СОФИЙСКИ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“

ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА

ПРОЕКТ ПО WEB

Тема: w22ed-001/55.1
Система за създаване и управление на БД и уеб настройки (dashboard)

Имена: Йоан Венелинов Йорданов, Стоян Красимиров Николов

Факултетни номера: 5MI0800321, 5MI0800347

Начална година: 2025

Програма: бакалавър, (КН)

Предмет: w25prj_KN_final

Имейл: ioaniordanov123@gmail.com, nikolovs2003@abv.bg

Курс: 4

Преподавател: проф. д-р Милен Петров

Дата: 2026-02-11

Съдържание

1. Условие

Всяко уеб приложение изисква специфична първоначална настройка за своята инсталация. Целта на проекта е да се автоматизира този процес чрез специализиран Dashboard. Системата трябва да позволява импортиране на списък с проекти, съдържащ данни за участници (факултетни номера), кратко име, версия и тип на средата (напр. 7777, 8888, myweb1, version 2, type mysql/apache2/xampp).

Въз основа на типа на импортираните данни, системата трябва да приложи съответния шаблон (за версия на БД или XAMPP), чрез който автоматизирано да се извършат следните действия:

- Създаване на потребител и база данни.
- Генериране на конфигурационни файлове (напр. config_db.php) и копирането им на предефиниран път (напр. webtech/w22_7777_8888_myweb1_v2/php/config_db.php).
- Преглед, редакция и импорт на прилежащи SQL скриптове (db.sql, db_init_data.sql и др.).

Йоан Йорданов 5MI0800321
Стоян Николов 5MI0800347

2. Въведение – извличане на изискванията

Настоящият проект цели създаването на централизирана система (Dashboard) за автоматизирано управление на учебни и тестови уеб проекти. Основният проблем, който системата решава, е времеемкото и податливо на грешки ръчно конфигуриране на средата (бази данни, потребители, конфигурационни файлове) при инсталиране на множество приложения.

2.1. Роли в системата

Системата обслужва две основни потребителски роли, всяка с дефинирани права на достъп:

1. Администратор:

- Има пълен достъп до всички модули на системата.
- Управлява глобалните настройки (сървъри, root права за БД).
- Извършва масов импорт (bulk import) на проекти.
- Дефинира и редактира шаблони за инсталация (версии на MySQL/XAMPP).
- Има достъп до административния панел за архивиране и възстановяване.

2. Потребител (Студент/Разработчик):

- Има достъп до Потребителския панел.
- Може да се регистрира и автентикира в системата.
- Управлява собствения си профил (промяна на данни, парола).
- Може да преглежда статуса на своите проекти (в зависимост от правата, дадени от администратора).

2.2. Функционални изисквания

Системата трябва да реализира следните специфични функции, групирани по модули:

А. Модул "Управление на проекти и импорт"

- Масов импорт: Системата трябва да приема списък (CSV/Text) с дефинирани полета (ФН, име, версия, тип) и да го парсва коректно.
- Генериране на конфигурации: Автоматично създаване на конфигурационни файлове (напр. config_db.php) на базата на шаблони и копирането им в директориите на проектите.
- Филтрация и търсене: Възможност за търсене на проекти по тагове, версии и имена.

Б. Модул "Бази данни (DB Management)"

- Създаване на ресурси: Автоматично създаване на база данни и асоцииран потребител с генерирана парола за всеки проект.
- SQL Екзекуция: Възможност за изпълнение на SQL скриптове върху една или множество бази едновременно.
- Миграции: Поддръжка на различни версии/ревизии на структурата на базата данни.
- Сървърни записи: Съхраняване на мета-данни за връзка (host, user, pass) за всяка създадена БД.

В. Модул "Архивиране и Възстановяване"

- Backup: Създаване на архиви на файловата структура и дъмп на базата данни за конкретен проект или група проекти.
- Версиониране: Управление на хронологията на архивите.

Г. Модул "Отчети и Експорт"

- Експорт към Excel: Генериране на справка с всички активни проекти и техните настройки.

2.3. Нефункционални изисквания

- Сигурност: Паролите на потребителите и достъпите до базите данни трябва да се съхраняват защитено. Root достъпът до MySQL сървъра трябва да е изолиран от обикновените потребители.
- Надеждност: При грешка в един от проектите при масов импорт, процесът не трябва да прекъсва работата на цялата система (transaction safety/error handling).
- Удобство на интерфейса (Usability): Интуитивен Dashboard дизайн за лесна навигация между стотици проекти.

2.4. Ползи от реализацията

- Ефективност: Драстично намаляване на времето за разгръщане на студентски проекти (от часове на минути).
- Стандартизация: Унифициране на конфигурационните файлове и структурите на базите данни, което улеснява проверката и оценяването.
- Минимизиране на грешките: Елиминиране на човешкия фактор при писане на конфигурационни файлове и даване на права в БД.
- Образователна стойност: Демонстрация на добри практики за DevOps и автоматизация на процеси.

3. Теория – анализ и проектиране на решението

За реализацията на системата е избран подходът на Client-Server архитектура, реализирана чрез Native PHP (без използване на тежки Framework-ци) за сървърната част и Native JavaScript/HTML за клиентската част. Това решение осигурява максимална производителност и пълен контрол върху процесите по управление на сървърните ресурси.

3.1. Архитектурен модел

Приложението следва принципите на MVC (Model-View-Controller) архитектурата, адаптирана за RESTful API комуникация:

- Model (Данни): Представен е от базата данни и SQL заявките, капсулирани в контролерите или специализирани класове. Използва се PDO за сигурна комуникация с базата данни.
- View (Изглед): Клиентската част (public/index.html, pages/), която консумира данни чрез AJAX заявки към API-то. Няма сървърно генериране на HTML (Server-Side Rendering), което отделя логиката от визуализацията.
- Controller (Контролер): Обработва HTTP заявките, валидира входа, взаимодейства с базата данни и връща JSON отговор (src/Controllers).

