпеці.

Під час роботи будуть вирішені наступні завдання:

- Вивчити сучасні технології та методи автоматизації процесу авторизації користувачів.
- Визначити основні вимоги до функціональності та безпеки системи.
- Спроектувати структуру та архітектуру системи з урахуванням вимог та потреб користувачів.
- Розробити програмне забезпечення системи, використовуючи сучасні технології розробки та безпеки.
- Провести тестування системи для перевірки її функціональності та надійності.
- В результаті роботи над кваліфікаційною роботою очікується отримання працездатної та безпечної автоматизованої системи формування та контролю авторизації користувачів, яка забезпечить зручний та безпечний доступ до сервісу.

1.5. Вимоги до програми або програмного виробу.

1.5.1. Вимоги до функціональних характеристик.

Програмний продукт acl_simple, повинен мати два інтерфейси, для адміністратора та клієнта сервісу централізованої авторизації. Програма буде відрізняти інтерфейси за допомогою url. Для клієнтських запитів буде /customer, для адміністратора /admin.

Інтерфейс для клієнта повинен мати такий функціонал:

- авторизація користувача у системі по захешованому паролю та логіну;
- отримання списку ролей авторизованого користувача.

Інтерфейс для адміністратора повинен мати такий функціонал:

- авторизація адміністратора у системі по захешованому паролю та логіну;
- створення нового користувача;
- видалення користувача;
- переглянути всіх існучих користувачів у системі;
- додати ролі користувачу;
- видалити ролі користувача;
- переглянути ролі користувача;
- переглянути всі можливі ролі у системі;
- створити нові ролі;
- видалити ролі із системи;

Організація вхідних та вихідних даних буде виконана використанням технології http протоколу. Дані будуть передаватися у тілі post-запиту у форматі json.

1.5.2. Вимоги до інформаційної безпеки.

Автоматизована система формування та контролю авторизації користувачів сервісу повинна задовольняти наступні вимоги до інформаційної безпеки:

1. Забезпечення сталого функціонування. Система повинна бути стійкою до впливу зовнішніх та внутрішніх загроз, таких як хакерські атаки, збої апаратного забезпечення, програмні помилки тощо.

2. Контроль вхідної та вихідної інформації. Система повинна забезпечувати перевірку та фільтрацію вхідних даних для запобігання вразливостям, таким як ін'єкції SQL. Механізми перевірки цілісності та автентифікації повинні бути встановлені для вихідної інформації, що надсилається користувачами.

3. Час відновлення після відмови. Система повинна мати механізми для виявлення та відновлення після відмови, щоб забезпечити мінімальний час перерви у функціонуванні системи. Регулярні резервні копії даних та наявність запланованих процедур відновлення допоможуть відновити систему до працездатного стану після відмови.

4. Захист від несанкціонованого доступу. Система повинна мати механізми ідентифікації та автентифікації користувачів, щоб забезпечити доступ лише авторизованим особам.

5. Забезпечення контролю та цілісності даних. Система повинна мати механізми контролю доступу до даних, щоб забезпечити, що тільки авторизовані користувачі мають доступ до конфіденційної інформації. Застосування методів шифрування для зберігання та передачі даних допоможе запобігти несанкціонованому доступу до них. Регулярне аудитування та моніторинг доступу до даних допоможуть виявити можливі порушення безпеки та забезпечити цілісність інформації.

Ці вимоги до інформаційної безпеки допоможуть забезпечити надійне функціонування системи та захистити конфіденційність та цілісність даних.

1.5.3. Вимоги до складу та параметрів технічних засобів.

Автоматизована система формування та контролю авторизації користувачів сервісу вимагає наявності певного складу технічних засобів, які забезпечують її функціонування та задовольняють встановлені вимоги. Нижче наведено основні технічні характеристики та параметри, які потрібно врахувати при розробці системи:

Сервери:

1. Система потребує наявності серверів для зберігання, обробки та надання доступу до даних.

2. Кількість серверів і їх конфігурація повинні бути достатніми для забезпечення швидкодії та масштабованості системи. Оскільки в нас не великий обсяг обробатуємих даних, буде достатньо одного сервера.

3. Параметри серверів, такі як обсяг пам'яті, кількість ядер процесора, швидкість мережі, повинні бути достатніми для ефективної обробки запитів користувачів та забезпечення швидкодії системи. У нашому випадку буде достатньо 8 ядер процесора Intel Core i7, операційна система Linux Ubuntu 22.04.2 LTS, відеокарта Intel(R) Xe Graphics (TGL GT2), ОЗУ 25 Гб.

4. Програмне забезпечення: мова Erlang 25.0.4 Ubuntu jammy 64 bit, компілятор Rebar3 з github репозиторію.

Бази даних:

1. Система вимагає наявності баз даних для зберігання інформації про користувачів, прав доступу та інші відповідні дані.

2. Вибір технології бази даних повинен враховувати вимоги до швидкодії, масштабованості та безпеки, тому вибір пав на СУД PostgreSQL.

3. Параметри бази даних, такі як обсяг дискового простору, швидкість операцій вводу-виводу, резервне копіювання та відновлення, повинні відповідати обсягу та потребам системи. Обсяг диску 1 Тб, ОС Linux Ubuntu 22.04.2 LTS, відеокарта Intel(R) Xe Graphics (TGL GT2), процесор Intel Core i7, ОЗУ 25 Гб.

Мережеве обладнання:

1. Для забезпечення зв'язку між компонентами системи та користувачами необхідно наявність мережевого обладнання.

2. Параметри мережевого обладнання, такі як пропускна здатність, швидкість передачі даних, стійкість до перебоїв, повинні бути достатніми для забезпечення стабільної та швидкої роботи системи. Кабель віта пара, та підтримка високошвидкісного Інтернету.

1.5.4. Вимоги до інформаційної та програмної сумісності.

Для успішної реалізації автоматизованої системи формування та контролю авторизації користувачів сервісу, важливо встановити вимоги до інформаційних структур, вихідних кодів, мов програмування та програмних засобів. У даній роботі розробка системи здійснюється на мові програмування Erlang, тому необхідно врахувати особливості цієї мови та вимоги щодо інформаційної та програмної сумісності.

Вимоги до інформаційних структур.

Для зберігання інформації про користувачів та їх авторизаційні дані використовується база даних. Вимоги до бази даних включають:

- Ефективність: база даних повинна забезпечувати швидкий доступ до інформації та оптимальні запити.
- Надійність: база даних повинна бути захищеною від втрати даних та відновлюватись у разі виникнення помилок чи збоїв.
- Скальованість: база даних повинна бути здатною масштабуватись для обробки зростаючого обсягу даних та навантаження.

Вимоги до програмного коду.

Код системи має бути написаний мовою програмування Erlang з використанням принципів функціонального програмування та акторної моделі.

Код повинен бути добре структурованим, легким для розуміння та модифікації, має бути дотримано принципів чистоти та ефективності.

Вимоги до мов програмування та програмних засобів.

Мова програмування Erlang версії 25.0.4 повинна бути встановлена та налаштована на комп'ютері розробника та на сервері, де буде виконуватись код.

Розробка та тестування системи може здійснюватись з використанням різних інтегрованих середовищ розробки (IDE), таких як Erlang/OTP, IntelliJ

IDEA, Emacs Erlang Mode або Visual Studio Code з додатком Erlang Extension. Для розробки автоматизованої системи формування та контролю авторизації користувачів сервісу буде використано IDE IntelliJ IDEA.

Вимоги до інтеграції з іншими системами.

Система повинна мати можливість взаємодіяти з іншими сервісами або додатками шляхом використання стандартних протоколів комунікації, таких як HTTP, REST або SOAP. Для цього проєкту буде використано HTTP, та форматування тіла запиту - JSON.

РОЗДІЛ 2.
ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи.

Функціональне призначення системи полягає у тому, що необхідно розробити сервіс для IT-інфраструктури за для того, щоб централізувати ідентифікацію, аутентифікацію та авторизацію у системі та маніпуляцію з правами доступу, а також створення нових прав у випадку розширення. Сервіси інфраструктури замість децентралізованої роботи з правами доступу будуть звертатись через протокол HTTP до автоматизованої системи формування та контролю авторизації та будуть отримувати список певних прав користувача за його логіном та паролем що був попередньо захешований алгоритмом sha. Таким чином, за принципом мінімізації поверхні атаки, буде створено вузьке місце в інфраструктурі, що допоможе суттєво знизити ризик кібератаки на критичну систему авторизації та полегшити її моніторинг завдяки доданому функціоналу адміністрування та логів.

2.2. Опис застосованих математичних методів.

У розробленій системі не використовуються математичні методи.

2.3. Опис використаних технологій та мов програмування.

Розглянемо мови програмування їх переваги та недоліки:

Erlang

Мова програмування Erlang була розроблена в 1986 році компанією Ericsson для використання у великих телекомунікаційних системах. Вона названа на честь норвезького математика і інженера Агнера Ерланга, який

вніс значний внесок у розробку теорії чергових систем, яка була використана в проектуванні телекомунікаційних мереж.

Ерланг був спочатку створений для побудови розподілених систем з високою надійністю, здатних обробляти великі обсяги паралельних операцій та забезпечувати м'яке відновлення після збоїв. Основні принципи, на яких ґрунтується Erlang, включають наступні:

1. Функціональне програмування: Erlang є чисто функціональною мовою, що означає, що вона спрямована на вирішення задач шляхом виконання функцій. Це дозволяє писати компактний та експресивний код, легкий у розумінні та тестуванні.

2. Паралельне та розподілене програмування: Erlang має вбудовану підтримку паралельного виконання та розподілених обчислень. Вона використовує легковагові процеси (lightweight processes), які управляються віртуальною машиною Erlang (BEAM). Це дозволяє створювати велику кількість процесів з невеликими витратами ресурсів та забезпечує ефективну обробку паралельних завдань.

3. Висока надійність: Erlang був спроєктований з орієнтацією на системи, які потребують високої надійності та доступності. Вона має вбудовані засоби для відновлення від збоїв, керування помилками та моніторингу процесів. Також вона має механізми для гарантованої доставки повідомлень та керування розподіленою конкуренцією.

Переваги Erlang:

1. Висока продуктивність та швидкодія: Erlang розроблена для обробки великого обсягу паралельних операцій, що дозволяє досягати високої продуктивності та ефективності розподіленого виконання.

2. Висока надійність та доступність: Erlang має вбудовані механізми для обробки збоїв та відновлення системи без перерви у роботі. Вона також підтримує гарантовану доставку повідомлень, що забезпечує надійну комунікацію між процесами.

3. Складність системи: Erlang дозволяє легко побудувати складні системи з використанням конкурентних процесів та повідомлень. Вона має вбудовані засоби для керування конкуренцією та розподіленою обробкою.

Недоліки Erlang:

1. Обмежені вибір інструментів та бібліотек: Хоча Erlang має декілька потужних інструментів та бібліотек, екосистема навколо неї менш розгалужена порівняно з іншими популярними мовами програмування. Це може призвести до необхідності розробки деяких компонентів самостійно або використання додаткових інструментів, які можуть бути менш оптимальними для Erlang.

2. Складність у навчанні: Erlang має свою специфіку та вимагає певного часу та зусиль для освоєння. Оскільки вона відрізняється від більшості інших мов програмування, розробникам може знадобитись час, щоб засвоїти її концепції та підходи.

3. Обмежена підтримка графічного інтерфейсу користувача: Erlang не є першим вибором для розробки графічного інтерфейсу користувача (GUI). Вона більше спрямована на серверні та розподілені застосування. Хоча існують інструменти для розробки GUI на Erlang, вони можуть бути менш розвинутими або менш популярними.

Elixir

Elixir - це мова програмування, яка побудована на віртуальній машині Erlang (BEAM) і має синтаксис, який нагадує Ruby. Ось короткий екскурс в історію, переваги та недоліки мови Elixir:

Мова Elixir була розроблена Хосе Валімом (José Valim) і вперше випущена у 2011 році. Вона була створена з метою поєднати потужність функціонального програмування з можливостями конкурентного та розподіленого програмування, які надає віртуальна машина Erlang. Elixir

швидко здобула популярність серед розробників, які шукали ефективний спосіб розробки масштабованих та надійних додатків.

Переваги мови Elixir:

1. Стійкість та надійність: Elixir ґрунтується на віртуальній машині Erlang, яка зарекомендувала себе у великих, розподілених системах з високим рівнем доступності. Це робить Elixir ідеальним вибором для розробки надійних та стійких додатків.

2. Паралельність та конкурентність: Elixir надає високорівневі абстракції для паралельного та конкурентного програмування. Завдяки механізмам акторів та співробітників (actors and supervisors), розробники можуть легко створювати розподілені та високопродуктивні додатки.

3. Складний додаток: Elixir надає розробникам можливість будувати складні додатки з легкістю завдяки концепції OTP (Open Telecom Platform). OTP пропонує готові шаблони для побудови надійних, масштабованих та стійких додатків.

4. Простий синтаксис: Синтаксис Elixir є зрозумілим і дружнім для розробників. Він нагадує синтаксис мови Ruby, що полегшує процес навчання та розробки.

Недоліки мови Elixir:

1. Недостатня кількість бібліотек: Одним з недоліків Elixir є те, що в порівнянні з іншими мовами програмування, екосистема бібліотек може бути менш розвиненою. Однак, ця ситуація поступово змінюється, і все більше бібліотек та фреймворків стають доступними.

2. Висока витривалість пам'яті: У віртуальній машині Erlang, на якій ґрунтується Elixir, є деякі обмеження на витривалість пам'яті. Це може призвести до накопичення сміття та зменшення продуктивності додатка.

3. Відсутність повноцінного набору інструментів: У порівнянні з деякими іншими мовами програмування, Elixir може бути не таким добре підтримуваним у сфері інструментів розробки та інфраструктури.

LFE(Lisp Flavored Erlang)

LFE (Lisp Flavored Erlang) - це мова програмування, яка поєднує функціональні можливості мови Erlang з синтаксисом Lisp. Ось короткий екскурс в історію, переваги та недоліки мови LFE:

Мова LFE була розроблена Робом Ван Ліндтом (Robert Virding) та Луїсом Гаспарі (Luis Gaspari) і вперше випущена в 2008 році. LFE створена з метою надати розробникам можливість використовувати синтаксис Lisp для розробки додатків, які працюють на віртуальній машині Erlang.

Переваги мови LFE:

1. Функціональність Erlang: LFE використовує віртуальну машину Erlang, що дозволяє розробникам використовувати всі функціональні можливості Erlang, такі як розділення на процеси, акторська модель та масштабованість.

2. Синтаксис Lisp: Синтаксис LFE базується на Lisp, що дає можливість використовувати потужні функціональні конструкції, такі як макроси, інтроспекція та маніпуляція деревами виразів.

3. Інтеграція з Erlang: LFE підтримує пряму інтеграцію з кодом Erlang, що дозволяє використовувати бібліотеки та фреймворки, написані на Erlang.

4. Розширюваність: Завдяки синтаксису Lisp, LFE є дуже розширюваною мовою. Розробники можуть використовувати макроси для створення нових мовних конструкцій та абстракцій.

Недоліки мови LFE:

1. Обмежена спільнота: У порівнянні з іншими мовами програмування, спільнота розробників LFE є менш чисельною. Це може обмежити доступність підтримки та ресурсів для навчання та розвитку.

2. Недостатня документація: Офіційна документація для LFE може бути обмеженою, що може ускладнити навчання та розробку в мові.

Joxa

Joxa - це мова програмування, спрямована на розробку розподілених та паралельних додатків. Ось короткий екскурс в історію, переваги та недоліки мови Joxa:

Мова Joxa була розроблена Кліффом Мунро (Cliff Moon) та Джо Різ (Joe Williams) і була представлена у 2010 році. Вона була створена з метою розширення мови Erlang, надання простого та ефективного способу програмування розподілених додатків, які працюють на віртуальній машині Erlang.

Переваги мови Joxa:

1. Інтеграція з Erlang: Joxa побудована на основі віртуальної машини Erlang, що дозволяє використовувати всі переваги Erlang, такі як масштабованість, розділення на процеси та акторська модель.
2. Простий синтаксис: Joxa використовує простий та лаконічний синтаксис, що дозволяє розробникам швидко писати код і знижує ймовірність помилок.
3. Розподілені можливості: Мова Joxa надає розробникам потужні засоби для розробки розподілених додатків. Вона підтримує механізми розподіленого обчислення та взаємодії з іншими вузлами мережі.

Недоліки мови Joxa:

1. Обмежена спільнота та ресурси: Через свою невелику популярність, мова Joxa має обмежену спільноту розробників та документацію. Це може ускладнити навчання та розвиток в мові.
2. Обмежена підтримка інструментів: На сьогоднішній день Joxa не має такого широкого спектру інструментів та бібліотек, які доступні для мови Erlang.

Clojerl

Clojerl - це мова програмування, яка поєднує функціональну мову Clojure з платформою Erlang. Ось короткий екскурс в історію, переваги та недоліки мови Clojerl:

Мова Clojerl була розроблена з метою поєднання потужностей Clojure, функціональної мови програмування для JVM, з масштабованістю та надійністю платформи Erlang. Розробка Clojerl розпочалася у 2011 році, і з того часу вона продовжує активно розвиватись та вдосконалюватись.

Переваги мови Clojerl:

1. Синтаксис Clojure: Clojerl використовує синтаксис мови Clojure, що дає змогу розробникам використовувати потужні функціональні конструкції, зокрема, замикання, неизмінні дані та потокові операції.

2. Інтеграція з Erlang: Clojerl може безпосередньо використовувати існуючі бібліотеки та фреймворки, створені для Erlang, що дає можливість розробникам використовувати багато функціональних можливостей Erlang, таких як акторська модель та масштабованість.

3. Масштабованість та надійність: Завдяки платформі Erlang, Clojerl може працювати в розподіленому середовищі з підтримкою багатопоточності та обробки повідомлень. Це забезпечує високу масштабованість та надійність додатків.

Недоліки мови Clojerl:

1. Обмежена спільнота та ресурси: У порівнянні з популярними мовами програмування Clojure та Erlang, Clojerl має меншу спільноту розробників та обмежений вибір бібліотек та інструментів.

2. Складність інтеграції: Хоча Clojerl має здатність використовувати Erlang-бібліотеки, інтеграція може бути складнішою та вимагати додаткових зусиль, особливо для розробників, які не мають досвіду з Erlang.

3. Відсутність повної сумісності з Clojure: Незважаючи на те, що Clojerl базується на Clojure, вона не є повністю сумісною з Clojure, і можуть виникати нюанси та відмінності у синтаксисі та функціональності.

Чому було обрано саме Erlang:

1. Стабільність та надійність: Erlang має довгу історію в розробці систем реального часу та телекомунікаційних додатків. Вона була проектована для забезпечення високої доступності та надійності, зокрема в розподілених середовищах. Ця надійність робить її привабливою для розробки критичних за надійністю систем.

2. Вбудована підтримка паралельності та розподіленості: Erlang має вбудовану підтримку акторської моделі програмування, що дозволяє легко створювати та керувати багатопоточними та розподіленими додатками. Це дозволяє ефективно використовувати багатоядерні та розподілені обчислювальні ресурси.

3. Масштабованість: Erlang має вбудовані механізми масштабування, такі як можливість запуску та керування багатьма процесами, розподіленим сховищем даних та механізмом обміну повідомленнями. Це робить її ефективною для побудови високонавантажених та масштабованих систем.

4. Зручність в розробці розподілених додатків: Erlang має вбудовані механізми для обміну повідомленнями та керування процесами на віддалених вузлах, що спрощує розробку розподілених додатків. Це дозволяє легко створювати системи, які працюють на різних вузлах та комунікують між собою.

5. Масивна стандартна бібліотека: Erlang має багатий набір стандартних бібліотек, які надають різноманітні функціональні можливості для розробки додатків. Це включає в себе бібліотеки для мережевого

програмування, роботи з процесами та потоками, обробки XML та багато інших.

Хоча Elixir, LFE, Clojerl і Joxa побудовані на базі Erlang та мають свої переваги, Erlang все ще залишається сильним вибором для розробки систем реального часу та розподілених додатків завдяки своїм особливостям, надійності та зручності в розробці.

У розробці важливим є вибір фреймворку. Для подальшої ефективної та якісної розробки було обрано фреймворк OTP (Open Telecom Platform).

OTP (Open Telecom Platform) є набором бібліотек, шаблонів та інструментів, які розширюють можливості мови Erlang та допомагають у створенні розподілених та надійних додатків. OTP надає готові компоненти, такі як супервізори (supervisors), генератори процесів (gen_servers) та інші, що спрощують розробку та підтримку системи авторизації. Використання OTP дозволяє стандартизувати розподілену обробку даних та керування процесами у системі.

