Молдавский Государственный Университет Факультет Математики и Информатики Департамент Информатики

## Индивидуальная работа: «Система управления тикетами»

Выполнил: Мамалига Артур, группа I2302  
Григорий Зайка, группа I2302  
Преподаватель: Никита Нартя  
Дата: 06.05.2025

## Кишинев 2025

**Оглавление**

1. Теоретическая часть  
   1.1 Формулировка задачи  
   1.2 Цель и основные этапы работы
2. Практическая часть  
   2.1 Краткое описание особенностей реализации приложения  
   2.2 Содержание и функциональные возможности  
   2.3 Сценарии взаимодействия пользователей  
   2.4 Структура базы данных
3. Выводы и ссылка на репозиторий
4. Ответы на контрольные вопросы
5. Список использованных источников
6. Технические требования

# 1. Теоретическая часть

## 1.1 Формулировка задачи

Разработать веб‑приложение средней сложности для создания и управления тикетами. Приложение должно поддерживать регистрацию и вход пользователей с ролями, создание, редактирование, удаление тикетов, изменение их статуса, поиск и фильтрацию, а также безопасное хранение паролей.

## 1.2 Цель и основные этапы работы

**Цель работы:** изучить и применить методы построения приложения с модульной архитектурой на чистом PHP, реализовать веб‑сервис с аутентификацией, механизмом управления данными и безопасностью.

### Этапы работы:

1. Анализ требований и проектирование архитектуры.
2. Настройка окружения Docker и базы MySQL.
3. Разработка каркаса (Router, Controller, Views).
4. Реализация аутентификации и ролей.
5. CRUD‑операции для тикетов.
6. Поиск и фильтрация.
7. Тестирование безопасности.

# 2. Практическая часть

## 2.1 Краткое описание особенностей реализации приложения

Приложение разработано на чистом PHP с использованием паттерна Front Controller и собственной реализацией автозагрузчика по PSR‑4–подобному принципу. Основные технические особенности:

**Front Controller**: вся маршрутизация запросов проходит через единый файл public/index.php, где инициализируется автозагрузка, подключается конфиг, обрабатываются сессии и регистрируются маршруты (Router::get(), Router::post()).

**Роутер** (app/Core/Router.php): простой класс, сопоставляющий URI и HTTP-метод с контроллером и методом, поддерживает 404-обработчик, позволяет легко расширять список маршрутов.

**Контроллеры** (app/Controllers): каждый контроллер отвечает за конкретную группу логики (аутентификация, управление тикетами, админ-функции). Базовый класс Controller реализует метод view(), который подгружает шаблоны с header/footer.

**Автозагрузка**: вместо Composer используется spl\_autoload\_register(), который на основании неймспейса App\ мапит вызовы классов на файлы в каталоге app/.

**Шаблонизация**: представления — это простые PHP-файлы в app/Views, разбитые на layout/header.php и layout/footer.php для общих элементов и на подпапки для отдельных страниц (auth, tickets, admin). Данные передаются через extract() в контроллере.

**Docker‑Compose**: в docker-compose.yml описаны сервисы db (MySQL 8.0) и app (Apache+PHP 8.0). Контейнер базы при старте подхватывает миграции из migrations/, автоматизируя создание схемы. Контейнер PHP устанавливает расширения PDO и настраивает корневую папку public/.

**Миграции**: SQL-скрипты в папке migrations/ создают таблицы users и tickets, а также дефолтного админа. При каждом docker-compose up они применяются автоматически.

**Безопасность**:

* SQL-запросы через PDO с подготовленными выражениями (prepare/execute).
* Хеширование паролей password\_hash(..., PASSWORD\_BCRYPT) и проверка password\_verify().
* Санитизация вывода htmlspecialchars(), чтобы избежать XSS.
* CSRF-токены в формах (скрытый <input> и проверка в контроллерах).
* Контроль доступа через $\_SESSION['user'] и флаг is\_admin.

**Поиск и фильтрация**: выполнены динамические SQL-конструкции в методе TicketController::index(), где к базовым условиям по статусам добавляются LIKE и точные сравнения по GET-параметрам.

## 2.2 Функциональные возможности

### Регистрация и вход

1. Форма регистрации с полями «Email», «Пароль», «Подтверждение пароля».
2. Хеширование пароля через password\_hash(PASSWORD\_BCRYPT).
3. Форма входа с проверкой через password\_verify().
4. Флеш-сообщения об ошибках и успехе.

### Роли и разграничение доступа

1. Три роли: Гость, Пользователь, Администратор.
2. Гостевой доступ: только чтение открытых тикетов.
3. Пользователь: создание тикетов, просмотр своих и закрытых.
4. Администратор: полный CRUD над любыми тикетами, управление пользователями (создание админов).