3.2. Декомпозиция на приложението

Файловата структура е организирана логически, за да раздели отговорностите на отделните модули:

- public/ – Публично достъпна директория.
  - api.php: Входна точка (Entry Point) за всички API заявки. Рутира заявките към съответния контролер.
  - assets/ & pages/: Съдържат статичните ресурси (CSS, JS, HTML) за потребителския интерфейс.
- src/ – Ядро на приложението (Back-end логика).
  - Auth/: Модул за автентикация и авторизация (AuthService). Управлява сесиите и правата за достъп (RBAC).
  - Controllers/: Съдържа бизнес логиката. Например ProjectController.php управлява създаването на проекти, добавянето на участници и извличането на списъци.
  - Db/: Отговаря за връзката с базата данни (Connection клас, реализиращ Singleton pattern за PDO инстанцията).
  - Http/: Помощни класове за обработка на HTTP отговори и статус кодове (Response).
- database/: Съдържа SQL скриптове за инициализация (init.sql) и конфигурация за Docker контейнеризация.

3.3. Проектиране на базата данни (ER Модел)

Базата данни db_dashboard е проектирана релационно (MySQL), като основните същности са:

А. Потребители и Права

- users: Съхранява основните данни (име, email, факултетен номер, хеширана парола).
- roles & user_roles: Реализират Many-to-Many връзка за ролите (напр. 'admin', 'user').
- sessions: Управлява активните сесии чрез токени, валидни за определен период.

Б. Проекти и Конфигурация

- projects: Централна таблица. Съдържа информация за проекта (код, име, версия) и връзка към собственика (owner_id).
- project_participants: Свързваща таблица (Many-to-Many), позволяваща на един проект да има множество участници (студенти), които да го управляват.
- templates: Дефинира шаблоните за инсталация. Използва поле body_json за гъвкаво съхранение на конфигурационни стъпки (напр. SQL команди за създаване на user/db), без да се налага промяна на схемата при нови типове шаблони.
- servers: Описва връзката към сървърите (host, port, root user), където ще се разгръщат проектите.

В. Оперативни данни

- backups: Регистър на създадените архиви, техния тип (sql/code) и локация.

4. Използвани технологии

За реализацията на проекта е избран технологичен стек, базиран на отворени стандарти и нативни решения. Този подход гарантира висока производителност, лесна преносимост и минимални зависимости от външни библиотеки.

4.1. Сървърна част (Back-end)

Използван е Native PHP (версия 8.x). Това решение позволява пълна оптимизация на кода и директен контрол върху HTTP заглавията и потока на данните.

Архитектура: Реализиран е собствен MVC (Model-View-Controller) модел с RESTful API функционалност.

4.2. Клиентска част (Front-end)

HTML5 / CSS3: Използвани за структурното и визуално оформление на интерфейса. Дизайнът е реализиран чрез стандартни CSS правила, осигуряващи консистентност и лекота на зареждане.

JavaScript (ES6+): Използван е Native JavaScript за динамичната логика на приложението.

AJAX / Fetch API: Комуникацията със сървъра се извършва асинхронно. Клиентът изпраща JSON заявки към PHP контролерите и динамично обновява DOM дървото на страницата, без да е необходимо пълно презареждане (Single Page Application поведение).

4.3. Система за управление на бази данни (DBMS)

MySQL (версия 8.0): Използвана като основно хранилище за данните на приложението.

JSON поддръжка: Използване на нативния JSON тип данни в MySQL за гъвкаво съхранение на конфигурационни параметри и логове, без нужда от промяна на схемата на базата.

4.4. Среда и Инфраструктура (DevOps)

Docker & Docker Compose: Проектът е контейнеризиран, което гарантира, че средата за разработка е идентична за всички участници в екипа.

Git: Системата за контрол на версиите се използва за проследяване на промените в кода и съвместна работа.

5. Инсталация, настройки и DevOps

Системата е проектирана за бързо и лесно разгръщане (deployment) както в локална среда за разработка, така и върху продукционни сървъри. Поддържат се два основни метода за инсталация: чрез Docker (за базата данни) или чрез стандартен XAMPP/LAMP стек.

5.1. Предварителни изисквания

За да функционира системата, е необходимо наличието на:

- PHP 8.0+ (с активирани разширения pdo_mysql и json).
- MySQL 8.0+ (или Docker за контейнеризация на базата).
- Терминал / Command Line Interface.

5.2. Конфигурация

Основните настройки за връзка с базата данни се намират във файла config.json. Преди стартиране се уверете, че данните съвпадат с вашата среда:

```
{
  "db": {
    "host": "127.0.0.1",
    "port": 3306,
    "name": "db_dashboard",
    "user": "root",
    "pass": "changeme",
    "charset": "utf8mb4"
  }
}
```

5.3. Инсталация (Вариант A: Docker + Local PHP)

Това е препоръчителният метод за разработка, който гарантира чиста инсталация на базата данни.

Стъпка 1: Стартиране на базата данни

В коренната директория на проекта изпълнете командата:

```
docker-compose up -d
```

Това ще стартира MySQL 8.0 контейнер на порт 3306 с парола changeme.

Стъпка 2: Инициализиране на структурата (Seeding)

Импортирайте началната схема и данни чрез команда(и чрез въвеждане на паролата changeme след нея):

```
mysql -h 127.0.0.1 -u root -p < database/init.sql
```

Стъпка 3: Стартиране на уеб сървъра

Използвайте вградения в PHP сървър за стартиране на приложението:

```
php -S localhost:8080 -t public
```

5.4. Инсталация (Вариант Б: XAMPP / Apache)

Ако използвате XAMPP, WAMP или MAMP:

Стъпка 1: Стартирайте Apache и MySQL модулите от контролния панел.