OTP був створений компанією Ericsson у 1996 році для підтримки розробки телекомунікаційних систем з високою надійністю та масштабованістю. Згодом він став доступним у відкритому вигляді та знайшов застосування у багатьох інших галузях, де вимагається побудова надійних та розподілених систем.

Серед переваг OTP можна виділити:

1. Надійність: OTP надає набір засобів для розробки надійних систем, включаючи механізми обробки винятків, керування життєвим циклом процесів, відновлення після відмови та моніторингу системи.
2. Масштабованість: OTP має підтримку для розподіленого обчислення, що дозволяє легко масштабувати додатки на декількох вузлах та використовувати їх ресурси ефективно.

3. Підтримка паралельного виконання: OTP надає концепцію акторів, яка дозволяє легко створювати та керувати багатопоточними додатками, що сприяє ефективному використанню багатоядерних систем.

4. Високорівневі абстракції: OTP надає високорівневі абстракції, такі як генеричні сервери, супервізори та реєстри процесів, що спрощують розробку та управління додатками.

Недоліки OTP:

1. Складність у навчанні: Початкове вивчення OTP може бути дещо складним, оскільки вимагає розуміння концепцій акторів та інших абстракцій, які відхиляються від традиційних підходів до програмування.

2. Обмежена екосистема: У порівнянні з іншими популярними мовами та фреймворками, екосистема OTP може бути менш розгалуженою, що може вплинути на доступність сторонніх бібліотек та ресурсів.

Не зважаючи на недоліки, OTP залишається потужним інструментом для розробки надійних та розподілених додатків, особливо у сферах, де вимагається висока доступність та масштабованість.

Також у сервісі буде використано веб-фреймворк **cowboy**.

Cowboy є високопродуктивним HTTP-сервером та веб-фреймворком для мови Erlang. Він надає зручні інструменти для розробки веб-додатків, включаючи маршрутизацію, обробку запитів та відповідей, підтримку веб-сокетів та багато іншого. Cowboy обраний для розробки back-end частини системи авторизації, оскільки він забезпечує швидкодію та ефективність в обробці HTTP-запитів.

Cowboy був розроблений у 2010 році командою розробників з компанії Nine Nines. Він почався як експериментальний проєкт з метою створення швидкого та ефективного веб-сервера, що використовує переваги мови Erlang. Згодом Cowboy набув популярності серед розробників Erlang-

додатків та став одним з основних виборів для веб-розробки на платформі Erlang.

Переваги Cowboy:

1. Швидкість та продуктивність: Cowboy є високопродуктивним веб-фреймворком, який працює на високому рівні продуктивності завдяки оптимізаціям та ефективному використанню мови Erlang та OTP.

2. Простота використання: Cowboy надає простий та зрозумілий API, що дозволяє розробникам швидко створювати веб-додатки та API. Він має гнучку архітектуру, яка дозволяє легко розширювати та налаштовувати його функціонал.

3. Підтримка протоколів: Cowboy підтримує різні веб-протоколи, такі як HTTP, WebSockets, SPDY та інші, що дозволяє розробникам створювати різноманітні веб-додатки та сервіси.

Недоліки Cowboy:

1. Обмежена екосистема: Cowboy, хоч і є популярним фреймворком, має менш розгалужену екосистему порівняно з іншими веб-фреймворками. Це може вплинути на доступність сторонніх бібліотек та ресурсів.

2. Відсутність підтримки масштабування: Cowboy не надає вбудованих механізмів масштабування та розподіленого виконання. Для цих завдань можуть знадобитись додаткові компоненти та інструменти.

Незважаючи на недоліки, Cowboy залишається популярним веб-фреймворком в середовищі Erlang завдяки своїй швидкості, простоті використання та гнучкій архітектурі. Він є хорошим вибором для розробки веб-додатків та API на платформі Erlang.

Для проєкту acl_simple була обрана СУБД PostgreSQL для зберігання та управління даними. Опис СУБД PostgreSQL та обґрунтування вибору цієї системи можуть бути наступними:

PostgreSQL є потужною об'єктно-реляційною системою керування базами даних (СУБД), яка надає широкі можливості для зберігання, організації та оптимізації даних. Вона пропонує розширений набір функцій, що включають підтримку SQL запитів, транзакційності, індексування, виконання збережених процедур, реплікацію даних та багато іншого.

Обґрунтування вибору PostgreSQL:

1. Надійність: PostgreSQL є однією з найбільш надійних СУБД, вона володіє механізмами забезпечення цілісності даних, відновлення після збоїв та захисту від втрати даних. Це важливо для системи, яка обробляє авторизацію користувачів та зберігає чутливу інформацію.

2. Масштабованість: PostgreSQL може працювати з великими обсягами даних та високими навантаженнями. Вона підтримує горизонтальне та вертикальне масштабування, що дозволяє розширювати систему під потреби росту обсягів даних та кількості користувачів.

3. Розширені можливості: PostgreSQL має багатий набір функцій та розширень, що дозволяють реалізовувати різноманітні завдання. Вона підтримує розширення SQL, географічні дані, розподілену обробку, повнотекстовий пошук та інші розширені можливості, які можуть бути корисними для реалізації системи авторизації користувачів.

4. Відкритість та спільнота: PostgreSQL є відкритим програмним забезпеченням з активною спільнотою розробників та користувачів. Це означає наявність безлічі документації, підтримки, патчів та оновлень, що забезпечує стабільність та розвиток системи в майбутньому.

5. Широке використання: PostgreSQL є однією з найпопулярніших вільних СУБД, вона використовується багатьма великими компаніями та проєктами. Це означає наявність експертів, готових допомогти з розробкою та підтримкою системи.

Обґрунтування вибору PostgreSQL в даній роботі може базуватися на вищезгаданих перевагах, а також на потребі в потужній, надійній та

розширюваній СУБД для забезпечення функціональності автоматизованої системи формування та контролю авторизації користувачів сервісу.

2.4. Опис структури системи та алгоритмів її функціонування.

У даному проєкті ми будемо використовувати **клієнт-серверну архітектуру**. Вона є широко використовуваною моделлю в розробці програмного забезпечення, включаючи серверні додатки та веб-системи. Вона передбачає взаємодію між двома типами компонентів: клієнтами й серверами.

У клієнт-серверній архітектурі, клієнти є користувачами або додатками, які взаємодіють з системою та надсилають запити до сервера. Сервери, зі свого боку, обробляють ці запити та забезпечують відповіді клієнтам. Ця модель передбачає розділення обов'язків між клієнтською та серверною стороною, що дозволяє досягти більшої модульності, масштабованості та керованості системи.

Структура проєкту aqcl_simple зазначена на рисунку 2.4.1. Вона була згенерована завдяки інструменту rebar3 та має три директорії: config, _build та apps/acl_simple. Config містить у собі конфігурацію до проєкту. Apps/acl_simple своєю чергою має декілька директорій напряму пов’язані із кодом та його тестами:

- “src”. Містить у собі самі модулі, в яких описана програма.
- “include”. Містить hrl скрипт де прописані базові макроси, що інкапсулюються у кожен модуль для їхнього використання.
- “test”. В ній знаходяться директорії, скрипти та модулі в яких описані тести для перевірки вірної роботи проєкту. На приклад common-тести та eunit-тести.

- “wrk”. В ній знаходяться скрипти написані на мові lua, для навантажувального тестування системи.

Рис. 2.4.1 Структура проєкту.

При процесі запуску програми усі скомпільовані файли, а саме сторонні бібліотеки, тести, сама програма acl_simple, завантажуються у директорію _build.

Архітектура проєкту побудована на базі акторної моделі.

Акторна модель є концепцією паралельного програмування, яка використовується в мові програмування Erlang та багатьох інших мовах, і є ключовим аспектом фреймворка OTP.

Акторна модель базується на ідеї процесів (акторів), які взаємодіють один з одним шляхом обміну повідомленнями. Кожен актор має свій внутрішній стан і може виконувати певні дії у відповідь на отримані повідомлення. Актори можуть створювати нові актори, надсилати їм повідомлення і відповідати на отримані повідомлення.

Є декілька основних принципів акторної моделі:

Ізоляція. Кожен актор має власне просторове обмеження, що означає, що його стан і поведінка є приватними. Взаємодія між акторами відбувається виключно через обмін повідомленнями.

Асинхронність. Актори взаємодіють асинхронно, що означає, що вони можуть надсилати повідомлення і продовжувати свою роботу без очікування відповіді.

Безпечність. Кожен актор має власний приватний стан і не може бути безпосередньо модифікований іншими акторами. Взаємодія між акторами відбувається шляхом обміну копіями повідомлень.

Відновлення збоїв. Актори можуть відновлюватися після збоїв, оскільки вони зберігають свій внутрішній стан. Це робить акторну модель відмінною для розробки надійних та стійких до збоїв систем.

Схему акторів у проєкті acl_simple можна зобразити таким чином як на рисинку 2.4.2.

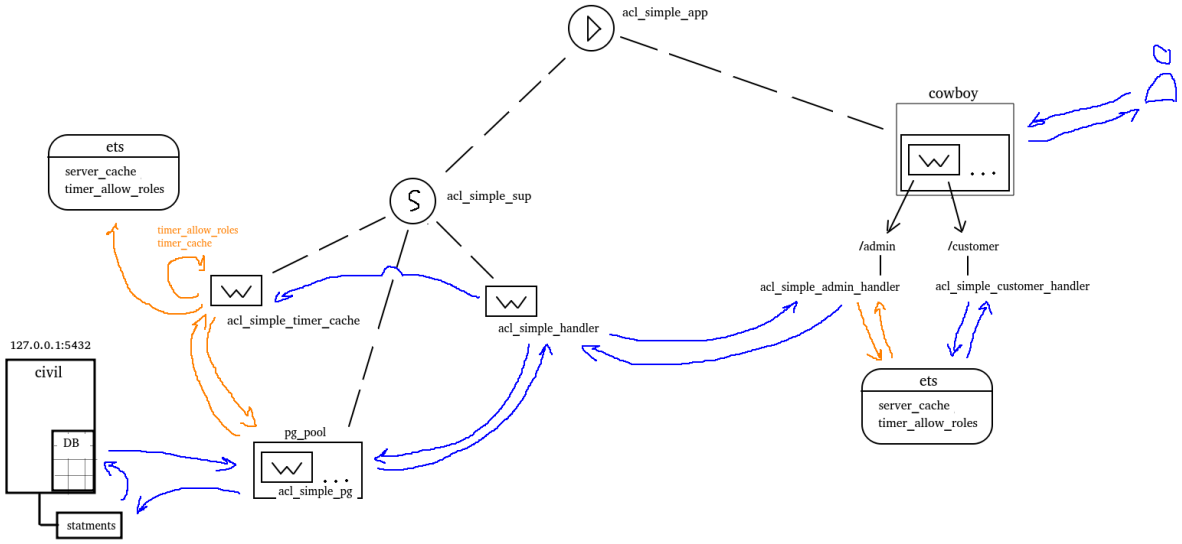


Рис. 2.4.2 Схема акторів.

Є кореневий актор, що реалізовано у модулі acl_simple_app, з якого починається робота програми. Цей процес створює актор-супервайзер, таблицю acl_simple в ETS-пам'яті, де буде зберігатися увесь кеш проєкту, та ініціалізує роботу веб-фреймворка cowboy, вказуючи порт, який виділяє додаток під себе в операційній системі, та url із вказаним модулем де є зворотня функція init/2 для подальшої обробки запиту. У випадку закінчення роботи програми, актор acl_simple_app надсилає повідомлення створеним акторам про закінчення роботи, а після цього закінчує роботу.

Актор реалізований у модулі acl_simple_sup є супервайзером. Він ініціалізується актором acl_simple_app. Він створює та моніторить такі воркери як acl_simple_server, acl_simple_timer_cache та ініціалізує пул pg_pool, процеси якого працюють по сценарію що зазначений у модулі acl_simple_pg.

Супервайзер (Supervisor) є важливою складовою фреймворка OTP (Open Telecom Platform) в мові програмування Erlang. Він використовується для керування та контролю за процесами в системі.

Супервайзер є спеціальним процесом, який відповідає за створення, запуск, моніторинг та відновлення інших процесів, відомих як дочірні процеси. Він забезпечує високу доступність та надійність системи, розв'язувати проблеми, пов'язані з відмовою процесів. Саме він є базисом парадигми “Let it crush”. Замість того, щоб намагатися передбачити всі можливі помилки та відновлювати систему вручну, парадигма "let it crash" пропонує дозволити виникнення помилок та недоліків і використовувати механізми в OTP для автоматичного відновлення системи.

Є декілька основних характеристик супервайзера.

Створення процесів. Супервайзер може створювати дочірні процеси під своїм керівництвом. Ці процеси можуть виконувати різні функції у системі.

Моніторинг. Супервайзер встановлює моніторинг за дочірніми процесами, що дозволяє йому виявляти їх відмови або недоступність.

Відновлення. У разі відмови дочірнього процесу, супервайзер може відновити його або перезапустити, виконавши задану стратегію відновлення.

Стратегії відновлення. Супервайзер може мати різні стратегії відновлення для дочірніх процесів, такі як перезапуск, заміна або припинення. Це дозволяє системі адаптуватись до помилок та продовжувати свою роботу.

Воркер acl_simple_timer_cache є процесом-таймером, що синхронізує кеш с базою даних. Він генерує два таймери: timer_cache та timer_allow_roles.

Timer_cache - надсилає sql-запити до бази даних, та оновлює дані у кеш-пам’яті. Обробляє отриманні дані формуючи мапу, де ключ - це логін користувача, а значення - його ролі. Також обробляє отриманні дані формуючи мапу захешованих паролів користувачів.

Timer_allow_roles - надсилає sql-запити до бази даних, та оновлює дані у кеш-пам’яті. Отримує всі існуючі ролі системи, та зберігає їх у кешу.

Воркер acl_simple_timer_cache також має функцію refresh_cache/0, що експортується. Вона надсилає сповіщення воркеру, що примушую терміново оновити кеш.

Структура бази даних реалізована на рисунку 2.4.3. Вона має чотири таблиці, що містять у собі дані, необхідні для роботи проєкту acl_simple: users, roles, allow_roles, admins.

Таблиця users має у собі такі стовпці:

- id. В ній зберігається згенерований на боці сервера токен uuid, що ідентифікує користувача у таблиці roles.
- name. Логін, який є ідентифікатором при процесі ідентифікації користувача в системі.
- passhash. Захешований алгоритмом sha пароль користувача, завдяки якому іде процес аутентифікації користувача. Зберігається у вигляді 20 чисел.

Таблиця roles має такі стовпці:

- user_id. Токен, що є ідентифікатором користувача.
- role. Роль, яка була надана користувачу. Завдяки цій колонці проходить операція авторизації клієнта - визначення прав.

Таблиця allow_roles має один стовпець role. Там зберігаються усі можливі ролі у системі авторизації користувачів. Своєю чергою у таблиці admins, зберігається логін адміністратора та його пароль у захешованому вигляді за алгоритмом sha.

Рис. 2.4.3. Структура БД.

2.5. Обґрунтування та організація вхідних та вихідних даних програми.

У рамках розробленої автоматизованої системи формування та контролю авторизації користувачів сервісу, вхідні дані мають бути отримані від користувачів системи.

Сервіс буде отримувати вхідні дані у тілах post-запитів протоколу http формату json. Тіло для запитів на будь-який url буде складатись із двох записів, авторизації (“auth”:{}) та параметрів (“parameters”:{}). У записі авторизації будуть поля захешованого паролю користувача (“passhash”) та його логіном (“login”). Пароль буде перевірено на складність та захешовано завдяки алгоритму sha на боці front-end. У записі параметрів буде зазначено поле методу, що є ідентифікатором операції з ролями, та інших аргументів що необхідні для цієї операції. Приклад тіла зазначено на рисунку 2.5.1.

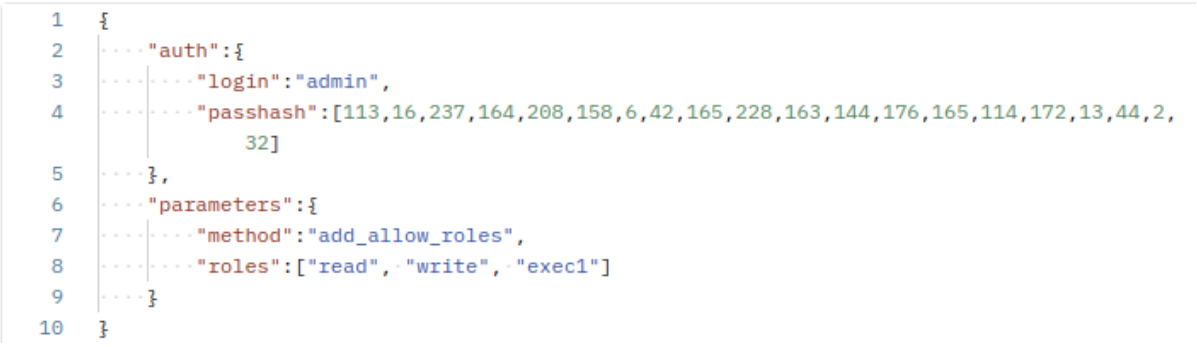


Рис. 2.5.1. Приклад тіла запиту.

Результатом програми роботи сервісу є при необхідності взаємодія із базою даних та відправлення відповіді користувачу у тілі post-запиту протоколу http формату json. Відповідь може бути успішною, чи відправити користувачу помилку. Приклад успіху виконаної операції зазначено на рисунку 2.5.2.

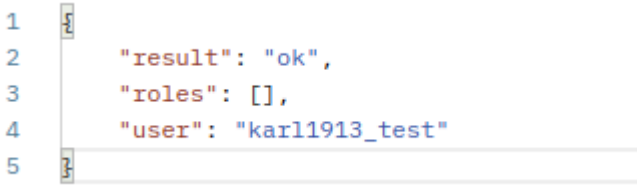


Рис. 2.5.2. Приклад успішної відповіді.

2.6. Опис розробленої системи.

2.6.1. Використані технічні засоби.

Для роботи даного продукту буде використано ноутбук компанії Dell, із процесором Intel Core i7, відеокартою Intel(R) Xe Graphics (TGL GT2) та з ОЗУ у розмірі 25 Гб, накопичувач на жорсткому диску обсягом 1 Тб, клавіатура та маніпулятор типу “миш”.

Вказані технічні засоби не мають бути обов’язково ідентичних характеристик для безперешкодної роботи системи.

2.6.2. Використані програмні засоби.

Для роботи даного продукту було встановлено і використано такі програмні засоби:

- Операційна система Linux Ubuntu 22.04.2 LTS;
- Мова Erlang 25.0.4 Ubuntu jammy;
- Компілятор Rebar3 з github репозиторію;
- Утиліти операційної системи Linux make та git;
- СУБД postgresql, та додаток pgAdmin4;
- IDE IntelliJ IDEA.

IntelliJ IDEA - це інтегроване середовище розробки (Integrated Development Environment, IDE), розроблене компанією JetBrains, яке надає широкі можливості для розробки програмного забезпечення. Воно підтримує багато мов програмування, включаючи Erlang.

IntelliJ IDEA надає зручний та потужний інтерфейс розробки для мови Erlang. Воно має багато корисних функцій, таких як підсвічування синтаксису, автодоповнення коду, налагодження, підтримка систем контролю версій та інше. Завдяки своїм функціям, IntelliJ IDEA полегшує процес розробки на мові Erlang, допомагає збільшити продуктивність розробника та забезпечує зручну роботу з проєктом.

2.6.3. Виклик та завантаження програми.

Спосіб виклику програми. Через термінал двома способами:

1. Створення релізу, та запуск скомпільованого файлу acl_simple, через консоль. Команди:

$ make rel

$ cd _build/prod/rel/acl_simple/bin/

$ acl_simple console

2. Запуск завдяки створеному bash-скрипту run.sh:

$ chmod +x run.sh

$ run.sh

Спосіб виклику common-тестів:

$ rebar3 ct --spec apps/acl_simple/test/common_tests/ct.spec

Спосіб виклику навантажувального тестування:

$ wrk -t2 -c3 -d1m -R5700 -s apps/acl_simple/wrk/show_all_users.lua http://127.0.0.1:1913

Обсяг не скомпільованого проєкту 103 Kb. Скомпільованого 70,8 Mb.

2.6.4. Опис інтерфейсу користувача.

У програмі є два інтерфейси, для клієнта з url customer, та для адміністратора з url admin. Розглянемо інтерфейс клієнта.

Інтерфейс користувача має лише один функціонал, а саме метод get_roles, який повертає список ролей що притаманні користувачу. На рисунку 2.6.4.1 зображено приклад авторизації користувача - отримання списку ролей.