### Управление тикетами

1. Создание тикета: минимум 6 полей (заголовок, описание, категория, приоритет, дата, флаг срочности).
2. При создании статус = Pending.
3. Редактирование: изменение любых полей и статуса (Pending-Open-Closed) только администратором.
4. Удаление: только админ может безвозвратно удалить тикет.
5. Просмотр списка: фильтрация по статусам в зависимости от роли; отображение даты создания.

### Поиск и фильтрация

1. Поиск по заголовку (LIKE '%…%').
2. Фильтрация по категории и приоритету (точное совпадение).
3. GET-параметры q, category, priority и динамическая подстановка в SQL.

### Безопасность и валидация

1. Client-side: HTML5-валидация (required, maxlength, type="date") + JavaScript-проверки.
2. Server-side: проверка всех полей в контроллере, сбор массива ошибок, флеш-сообщения.
3. CSRF-защита: токен в сессии + скрытое поле.
4. XSS-защита: htmlspecialchars() для всякого вывода.
5. SQL-инъекции: PDO + prepare()/execute().

### Контейнеризация и автоматизация

1. Docker-Compose: два сервиса (db и app).
2. MySQL-миграции в папке migrations/ автоматически применяются при старте базы.
3. Apache + PHP настроены на public/ как DocumentRoot, mod\_rewrite для ЧПУ.

## 2.3 Сценарии взаимодействия пользователей

### 2.3.1 Сценарий «Гость» (неавторизованный пользователь)

1. **Открытие главной страницы.**
   * URL: http://<host\_address>:8080/
   * В шапке видны ссылки: «Main», «Login », «Sign Up », «Ticket List».
   * Ни «Create Ticket», ни «Logout» не доступны.
2. **Просмотр списка тикетов.**
   * Пользователь нажимает « Ticket List».
   * Переход на http://<хост>:8080/tickets.
   * Отображаются только тикеты со статусом Open.
   * В каждом элементе списка видно:
     + Номер и заголовок ([#12] Web Server Failure),
     + Категория (Server),
     + Дата создания,
     + **нет** метки статуса, так как для гостей статус не выводится.
3. **Поиск и фильтрация.**
   * Над списком находится форма с полем «Поиск по заголовку» и двумя выпадающими списками «Categories » и «Priority».
   * Пользователь вводит фразу (например, Server) и нажимает «Искать».
   * Браузер делает GET-запрос на ...?q=Server.
   * Список обновляется, показывая только подходящие тикеты.
4. **Попытка создать тикет.**
   * Пользователь кликает «Создать тикет» (ссылка не отображается) или вручную заходит на /ticket/create.
   * Система редиректит на /login с кодом 302, так как создание доступно только при авторизации.
5. **Попытка просмотра закрытых или pending-тикетов.**
   * Если гость по прямой ссылке попытается зайти на /ticket?id=5 где status=Closed или Pending, — система перенаправит его обратно на /tickets.

### 2.3.2 Сценарий «Зарегистрированный пользователь»

1. **Регистрация нового аккаунта.**
   * URL: http://<host\_address>:8080/register.
   * Форма содержит поля:
     + Email (текстовое поле, required, тип email),
     + Пароль (парольное поле, required, minlength=6),
     + Повтор пароля (парольное поле).
   * При вводе некорректных данных (пароли не совпали, слишком короткий пароль) форма не отправляется или возвращается с ошибкой из $\_SESSION['errors'].
   * При успехе пользователь видит флеш-сообщение «Registration successful. Please log in.» и попадает на /login.
2. **Вход в систему.**
   * URL: http://< host\_address >:8080/login.
   * Поля: Email, Пароль.
   * После успешного входа устанавливается $\_SESSION['user'] и $\_SESSION['success'] = 'You logged in as …'.
   * Редирект на главную страницу /.
   * Шапка теперь показывает:   
     «Main, user@example.com | Create Ticket | Ticket List | Logout».
3. **Создание тикета.**
   * URL: /ticket/create.
   * Форма с 6 полями:
     + Заголовок (<input type="text" required maxlength="255">),
     + Описание (<textarea required>),
     + Категория (<select>),
     + Приоритет (<input type="radio"> группы),
     + Срок выполнения (<input type="date">),
     + Флаг срочности (<input type="checkbox">).
   * Клиент-сайд валидация: JS-проверка непустоты, длины, даты не в прошлом.
   * Серверная валидация в методе create(): сбор массива ошибок, редирект обратно при наличии $\_SESSION['errors'].
   * При успехе в БД вставляется запись с status = 'Pending', флеш «Ticket created, pending approval.», редирект на /tickets.
4. **Просмотр собственных тикетов.**
   * Переход на /tickets:
     + Список включает тикеты со статусами Open и Closed (но не Pending).
     + Предусмотрена сортировка по дате created\_at DESC.
5. **Просмотр деталей тикета.**
   * URL: /ticket?id=XYZ.
   * Отображаются все поля тикета: заголовок, описание, категория, приоритет, срок, срочность, статус, дата создания.
   * Нет кнопок «Edit», «Delete» и формы смены статуса для обычного пользователя.
   * Ссылка «Back» ведёт на /tickets.
6. **Выход из системы.**
   * При клике на «Logout» вызывается метод logout(), очищаются сессии, редирект на /.
   * Шапка возвращается в «гостевой» вид.