Стъпка 2: Отворете phpMyAdmin и създайте нова база данни с име db_dashboard.

Стъпка 3: Импортирайте файла database/init.sql в създадената база.

Стъпка 4: Копирайте папката на проекта в папката htdocs.

Стъпка 5: Настройте config.json според вашите XAMPP настройки.

Стъпка 6: Добавете този код в края на C:\xamppi\apache\conf\extra\httpd-vhosts.conf

```
<VirtualHost *:80>
    DocumentRoot "C:/xampp/htdocs/db-dashboard/public"
    ServerName localhost
    <Directory "C:/xampp/htdocs/db-dashboard/public">
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

(като може да се наложи да редактирате низовете, така че да съвпадат с вашата инсталационна папка за xampp).

5.5. Достъп и Потребителски акаунти

След успешно стартиране, отворете браузъра на адрес:

URL: http://localhost:80/index.html

Системата разполага с предварително създадени потребители за тестване (дефинирани в init.sql):

Роля	Email	Парола	Описание
Admin	admin@example.com	password	Пълен достъп до всички настройки и проекти.
User	demo@example.com	password	Ограничен достъп само до собствени проекти.

5.6. Работен процес

Стандартният алгоритъм за работа със системата след инсталация е следният:

1. Вход в системата с администраторския акаунт.
2. Дефиниране на ресурси: Добавяне на сървър (Servers) и шаблони (Templates) за конфигурация.
3. Импорт на проекти: Използване на функцията за масов импорт (Bulk Import) или ръчно създаване на нов проект.
4. Генериране на действия: Стартиране на автоматизирани задачи (създаване на БД, потребители, бекъпи) за избраните проекти.

6. Кратко ръководство на потребителя

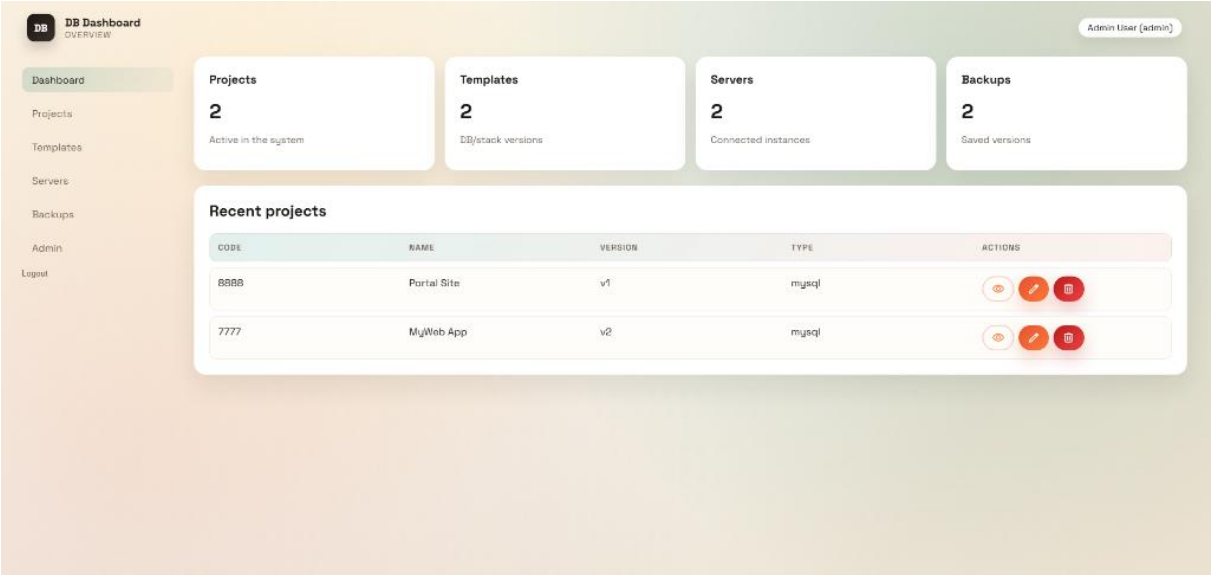
Този раздел описва основните стъпки за работа със системата DB Dashboard. Интерфейсът е проектиран да бъде интуитивен, като навигацията се извършва чрез страничното меню вляво.

6.1. Начален екран (Dashboard)

След успешен вход в системата, потребителят бива пренасочен към началния екран ("Overview"). Тук се визуализира обобщена информация за състоянието на системата чрез информационни карти:

- Projects: Брой активни проекти.