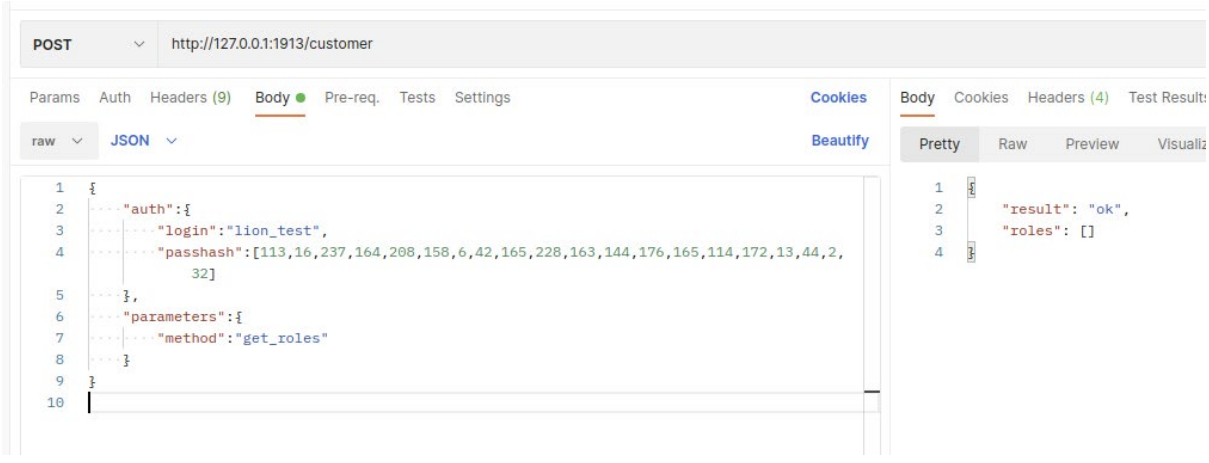


Рис. 2.6.4.1. Приклад авторизації користувача.

Також можна подивитися логування дій на сервері. Логування дії авторизації користувача зазначено на рисунку 2.6.4.2.

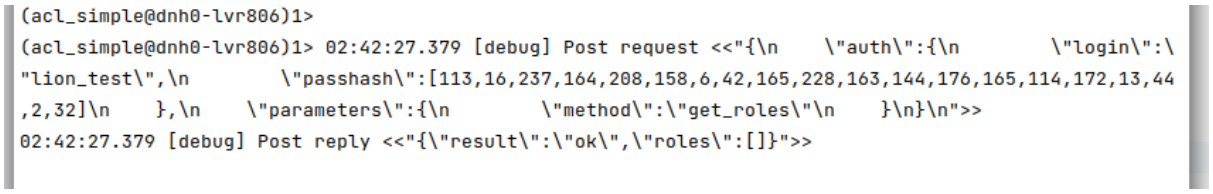


Рис. 2.6.4.2. Приклад логування при авторизації користувача.

У випадку якщо буде не вірно введений пароль, чи логін програма видасть помилку у відповіді користувачу як на рисунку 2.6.4.3, та зафіксує помилки у журналі на рисунку 2.6.4.4.

Рис. 2.6.4.3. Відповідь не пройденої аутентифікації.

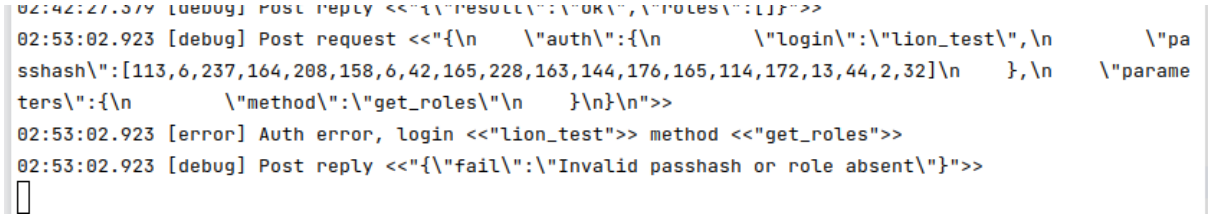


Рис. 2.6.4.4. Логування помилки аутентифікації.

У випадку не вірного вводу, не закриття дужок, чи не зазначення обов’язкових записів, буде виведена помилка як на рисунку 2.6.4.5. Логування помилки зазначено на рисунку 2.6.4.6.

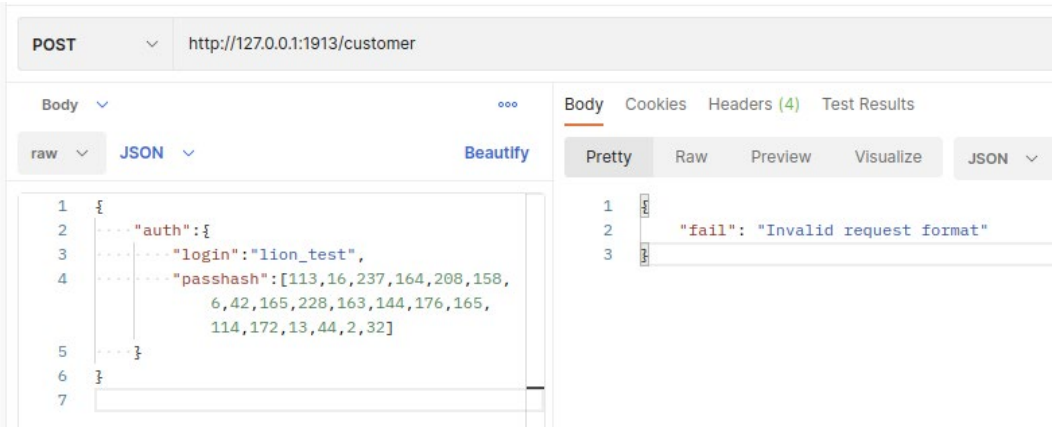


Рис. 2.6.4.5. Помилка не вірного вводу користувачем.

```
02:58:06.043 [debug] Post request <<"{\n    \"auth\":{\n        \"login\":\"lion_test\",\n        \"pa
sshash\":[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,32]\n    }\n}\n">>
02:58:06.043 [error] Parse map error #{<<"auth">> => #{<<"login">> => <<"lion_test">>,<<"passhash">> =
> [113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,32]}} Reason {badmatch,#{<<"aut
h">> => #{<<"login">> => <<"lion_test">>,<<"passhash">> => [113,16,237,164,208,158,6,42,165,228,163,14
4,176,165,114,172,13,44,2,32]}}}
02:58:06.043 [debug] Post reply <<"{\"fail\":\"Invalid request format\"}">>
```

Рис. 2.6.4.6. Логування помилки не вірного вводу користувачем.

Розглянемо інтерфейс адміністратора. Інтерфейс адміністратора має великий функціонал методів роботи із користувачами, ролями користувачів та ролями самої системи.

Show_all_users.

Після успішної аутентифікації адміністратора в системі, завдяки параметрам запису “auth”, проходить ідентифікація методу. Метод show_all_users отримує із кеш-пам’яті усіх користувачів системи, та формує відповідь. Приклад запиту, та відповіді зображено на рисунку 2.6.4.7.

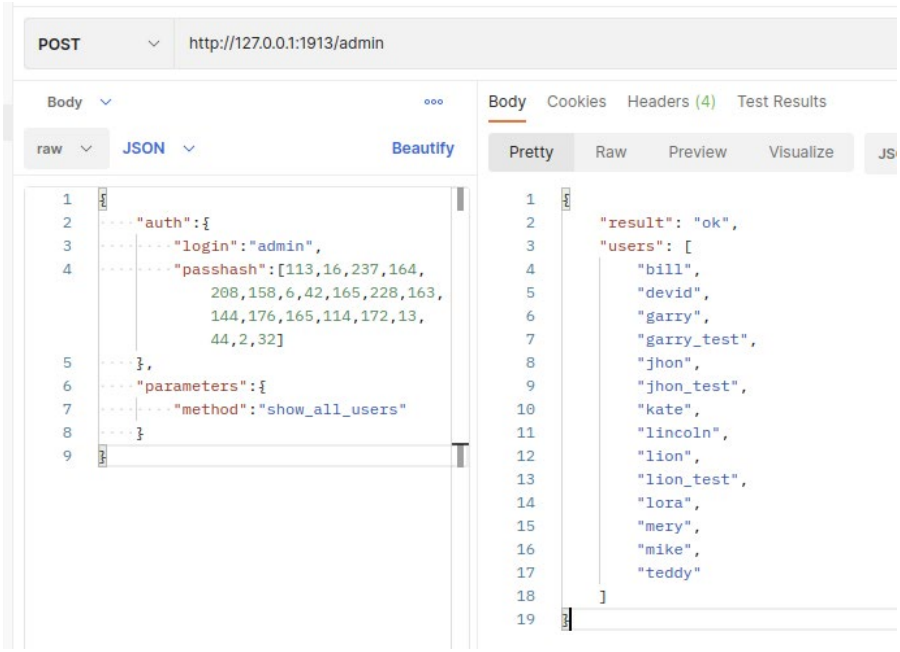


Рис. 2.6.4.7. Приклад успішної роботи методу show_all_users.

Також для подальшого моніторингу сервіс займається логуванням отриманих запитів, помилок, та інших дій (рис. 2.6.4.8).

```
03:11:43.239 [debug] Post request <<"{\n    \"auth\":{\n        \"login\":\"admin\",\n        \"passha
sh\":[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,32]\n    },\n    \"parameter
s\":{\n        \"method\":\"show_all_users\"\n    }\n}">>
03:11:43.239 [debug] Post reply <<"{\"result\":\"ok\",\"users\":[\"bill\",\"devid\",\"garry\",\"garry_
test\",\"jhon\",\"jhon_test\",\"kate\",\"lincoln\",\"lion\",\"lion_test\",\"lora\",\"mery\",\"mike\",\
"teddy\"]}">>
----- ----- -----
```

Рис. 2.6.4.8. Приклад логування успішної роботи методу show_all_users.

Метод із базою даних не взаємодіє.

User_delete.

Після успішної аутентифікації адміністратора в системі, завдяки параметрам запису “auth”, проходить ідентифікація методу. Метод user_delete видаляє користувача із системи, проводячи необхідні перевірки з існуючими користувачами, взаємодіє з базою даних, та

видаляє користувача і всі його ролі із кеш-пам’яті. Формує успішну відповідь. Приклад запиту, та відповіді зображено на рисунку 2.6.4.9.

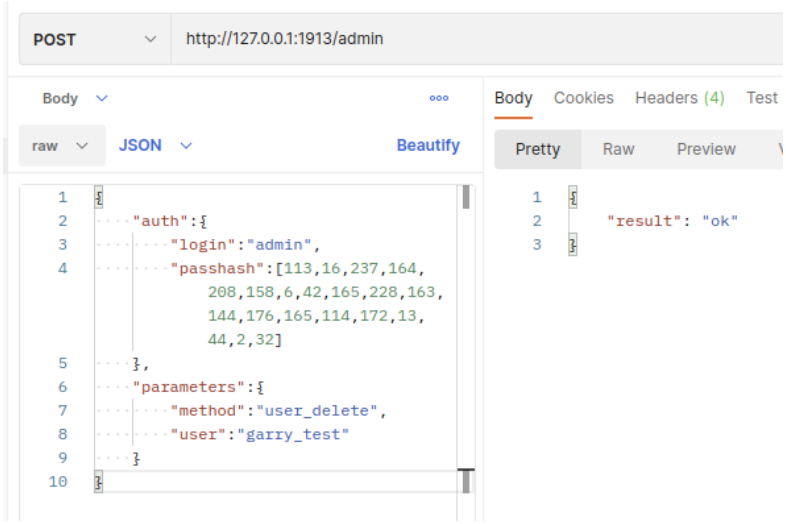


Рис. 2.6.4.9. Приклад успішної роботи методу user_delete.

Також для подальшого моніторингу сервіс займається логуванням отриманих запитів, помилок, та інших дій (рис. 2.6.4.10).

```
----- ----- -----
03:15:56.146 [debug] Post request <<"{\n    \"auth\":{\n        \"login\":\"admin\",\n        \"passha
sh\":[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,32]\n    },\n    \"parameter
s\":{\n        \"method\":\"user_delete\",\n        \"user\":\"garry_test\"\n    }\n}">>
03:15:56.149 [debug] Post reply <<"{\"result\":\"ok\"}">>
----- ----- -----
```

Рис. 2.6.4.10. Приклад логування успішної роботи методу user_delete.

Переглядаючи базу даних після операції методу user_delete, ми не побачимо видаленого користувача garry_test (рис. 2.6.4.11).

```
1  SELECT * FROM users;
```

Data Output | Messages | Notifications

	id [PK] character varying (50)	name character varying (50)	passhash json
1	c19884d2f99711ed968af4ee08f04a03	lora	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
2	c9fca5e8ffcf11eda130f4ee08f04a03	lion_test	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
3	d65e9d20f99711ed846df4ee08f04a03	lincoln	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
4	eb6d810ef99711ed8f45f4ee08f04a03	bill	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
5	1ce66aa2f99811edbe1bf4ee08f04a03	lion	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
6	0537fd8af99811ed883ff4ee08f04a03	teddy	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
7	d263bd58f96b11eda225f4ee08f04a03	mike	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
8	1022246ef99811edba7df4ee08f04a03	garry	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
9	4b2f191ef7f311eda33cf4ee08f04a03	jhon_test	[213,241,46,83,161,130,192,98,182,191,48,193,68,81,83,250,255,18,38,1...
10	6d659418f99711eda382f4ee08f04a03	jhon	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
11	73e9458cf99711edae4ef4ee08f04a03	devid	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
12	a4a64562f99711edbc83f4ee08f04a03	mery	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
13	a9648654f99711eda24af4ee08f04a03	kate	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...

Рис. 2.6.4.11. Перегляд зміненої БД після операції методу user_delete.

User_add.

Після успішної аутентифікації адміністратора в системі, завдяки параметрам запису “auth”, проходить ідентифікація методу. Метод user_add створює користувача у системи, проводячи необхідні перевірки з користувачами що вже існують, взаємодіє з базою даних, та додає користувача, що не має ролей до кеш-пам’яті. Формує успішну відповідь. Приклад запиту, та відповіді зображено на рисунку 2.6.4.12.

POST http://127.0.0.1:1913/admin

Body raw JSON Beautify

```
{
    "auth":{
        "login":"admin",
        "passhash":[113,16,237,164,208,
            158,6,42,165,228,163,144,
            176,165,114,172,13,44,2,32]
    },
    "parameters":{
        "method":"user_add",
        "user":"garry_test",
        "passhash":[113,16,237,164,208,
            158,6,42,165,228,163,144,
            176,165,114,172,13,44,2, 32]
    }
}
```

Body Cookies Headers (4) Test Re

Pretty Raw Preview Visu

```
{
    "result": "ok"
}
```

Рис. 2.6.4.12. Приклад успішної роботи методу user_add.

Також для подальшого моніторингу сервіс займається логуванням отриманих запитів, помилок, та інших дій (рис. 2.6.4.13).

```
03:20:32.809 [debug] Post request <<"{\n    \"auth\":{\n        \"login\":\"admin\",\n        \"passha
sh\":[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,32]\n    },\n    \"parameter
s\":{\n        \"method\":\"user_add\",\n        \"user\":\"garry_test\",\n        \"passhash\":[113,1
6,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2, 32]\n    }\n}">>
03:20:32.813 [debug] Post reply <<"{\"result\":\"ok\"}">>
----- ----- -----
```

Рис. 2.6.4.13. Приклад логування успішної роботи методу user_add.

Переглядаючи базу даних після операції методу user_add, ми побачимо новоствореного користувача garry_test (рис. 2.6.4.14).

```
1  SELECT * FROM users;
```

Data Output | Messages | Notifications

	id [PK] character varying (50)	name character varying (50)	passhash json
1	c19884d2f99711ed968af4ee08f04a03	lora	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
2	c9fca5e8ffcf11eda130f4ee08f04a03	lion_test	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
3	d65e9d20f99711ed846df4ee08f04a03	lincoln	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
4	eb6d810ef99711ed8f45f4ee08f04a03	bill	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
5	1ce66aa2f99811edbe1bf4ee08f04a03	lion	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
6	0537fd8af99811ed883ff4ee08f04a03	teddy	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
7	d263bd58f96b11eda225f4ee08f04a03	mike	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
8	1022246ef99811edba7df4ee08f04a03	garry	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
9	4b2f191ef7f311eda33cf4ee08f04a03	jhon_test	[213,241,46,83,161,130,192,98,182,191,48,193,68,81,83,250,255,18,38,1...
10	0169e226044311eeb898f4ee08f04a...	garry_test	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
11	6d659418f99711eda382f4ee08f04a03	jhon	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
12	73e9458cf99711edae4ef4ee08f04a03	devid	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
13	a4a64562f99711edbc83f4ee08f04a03	mery	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...
14	a9648654f99711eda24af4ee08f04a03	kate	[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,...

Рис. 2.6.4.14. Перегляд зміненої БД після операції методу user_add.

Show_roles.

Після успішної аутентифікації адміністратора в системі, завдяки параметрам запису “auth”, проходить ідентифікація методу. Метод

show_roles отримує із кеш-пам’яті усі надані ролі користувачу системи, та виводить їх у відповіді. Приклад запиту, та відповіді зображено на рисунку 2.6.4.15.

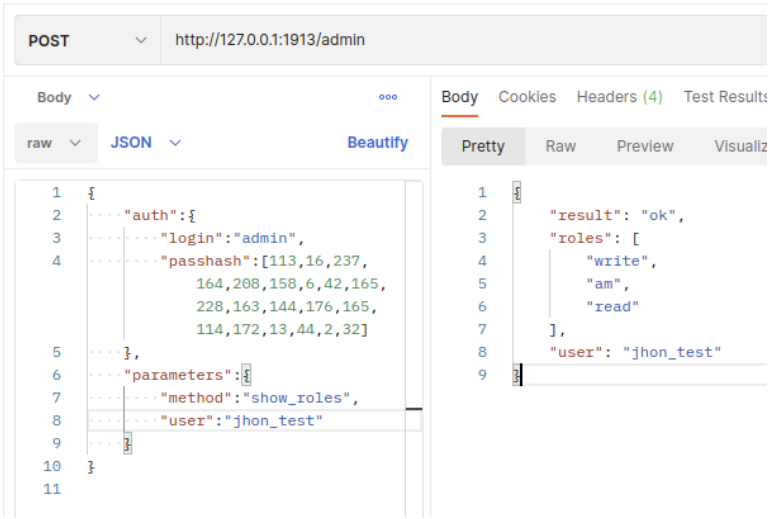


Рис. 2.6.4.15. Приклад успішної роботи методу show_roles.

Також для подальшого моніторингу сервіс займається логуванням отриманих запитів, помилок, та інших дій (рис. 2.6.4.16).

```
----- ----- -----
03:29:23.684 [debug] Post request <<"{\n    \"auth\":{\n        \"login\":\"admin\",\n        \"passha
sh\":[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,32]\n    },\n    \"parameter
s\":{\n        \"method\":\"show_roles\",\n        \"user\":\"jhon_test\"\n    }\n}\n">>
03:29:23.684 [debug] Post reply <<"{\"result\":\"ok\",\"roles\":[\"write\",\"am\",\"read\"],\"user\":\
"jhon_test\"}">>
----- ----- -----
```

Рис. 2.6.4.16. Приклад логування успішної роботи методу show_roles.

Порівнюючи дані з відповіді на запит та даних БД, знаходимо їх ідентичними (рис. 2.6.4.17).

```
1  SELECT role, name FROM roles r1, users u1 WHERE r1.user_id=u1.id;
```

Data Output | Messages | Notifications

	role character varying (10)	name character varying (50)
1	read	lion
2	am	lion
3	Am_3	mike
4	creat	mike
5	am	mike
6	read	jhon_test
7	am	jhon_test
8	write	jhon_test
9	read	jhon
10	am	jhon
11	am	mery

Рис. 2.6.4.17. Перегляд ролей користувачів у базі даних.

Roles_add.

Після успішної аутентифікації адміністратора в системі, завдяки параметрам запису “auth”, проходить ідентифікація методу. Метод roles_add проводить перевірку на існування введених ролей у системі, надає ролі користувачу. Взаємодіє з базою даних, та додає ролі користувачу у кеш-пам’яті. Формує успішну відповідь. Приклад запиту, та відповіді зображено на рисунку 2.6.4.18.

POST http://127.0.0.1:1913/admin

Body raw JSON Beautify

```
{
    "auth":{
        "login":"admin",
        "passhash":[113,16,237,164,208,
            158,6,42,165,228,163,144,
            176,165,114,172,13,44,2,32]
    },
    "parameters":{
        "method":"roles_add",
        "user":"jhon_test",
        "roles":["lm1", "am2"]
    }
}
```

Body Cookies Headers (4) Test

Pretty Raw Preview

```
{
    "result": "ok"
}
```

Рис. 2.6.4.18. Приклад успішної роботи методу roles_add.