### 2.3.3 Сценарий «Администратор»

1. **Вход под админом.**
   * Email: admin, Пароль: admin.
   * После логина шапка показывает:   
     «Welcome, admin | Create Ticket | Ticket List | Create New Admin | Logout».
2. **Просмотр всех тикетов.**
   * /tickets показывает тикеты всех трёх статусов (Pending, Open, Closed).
   * Рядом с каждым заголовком отображается цветная метка:
     + Оранжевая — Pending,
     + Зелёная — Open,
     + Красная — Closed.
3. **Подробный просмотр и управление тикетом.**
   * /ticket?id=XYZ — на странице деталей выводятся кнопки:
     + **Edit** (ссылка на /ticket/edit?id=XYZ),
     + **Delete** (ссылка /ticket/delete?id=XYZ с confirm()),
     + **Change Status** — выпадающий <select> со статусами и кнопка «Change Status».
4. **Изменение статуса.**
   * Админ выбирает статус Open или Closed для тикета Pending и жмёт «Change Status».
   * Метод changeStatus() обновляет поле status, устанавливает флеш-сообщение и возвращает на /ticket?id=XYZ.
5. **Редактирование тикета.**
   * /ticket/edit?id=XYZ — форма, аналогичная созданию, но предзаполненная текущими значениями, включая выпадающий селект status.
   * После сабмита в методе edit() производится та же валидация, и при успехе данные обновляются.
6. **Удаление тикета.**
   * При клике на «Delete» открывается JS-подтверждение.
   * По подтверждению вызывается /ticket/delete?id=XYZ, выполняется DELETE FROM tickets WHERE id = XYZ, появляется флеш «Ticket deleted.» и редирект на /tickets.
7. **Создание нового администратора.**
   * URL: /admin/users/create.
   * Форма с полями Email, Пароль, Повтор пароля.
   * Валидация аналогична регистрации пользователя, но в БД в поле is\_admin выставляется 1.
   * В случае успеха флеш «Admin account created: email@example.com» и остаётся на той же форме.
8. **Выход администратора.**
   * Аналогично пользователю вызывается logout(), переход на /, шапка становится гостевой.

## 2.4 Структура базы данных

### Таблица users:

**id** INT PK,

**email** VARCHAR(255),

**password\_hash** VARCHAR(255),

**is\_admin** TINYINT,

**created\_at** TIMESTAMP

### Таблица tickets:

**id** INT PK,

**user\_id** INT (FK → users),

**title** VARCHAR(255),

**description** TEXT,

**category** VARCHAR(100),

**priority** VARCHAR(20),

**due\_date** DATE,

**is\_urgent** TINYINT,

**status** VARCHAR(20),

**created\_at** TIMESTAMP

# 3. Выводы и ссылка на репозиторий

В результате выполненной работы разработано функционирующее веб‑приложение для управления тикетами с учётными записями и распределением ролей. Приложение соответствует заданным требованиям.  
**Ссылка на репозиторий:** <https://github.com/slendchat/php_individual_work>

# 4. Ответы на контрольные вопросы

1. Как реализована аутентификация?

Через «Front Controller» и AuthController: формы /login и /register, при входе сверка чистого пароля с хэшем из БД (password\_verify), после успешной проверки в $\_SESSION['user'] сохраняются id, email и флаг is\_admin.

1. Какие хэш‑функции используются для паролей?

Встроенная PHP-функция password\_hash(..., PASSWORD\_BCRYPT) (алгоритм bcrypt), проверка — password\_verify().

1. Какие меры безопасности применены?

Все SQL — через PDO с подготовленными запросами (prepare/execute).

Вывод данных — через htmlspecialchars().

Валидация форм на клиенте (HTML5/JS) и на сервере.

CSRF-токены в скрытых полях форм.

Контроль доступа по $\_SESSION['user'] и флагу is\_admin.

1. Как работает поиск и фильтрация?

Метод TicketController::index() читает GET-параметры q, category, priority, динамически строит массив WHERE-условий (включая статус в зависимости от роли) и выполняет SELECT … WHERE … с LIKE для текста и точными сравнениями для категорий/приоритетов.

# 5. Список использованных источников

1. PHP Manual (<https://www.php.net/manual/>)
2. Docker Documentation (<https://docs.docker.com/>)
3. MDN Web Docs (<https://developer.mozilla.org/>)