- Templates: Налични конфигурационни шаблони.
- Servers: Брой свързани сървъри за бази данни.
- Backups: Брой създадени архиви.

В долната част на екрана се намира секцията "Recent projects", която предоставя бърз достъп до последните добавени или редактирани проекти.



Фиг. 1. Начален екран с обобщени метрики.

6.2. Управление на проекти (Projects)

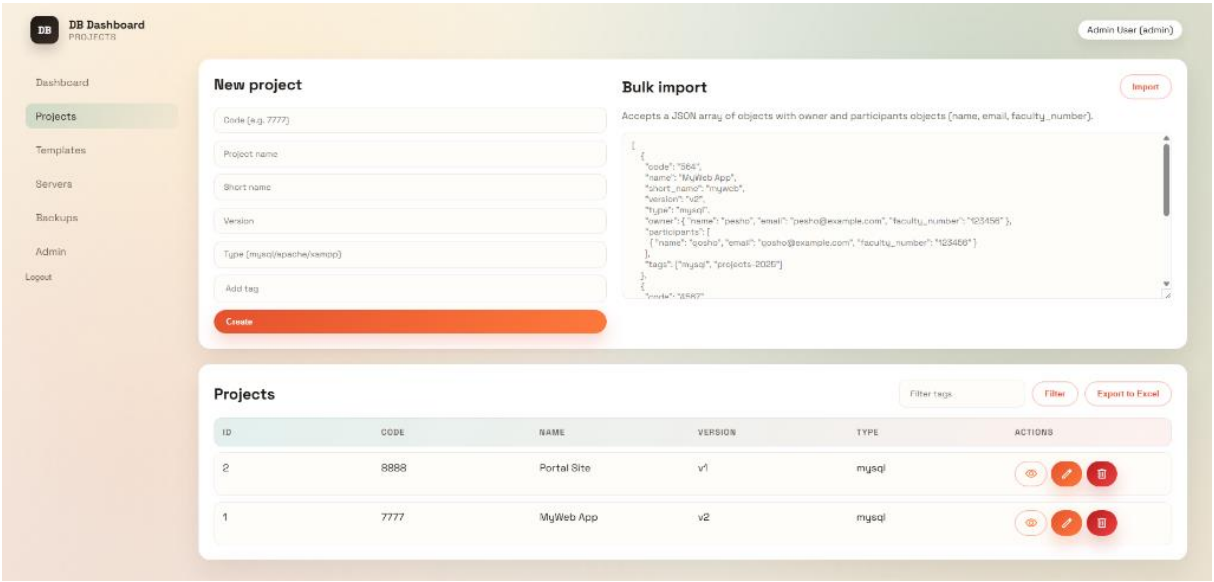
Секция Projects е основният работен модул. Тук потребителите могат да създават нови проекти, да разглеждат списъка с текущи и да извършват масов импорт.

Създаване на нов проект:

Формата изисква въвеждане на уникален код (Code), име на проекта, версия и тип на базата данни. Полетата за собственик и участници се попълват автоматично или се избират от списък (за администратори).

Масов импорт (Bulk Import):

В дясната част на екрана е разположен панелът за масов импорт. Той приема JSON масив, съдържащ дефиниции за множество проекти едновременно. Системата автоматично парсира данните и създава съответните записи и потребители, ако те не съществуват.



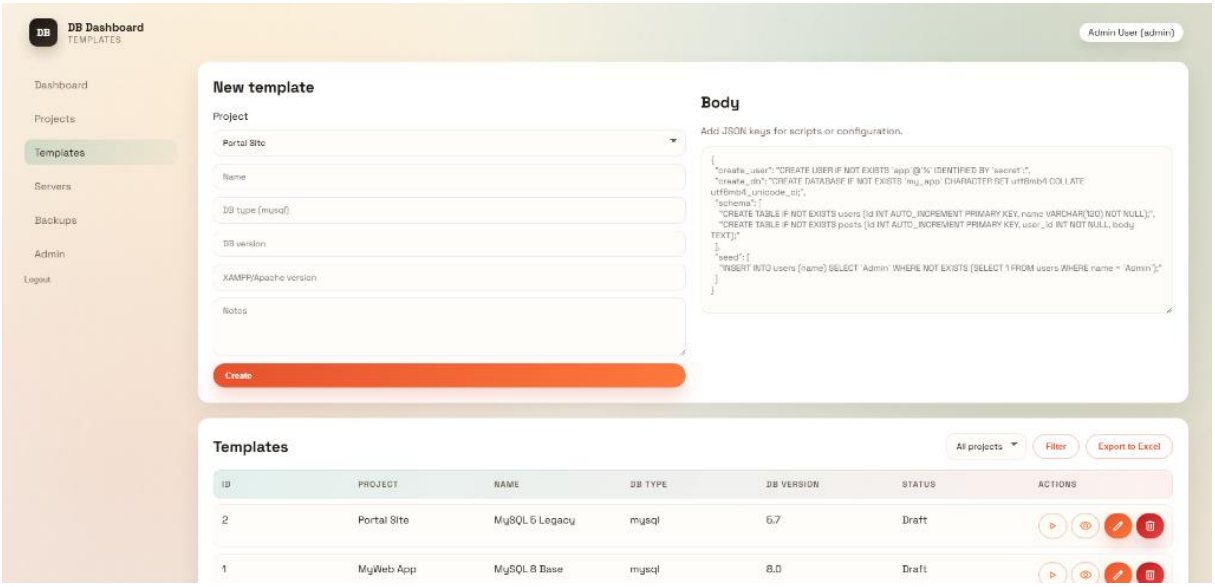
Фиг. 2. Екран за създаване на проекти и панел за JSON импорт.

6.3. Шаблони за конфигурация (Templates)

Шаблоните дефинират "рецептата", по която се създава средата за даден проект. Те позволяват стандартизация на настройките (напр. MySQL 8 Base или Legacy MySQL 5).

Основни данни: Избор на проект, към който се прикача шаблонът, и версия на DB/Stack.

Body (JSON): Текстово поле за въвеждане на специфични скриптове под формата на JSON. Тук се описват SQL командите за създаване на потребители (create_user), бази данни (create_db) и начални данни (seed).



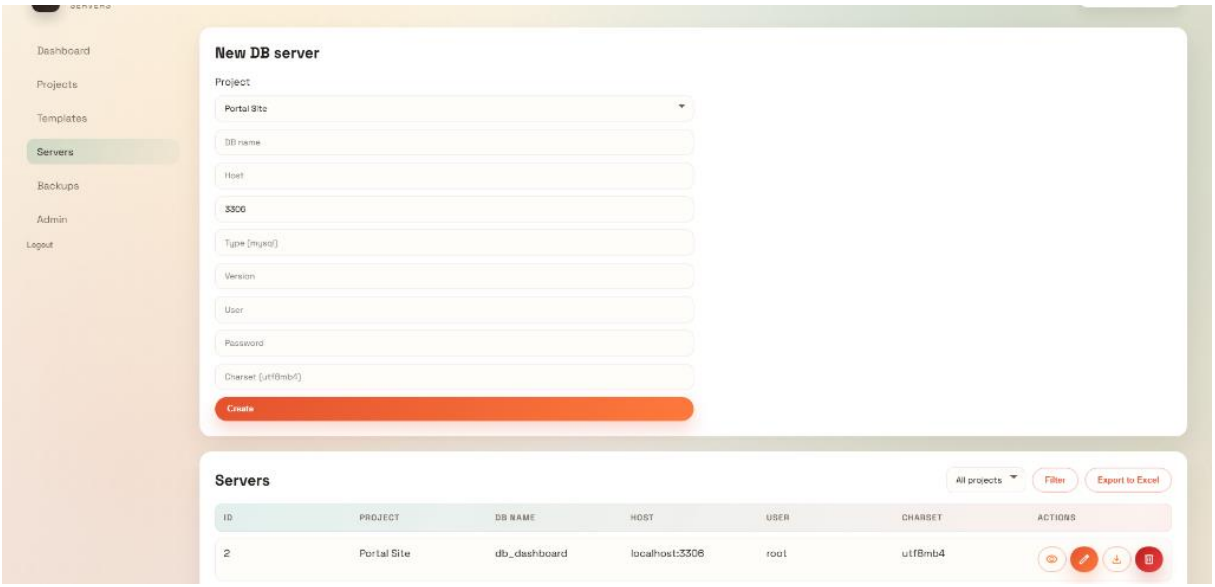
Фиг. 3. Редактор на шаблони с JSON конфигурация.

6.4. Сървъри (Servers)

В тази секция се дефинират връзките към физическите или виртуални сървъри за бази данни.

При добавяне на нов сървър ("New DB server") се въвеждат:

- Host & Port: Адресът на сървъра (напр. localhost:3306).
- User & Password: Данни за административен достъп (root), необходими за създаването на нови бази.