Також для подальшого моніторингу сервіс займається логуванням отриманих запитів, помилок, та інших дій (рис. 2.6.4.19).

```
----- ----- -----
03:36:31.996 [debug] Post request <<"{\n    \"auth\":{\n        \"login\":\"admin\",\n        \"passha
sh\":[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,32]\n    },\n    \"parameter
s\":{\n        \"method\":\"roles_add\",\n        \"user\":\"jhon_test\",\n        \"roles\":[\"lm1\",
 \"am2\"]\n    }\n}">>
03:36:31.998 [debug] Add role <<"lm1">> for user <<"jhon_test">>
03:36:31.999 [debug] Add role <<"am2">> for user <<"jhon_test">>
03:36:31.999 [debug] Post reply <<"{\"result\":\"ok\"}">>
----- ----- -----
```

Рис. 2.6.4.19. Приклад логування успішної роботи методу roles_add.

Переглядаючи базу даних після операції методу roles_add, ми побачимо нові ролі у користувача jhon_test (рис. 2.6.4.20).

```
1  SELECT role, name FROM roles r1, users u1 WHERE r1.user_id=u1.id;
```

Data Output | Messages | Notifications

	role character varying (10)	name character varying (50)
1	read	lion
2	am	lion
3	Am_3	mike
4	creat	mike
5	am	mike
6	read	jhon_test
7	am	jhon_test
8	am2	jhon_test
9	lm1	jhon_test
10	write	jhon_test
11	read	jhon
12	am	jhon
13	am	mery

Рис. 2.6.4.20. Перегляд зміненої БД після операції методу roles_add.

Roles_delete.

Після успішної аутентифікації адміністратора в системі, завдяки параметрам запису “auth”, проходить ідентифікація методу. Метод roles_delete проводить перевірку на існування введених ролей у системі, після чого видаляє у користувача надані попередньо ролі. Взаємодіє з базою

даних, та видаляє ролі користувача у кеш-пам’яті. Формує успішну відповідь. Приклад запиту, та відповіді зображено на рисунку 2.6.4.21.

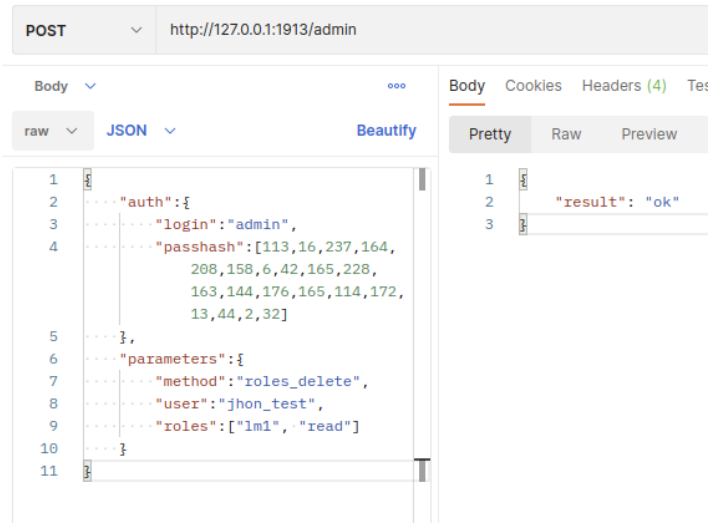


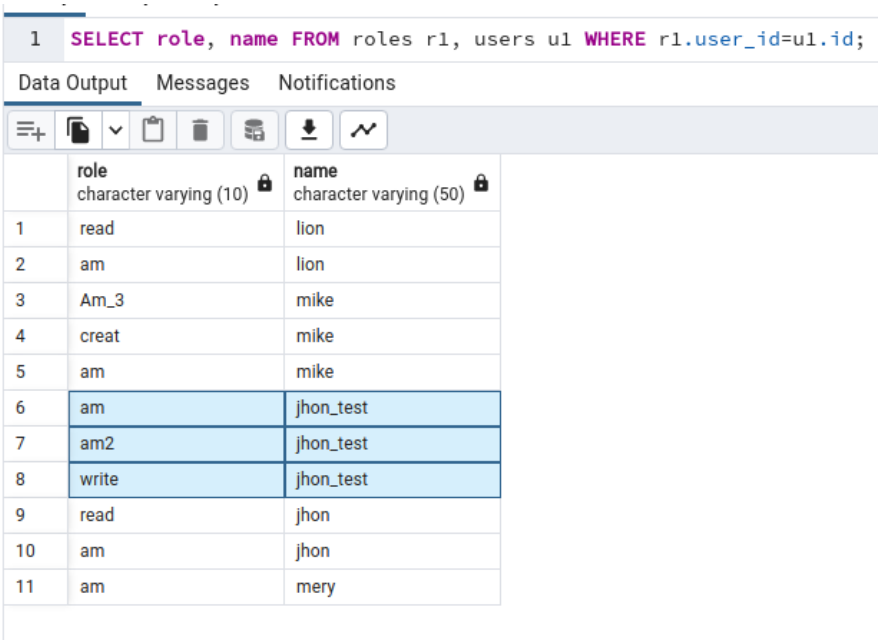
Рис. 2.6.4.21. Приклад успішної роботи методу roles_delete.

Також для подальшого моніторингу сервіс займається логуванням отриманих запитів, помилок, та інших дій (рис. 2.6.4.22).

```
----- ----- -----
03:41:21.568 [debug] Post request <<"{\n    \"auth\":{\n        \"login\":\"admin\",\n        \"passha
sh\":[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,32]\n    },\n    \"parameter
s\":{\n        \"method\":\"roles_delete\",\n        \"user\":\"jhon_test\",\n        \"roles\":[\"lm1
\", \"read\"]\n    }\n}">>
03:41:21.572 [debug] Role <<"lm1">> was delete in user <<"jhon_test">>
03:41:21.573 [debug] Role <<"read">> was delete in user <<"jhon_test">>
03:41:21.573 [debug] Post reply <<"{\"result\":\"ok\"}">>
----- ----- -----
```

Рис. 2.6.4.22. Приклад логування успішної роботи методу roles_delete.

Переглядаючи базу даних після операції методу roles_delete, ми не побачимо визначені ролі у користувача jhon_test (рис. 2.6.4.23).



```
1  SELECT role, name FROM roles r1, users u1 WHERE r1.user_id=u1.id;
```

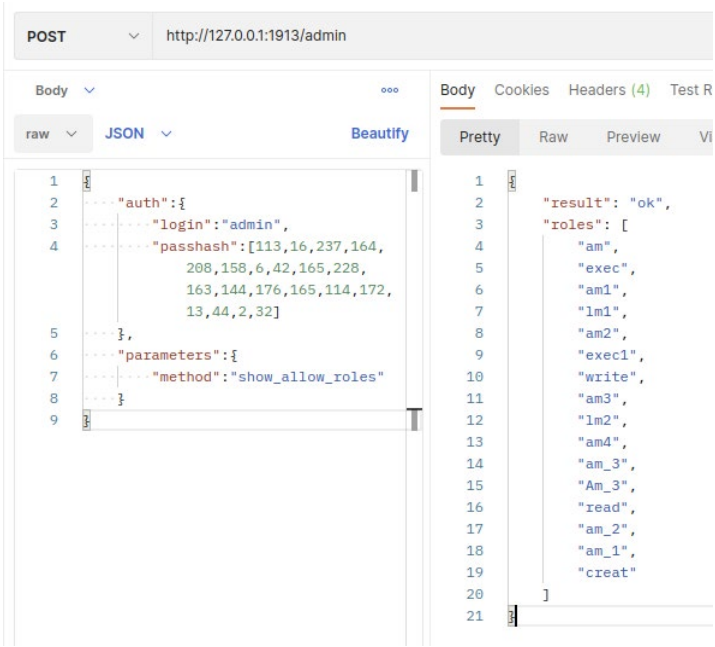
Data Output | Messages | Notifications

	role character varying (10)	name character varying (50)
1	read	lion
2	am	lion
3	Am_3	mike
4	creat	mike
5	am	mike
6	am	jhon_test
7	am2	jhon_test
8	write	jhon_test
9	read	jhon
10	am	jhon
11	am	mery

Рис. 2.6.4.23. Перегляд зміненої БД після операції методу roles_delete.

Show_allow_roles.

Після успішної аутентифікації адміністратора в системі, завдяки параметрам запису “auth”, проходить ідентифікація методу. Метод show_allow_roles отримує із кеш-пам’яті усі ролі системи, та виводить їх у відповіді. Приклад запиту, та відповіді зображено на рисунку 2.6.4.24.



POST http://127.0.0.1:1913/admin

Body raw JSON Beautify

```
{
    "auth":{
        "login":"admin",
        "passhash":[113,16,237,164,
            208,158,6,42,165,228,
            163,144,176,165,114,172,
            13,44,2,32]
    },
    "parameters":{
        "method":"show_allow_roles"
    }
}
```

Body Cookies Headers (4) Test R

Pretty Raw Preview Vi

```
{
    "result": "ok",
    "roles": [
        "am",
        "exec",
        "am1",
        "lm1",
        "am2",
        "exec1",
        "write",
        "am3",
        "lm2",
        "am4",
        "am_3",
        "Am_3",
        "read",
        "am_2",
        "am_1",
        "creat"
    ]
}
```

Рис. 2.6.4.24. Приклад успішної роботи методу show_allow_roles.

Також для подальшого моніторингу сервіс займається логуванням отриманих запитів, помилок, та інших дій (рис. 2.6.4.24).

```
----- ----- -----
03:49:30.625 [debug] Post request <<"{\n    \"auth\":{\n        \"login\":\"admin\",\n        \"passha
sh\":[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2,32]\n    },\n    \"parameter
s\":{\n        \"method\":\"show_allow_roles\"\n    }\n}">>
03:49:30.625 [debug] Post reply <<"{\"result\":\"ok\",\"roles\":[\"am\",\"exec\",\"am1\",\"lm1\",\"am2
\",\"exec1\",\"write\",\"am3\",\"lm2\",\"am4\",\"am_3\",\"Am_3\",\"read\",\"am_2\",\"am_1\",\"creat\"]
}">>
----- ----- -----
```

Рис. 2.6.4.24. Приклад логування успішної роботи методу show_allow_roles.

Порівнюючи дані з відповіді на запит та даних БД, знаходимо їх ідентичними (рис. 2.6.4.25).

```
1  SELECT * FROM allow_roles;
```

Data Output | Messages | Notifications

	role [PK] character varying
1	am
2	exec
3	am1
4	lm1
5	am2
6	exec1
7	write
8	am3
9	lm2
10	am4
11	am_3
12	Am_3
13	read
14	am_2
15	am_1
16	creat

Рис. 2.6.4.25. Перегляд зміненої БД після операції методу show_allow_roles.

Add_allow_roles.

Після успішної аутентифікації адміністратора в системі, завдяки параметрам запису “auth”, проходить ідентифікація методу. Метод add_allow_roles проводить перевірку на відсутність введених ролей у системі та створює їх. Взаємодіє з базою даних, та додає ролі у кеш-пам’ять. Формує успішну відповідь. Приклад запиту, та відповіді зображено на рисунку 2.6.4.26.

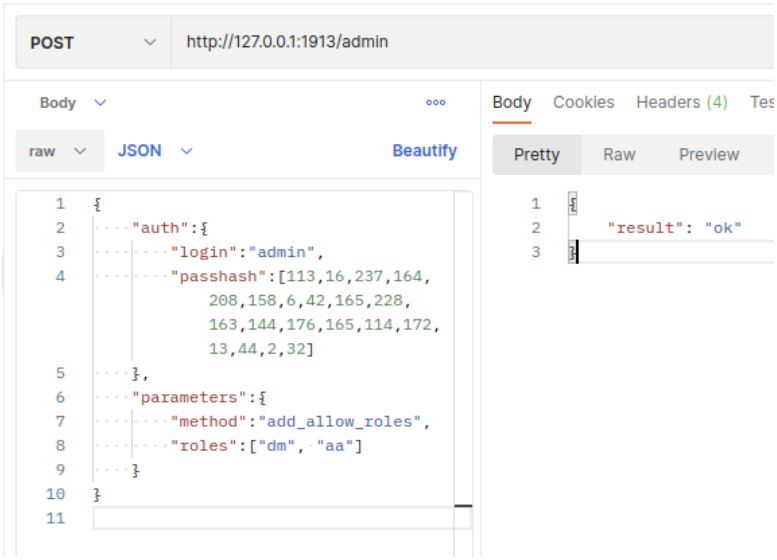


Рис. 2.6.4.26. Приклад успішної роботи методу add_allow_roles.

Також для подальшого моніторингу сервіс займається логуванням отриманих запитів, помилок, та інших дій (рис. 2.6.4.27).

```
(acl_simple@dnh0-lvr806)1> 03:59:10.482 [debug] Post request <<"{\n    \"auth\":{\n        \"login\":
\"admin\",\n        \"passhash\":[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2
,32]\n    },\n    \"parameters\":{\n        \"method\":\"add_allow_roles\",\n        \"roles\":[\"dm\
", \"aa\"]\n    }\n}\n">>
03:59:10.486 [debug] Post reply <<"{\"result\":\"ok\"}">>
----- ----- -----
```

Рис. 2.6.4.27. Приклад логування успішної роботи методу add_allow_roles.

Переглядаючи базу даних після операції методу add_allow_roles, ми побачимо новостворені ролі у системі (рис. 2.6.4.28).

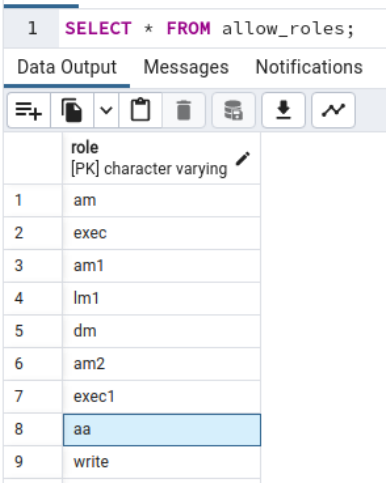


Рис. 2.6.4.28. Перегляд зміненої БД після операції методу add_allow_roles.

Delete_allow_roles.

Після успішної аутентифікації адміністратора в системі, завдяки параметрам запису “auth”, проходить ідентифікація методу. Метод delete_allow_roles проводить перевірку на існування введених ролей у системі, після чого видаляє попередньо створені ролі. Взаємодіє з базою даних, та видаляє ролі у кеш-пам’яті. Формує успішну відповідь. Приклад запиту, та відповіді зображено на рисунку 2.6.4.29.

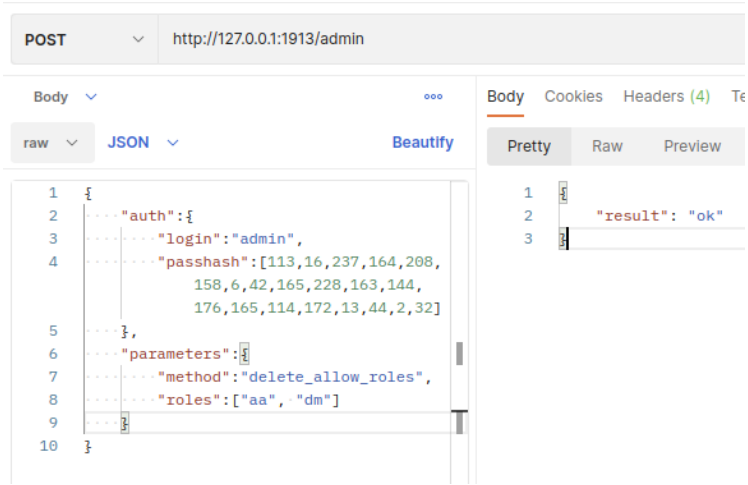


Рис. 2.6.4.29. Приклад успішної роботи методу delete_allow_roles.

Також для подальшого моніторингу сервіс займається логуванням отриманих запитів, помилок, та інших дій (рис. 2.6.4.30).

```
(acl_simple@dnh0-lvr806)1> 04:03:02.885 [debug] Post request <<"{\n    \"auth\":{\n        \"login\":
\"admin\",\n        \"passhash\":[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,172,13,44,2
,32]\n    },\n    \"parameters\":{\n        \"method\":\"delete_allow_roles\",\n        \"roles\":[\"
aa\", \"dm\"]\n    }\n}">>
04:03:02.888 [debug] Post reply <<"{\"result\":\"ok\"}">>
----- ----- -----
```

Рис. 2.6.4.30. Приклад логування роботи методу delete_allow_roles.

Переглядаючи базу даних після роботи методу delete_allow_roles, ми не побачимо визначені ролі у системі (рис. 2.6.4.31).

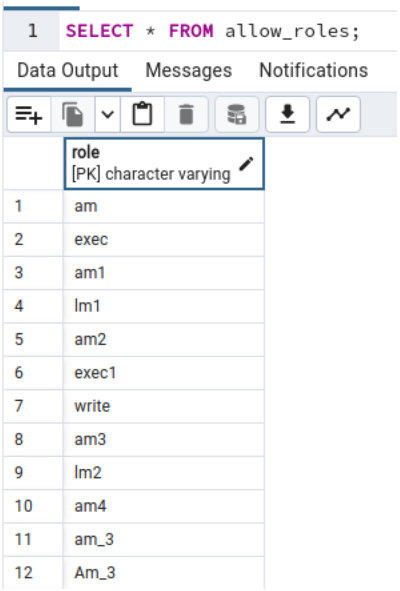


Рис. 2.6.4.31. Перегляд зміненої БД після роботи методу delete_allow_roles.

2.7. Робота тестів та аналіз результату навантажувального тестування.

Приклад роботи common-тестів. Запуск common-тестів виконується завдяки команді:

$ rebar3 ct --spec apps/acl_simple/test/common_tests/ct.spec

Результат роботи common-тестів у терміналі зазначено на рисунку 2.7.1, 2.7.2, 2.7.3.

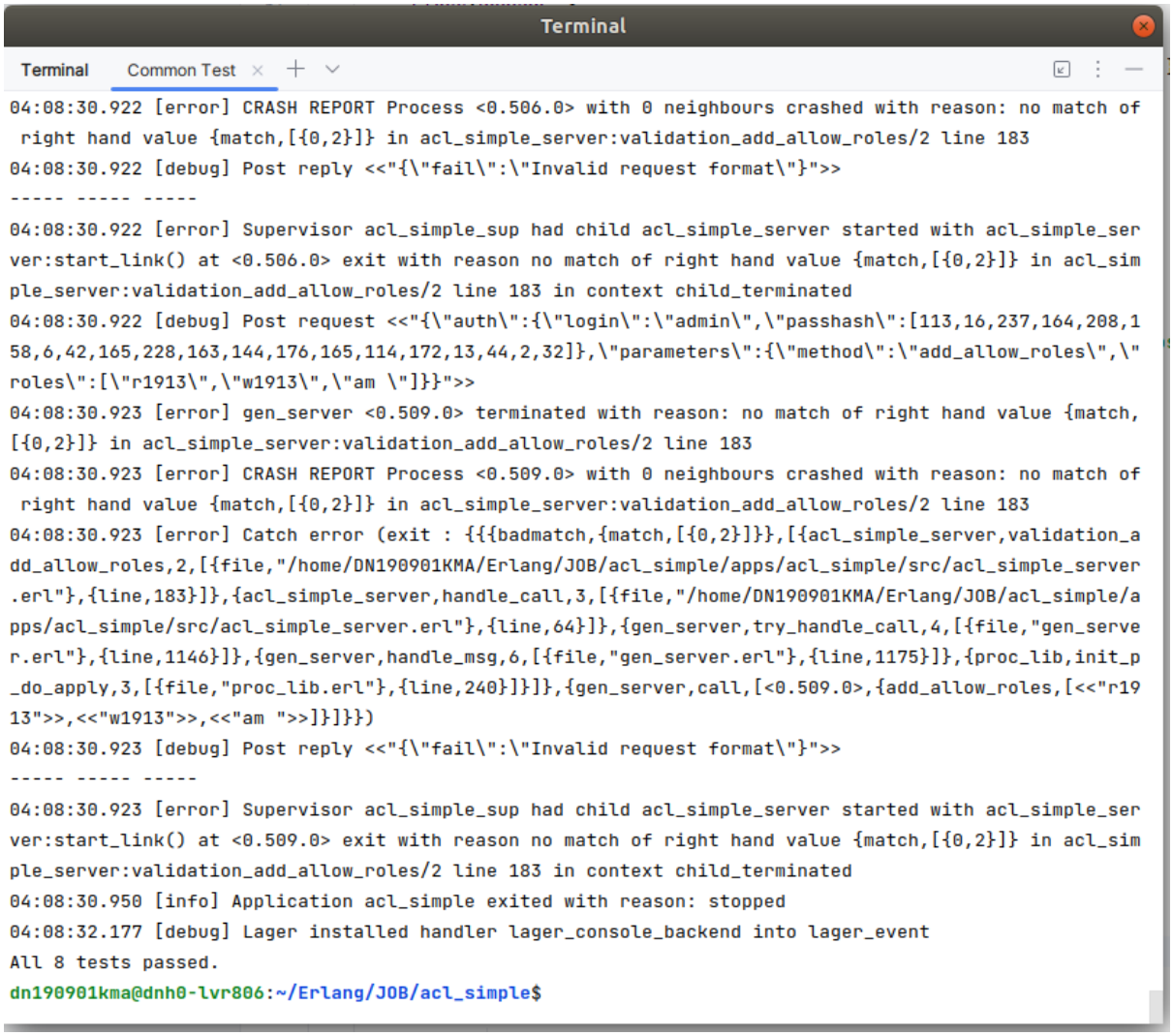
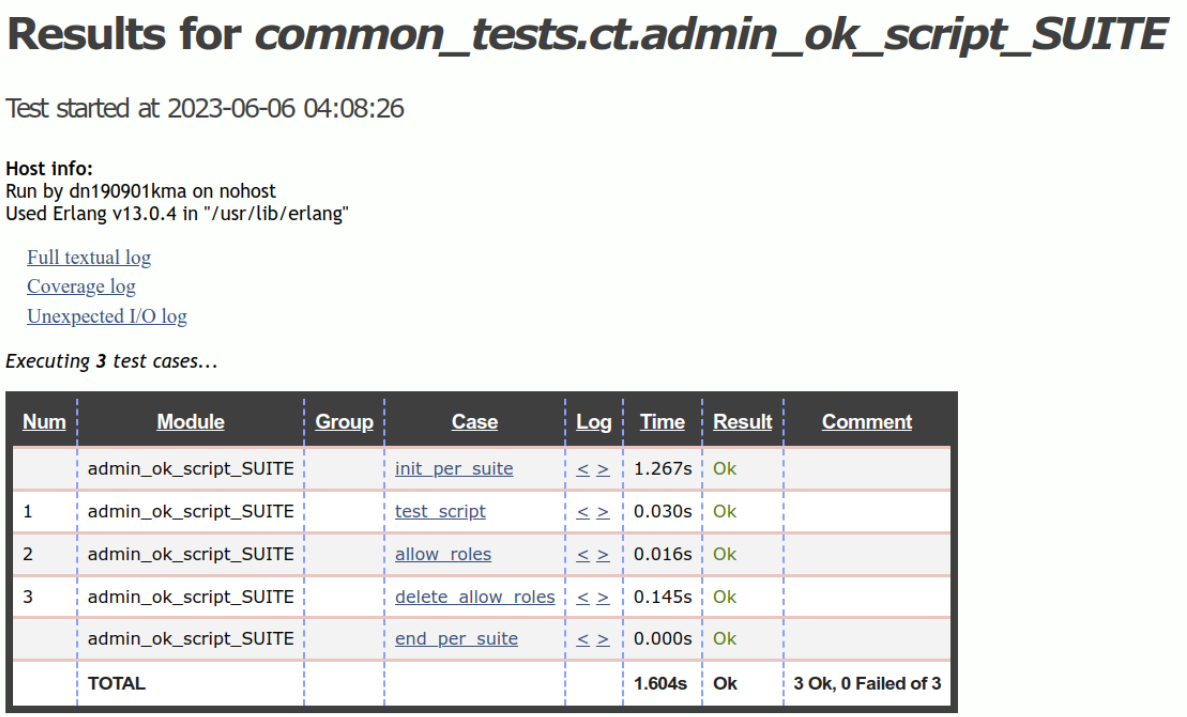


Рис. 2.7.1. Результат роботи common-тестів у консолі.

Test Name	Ok	Failed	Skipped (User/Auto)	Missing Suites
common_tests.ct.admin_ok_script_SUITE	3	0	0 (0/0)	0
common_tests.ct.customer_ok_SUITE	1	0	0 (0/0)	0
common_tests.ct.admin_fail_SUITE	4	0	0 (0/0)	0
Total	**8**	**0**	**0 (0/0)**	**0**

Рис. 2.7.2. Результат роботи common-тестів у html.



Results for *common_tests.ct.admin_ok_script_SUITE*

Test started at 2023-06-06 04:08:26

Host Info:
Run by dn190901kma on nohost
Used Erlang v13.0.4 in "/usr/lib/erlang"

Full textual log
Coverage log
Unexpected I/O log

*Executing **3** test cases...*

Num	Module	Group	Case	Log	Time	Result	Comment
	admin_ok_script_SUITE		init_per_suite	< >	1.267s	Ok	
1	admin_ok_script_SUITE		test_script	< >	0.030s	Ok	
2	admin_ok_script_SUITE		allow_roles	< >	0.016s	Ok	
3	admin_ok_script_SUITE		delete_allow_roles	< >	0.145s	Ok	
	admin_ok_script_SUITE		end_per_suite	< >	0.000s	Ok	
	TOTAL				**1.604s**	**Ok**	**3 Ok, 0 Failed of 3**

Рис. 2.7.3. Результат роботи кейсів common-тесту у html.

Результат тестів успішний.

Приклад навантажувального тестування. Для того, щоб побачити навантаження на програму було використано утиліти wrk та observer. Приклад роботи observer у звичайному режимі зазначено на рисунку 2.7.4.

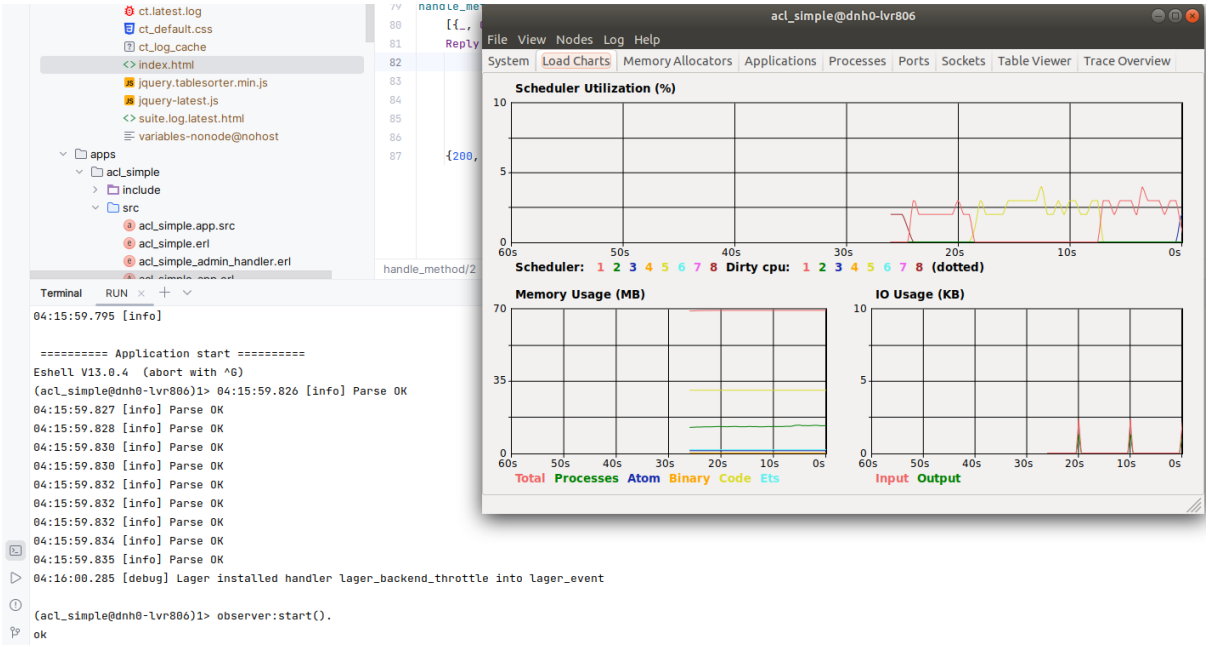


Рис. 2.7.4. Приклад роботи observer у звичайному режимі.

Навантаження програми буде проведено завдяки команді:

$ wrk -t2 -c3 -d1m -R5700 -s apps/acl_simple/wrk/show_all_users.lua http://127.0.0.1:1913.

Результат роботи утілити observer та навантажувального тестування wrk зазначено на рисунку 2.7.5.

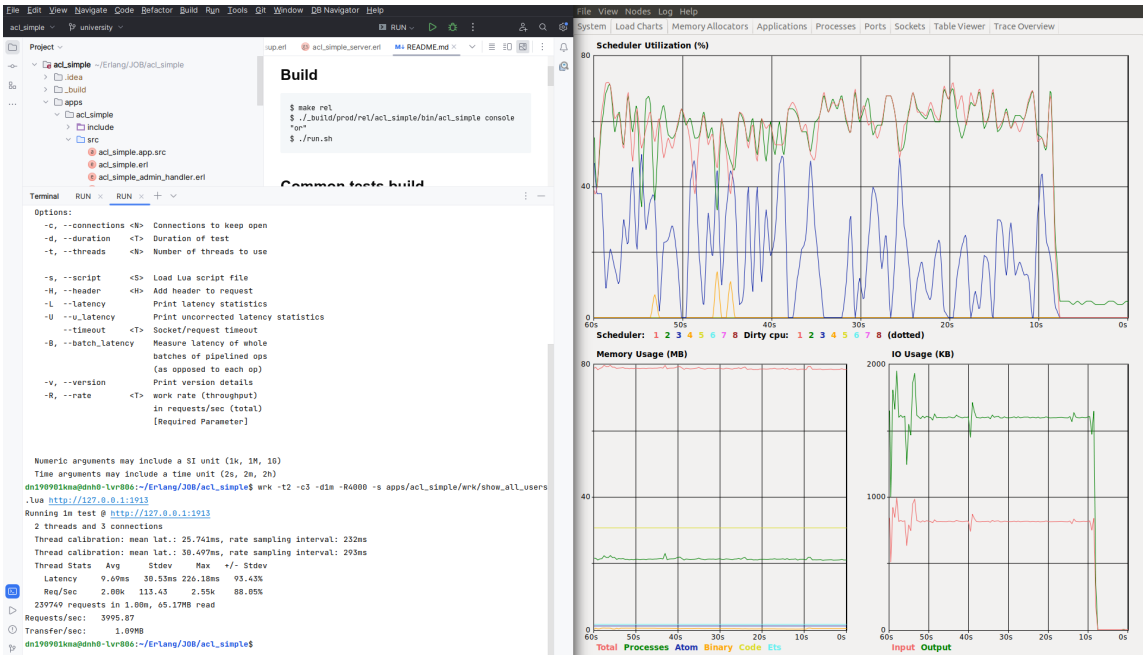


Рис. 2.7.5. Приклад роботи observer у звичайному режимі.

Аналізуючи результат тесту, можемо контактувати що сервіс витримує навантаження у розмірі 3 підключень, 2 потоків та швидкості 4000 запитів у секунду. Час тесту 1 хвилина. Максимальна затримка запиту сягала 226 мс, що є хорошим результатом.

Така швидкодія забезпечується завдяки кешуванню даних отриманих з БД.

РОЗДІЛ 3. ЕКОНОМІЧНИЙ РОЗДІЛ

3.1 Розрахунок трудомісткості та вартості розробки програмного продукту.

Початкові дані:

1. передбачуване число операторів програми – 3386;
2. коефіцієнт складності програми – 1,3;
3. коефіцієнт корекції програми в ході її розробки – 0,07;
4. годинна заробітна плата програміста – 150 грн/год;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,3;

Середня зарплата програміста у годину була вирахувана виходячи з даних «Української спільноти програмістів (DOU)». Середньоукраїнська заробітна плата програміста, який пише програми мовою програмування Erlang і з досвідом роботи близько року дорівнює коливається від 800 до 1000 американських доларів в місяць[13], тож очікувана заробітна плата має бути в цих рамках. При поточному курсі валют НБУ станом на початок грудень 2022 року один американський долар дорівнює 36,85 грн[14], тому середня зарплата в гривнях дорівнює 35780 грн (971 доларів США).

6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,2;
7. вартість машино-години ЕОМ – 23,16 грн/год.

Комп'ютер споживає приблизно 250 Ватт / год, що дорівнює 0,25 кВт / год. Вартість одного кВт на годину від постачальника Yasno[15] на стан середини 2023 року дорівнює 2,64 грн в незалежності від обсягу споживання. Отже, вартість електроенергії споживаної комп'ютером за годину дорівнює 0,66 грн. Ціна довгострокової оренди ноутбука дорівнює 4000 грн на місяць[16]. Ціна оренди на годину буде дорівнювати 22,72 грн. Остаточна вартість машино-години ЕОМ дорівнює 23,38 грн за годину.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ у можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл}^{к} + t_д, \text{ людино-годин.} \quad (3.1)$$

t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму розв'язання задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}^{к}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p) \quad (3.2)$$

q – передбачуване число операторів;

C – коефіцієнт складності програми;

p – коефіцієнт кореляції програми в ході її розробки.

$Q = 3386 \cdot 1{,}3 \cdot (1 + 0{,}07) = 4709{,}926;$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75...85) \cdot K'}, \text{ людино-годин.} \quad (3.3)$$

В – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

К – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

$$t_u = \frac{4709{,}926 \cdot 1{,}3}{85 \cdot 1{,}2} = 60{,}02, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму розв'язання задачі:

$$t_a = \frac{Q}{(20...25) \cdot K'}; \quad (3.4)$$

$$t_a = \frac{4709{,}926}{20 \cdot 1{,}2} = 196{,}24, \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_a = \frac{Q}{(20...25) \cdot K'}; \quad (3.5)$$

$$t_a = \frac{4709{,}926}{25 \cdot 1{,}2} = 157, \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5) \cdot K'}; \quad (3.6)$$

$$t_{отл} = \frac{4709{,}926}{5 \cdot 1{,}2} = 785, \text{ людино-годин}$$

- за умови комплексного налагодження завдання:

$t_{отл}^{к} = 1{,}2 \cdot t_{отл}$; (3.7)

$t_{отл}^{к} = 1{,}2 \cdot 785 = 942$, людино-годин

Витрати праці на підготовку документації:

$t_{д} = t_{др} + t_{до}$; (3.8)

$t_{др}$– трудомісткість підготовки матеріалів і рукопису

$t_{др} = \frac{Q}{(15...20) \cdot K'}$; (3.9)

$t_{др} = \frac{4709{,}926}{20 \cdot 1{,}2} = 196{,}24$, людино-годин.

$t_{до}$– трудомісткість редагування, печатки й оформлення документації

$t_{до} = 0{,}75 \cdot t_{др}$; (3.10)

$t_{до} = 0{,}75 \cdot 196{,}24 = 147{,}18$, людино-годин.

$t_{д} = 196{,}24 + 147{,}18 = 343{,}42$, людино-годин.

Отримаємо трудомісткість розробки програмного забезпечення:

t = 50 + 60,02 + 196,24 + 157 + 942 + 343,42 = 1748,66, людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 1748,66 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн.} \quad (3.11)$$

$З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн.} \quad (3.12)$$

t – загальна трудомісткість, людино-годин;

$C_{ПР}$ – середня годинна заробітна плата програміста, грн/година

$З_{ЗП} = 1748{,}66 \cdot 150 = 262299, \text{ грн.}$

$З_{МВ}$ – вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot C_{М}, \text{ грн.} \quad (3.13)$$

$t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$C_{М}$ – вартість машино-години ЕОМ, грн/год.

$З_{МВ} = 785 \cdot 23{,}38 = 18353{,}3 \text{ грн.}$

$K_{ПО}$ = 262299 + 18353,3 = 280652,3 грн.

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ мес.} \quad (3.14)$$

B_k- число виконавців;

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні F_p=176 годин).

$T = \frac{1748{,}66}{1 \cdot 176} = 9{,}9$ міс.

Висновки. На розробку даного програмного забезпечення піде 1748,66 людино-години. Тобто, ймовірна очікувана тривалість розробки складатиме 9,9 місяця при стандартному 40-годинному робочому тижні та 176-годинному 66 робочому місяці. Очікувані витрати на створення програмного забезпечення складатимуть 280652,3 грн.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний сайт Elixir: elixir-lang.org.
2. "Programming Elixir" (Програмування на Elixir) автора Дейва Томаса: pragprog.com/book/elixir16/programming-elixir-1-6.

3. "Elixir in Action" (Elixir на дії) автора Саші Марік: www.manning.com/books/elixir-in-action-second-edition
4. Офіційний сайт LFE: https://lfe.io.
5. Репозиторій LFE на GitHub: https://github.com/lfe/lfe.
6. Офіційний репозиторій Joxa на GitHub: github.com/joxa/joxa
7. Інформація про Cowboy була отримана з офіційної документації Cowboy: https://ninenines.eu/docs/en/cowboy/2.9/guide/
8. Інформація була використана з офіційного веб-сайту PostgreSQL: https://www.postgresql.org/
9. Joe Armstrong, "Programming Erlang: Software for a Concurrent World" (http://shop.oreilly.com/product/9781937785536.do)
10. Francesco Cesarini, Simon Thompson, "Erlang Programming" (http://shop.oreilly.com/product/9780596518189.do)
11. Joe Armstrong. "Programming Erlang: Software for a Concurrent World". Pragmatic Bookshelf, 2013.
12. Francesco Cesarini, Simon Thompson. "Erlang Programming: A Concurrent Approach to Software Development". O'Reilly Media, 2009.
13. Веб-сайт «Українська спільнота програмістів», URL: https://jobs.dou.ua/salaries/?period=2022-12&position=Middle%20SE&technology=Elixir
14. Веб-сайт НБУ, URL: https://bank.gov.ua/ua/markets/exchangerate-chart?cn%5B%5D=USD
15. Веб-сайт постачальника електроенергії Yasno, URL: https://yasno.com.ua
16. Оренда ноутбука, URL: https://komputers.com.ua/arenda-kompyutera/
17. Офіційний веб-сайт IntelliJ IDEA: https://www.jetbrains.com/idea/