- Charset: Кодировка по подразбиране (напр. utf8mb4).



Фиг. 4. Конфигурация на връзка с MySQL сървър.

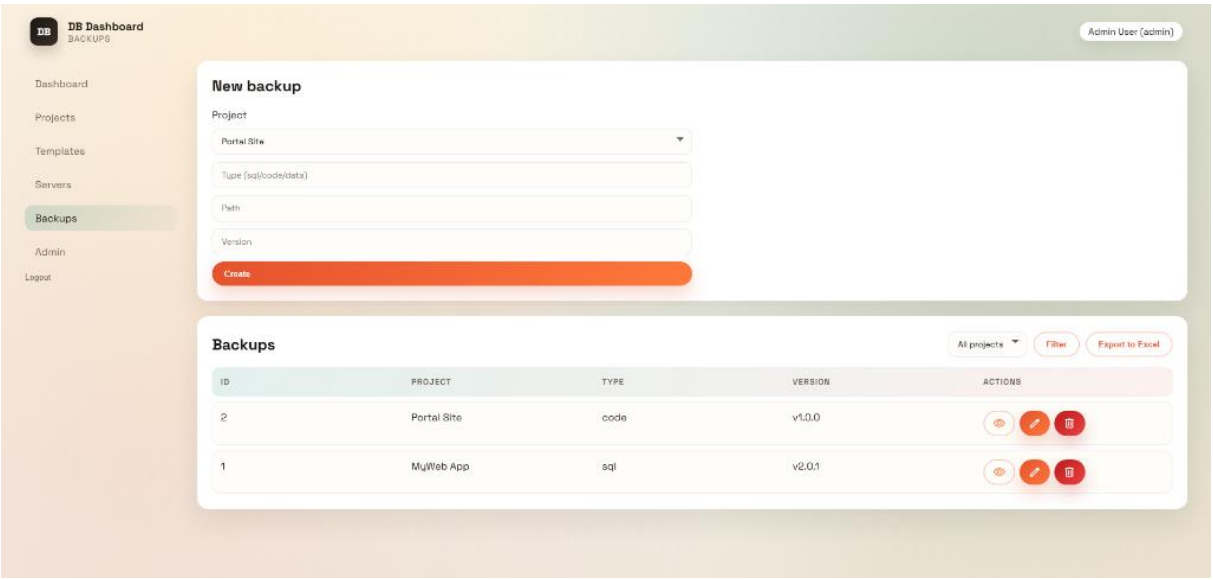
6.5. Архивиране (Backups)

Модулът позволява генериране на ръчни архиви на проектите. Потребителят избира проект и тип на архива:

SQL: Дъмп на базата данни (структура и данни).

Code: Архив на файловата система (ако е приложимо).

Таблицата по-долу показва история на всички генерирани архиви с възможност за преглед и изтриване.



Фиг. 5. Интерфейс за създаване и управление на архиви.

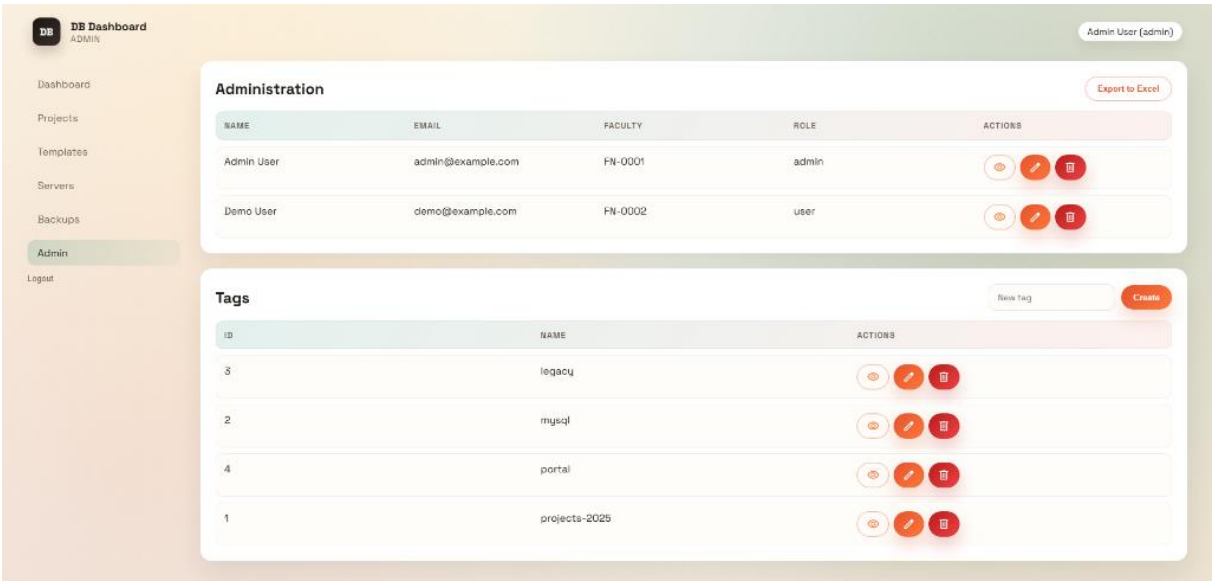
6.6. Администрация (Admin)

Тази секция е достъпна само за потребители с роля admin. Тя предоставя инструменти за:

Управление на потребители: Преглед на всички регистрирани потребители, техните имейли, факултетни номера и роли.

Управление на тагове (Tags): Създаване и редактиране на тагове за категоризация на проектите (напр. legacy, projects-2025).

Експорт: Възможност за изтегляне на пълна справка за потребителите в Excel формат.



Фиг. 6. Административен панел за управление на потребители и номенклатури.

7. Примерни данни

За целите на демонстрацията и първоначалното тестване на системата са подготвени набор от тестови данни, включващи потребителски акаунти, конфигурационни файлове и примерни JSON структури за импорт на проекти.

7.1. Потребителски акаунти (Credentials)

Системата разполага с два предварително дефинирани потребителя с различни нива на достъп. Данните се зареждат автоматично при изпълнение на скрипта database/init.sql.

Роля	Email	Парола	Факултетен Номер	Описание
Admin	admin@example.com	password	FN-0001	Пълен достъп до всички настройки и проекти.
User	demo@example.com	password	FN-0002	Ограничен достъп само до собствени проекти.

Забележка: Паролите в базата данни се съхраняват като хеш (bcrypt). За тестови цели, началната парола и за двата акаунта е password.

7.2. Конфигурация на базата данни

За връзка между PHP приложението и MySQL сървъра се използва конфигурационен файл storage/config.json. Примерно съдържание за локална Docker среда:

```
{
    "db_host": "127.0.0.1",
    "db_port": 3306,
    "db_name": "db_dashboard",
    "db_user": "root",
    "db_pass": "changeme",
    "db_charset": "utf8mb4"
}
```

7.3. Тестови данни за импорт

Системата поддържа масов импорт на проекти чрез JSON формат. По-долу е представен валиден пример, който може да бъде използван в секция Projects -> Bulk Import:

```
[
  {
    "code": "WEB-2025",
    "name": "E-Commerce Project",
    "short_name": "shop_v1",
    "version": "1.0",
    "type": "mysql",
    "owner": {
      "name": "Ivan Petrov",
      "email": "ivan@example.com",
      "faculty_number": "123456"
    },
    "participants": [
      {
        "name": "Maria Georgieva",
        "email": "maria@example.com",
        "faculty_number": "654321"
      }
    ],
    "tags": ["ecommerce", "php-course"]
  }
]

```

При този импорт системата автоматично ще създаде потребителите Ivan Petrov и Maria Georgieva, ако не съществуват, и ще им генерира служебни пароли.

7.4. Примерна конфигурация на шаблон

В секция Templates, полето Body приема JSON обект, описващ SQL командите за създаване на средата. Пример за стандартен шаблон:

```
{
    "create_user": "CREATE USER IF NOT EXISTS 'app_user'@'%' IDENTIFIED BY 'secret_pass';",
    "create_db": "CREATE DATABASE IF NOT EXISTS `app_db_v1` CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;",
    "schema": [
        "CREATE TABLE IF NOT EXISTS users (id INT AUTO_INCREMENT PRIMARY KEY, username VARCHAR(50));",
        "CREATE TABLE IF NOT EXISTS logs (id INT AUTO_INCREMENT PRIMARY KEY, message TEXT);"
    ],
    "seed": [
        "INSERT INTO users (username) VALUES ('admin');"
    ]
}

```

7.5. Разположение на тестовите скриптове

Основните файлове, необходими за инициализация и тестване, се намират в следните директории:

- database/init.sql: SQL скрипт за създаване на схемата на базата данни и попълване на началните (seed) данни.
- public/api.php: Входна точка за API заявките (може да се тества с Postman/cURL).
- docker-compose.yml: Дефиниция на тестовата инфраструктура (MySQL контейнер).

8. Описание на програмния код

Програмната реализация на системата следва стриктна Model-View-Controller (MVC) архитектура. Приложението е разделено на два основни слоя:

Backend (API): Реализиран на чист PHP, обработващ заявки и връзка с базата данни.

Frontend (UI): Реализиран с JavaScript модули, комуникиращи с бекенда чрез JSON.

8.1. Структура на файловата система

Проектът е организиран в следните основни директории:

- src/: Съдържа основната логика (Backend).
- Auth/: Логика за автентикация и сесии (AuthService.php).
- Controllers/: Класове, обработващи входните данни (ProjectController.php, ServerController.php и др.).
- Db/: Управление на базата данни (Connection.php).
- Http/: Системни класове за рутиране и HTTP отговори (Router.php, Response.php).
- public/: Публично достъпна директория.
- api.php: Входна точка (Entry Point) за всички сървърни заявки.
- *.js: Клиентски скриптове, разделени по модули (projects.js, dashboard.js).
- storage/: Съдържа конфигурационни файлове (config.json) и логове.

8.2. Сървърно ядро и Рутиране

Вместо използване на готова библиотека, е реализиран собствен „рутер“. Всички заявки минават през public/api.php, който инициализира приложението и предава управлението на рутера.

Този код демонстрира как системата прихваща HTTP метода и пътя, за да извика съответния контролер.