ДОДАТОК А

КОД ПРОГРАМИ

1. Проєкт acl_simple:

rebar.config

```
%% -*- mode: Erlang; -*-
{erl_opts, [{debug_info}]}.
{deps, [
   {cowboy,  {git, "https://github.com/extend/cowboy.git",       {tag, "2.9.0"}}},
   {jsone,   {git, "https://github.com/sile/jsone.git",          {tag, "1.7.0"}}},
   {lager,   {git, "https://github.com/erlang-lager/lager.git",  {tag, "3.9.1"}}},
   {epgsql,  {git, "https://github.com/epgsql/epgsql.git",       {tag, "4.6.0"}}},
   {poolboy, {git, "https://github.com/devinus/poolboy.git",     {tag, "1.5.2"}}},
   {uuid,    {git, "https://github.com/avtobiff/erlang-uuid.git", {tag, "v0.5.1"}}}
]}.

{relx, [{release, {acl_simple, "1.3.54"},
 [acl_simple,
   poolboy,
   jsone,
   lager,
   sasl,
   uuid,
   {observer, load},  %% Allow live debugging of server
   {wx, load},
   {runtime_tools, load},  %% Required by observer
   cowboy,
   epgsql
 ]},

 {sys_config, "./config/sys.config"},
 {vm_args, "./config/vm.args"},

 {dev_mode, true},
 {include_erts, true},

 {extended_start_script, true}]
}.

{profiles, [{prod, [{relx, [{dev_mode, false},
```

```
    {include_erts, true}]}]
  }]
}.
```

run.sh

```
#! /bin/bash

appName=acl_simple

pathFile=$(realpath $0)
pathDir=$(dirname $pathFile)

cd $pathDir
make rel
cd ./_build/prod/rel/$appName/bin/
./$appName console
```

makefile

```
all: get-deps compile
get-deps:
	@rebar3 get-deps
compile:
	@rebar3 compile
clean:
	@rm -rf _build/
	@rm rebar.lock
eunit:
	@rebar3 eunit
ct:
	@rebar3 ct
tests: eunit ct
default:
```

```
	@rebar3 release
rel:
	@rebar3 as prod release
relclean:
	@rm -rf _build/
dialyze:
	@rebar3 dialyzer
docs:
	@rebar3 edoc
```

sys.config

```
[
  {acl_simple, [
    {listen_port, 1913},
    {timer_cache, 10000},
    {timer_allow_roles, 90000}
  ]},

  {poolboy, [
    {pools, [
      {pg_pool,
        [
          {name, {local, pg_pool}},
          {strategy, fifo},
          {worker_module, acl_simple_pg},
          {size, 10},
          {max_overflow, 5}
        ], [
          {host, "127.0.0.1"},
          {port, 5432},
          {username, "trembita"},
          {password, "trembita"},
          {database, "civil"}
        ]}
    ]}
```

```
]},

{lager, [
   %% What handlers to install with what arguments
   %% The defaults for the logfiles are to rotate the files when
   %% they reach 10Mb or at midnight, whichever comes first, and keep
   %% the last 31 rotations.
   {handlers, [
      {lager_console_backend, [{level, debug}]},
      {lager_file_backend, [
         {file, "error.log"},
         {level, error},
         {size, 1073741824}, %% 1Gb
         {date, "$D0"},
         {count, 20}
      ]},
      {lager_file_backend, [
         {file, "warning.log"},
         {level, warning},
         {size, 1073741824}, %% 1Gb
         {date, "$D0"},
         {count, 20}
      ]},
      {lager_file_backend, [
         {file, "info.log"},
         {level, info},
         {size, 1073741824}, %% 1Gb
         {date, "$D0"},
         {count, 7}
      ]},
      {lager_file_backend, [
         {file, "debug.log"},
         {level, debug},
         {size, 1073741824}, %% 1Gb
         {date, "$D0"},
         {count, 2}
      ]}
```

```
        ]},

        %% Whether to write a crash log, and where.
        %% Commented/omitted/undefined means no crash logger.
        {crash_log, "crash.log"},

        %% Maximum size in bytes of events in the crash log - defaults to 65536
        {crash_log_msg_size, 65536},

        %% Maximum size of the crash log in bytes, before its rotated, set
        %% to 0 to disable rotation - default is 0
        {crash_log_size, 10485760},

        %% What time to rotate the crash log - default is no time
        %% rotation.
        {crash_log_date, "$D0"},

        %% Number of rotated crash logs to keep, 0 means keep only the
        %% current one - default is 0
        {crash_log_count, 5},

        %% Whether to redirect error_logger messages into lager - defaults to true
        {error_logger_redirect, true},

        {error_logger_hwm, 400}
    ]}
].
```

acl_simple.hrl

```
-author('Mykhailo Krasniuk <miha.190901@gmail.com>').

-define(LOG_DEBUG(Format, Args),    lager:log(debug,    self(), Format, Args)).
-define(LOG_INFO(Format, Args),     lager:log(info,     self(), Format, Args)).
-define(LOG_WARNING(Format, Args),  lager:log(warning,  self(), Format, Args)).
```

```
-define(LOG_ERROR(Format, Args),    lager:log(error,    self(), Format, Args)).
-define(LOG_CRITICAL(Format, Args), lager:log(critical, self(), Format, Args)).


-define(JSON_ERROR(Req),                #{<<"result">> => <<"error">>, <<"discription">> =>
list_to_binary(Req)}).
-define(JSON_OK,                        #{<<"result">> => <<"ok">>}).
-define(JSON_SHOW_ALLOW_ROLES(ListRoles), #{<<"result">> => <<"ok">>, <<"roles">> =>
ListRoles}).
-define(JSON_USERS(Users),              #{<<"result">> => <<"ok">>, <<"users">> => Users}).
-define(JSON_CUSTOM_GET_ROLES_OK(Roles),  #{<<"result">> => <<"ok">>, <<"roles">> =>
Roles}).
-define(JSON_ROLES_OF_USER(UserName, Roles),
   #{<<"result">> => <<"ok">>,
      <<"user">> => UserName,
      <<"roles">> => Roles}).
```

acl_simple.app.src

```
{application, acl_simple,
 [{description, "An OTP application"},
  {vsn, "1.3.54"},
  {registered, []},
  {mod, {acl_simple_app, []}},
  {applications,
   [kernel,
    stdlib,
    cowboy,
    lager
   ]},
  {env,[]},
  {modules, []},
  {licenses, ["Apache 2.0"]},
  {links, []}
 ]}.
```

acl_simple.erl

```
-module(acl_simple). %% Module for common tests
-author('Mykhailo Krasniuk <miha.190901@gmail.com>').

-export([start/0, stop/0]).



start() ->
    start(acl_simple).

stop() ->
    application:stop(acl_simple).



%% ====================================

start(AppName) ->
    F = fun({App, _, _}) -> App end,
    ok = load(AppName),
    {ok, Dependencies} = application:get_key(AppName, applications),
    [begin
        RunningApps = lists:map(F, application:which_applications()),
        case lists:member(A, RunningApps) of
            true ->
                ok;
            false ->
                ok = start(A)
        end
     end || A <- Dependencies],
    ok = application:start(AppName).

load(AppName) ->
    F = fun({App, _, _}) -> App end,
```

```
    LoadedApps = lists:map(F, application:loaded_applications()),
    case lists:member(AppName, LoadedApps) of
        true ->
            ok;
        false ->
            ok = application:load(AppName)
    end.
```

acl_simple_admin_handler.erl

```
-module(acl_simple_admin_handler).
-author('Mykhailo Krasniuk <miha.190901@gmail.com>').

-export([init/2]).

-include("acl_simple.hrl").

init(Req, Opts) ->
    Method = cowboy_req:method(Req),
    HasBody = cowboy_req:has_body(Req),
    Resp = handle_post(Method, HasBody, Req),
    {ok, Resp, Opts}.

handle_post(<<"POST">>, true, Req) ->
    handle_req(Req);
handle_post(<<"POST">>, false, Req) ->
    ?LOG_ERROR("Missing body ~p~n", [Req]),
    cowboy_req:reply(400, #{}, <<"Missing body.">>, Req);
handle_post(Method, _, Req) ->
    ?LOG_ERROR("Method ~p not allowed ~p~n", [Method, Req]),
    cowboy_req:reply(405, Req).

handle_req(Req) ->
    {ok, Body, _Req} = cowboy_req:read_body(Req),
    ?LOG_DEBUG("Post request ~p", [Body]),
```

```
    {Status, Json} = handle_body(Body),
    ?LOG_DEBUG("Post reply ~p~n----- ----- -----~n", [Json]),
    cowboy_req:reply(Status, #{<<"content-type">> => <<"application/json; charset=UTF-8">>},
Json, Req).

handle_body(Body) ->
    try jsone:decode(Body) of
        Map ->
            handle_package(Map)
    catch
        Exception:Reason ->
            ?LOG_ERROR("jsone:decode error (~p : ~p)", [Exception, Reason]),
            {206, jsone:encode(#{<<"fail">> => <<"Invalid request format">>})}
    end.

handle_package(Map) ->
    try
        #{<<"auth">> := AuthPack} = Map,
        #{<<"login">> := Login, <<"passhash">> := PassHash} = AuthPack,
        #{<<"parameters">> := ParamPack} = Map,
        handle_parameters({Login, PassHash}, ParamPack)
    catch
        Class:Reason ->
            ?LOG_ERROR("Catch error (~p : ~p)", [Class, Reason]),
            {206, jsone:encode(#{<<"fail">> => <<"Invalid request format">>})}
    end.

handle_parameters({Login, PassHash}, #{<<"method">> := Method} = ParamPack) ->
    case auth(Login, PassHash, Method) of
        true ->
            handle_method(Method, ParamPack);
        false ->
            ?LOG_ERROR("Auth error, login ~p method ~p~n", [Login, Method]),
            {206, jsone:encode(#{<<"fail">> => <<"Invalid passhash or role absent">>})}
    end.

-spec auth(binary(), list(), binary()) -> true|false.
```

```
auth(Login, Hash, _Method) ->
    {ok, _, [{JsonHash}]} = acl_simple_pg:select("get_admin_passhash", [Login]),
    RealHash = jsone:decode(JsonHash),
    RealHash == Hash.

-spec handle_method(binary(), map()) -> {integer(), binary()}.
handle_method(<<"user_add">>, Map) ->
    [{_, Pid}] = ets:lookup(acl_simple, acl_simple_server),
    #{<<"user">> := NewUser, <<"passhash">> := PassHash} = Map,
    Reply = gen_server:call(Pid, {user_add, NewUser, PassHash}),
    {200, jsone:encode(Reply)};
handle_method(<<"user_delete">>, Map) ->
    [{_, Pid}] = ets:lookup(acl_simple, acl_simple_server),
    User = maps:get(<<"user">>, Map),
    Reply = gen_server:call(Pid, {user_delete, User}),
    {200, jsone:encode(Reply)};
handle_method(<<"show_all_users">>, _Map) ->
    [{_, Pid}] = ets:lookup(acl_simple, acl_simple_server),
    Reply = gen_server:call(Pid, {show_all_users}),
    {200, jsone:encode(Reply)};
% --- roles ---
handle_method(<<"roles_add">>, Map) ->
    [{_, Pid}] = ets:lookup(acl_simple, acl_simple_server),
    User = maps:get(<<"user">>, Map),
    Roles = maps:get(<<"roles">>, Map),
    Reply = gen_server:call(Pid, {roles_add, User, Roles}),
    {200, jsone:encode(Reply)};
handle_method(<<"roles_delete">>, Map) ->
    [{_, Pid}] = ets:lookup(acl_simple, acl_simple_server),
    User = maps:get(<<"user">>, Map),
    Roles = maps:get(<<"roles">>, Map),
    Reply = gen_server:call(Pid, {roles_delete, User, Roles}),
    {200, jsone:encode(Reply)};
handle_method(<<"show_roles">>, Map) ->
    [{_, Pid}] = ets:lookup(acl_simple, acl_simple_server),
    User = maps:get(<<"user">>, Map),
    Reply = gen_server:call(Pid, {show_roles, User}),
```

```
    {200, jsone:encode(Reply)};
% --- allow_roles ---
handle_method(<<"show_allow_roles">>, _Map) ->
    [{_, Pid}] = ets:lookup(acl_simple, acl_simple_server),
    Reply = gen_server:call(Pid, {show_allow_roles}),
    {200, jsone:encode(Reply)};
handle_method(<<"add_allow_roles">>, Map) ->
    [{_, Pid}] = ets:lookup(acl_simple, acl_simple_server),
    #{<<"roles">> := ListRoles} = Map,
    Reply = gen_server:call(Pid, {add_allow_roles, ListRoles}),
    {200, jsone:encode(Reply)};
handle_method(<<"delete_allow_roles">>, Map) ->
    [{_, Pid}] = ets:lookup(acl_simple, acl_simple_server),
    #{<<"roles">> := ListRoles} = Map,
    Reply = gen_server:call(Pid, {delete_allow_roles, ListRoles}),
    {200, jsone:encode(Reply)}.
```

acl_simple_app.erl

```
-module(acl_simple_app).
-author('Mykhailo Krasniuk <miha.190901@gmail.com>').
-behaviour(application).

-include("acl_simple.hrl").

-export([start/2, stop/1]).

%%noinspection Erlang17Syntax
start(_StartType, _StartArgs) ->
    ?LOG_INFO("~n~n ========== Application start ==========", []),
    acl_simple = ets:new(acl_simple, [set, public, named_table]),
    true = ets:insert(acl_simple, [{server_cache, #{}}]),
    % ----------
    {ok, Port} = application:get_env(acl_simple, listen_port),
    Dispatch = cowboy_router:compile([
```

```
        {'_', [
            {"/admin", acl_simple_admin_handler, []},
            {"/customer", acl_simple_customer_handler, []}
        ]}
    ]),
    {ok, _} = cowboy:start_clear(http, [{port, Port}], #{
        env => #{dispatch => Dispatch}
    }),
    acl_simple_sup:start_link().

stop(_State) ->
    ok = cowboy:stop_listener(http).
```

acl_simple_customer_handler.erl

```
-module(acl_simple_customer_handler).
-author('Mykhailo Krasniuk <miha.190901@gmail.com>').

-export([init/2]).

-include("acl_simple.hrl").


init(Req, Opts) ->
    Method = cowboy_req:method(Req),
    HasBody = cowboy_req:has_body(Req),
    Resp = handle_post(Method, HasBody, Req),
    {ok, Resp, Opts}.

handle_post(<<"POST">>, true, Req) ->
    handle_req(Req);
handle_post(<<"POST">>, false, Req) ->
    ?LOG_ERROR("Missing body ~p~n", [Req]),
```

```
    cowboy_req:reply(400, #{}, <<"Missing body.">>, Req);
handle_post(Method, _, Req) ->
    ?LOG_ERROR("Method ~p not allowed ~p~n", [Method, Req]),
    cowboy_req:reply(405, Req).

handle_req(Req) ->
    {ok, Body, _Req} = cowboy_req:read_body(Req),
    ?LOG_DEBUG("Post request ~p~n", [Body]),
    {Status, Json} = handle_body(Body),
    ?LOG_DEBUG("Post reply ~p~n", [Json]),
    cowboy_req:reply(Status, #{<<"content-type">> => <<"application/json; charset=UTF-8">>},
Json, Req).

handle_body(Body) ->
    try jsone:decode(Body) of
        Map ->
            handle_package(Map)
    catch
        exit:{timeout, Other} ->
            ?LOG_ERROR("503; server overloaded (exit : {timeout, ~p})", [Other]),
            {503, <<"server overloaded">>};
        Exception:Reason ->
            ?LOG_ERROR("400; invalid syntax of json (~p : ~p)", [Exception, Reason]),
            {400, <<"invalid syntax of json">>}
    end.

handle_package(Map) ->
    try
        #{<<"auth">> := AuthPack} = Map,
        #{<<"login">> := Login, <<"passhash">> := PassHash} = AuthPack,
        #{<<"parameters">> := Param} = Map,
        handle_parameters({Login, PassHash}, Param)
    catch
        _Class:Reason ->
            ?LOG_ERROR("Parse map error ~p Reason ~p~n", [Map, Reason]),
            {206, jsone:encode(#{<<"fail">> => <<"Invalid request format">>})}
    end.
```

```
handle_parameters({Login, PassHash}, #{<<"method">> := Method} = Param) ->
    case auth(Login, PassHash, Method) of
        true ->
            handle_method(Method, Param#{<<"login">> => Login});
        false ->
            ?LOG_ERROR("Auth error, login ~p method ~p~n", [Login, Method]),
            {206, jsone:encode(#{<<"fail">> => <<"Invalid passhash or role absent">>})}
    end.

-spec auth(binary(), list(), binary()) -> true|false.
auth(Login, Hash, _Method) ->
    [{_, CachePassHash}] = ets:lookup(acl_simple, customer_passhash),
    case maps:get(Login, CachePassHash, undefined) of
        undefined ->
            ?LOG_ERROR("Inditification fail", []),
            false;
        RealHash ->
            RealHash == Hash
    end.


%% ================== handle_method ==================

handle_method(<<"get_roles">>, #{<<"login">> := Login} = _Param) ->
    [{_, Cache}] = ets:lookup(acl_simple, server_cache),
    Reply = case maps:find(Login, Cache) of
                {ok, Roles} ->
                    ?JSON_CUSTOM_GET_ROLES_OK(Roles);
                error ->
                    ?JSON_ERROR("Login '" ++ binary_to_list(Login) ++ "' is not exist")
            end,
    {200, jsone:encode(Reply)}.
```

acl_simple_pg.erl

```
-module(acl_simple_pg).
-author('Mykhailo Krasniuk <miha.190901@gmail.com>').
-behavior(gen_server).

-include("acl_simple.hrl").

-export([start_link/1]). % Export for poolboy
-export([init/1, handle_call/3, handle_cast/2, handle_info/2, terminate/2, code_change/3]). % Export
for gen_server
-export([select/2, insert/2, delete/2]).

% =====================================================
% Clients functions
% =====================================================

start_link(Args) ->
    gen_server:start_link(?MODULE, Args, []).

select(Statement, Args) ->
    case poolboy:checkout(pg_pool) of
        full ->
            ?LOG_ERROR("All workers are busy. Pool(~p)", [pg_pool]),
            {error, full_pool};
        WorkerPid ->
            Reply = gen_server:call(WorkerPid, {select, Statement, Args}),
            ok = poolboy:checkin(pg_pool, WorkerPid),
            Reply
    end.

insert(Statement, Args) ->
    case poolboy:checkout(pg_pool) of
        full ->
            ?LOG_ERROR("All workers are busy. Pool(~p)", [pg_pool]),
            {error, full_pool};
        WorkerPid ->
```

```
        Reply = gen_server:call(WorkerPid, {insert, Statement, Args}),
        ok = poolboy:checkin(pg_pool, WorkerPid),
        Reply
    end.

delete(Statement, Args) ->
    case poolboy:checkout(pg_pool) of
        full ->
            ?LOG_ERROR("All workers are busy. Pool(~p)", [pg_pool]),
            {error, full_pool};
        WorkerPid ->
            Reply = gen_server:call(WorkerPid, {delete, Statement, Args}),
            ok = poolboy:checkin(pg_pool, WorkerPid),
            Reply
    end.



% ====================================================
% Inverse functions
% ====================================================

init(Args) ->
    TConn = erlang:send_after(10, self(), connect),
    {ok, #{connect_arg => Args,
        timer_connect => TConn,
        connection => undefined}}.

terminate(_, _State) ->
    ok.

handle_call({insert, Statement, Args}, _From, State) ->
    #{connection := Conn} = State,
    Reply = send_to_bd(Conn, Statement, Args),
    {reply, Reply, State};
handle_call({select, Statement, Args}, _From, State) ->
    #{connection := Conn} = State,
    Reply = send_to_bd(Conn, Statement, Args),
```

```
    {reply, Reply, State};
handle_call({delete, Statement, Args}, _From, State) ->
    #{connection := Conn} = State,
    Reply = send_to_bd(Conn, Statement, Args),
    {reply, Reply, State}.

handle_cast(_Data, State) ->
    {noreply, State}.

handle_info(connect, State) -> % initialization
    #{connect_arg := Arg, timer_connect := TConn} = State,
    _ = erlang:cancel_timer(TConn),
    State1 = case epgsql:connect(Arg) of
            {ok, Pid} ->
               ok = parse(Pid),
               State#{connection := Pid};
            {error, _Error} ->
              % ?LOG_ERROR("db connect error ~p", [Error]),
             %  ok = timer:sleep(1000),
               TConn1 = erlang:send_after(500, self(), connect),
               State#{connection := undefined, timer_connect := TConn1}
         end,
    {noreply, State1};
handle_info(_Data, State) ->
    {noreply, State}.

code_change(_OldVsn, State, _Extra) ->
    {ok, State}.


% ===================================================
% Help-functions for inverse functions
% ===================================================

parse(Conn) ->
    ?LOG_INFO("Parse OK", []),
```

```
    {ok, _} = epgsql:parse(Conn, "delete_allow_role_in_roles", "DELETE FROM roles WHERE role
= $1", [varchar]),
    {ok, _} = epgsql:parse(Conn, "get_allow_roles", "SELECT role FROM allow_roles", []),
    {ok, _} = epgsql:parse(Conn, "add_allow_role", "INSERT INTO allow_roles (role) VALUES
($1)", [varchar]),
    {ok, _} = epgsql:parse(Conn, "delete_allow_role", "DELETE FROM allow_roles WHERE role =
$1", [varchar]),
    {ok, _} = epgsql:parse(Conn, "user_add", "INSERT INTO users (id, name, passhash) VALUES
($1, $2, $3)", [varchar, varchar, json]),
    {ok, _} = epgsql:parse(Conn, "get_passhash", "SELECT passhash FROM users WHERE name =
$1", [varchar]),
    {ok, _} = epgsql:parse(Conn, "get_admin_passhash", "SELECT passhash FROM admins WHERE
login = $1", [varchar]),
    {ok, _} = epgsql:parse(Conn, "roles_add_by_name", "INSERT INTO roles (user_id, role)
VALUES ((SELECT id FROM users WHERE name = $1), $2)", [varchar, varchar]),
    {ok, _} = epgsql:parse(Conn, "get_all_users", "SELECT name, passhash FROM users", []),
    {ok, _} = epgsql:parse(Conn, "get_roles_by_name", "SELECT role FROM roles WHERE user_id
= (SELECT id FROM users WHERE name = $1)", [varchar]),
    {ok, _} = epgsql:parse(Conn, "users_delete_by_name", "DELETE FROM users WHERE name =
$1", [varchar]),
    {ok, _} = epgsql:parse(Conn, "roles_delete_by_name", "DELETE FROM roles WHERE user_id =
(SELECT id FROM users WHERE name = $1) AND role = $2", [varchar, varchar]),
    ok.

send_to_bd(undefined, _, _) ->
    ?LOG_ERROR("acl_simple_pg: no connect to db", []),
    {error, no_connect};
send_to_bd(Conn, Statement, Args) -> % INTERFACE between prepared_query of DB, and
handle_call...
    case epgsql:prepared_query(Conn, Statement, Args) of
        {error, Error} ->
            ?LOG_ERROR("PostgreSQL prepared_query error(~p): ~p~n", [Statement, Error]),
            {error, Error};
        Other -> Other
    end.
```

acl_simple_server.erl

```
-module(acl_simple_server).
-author('Mykhailo Krasniuk <miha.190901@gmail.com>').
-behavior(gen_server).

-include("acl_simple.hrl").

-export([init/1, handle_call/3, handle_cast/2, handle_info/2, terminate/2, code_change/3]).
-export([start_link/0, stop/1]).



% ====================================================
% Users functions
% ====================================================

start_link() ->
   gen_server:start_link(?MODULE, [], []).
stop(Pid) ->
   gen_server:call(Pid, terminate).



% ====================================================
% Inverse functions
% ====================================================

init([]) ->
   true = ets:insert(acl_simple, [{acl_simple_server, self()}]),
   {ok, []}.

terminate(_, _State) ->
   ok.

handle_call({user_add, UserName, PassHash}, _From, State) ->
   Reply = user_add_handler(UserName, PassHash),
```

```
    {reply, Reply, State};
handle_call({user_delete, UserName}, _From, State) ->
    Reply = user_delete_handler(UserName),
    {reply, Reply, State};
handle_call({show_all_users}, _From, State) ->
    [{_, Cache}] = ets:lookup(acl_simple, server_cache),
    Users = maps:keys(Cache),
    Reply = ?JSON_USERS(Users),
    {reply, Reply, State};
handle_call({roles_add, UserName, Roles}, _From, _State) ->
    Reply = roles_add_handler(UserName, Roles),
    {reply, Reply, []};
handle_call({roles_delete, UserName, Roles}, _From, State) ->
    Reply = roles_delete_handler(UserName, Roles),
    {reply, Reply, State};
handle_call({show_roles, UserName}, _From, _State) ->
    [{_, Cache}] = ets:lookup(acl_simple, server_cache),
    Reply = case maps:find(UserName, Cache) of
                {ok, Roles} ->
                    ?JSON_ROLES_OF_USER(UserName, Roles);
                error ->
                    ?JSON_ERROR("User '" ++ binary_to_list(UserName) ++ "' is not exist")
            end,
    {reply, Reply, []};
handle_call({show_allow_roles}, _From, State) ->
    [{_, AllowRoles}] = ets:lookup(acl_simple, allow_roles),
    Reply = ?JSON_SHOW_ALLOW_ROLES(AllowRoles),
    {reply, Reply, State};
handle_call({add_allow_roles, ListRoles}, _From, State) ->
    [{_, AllowRoles}] = ets:lookup(acl_simple, allow_roles),
    ok = validation_add_allow_roles(ListRoles, AllowRoles),
    Reply = add_allow_roles_handler(ListRoles),
    {reply, Reply, State};
handle_call({delete_allow_roles, ListRoles}, _From, State) ->
    [{_, AllowRoles}] = ets:lookup(acl_simple, allow_roles),
    ok = validation_roles(ListRoles, AllowRoles),
    Reply = delete_allow_roles_handler(ListRoles),
```

```
    {reply, Reply, State};
handle_call(_Msg, _From, State) ->
    {reply, unknown_req, State}.

handle_cast(_Msg, State) ->
    {noreply, State}.

handle_info(terminate, State) ->
    {stop, normal, ok, State};
handle_info(_Msg, State) ->
    {noreply, State}.

code_change(_OldVsn, State, _Extra) ->
    {ok, State}.



% ====================================================
% Help-functions for inverse functions
% ====================================================

-spec add_allow_roles_handler(binary()) -> map().
add_allow_roles_handler([]) ->
    ?JSON_OK;
add_allow_roles_handler([Role|ListRoles]) ->
    [{_, AllowRoles}] = ets:lookup(acl_simple, allow_roles),
    case lists:member(Role, AllowRoles) of
        true ->
            add_allow_roles_handler(ListRoles);
        false ->
            {ok, _} = acl_simple_pg:insert("add_allow_role", [Role]),
            true = ets:insert(acl_simple, [{allow_roles, [Role|AllowRoles]}]),
            add_allow_roles_handler(ListRoles)
    end.

-spec delete_allow_roles_handler(list()) -> map().
delete_allow_roles_handler([]) ->
    ?JSON_OK;
```

```
delete_allow_roles_handler([Role|ListRoles]) ->
    [{_, AllowRoles}] = ets:lookup(acl_simple, allow_roles),
    {ok, _} = acl_simple_pg:delete("delete_allow_role_in_roles", [Role]),
    {ok, _} = acl_simple_pg:delete("delete_allow_role", [Role]),
    ok = acl_simple_timer_cache:refresh_cache(),
    true = ets:insert(acl_simple, [{allow_roles, AllowRoles -- [Role]}]),
    delete_allow_roles_handler(ListRoles).

-spec user_add_handler(binary(), list()) -> map().
user_add_handler(User, PassHash) ->
    [{_, Cache}] = ets:lookup(acl_simple, server_cache),
    [{_, CachePassHash}] = ets:lookup(acl_simple, customer_passhash),
    ok = validation_user_add(User, Cache),
    ok = validation_passhash(PassHash),
    Uuid = list_to_binary(uuid:to_string(simple, uuid:uuid1())),
    % binary_to_list(crypto:hash(sha, <<"1234">>)).
    {ok, _} = acl_simple_pg:insert("user_add", [Uuid, User, jsone:encode(PassHash)]),
    true = ets:insert(acl_simple, [{server_cache, Cache#{User => []}}]),
    true = ets:insert(acl_simple, [{customer_passhash, CachePassHash#{User => PassHash}}]),
    ?JSON_OK.

-spec user_delete_handler(binary()) -> map().
user_delete_handler(User) ->
    [{_, Cache}] = ets:lookup(acl_simple, server_cache),
    [{_, CachePassHash}] = ets:lookup(acl_simple, customer_passhash),
    ok = validation_user(User, Cache),
    #{User := RolesList} = Cache,
    ok = delete_roles_in_db(RolesList, User),
    {ok, _} = acl_simple_pg:delete("users_delete_by_name", [User]),
    NewCache = maps:remove(User, Cache),
    true = ets:insert(acl_simple, [{server_cache, NewCache}]),
    NewCachePassHash = maps:remove(User, CachePassHash),
    true = ets:insert(acl_simple, [{customer_passhash, NewCachePassHash}]),
    ?JSON_OK.

-spec roles_add_handler(binary(), list()) -> map().
roles_add_handler(User, Roles) ->
```

```
    [{_, Cache}] = ets:lookup(acl_simple, server_cache),
    [{_, AllowRoles}] = ets:lookup(acl_simple, allow_roles),
    ok = validation_user(User, Cache),
    ok = validation_roles(Roles, AllowRoles),
    #{User := RolesOld} = Cache,
    RolesAdd = insert_roles_in_db(User, RolesOld, Roles),
    case lists:member(error, RolesAdd) of
        true ->
            RolesAdd1 = lists:delete(error, RolesAdd),
            NewMap = Cache#{User := RolesOld ++ RolesAdd1},
            true = ets:insert(acl_simple, [{server_cache, NewMap}]),
            ?JSON_ERROR("db error");
        false ->
            NewCache = Cache#{User := RolesOld ++ RolesAdd},
            true = ets:insert(acl_simple, [{server_cache, NewCache}]),
            ?JSON_OK
    end.

-spec roles_delete_handler(binary(), list()) -> map().
roles_delete_handler(User, Roles) ->
    [{_, Cache}] = ets:lookup(acl_simple, server_cache),
    [{_, AllowRoles}] = ets:lookup(acl_simple, allow_roles),
    ok = validation_user(User, Cache),
    ok = validation_roles(Roles, AllowRoles),
    #{User := OldRoles} = Cache,
    ok = delete_roles_in_db(Roles, User),
    NewCache = Cache#{User := OldRoles -- Roles},
    true = ets:insert(acl_simple, [{server_cache, NewCache}]),
    ?JSON_OK.


% ==================================================

-spec validation_add_allow_roles(list(), list()) -> ok.
validation_add_allow_roles([], _) ->
    ok;
validation_add_allow_roles([Role|RoleList], AllowRoles) ->
```

```
    LenRole = byte_size(Role),
    {match, [{0, LenRole}]} = re:run(Role, "[a-zA-Z][a-zA-Z_\\d]*", [unicode]),
    validation_add_allow_roles(RoleList, AllowRoles).

-spec validation_roles(list(), list()) -> ok.
validation_roles([], _) ->
    ok;
validation_roles([Role|RoleList], AllowRoles) ->
    LenRole = byte_size(Role),
    {match, [{0, LenRole}]} = re:run(Role, "[a-zA-Z][a-zA-Z_\\d]*", [unicode]),
    true = lists:member(Role, AllowRoles),
    validation_roles(RoleList, AllowRoles).

-spec validation_user_add(binary(), map()) -> ok.
validation_user_add(User, Cache) ->
    LenRole = byte_size(User),
    {match, [{0, LenRole}]} = re:run(User, "[a-zA-Z][a-zA-Z_\\d]*", [unicode]),
    error = maps:find(User, Cache),
    ok.

-spec validation_user(binary(), map()) -> ok.
validation_user(User, Cache) ->
    LenRole = byte_size(User),
    {match, [{0, LenRole}]} = re:run(User, "[a-zA-Z][a-zA-Z_\\d]*", [unicode]),
    {ok, _} = maps:find(User, Cache),
    ok.

-spec validation_passhash(list()) -> ok.
validation_passhash(PassHash) ->
    20 = length(PassHash),
    PassHash = lists:map(fun(Num) -> true = Num > 0, true = Num < 256, Num end, PassHash),
    ok.



-spec delete_roles_in_db(list(), binary()) -> ok | {error, any()}.
delete_roles_in_db([], _) -> ok;
```

```
delete_roles_in_db([Role|T], User) ->
    case acl_simple_pg:delete("roles_delete_by_name", [User, Role]) of
        {ok, _} ->
            ?LOG_DEBUG("Role ~p was delete in user ~p", [Role, User]),
            delete_roles_in_db(T, User);
        {error, Error} ->
            ?LOG_ERROR("db error ~p", [Error]),
            {error, Error}
    end.

-spec insert_roles_in_db(binary(), list(), list()) -> list().
insert_roles_in_db(_, _RolesOld, []) ->
    [];
insert_roles_in_db(User, RolesOld, [Role|T]) ->
    case lists:member(Role, RolesOld) of
        false ->
            case acl_simple_pg:insert("roles_add_by_name", [User, Role]) of
                {error, Error} ->
                    ?LOG_ERROR("db error ~p", [{error, Error}]),
                    [error];
                {ok, _} ->
                    ?LOG_DEBUG("Add role ~p for user ~p", [Role, User]),
                    [Role | insert_roles_in_db(User, RolesOld, T)]
            end;
        true ->
            insert_roles_in_db(User, RolesOld, T)
    end.
```

acl_simple_sup.erl

```
-module(acl_simple_sup).
-author('Mykhailo Krasniuk <miha.190901@gmail.com>').
```

```
-behaviour(supervisor).

-export([start_link/0]).
-export([init/1]).

-define(SERVER, ?MODULE).

start_link() ->
   supervisor:start_link({local, ?SERVER}, ?MODULE, []).

init([]) ->
   {ok, Args} = application:get_env(poolboy, pools),
   PoolSpecs = lists:map(fun({Name, PoolArgs, WorkerArgs}) ->
      poolboy:child_spec(Name, PoolArgs, WorkerArgs)
                  end, Args),
   SupFlags = {one_for_one, 5, 100},
   Server = {acl_simple_server, {acl_simple_server, start_link, []}, permanent, 1000, worker, []},
   TimerCache = {acl_simple_timer_cache, {acl_simple_timer_cache, start_link, []}, permanent,
1000, worker, []},

   {ok, {SupFlags, PoolSpecs ++ [TimerCache, Server]}}.
```

acl_simple_timer_cache.erl

```
-module(acl_simple_timer_cache).
-author('Mykhailo Krasniuk <miha.190901@gmail.com>').
-behavior(gen_server).

-include("acl_simple.hrl").

-export([start_link/0]).
-export([init/1, handle_call/3, handle_cast/2, handle_info/2, terminate/2, code_change/3]).
```

```
-export([refresh_cache/0]).

% ==================================================
% Users functions
% ==================================================

start_link() ->
    gen_server:start_link(?MODULE, [], []).

refresh_cache() ->
    [{_, Pid}] = ets:lookup(acl_simple, acl_simple_timer_cache),
    _Timer = erlang:send_after(10, Pid, refresh_cache),
    ok.



% ==================================================
% Inverse functions
% ==================================================

init([]) ->
    true = ets:insert(acl_simple, [{acl_simple_timer_cache, self()}]),
    {ok, PauseTime} = application:get_env(acl_simple, timer_cache),
    {ok, PauseAllowRoles} = application:get_env(acl_simple, timer_allow_roles),
    TCache = erlang:send_after(200, self(), {timer_cache, PauseTime}),
    TAllowRoles = erlang:send_after(205, self(), {timer_allow_roles, PauseAllowRoles}),
    {ok, #{timer_cache => TCache,
          timer_allow_roles => TAllowRoles
    }}.

terminate(_, _State) ->
    ok.
code_change(_OldVsn, State, _Extra) ->
    {ok, State}.
handle_call(_Data, _From, State) ->
    {reply, unknown_req, State}.
handle_cast(_Data, State) ->
    {noreply, State}.
```

```
handle_info({timer_cache, PauseTime}, #{timer_cache := T} = State) ->
    _ = erlang:cancel_timer(T),
    Timer = case timer_cache_handler() of
                {error, _} ->
                    erlang:send_after(2000, self(), {timer_cache, PauseTime});
                {MapCache, MapPassHash} ->
                    true = ets:insert(acl_simple, [{server_cache, MapCache}]),
                    true = ets:insert(acl_simple, [{customer_passhash, MapPassHash}]),
                    erlang:send_after(PauseTime, self(), {timer_cache, PauseTime})
            end,
    {noreply, State#{timer_cache := Timer}};
handle_info({timer_allow_roles, PauseTime}, #{timer_allow_roles := T} = State) ->
    _ = erlang:cancel_timer(T),
    Timer = case allow_roles_handler() of
                {error, _} ->
                    erlang:send_after(2000, self(), {timer_allow_roles, PauseTime});
                List ->
                    true = ets:insert(acl_simple, [{allow_roles, List}]),
                    erlang:send_after(PauseTime, self(), {timer_allow_roles, PauseTime})
            end,
    {noreply, State#{timer_allow_roles := Timer}};
handle_info(refresh_cache, State) ->
    {Cache, CachePassHash} = timer_cache_handler(),
    true = ets:insert(acl_simple, [{server_cache, Cache}]),
    true = ets:insert(acl_simple, [{customer_passhash, CachePassHash}]),
    AllowRoles = allow_roles_handler(),
    true = ets:insert(acl_simple, [{allow_roles, AllowRoles}]),
    {noreply, State};
handle_info(_Data, State) ->
    {noreply, State}.


% ==================================================
% Help-functions for inverse functions
% ==================================================
```

```
-spec allow_roles_handler() -> list() | {error, any()}.
allow_roles_handler() ->
    case acl_simple_pg:select("get_allow_roles", []) of
        {error, Error} ->
            {error, Error};
        {ok, _, AllowRoles} ->
            handler_convert_to_map(AllowRoles)
    end.

-spec timer_cache_handler() -> {error, any()} | {map(), map()}.
timer_cache_handler() ->
    case acl_simple_pg:select("get_all_users", []) of
        {error, Error} ->
            {error, Error};
        {ok, _, Users} ->
            convert_to_map(Users, #{}, #{})
    end.

-spec convert_to_map(list(), #{}, #{}) -> {map(), map()}.
convert_to_map([], MapCache, MapPassHash) ->
    {MapCache, MapPassHash};
convert_to_map([{Name, PassHash} | T], MapCache, MapPassHash) ->
    {ok, _, RolesList_Dirty} = acl_simple_pg:select("get_roles_by_name", [Name]),
    RolesList = handler_convert_to_map(RolesList_Dirty),
    MapCache1 = MapCache#{Name => RolesList},
    MapPassHash1 = MapPassHash#{Name => jsone:decode(PassHash)},
    convert_to_map(T, MapCache1, MapPassHash1).


%% ----------------------------------------

-spec handler_convert_to_map(list()) -> list().
handler_convert_to_map([]) ->
    [];
handler_convert_to_map([{Role} | T]) ->
    [Role | handler_convert_to_map(T)].
```

2. Common тести.

sys_config.hrl

```
-author('Mykhailo Krasniuk <miha.190901@gmail.com>').

-define(START_ENV,
  [
    {acl_simple, [
      {listen_port, 1913},
      {timer_cache, 10000},
      {timer_allow_roles, 90000}
    ]},
    {poolboy, [
      {pools, [
        {pg_pool, [
          {name, {local, pg_pool}},
          {strategy, fifo},
          {worker_module, acl_simple_pg},
          {size, 10},
          {max_overflow, 5}
        ], [
          {host, "127.0.0.1"},
          {port, 5432},
          {username, "trembita"},
          {password, "trembita"},
          {database, "civil"}]}
      ]}
    ]},
    {lager, [
      {handlers, [
        {lager_console_backend, [{level, debug}]},
        {lager_file_backend, [
```

```
            {file, "error.log"},
            {level, error},
            {size, 1073741824}, %% 1Gb
            {date, "$D0"},
            {count, 20}
        ]},
        {lager_file_backend, [
            {file, "warning.log"},
            {level, warning},
            {size, 1073741824}, %% 1Gb
            {date, "$D0"},
            {count, 20}
        ]},
        {lager_file_backend, [
            {file, "info.log"},
            {level, info},
            {size, 1073741824}, %% 1Gb
            {date, "$D0"},
            {count, 7}
        ]},
        {lager_file_backend, [
            {file, "debug.log"},
            {level, debug},
            {size, 1073741824}, %% 1Gb
            {date, "$D0"},
            {count, 2}
        ]}
    ]},
    {crash_log, "crash.log"},
    {crash_log_msg_size, 65536},
    {crash_log_size, 10485760},
    {crash_log_date, "$D0"},
    {crash_log_count, 5},
    {error_logger_redirect, true},
    {error_logger_hwm, 400}
  ]}
].
```