```php
// public/api.php
require __DIR__ . '/../src/bootstrap.php'; // Зареждане на автолоудър и настройки

use App\Http\Router;

// Дефиниране на маршрутите (Routes)
// Формат: [HTTP Метод, URL Път, Контролер, Метод на контролера]
Router::add('GET', '/projects', 'ProjectController', 'list');
Router::add('POST', '/projects', 'ProjectController', 'create');
Router::add('POST', '/login', 'AuthController', 'login');

// Обработка на текущата заявка
Router::handle();

```

Фрагмент 1: Инициализация и Рутиране (api.php & Router.php)

8.3. Работа с База данни (Singleton Pattern)

За връзката с MySQL се използва класът Connection, който имплементира шаблона Singleton. Това гарантира, че по време на изпълнение на скрипта ще има само една активна връзка към базата данни, което оптимизира ресурсите.

```php
namespace App\Db;
use PDO;

class Connection {
    private static ?PDO $instance = null;

    public static function get(): PDO {
        // Ако инстанцията вече съществува, я връщаме директно
        if (self::$instance === null) {
            // Зареждане на конфигурацията от JSON файл
            $config = json_decode(file_get_contents(__DIR__ . '/../../storage/config.json'), true);

            // Създаване на нова PDO връзка с настройки за грешки и кодировка
            $dsn = "mysql:host={$config['db_host']};dbname={$config['db_name']};charset=utf8mb4";
            self::$instance = new PDO($dsn, $config['db_user'], $config['db_pass'], [
                PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION, // Хвърляне на изключения при SQL грешки
                PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC // Връщане на резултатите като асоциативен масив
            ]);
        }
        return self::$instance;
    }
}

```

Фрагмент 2: Връзка с базата данни (src/Db/Connection.php)

8.4. Логика и Сигурност

Основната логика е концентрирана в контролерите. Те проверяват правата на потребителя чрез AuthService и изпълняват заявки към базата данни.

Този фрагмент показва как администраторите виждат всички проекти, докато обикновените потребители виждат само тези, в които участват. Използват се Prepared Statements за защита от SQL Injection.

```php
public function list(): void
{
    $auth = new AuthService();
    // 1. Проверка дали потребителят е логнат
    $current = $auth->requireUser();
    $pdo = Connection::get();

    // 2. Логика според ролята на потребителя
    if ($current['role'] === 'admin') {
        // Администраторът вижда всичко
        $sql = 'SELECT p.*, u.name AS owner_name
                FROM projects p
                JOIN users u ON u.id = p.owner_id
                ORDER BY p.id DESC';
        $stmt = $pdo->query($sql);
    } else {
        // Обикновеният потребител вижда само своите проекти или тези, в които е участник
        // Използва се параметризирана заявка (:user_id) за сигурност
        $sql = 'SELECT DISTINCT p.*, u.name AS owner_name
                FROM projects p
                JOIN users u ON u.id = p.owner_id
                LEFT JOIN project_participants pp ON pp.project_id = p.id
                WHERE p.owner_id = :user_id OR pp.user_id = :user_id
                ORDER BY p.id DESC';

        $stmt = $pdo->prepare($sql);
        $stmt->execute([':user_id' => (int) $current['id']]);
    }

    // 3. Връщане на резултата като JSON
    Response::json($stmt->fetchAll());
}

```

Фрагмент 3: Разграничаване на правата за достъп (src/Controllers/ProjectController.php)

8.5. Интерфейс

Клиентската част е изградена модулно. Всеки екран има свой JavaScript файл, който се грижи за зареждането на данните и обновяването на DOM дървото.

Пример за извличане на данните от API-то и динамично генериране на таблица.

```
async function loadProjects() {
    try {
        // Изпращане на GET заявка към бекенда
        const response = await fetch('/api.php/projects');

        if (!response.ok) throw new Error('Failed to load projects');

        const projects = await response.json();