admin_fail_SUITE.erl

```
-module(admin_fail_SUITE).
-author('Mykhailo Krasniuk <mykhailo.krasniuk@privatbank.ua>').

-include_lit("common_test/include/ct.hrl").
-include("include/sys_config.hrl").

-export([all/0, init_per_suite/1, end_per_suite/1, groups/0]).

-export([missing_body/1, bad_http_method/1, parse_body_fail/1, add_allow_roles/1]).

-define(URL_ADMIN, "http://127.0.0.1:1913/admin").
-define(HEADERS, [{"Content-type", "application/json;charset=UTF-8"}]).



%% ====================================
%% Export for Common Tests
%% ====================================

init_per_suite(Config) ->
  %  ok = application:set_env(?START_ENV),
 %   ok = acl_simple:start(),
    Config.

groups() ->
    [].

end_per_suite(Config) ->
    ok = acl_simple:stop(),
    Config.



all() ->
```

```
    [
        missing_body,
        bad_http_method,
        parse_body_fail,
        add_allow_roles
    ].


%% ==================================================
%% Cases
%% ==================================================

missing_body(_Config) ->
    {ok, {{_, 400, _}, _Headers, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ""},
        [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("RespBody = ~p", [RespBody]),
    <<"Missing body.">> = RespBody.

bad_http_method(_Config) ->
    {ok, {{_, 405, _}, _Headers, []}} = httpc:request(get, {?URL_ADMIN, []}, [{timeout, 4000}], []).

parse_body_fail(_Config) ->
    ReqBody = "{\"parameters\":{\"method\":\"show_allow_roles\"",
    {ok, {{_, 206, _}, _Headers, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
        [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("RespBody = ~p", [RespBody]),
    #{<<"fail">> := <<"Invalid request format">>} = jsone:decode(RespBody).

add_allow_roles(_Config) ->
    #{<<"fail">> := <<"Invalid request format">>} = request(add_allow_roles, {[<<"1a">>]}),
    #{<<"fail">> := <<"Invalid request format">>} = request(add_allow_roles, {[<<"_a">>]}),
    #{<<"fail">> := <<"Invalid request format">>} = request(add_allow_roles, {[<<"am./">>]}),
    #{<<"fail">> := <<"Invalid request format">>} = request(add_allow_roles, {[<<"am ">>]}),
    #{<<"fail">> := <<"Invalid request format">>} = request(add_allow_roles, {[<<"r1913">>,
<<"w1913">>, <<"am ">>]}).
```

```
% ----------------------------------

-spec request(atom(), tuple()) -> ok | list().
request(add_allow_roles, {Roles}) ->
    Body = #{<<"auth">> => #{<<"login">> => <<"admin">>, <<"passhash">> =>
binary_to_list(crypto:hash(sha, <<"1234">>))},
        <<"parameters">> => #{<<"method">> => <<"add_allow_roles">>, <<"roles">> => Roles}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
        [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("show_allow_roles ~n RespBody = ~p", [RespBody]),
    jsone:decode(RespBody);
request(delete_allow_roles, {Roles}) ->
    Body = #{<<"auth">> => #{<<"login">> => <<"admin">>, <<"passhash">> =>
binary_to_list(crypto:hash(sha, <<"1234">>))},
        <<"parameters">> => #{<<"method">> => <<"delete_allow_roles">>, <<"roles">> => Roles}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
        [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("show_allow_roles ~n RespBody = ~p", [RespBody]),
    #{<<"result">> := <<"ok">>} = jsone:decode(RespBody),
    ok;

request(user_add, {User, PassHash}) ->
    Body = #{<<"auth">> => #{<<"login">> => <<"admin">>,
        <<"passhash">> => binary_to_list(crypto:hash(sha, <<"1234">>))},
        <<"parameters">> => #{<<"method">> => <<"user_add">>,
            <<"user">> => User,
            <<"passhash">> => PassHash}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
        [{timeout, 4000}], [{body_format, binary}]),
```

```
    ok = ct:pal("user_add ~p ~n RespBody = ~p", [User, RespBody]),
    #{<<"result">> := <<"ok">>} = jsone:decode(RespBody),
    ok;
request(user_delete, {User}) ->
    Body = #{<<"auth">> => #{<<"login">> => <<"admin">>,
        <<"passhash">> => binary_to_list(crypto:hash(sha, <<"1234">>))},
        <<"parameters">> => #{<<"method">> => <<"user_delete">>,
            <<"user">> => User}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
        [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("user_delete ~p ~n RespBody = ~p", [User, RespBody]),
    #{<<"result">> := <<"ok">>} = jsone:decode(RespBody),
    ok;
request(show_all_users, {}) ->
    Body = #{<<"auth">> => #{<<"login">> => <<"admin">>,
        <<"passhash">> => binary_to_list(crypto:hash(sha, <<"1234">>))},
        <<"parameters">> => #{<<"method">> => <<"show_all_users">>}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
        [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("show_all_users ~n RespBody = ~p", [RespBody]),
    #{<<"result">> := <<"ok">>,
        <<"users">> := ListUsers} = jsone:decode(RespBody),
    ListUsers;

request(show_roles, {User}) ->
    Body = #{<<"auth">> => #{<<"login">> => <<"admin">>, <<"passhash">> =>
binary_to_list(crypto:hash(sha, <<"1234">>))},
        <<"parameters">> => #{<<"method">> => <<"show_roles">>, <<"user">> => User}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
        [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("show_roles ~p ~n RespBody = ~p", [User, RespBody]),
```

```
    #{<<"result">> := <<"ok">>,
      <<"user">> := User,
      <<"roles">> := ListRoles} = jsone:decode(RespBody),
    ListRoles;
request(roles_add, {User, Roles}) ->
    Body = #{<<"auth">> => #{<<"login">> => <<"admin">>,
      <<"passhash">> => binary_to_list(crypto:hash(sha, <<"1234">>))},
      <<"parameters">> => #{<<"method">> => <<"roles_add">>,
        <<"user">> => User,
        <<"roles">> => Roles}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
      [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("roles_add ~p ~n RespBody = ~p", [{User, Roles}, RespBody]),
    #{<<"result">> := <<"ok">>} = jsone:decode(RespBody),
    ok;
request(roles_delete, {User, Roles}) ->
    Body = #{<<"auth">> => #{<<"login">> => <<"admin">>,
      <<"passhash">> => binary_to_list(crypto:hash(sha, <<"1234">>))},
      <<"parameters">> => #{<<"method">> => <<"roles_delete">>,
        <<"user">> => User,
        <<"roles">> => Roles}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
      [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("roles_delete ~p ~n RespBody = ~p", [{User, Roles}, RespBody]),
    #{<<"result">> := <<"ok">>} = jsone:decode(RespBody),
    ok.
```

admin_ok_script_SUITE.erl

```
-module(admin_ok_script_SUITE).
-author('Mykhailo Krasniuk <miha.190901@gmail.com>').
```

```
-include_lit("common_test/include/ct.hrl").
-include("include/sys_config.hrl").

-export([all/0, init_per_suite/1, end_per_suite/1, groups/0]).
-export([test_script/1, allow_roles/1, delete_allow_roles/1]).

-define(URL_ADMIN, "http://127.0.0.1:1913/admin").
-define(HEADERS, [{"Content-type", "application/json;charset=UTF-8"}]).



%% ====================================
%% Export for Common Tests
%% ====================================

init_per_suite(Config) ->
    ok = application:set_env(?START_ENV),
    ok = acl_simple:start(),

    ok = timer:sleep(1000),
    AllowRoles = request(show_allow_roles, {}),
    case length(AllowRoles) of
        0 ->
            ok = request(add_allow_roles, {[<<"read">>, <<"write">>, <<"exec">>]});
        _Other ->
            ok
    end,
    case lists:member(<<"r1913">>, AllowRoles) of
        false ->
            ok;
        true ->
            ok = request(delete_allow_roles, {[<<"r1913">>]})
    end,
    case lists:member(<<"w1913">>, AllowRoles) of
        false ->
            ok;
        true ->
```

```
        ok = request(delete_allow_roles, {[<<"w1913">>]})
    end,
    case lists:member(<<"e1913">>, AllowRoles) of
        false ->
            ok;
        true ->
            ok = request(delete_allow_roles, {[<<"e1913">>]})
    end,
    ListUsers = request(show_all_users, {}),
    IsMike = lists:member(<<"mike1913_test">>, ListUsers),
    IsKarl = lists:member(<<"karl1913_test">>, ListUsers),
    case {IsMike, IsKarl} of
        {false, false} ->
            ok;
        {true, true} ->
            ok = request(user_delete, {<<"mike1913_test">>}),
            ok = request(user_delete, {<<"karl1913_test">>});
        {_, true} ->
            ok = request(user_delete, {<<"karl1913_test">>});
        {true, _} ->
            ok = request(user_delete, {<<"mike1913_test">>})
    end,
    Config.

end_per_suite(Config) ->
   % ok = acl_simple:stop(),
    Config.

groups() ->
    [].

all() ->
    [
        test_script,
        allow_roles,
        delete_allow_roles
    ].
```

```
%% ---------------------------------
%% Cases
%% ---------------------------------

test_script(_Config) ->
    PassHash = binary_to_list(crypto:hash(sha, <<"1234">>)),
    ok = request(user_add, {<<"mike1913_test">>, PassHash}),
    ok = request(user_add, {<<"karl1913_test">>, PassHash}),
    Users = request(show_all_users, {}),
    true = lists:member(<<"mike1913_test">>, Users),
    true = lists:member(<<"karl1913_test">>, Users),
    AllowRoles = request(show_allow_roles, {}),
    Role1 = hd(AllowRoles),
    Role2 = lists:nth(2, AllowRoles),
    Role3 = lists:nth(3, AllowRoles),
    ok = request(roles_add, {<<"karl1913_test">>, [Role1]}),
    ok = request(roles_add, {<<"karl1913_test">>, [Role1, Role2]}),
    [Role1, Role2] = request(show_roles, {<<"karl1913_test">>}),
    ok = request(roles_delete, {<<"karl1913_test">>, [Role3]}),
    ok = request(roles_delete, {<<"karl1913_test">>, [Role3, Role1]}),
    [Role2] = request(show_roles, {<<"karl1913_test">>}),
    ok = request(roles_delete, {<<"karl1913_test">>, [Role2]}),
    Roles1 = request(show_roles, {<<"karl1913_test">>}),
    [] = Roles1,
    ok = request(roles_add, {<<"karl1913_test">>, [Role3, Role2]}),
    ok = request(user_delete, {<<"mike1913_test">>}),
    Users1 = request(show_all_users, {}),
    false = lists:member(<<"mike1913_test">>, Users1),
    ok = request(user_delete, {<<"karl1913_test">>}),
    Users2 = request(show_all_users, {}),
    false = lists:member(<<"mike1913_test">>, Users2).

allow_roles(_Config) ->
    AllowRoles = request(show_allow_roles, {}),
    ok = request(add_allow_roles, {[<<"r1913">>, <<"w1913">>]}),
```

```
    ok = request(add_allow_roles, {[<<"r1913">>, <<"w1913">>, <<"e1913">>]}),
    AllowRoles1 = request(show_allow_roles, {}),
    true = lists:member(<<"r1913">>, AllowRoles1),
    true = lists:member(<<"w1913">>, AllowRoles1),
    true = lists:member(<<"e1913">>, AllowRoles1),
    ok = request(delete_allow_roles, {[<<"r1913">>, <<"w1913">>, <<"e1913">>,
hd(AllowRoles)]}),
    AllowRoles2 = request(show_allow_roles, {}),
    false = lists:member(hd(AllowRoles), AllowRoles2),
    ok = request(add_allow_roles, {[hd(AllowRoles)]}),
    AllowRoles = request(show_allow_roles, {}).

delete_allow_roles(_Config) ->
    PassHash = binary_to_list(crypto:hash(sha, <<"1234">>)),
    ok = request(user_add, {<<"mike1913_test">>, PassHash}),
    ok = request(user_add, {<<"karl1913_test">>, PassHash}),
    ok = request(add_allow_roles, {[<<"r1913">>, <<"w1913">>, <<"e1913">>]}),
    AllowRoles = request(show_allow_roles, {}),
    true = lists:member(<<"e1913">>, AllowRoles),
    ok = request(roles_add, {<<"karl1913_test">>, [<<"r1913">>, <<"w1913">>]}),
    ok = request(roles_add, {<<"mike1913_test">>, [<<"r1913">>, <<"w1913">>, <<"e1913">>]}),
    KarlRoles = request(show_roles, {<<"karl1913_test">>}),
    MikeRoles = request(show_roles, {<<"mike1913_test">>}),
    true = lists:member(<<"r1913">>, KarlRoles),
    true = lists:member(<<"w1913">>, MikeRoles),
    ok = request(delete_allow_roles, {[<<"r1913">>, <<"w1913">>]}),
    ok = timer:sleep(100),
    [<<"e1913">>] = request(show_roles, {<<"mike1913_test">>}),
    [] = request(show_roles, {<<"karl1913_test">>}),
    AllowRoles1 = request(show_allow_roles, {}),
    false = lists:member(<<"w1913">>, AllowRoles1),
    true = lists:member(<<"e1913">>, AllowRoles1),
    ok = request(delete_allow_roles, {[<<"e1913">>]}),
    ok = request(user_delete, {<<"mike1913_test">>}),
    Users1 = request(show_all_users, {}),
    false = lists:member(<<"mike1913_test">>, Users1),
    ok = request(user_delete, {<<"karl1913_test">>}),
```

```
    Users2 = request(show_all_users, {}),
    false = lists:member(<<"mike1913_test">>, Users2).


% ----------------------------------


-spec request(atom(), tuple()) -> ok | list().
request(show_allow_roles, {}) ->
    Body = #{<<"auth">> => #{<<"login">> => <<"admin">>, <<"passhash">> =>
binary_to_list(crypto:hash(sha, <<"1234">>))},
        <<"parameters">> => #{<<"method">> => <<"show_allow_roles">>}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
        [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("show_allow_roles ~n RespBody = ~p", [RespBody]),
    #{<<"result">> := <<"ok">>,
        <<"roles">> := AllowRoles} = jsone:decode(RespBody),
    AllowRoles;
request(add_allow_roles, {Roles}) ->
    Body = #{<<"auth">> => #{<<"login">> => <<"admin">>, <<"passhash">> =>
binary_to_list(crypto:hash(sha, <<"1234">>))},
        <<"parameters">> => #{<<"method">> => <<"add_allow_roles">>, <<"roles">> => Roles}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
        [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("show_allow_roles ~n RespBody = ~p", [RespBody]),
    #{<<"result">> := <<"ok">>} = jsone:decode(RespBody),
    ok;
request(delete_allow_roles, {Roles}) ->
    Body = #{<<"auth">> => #{<<"login">> => <<"admin">>, <<"passhash">> =>
binary_to_list(crypto:hash(sha, <<"1234">>))},
        <<"parameters">> => #{<<"method">> => <<"delete_allow_roles">>, <<"roles">> => Roles}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
```

```
        [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("show_allow_roles ~n RespBody = ~p", [RespBody]),
    #{<<"result">> := <<"ok">>} = jsone:decode(RespBody),
    ok;

request(user_add, {User, PassHash}) ->
    Body = #{<<"auth">> => #{<<"login">> => <<"admin">>,
                             <<"passhash">> => binary_to_list(crypto:hash(sha, <<"1234">>))},
             <<"parameters">> => #{<<"method">> => <<"user_add">>,
                                   <<"user">> => User,
                                   <<"passhash">> => PassHash}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
        [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("user_add ~p ~n RespBody = ~p", [User, RespBody]),
    #{<<"result">> := <<"ok">>} = jsone:decode(RespBody),
    ok;
request(user_delete, {User}) ->
    Body = #{<<"auth">> => #{<<"login">> => <<"admin">>,
                             <<"passhash">> => binary_to_list(crypto:hash(sha, <<"1234">>))},
             <<"parameters">> => #{<<"method">> => <<"user_delete">>,
                                   <<"user">> => User}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
        [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("user_delete ~p ~n RespBody = ~p", [User, RespBody]),
    #{<<"result">> := <<"ok">>} = jsone:decode(RespBody),
    ok;
request(show_all_users, {}) ->
    Body = #{<<"auth">> => #{<<"login">> => <<"admin">>,
                             <<"passhash">> => binary_to_list(crypto:hash(sha, <<"1234">>))},
             <<"parameters">> => #{<<"method">> => <<"show_all_users">>}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
```

```
      [{timeout, 4000}], [{body_format, binary}]),
   ok = ct:pal("show_all_users ~n RespBody = ~p", [RespBody]),
   #{<<"result">> := <<"ok">>,
      <<"users">> := ListUsers} = jsone:decode(RespBody),
   ListUsers;

request(show_roles, {User}) ->
   Body = #{<<"auth">> => #{<<"login">> => <<"admin">>, <<"passhash">> =>
binary_to_list(crypto:hash(sha, <<"1234">>))},
          <<"parameters">> => #{<<"method">> => <<"show_roles">>, <<"user">> => User}},
   ReqBody = jsone:encode(Body),
   {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
      [{timeout, 4000}], [{body_format, binary}]),
   ok = ct:pal("show_roles ~p ~n RespBody = ~p", [User, RespBody]),
   #{<<"result">> := <<"ok">>,
      <<"user">> := User,
      <<"roles">> := ListRoles} = jsone:decode(RespBody),
   ListRoles;
request(roles_add, {User, Roles}) ->
   Body = #{<<"auth">> => #{<<"login">> => <<"admin">>,
                  <<"passhash">> => binary_to_list(crypto:hash(sha, <<"1234">>))},
         <<"parameters">> => #{<<"method">> => <<"roles_add">>,
                  <<"user">> => User,
                  <<"roles">> => Roles}},
   ReqBody = jsone:encode(Body),
   {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
      [{timeout, 4000}], [{body_format, binary}]),
   ok = ct:pal("roles_add ~p ~n RespBody = ~p", [{User, Roles}, RespBody]),
   #{<<"result">> := <<"ok">>} = jsone:decode(RespBody),
   ok;
request(roles_delete, {User, Roles}) ->
   Body = #{<<"auth">> => #{<<"login">> => <<"admin">>,
                  <<"passhash">> => binary_to_list(crypto:hash(sha, <<"1234">>))},
         <<"parameters">> => #{<<"method">> => <<"roles_delete">>,
                  <<"user">> => User,
```

```
                        <<"roles">> => Roles}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_ADMIN, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
        [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("roles_delete ~p ~n RespBody = ~p", [{User, Roles}, RespBody]),
    #{<<"result">> := <<"ok">>} = jsone:decode(RespBody),
    ok.
```

customer_ok_SUITE.erl

```
-module(customer_ok_SUITE).
-author('Mykhailo Krasniuk <mykhailo.krasniuk@privatbank.ua>').

-include_lit("common_test/include/ct.hrl").
-include("include/sys_config.hrl").

-export([all/0, init_per_suite/1, end_per_suite/1, groups/0]).
-export([get_roles/1]).

-define(URL_CUSTOMER, "http://127.0.0.1:1913/customer").
-define(HEADERS, [{"Content-type", "application/json;charset=UTF-8"}]).


%% ====================================
%% Export for Common Tests
%% ====================================

init_per_suite(Config) ->
    %  ok = application:set_env(?START_ENV),
    %   ok = acl_simple:start(),
    Config.
```

```
groups() ->
    [].

end_per_suite(Config) ->
    %ok = acl_simple:stop(),
    Config.



all() ->
    [
        get_roles
    ].



%% ====================================================
%% Cases
%% ====================================================

get_roles(_Config) ->
    Body = #{<<"auth">> => #{<<"login">> => <<"mike">>, <<"passhash">> =>
binary_to_list(crypto:hash(sha, <<"1234">>))},
        <<"parameters">> => #{<<"method">> => <<"get_roles">>}},
    ReqBody = jsone:encode(Body),
    {ok, {_, _, RespBody}} = httpc:request(post, {?URL_CUSTOMER, ?HEADERS,
"application/json;charset=UTF-8", ReqBody},
        [{timeout, 4000}], [{body_format, binary}]),
    ok = ct:pal("get_roles ~p ~n RespBody = ~p", [<<"mike">>, RespBody]),
    #{<<"result">> := <<"ok">>,
        <<"roles">> := _ListRoles} = jsone:decode(RespBody).
```

ct.spec

```
{suites, "./ct/",
   [
      admin_ok_script_SUITE,
      customer_ok_SUITE,
      admin_fail_SUITE
   ]
}.
```

3. Навантажувальне тестування wrk. Скрипти:

customer_get_roles.lua

```
request = function()
   path = "/customer"
   wrk.headers["Content-Type"] = "application/json; charset=utf-8"
   wrk.body =
'{"auth":{"login":"lion_test","passhash":[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,
172,13,44,2,32]},"parameters":{"method":"get_roles"}}'
   return wrk.format("POST", path)
end

response = function(status, headers, body)
   --io.write('----------------------------\n')
   --io.write('status = ' .. status ..'\n')
   --io.write('body = '.. body ..'\n')
end
```

show_all_users.lua

```
request = function()
   path = "/admin"
   wrk.headers["Content-Type"] = "application/json; charset=utf-8"
   wrk.body =
'{"auth":{"login":"admin","passhash":[113,16,237,164,208,158,6,42,165,228,163,144,176,165,114,17
2,13,44,2,32]},"parameters":{"method":"show_all_users"}}'
   return wrk.format("POST", path)
end

response = function(status, headers, body)
   --io.write('----------------------------\n')
   --io.write('status = ' .. status ..'\n')
   --io.write('body = '.. body ..'\n')
end
```

4. База даних, створення таблиць:

admins

```
-- Table: public.admins

-- DROP TABLE IF EXISTS public.admins;

CREATE TABLE IF NOT EXISTS public.admins
(
   login character varying COLLATE pg_catalog."default" NOT NULL,
   passhash json NOT NULL
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.admins
   OWNER to trembita;
```

allow_roles

```
-- Table: public.allow_roles

-- DROP TABLE IF EXISTS public.allow_roles;

CREATE TABLE IF NOT EXISTS public.allow_roles
(
    role character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT allow_roles_pkey PRIMARY KEY (role)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.allow_roles
    OWNER to trembita;
```

roles

```
-- Table: public.roles

-- DROP TABLE IF EXISTS public.roles;

CREATE TABLE IF NOT EXISTS public.roles
(
    user_id character varying(50) COLLATE pg_catalog."default" NOT NULL,
    role character varying(10) COLLATE pg_catalog."default" NOT NULL
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.roles
    OWNER to trembita;
```

users

```
-- Table: public.users

-- DROP TABLE IF EXISTS public.users;

CREATE TABLE IF NOT EXISTS public.users
(
    id character varying(50) COLLATE pg_catalog."default" NOT NULL,
    name character varying(50) COLLATE pg_catalog."default" NOT NULL,
    passhash json NOT NULL,
    CONSTRAINT users_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.users
    OWNER to trembita;
```

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу бакалавра

на тему:

«Автоматизована система формування та контролю авторизації користувачів»

студента групи 122-19-3 Краснюка Михайла Андрійовича

ДОДАТОК В

Перелік файлів на диску

ПЕРЕЛІК ДОКУМЕНТІВ НА МАГНІТНОМУ НОСІЇ

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Краснюк.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Краснюк.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
Краснюк.rar	Архів. Містить коди програми та скомпільовану програму.