        // Рендиране на таблицата
        projectsTable.innerHTML = projects.map(p => `
            <tr>
                <td>${p.code}</td>
                <td>${p.name}</td>
                <td>${p.version}</td>
                <td>
                    <button onclick="viewProject(${p.id})">View</button>
                </td>
            </tr>
        `).join('');
    } catch (error) {
        console.error('Error:', error);
    }
}
```

Фрагмент 4: Асинхронна комуникация (public/projects.js)

9. Приноси на студента, ограничения и възможности за бъдещо разширение

Проектът е разработен от екип от двама студенти, като функционалностите са разделени на логически модули, за да се гарантира независимост при разработката и лесна интеграция (Merge) на кода.

9.1. Разпределение на задачите

Стоян Красимиров Николов:

- Управление на сървъри: Добавяне на DB сървъри с root права и проверка на връзката.
- Шаблони (Templates): Създаване на типови конфигурации за приложения (групово с филтър) и скриптове за инсталация.
- Изпълнение на скриптове: Логика за стартиране на SQL/Shell скриптове върху една или повече бази.
- Архивиране (Backups): Логика за дъмп на бази данни и файлове, управление на версиите на архивите и възстановяване.
- DevOps: Настройка на docker-compose и init.sql.

Йоан Венелинов Йорданов:

- Автентикация: Регистрация, вход, сесии и защита на рутовете (AuthService).
- Административен панел: Управление на потребители, роли и права.
- Проекти & Тагове: CRUD операции за проекти, филтрация по тагове и търсене.
- Импорт/Експорт: Реализация на Bulk Import (JSON) и Експорт към Excel.
- Frontend: UI дизайн, JavaScript логика за SPA поведение и AJAX заявки.

9.2. Ограничения на текущата версия

- Системата поддържа основно MySQL бази данни. Поддръжка за PostgreSQL или MongoDB изисква разширяване на класа Connection.
- Архивирането на файлове (Code Backup) работи само локално, тъй като PHP скриптът няма достъп до отдалечени файлови системи без SSH/FTP модул.

9.3. Възможности за бъдещо разширение

- SSH Интеграция: Добавяне на модул за отдалечено изпълнение на команди върху сървърите.
- Scheduler: Автоматично създаване на бекъпи по график (Cron jobs).
- CI/CD Hooks: Възможност за автоматичен деплой при push в Git хранилище.

10. Използване на AI – как и защо

По време на разработката са използвани AI инструменти (ChatGPT/Gemini) като помощни средства за оптимизация на процеса.

- Генериране на тестови данни: Създаване на JSON структури за Bulk Import и SQL заявки за попълване на базата с dummy users.
- Frontend стилизиране: Генериране на CSS цветови палитри и Flexbox/Grid структури за респонсив дизайн.
- Документация: Структуриране и форматиране на техническата документация.

11. Какво научих (най-важните неща, които сте научили по време на курса и при разработване на проекта – за всеки студент)

Работата по проекта беше разпределена така, че всеки от екипа да навлезе дълбоко в специфична област на уеб разработката и системната администрация.

11.1. Стоян Красимиров Николов (Инфраструктура и DB Управление)

Фокусът върху "backend-heavy" задачите и управлението на сървъри ми позволи да усвоя следните ключови умения:

- Системно взаимодействие чрез PHP: Научих се как да използвам PHP не само за генериране на HTML, а като инструмент за управление на процеси – изпълнение на системни команди (за mysqldump), работа с файловата система (създаване на архиви) и парсване на конфигурационни файлове.
- Advanced PDO & SQL: Разбрах тънкостите при управлението на динамични връзки към бази данни (Dynamic Connections) – как да превключвам контекста между различни сървъри по време на една заявка, без да нарушавам Singleton шаблона.
- Автоматизация и Шаблони: Придобих опит в създаването на гъвкави структури (JSON шаблони), които автоматизират рутинни задачи като създаване на потребители и права, което е основата на DevOps практиките.
- Docker Networking: Задълбочих знанията си за това как контейнерите комуникират помежду си и как правилно да се конфигурират портове и мрежи, за да може приложението да достъпва външни или локални бази данни.

11.2. Йоан Венелинов Йорданов (Потребителски интерфейс и Бизнес логика)

Разработката на клиентската част и административния панел ми даде практически опит в изграждането на цялостни приложения:

- SPA Архитектура (Single Page Application): Научих как да организирам Frontend кода модулно, използвайки ES6 Modules и fetch API за асинхронна комуникация, избягвайки презареждането на страницата – нещо, което модерните frameworks правят автоматично, но тук реализирахме "от нулата".
- Автентикация и Сигурност: Разбрах в дълбочина механизма на сесиите (Session Management), защитата на маршрути (Route Guarding) и правилното съхранение на пароли (Hashing), както и разграничаването на права между admin и user.
- Обработка на данни (Data Mapping): Сблъсках се с предизвикателството да трансформирам данни от различни формати – парсване на големи JSON масиви при Bulk Import и генериране на файлове за експорт, както и валидация на входните данни преди те да стигнат до базата.
- Релационни връзки: Усвоих писането на оптимизирани SQL заявки с множество JOIN клаузи за свързване на потребители, проекти и тагове в една справка.

12. Dev(sec)Ops – подкарване на проекта – особености

12.1. Хранилище на кода

Проектът е качен в GitLab на ФМИ:

URL:_____

12.2. Линк към разгърнатия проект

Демо версия на приложението е достъпна в мрежата на ФМИ:

URL:_____

12.3. Инструкции за инсталация (Deployment)

За успешното подкарване на сървъра на ФМИ:

1. Клониране: git clone.
2. База данни: Импортирайте файла database/init.sql (различен от локалния, съдържа само структура без тестови данни).
3. Настройки: Създайте файл storage/config.json със следните параметри за продукционната среда:

```
1. {
2.     "db_host": "mysql_container",
3.     "db_user": "root",
4.     "db_pass": "secure_pass_fmi",
5.     "db_name": "db_dashboard_prod"
6. }
```

13. Използвани източници

[1] The PHP Group, "PHP Manual - Data Objects (PDO)", Online Documentation, Публикувано на: php.net, последно посетен на: 2026-02-05.

[2] Oracle Corporation, "MySQL 8.0 Reference Manual - The JSON Data Type", Technical Documentation, Публикувано на: dev.mysql.com, последно посетен на: 2026-02-06.

[3] Mozilla Developer Network (MDN), "Fetch API - Using Fetch", Web Documentation, Публикувано на: developer.mozilla.org, последно посетен на: 2026-01-25.

[4] Docker Inc., "Docker Compose Overview", Technical Documentation, Публикувано на: docs.docker.com, последно посетен на: 2026-01-28.

СОФИЙСКИ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“

ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА

Предал 1 (подпис): …….

/5MI0800321, Йоан Йорданов, КН, гр.1/

Предал 2 (подпис): …….

/5MI0800347, Стоян Николов, КН, гр.1/

Приел (подпис): …….

/проф. д-р Милен Петров/