Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное   
образовательное учреждение высшего образования   
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторным работам

Дисциплина: «Технологии сетевого программирования»

Тема: «Проектирование веб-приложения BusinessPlanner»

Выполнили:

Логанова Ю.В.

Поликарпова А.М.

Группа: 6304-010302D

Самара, 2025

Содержание

[Введение 3](#_Toc199621173)

[1 Описание структуры контейнеризации и настройки окружения; 4](#_Toc199621174)

[1.1 Docker-образы 4](#_Toc199621175)

[1.2 Docker Compose 4](#_Toc199621176)

[2 Описание API 7](#_Toc199621177)

[3 Архитектура приложения 9](#_Toc199621178)

[3.1 Схема взаимодействия компонентов 9](#_Toc199621179)

[3.2 Авторизация и аутентификация пользователей 10](#_Toc199621180)

[3.2.1 Аутентификация 10](#_Toc199621181)

[3.2.2 Авторизация 10](#_Toc199621182)

[3.3 Схема базы данных 10](#_Toc199621183)

[3.4 Безопасность 11](#_Toc199621184)

[4 Демонстрация работы контейнеров и приложения 12](#_Toc199621185)

[Приложение 15](#_Toc199621186)

Введение

Цель нашей работы – написать web-приложение BusinessPlanner, которое позволит эффективно отслеживать задачи и структурировать их.

Приложение реализовано на базе стека технологий:

* Backend: java 17, Spring Boot
* Frontend: Thymeleaf, HTML
* База данных: PostgreSQL
* Контейнеризация: Docker

Приложение разрабатывалось как учебный проект, с целью освоения и демонстрации навыков в области серверной разработки, проектирования структуры приложения, настройки безопасной аутентификации, работы с базой данных и т.д.

1. Описание структуры контейнеризации и настройки окружения;

Контейнеризация приложения BusinessPlanner реализована с использованием Docker и Docker Compose, что обеспечивает изолированное, воспроизводимое и масштабируемое развертывание системы.

* 1. Docker-образы

Образ PostgreSQL представлен файлом DockerfileDB. Он нужен для:

* Развертывания контейнера с СУБД PostgreSQL для постоянного хранения данных приложения между перезагрузками;
* Автоматической инициализации БД с предустановленными параметрами (пользователь, пароль, название БД).

Данные сохраняются на хост-машине через томы (volumes) Docker, что обеспечивает их устойчивость при перезапуске контейнера.

Образ Spring-приложения представлен файлом DockerfileAPP. Его назначение состоит в следующем:

* Развертывание backend-части приложения (Rest API и Thymeleaf);
* Подключение к PostgreSQL через переменные окружения.

Образ подвергается многоэтапной сборке, за счет чего оптимизируется размер образа:

* Этап builder: используется eclipse-temurin:17-jdk для компиляции кода (Maven, Java)
* Этап runtime: облегченный образ eclipse-temurin:17-jre для запуска JAR-файла.

Контейнеры объединены в одну виртуальную сеть с типом bridge. Это позволяет им безопасно взаимодействовать друг с другом внутри одной среды исполнения.

* 1. Docker Compose

Файл docker-compose.yml (рисунок 1) определяет взаимодействие сервисов и используется для управления контейнерами. Он описывает конфигурацию сервисов и их связи:

1. Сервис postgres
   1. Использует DockerfileDB для сборки образа.
   2. Переменные окружения (POSTGRES\_USER, POSTGRES\_PASSWORD, POSTGRES\_DB) задают параметры БД (environment).
   3. Подключен к внутренней сети businessplanner\_network для безопасного взаимодействия с приложением.
2. Сервис app
   1. Запускается только после готовности PostgreSQL (healthcheck проверка).
   2. Переменные окружения SPRING\_DATASOURCE\_URL, SPRING\_DATASOURCE\_USERNAME) настраивают подключение к БД.
   3. restart: unless-stopped – перезапуск для автоматического восстановления после сбоев
3. Общие настройки:
   * Сеть (businessplanner\_network) позволяет изолировать трафик между контейнерами и обращаться к БД по DNS-имени postgres.
   * Том postgres\_data сохраняет данные между перезапусками.

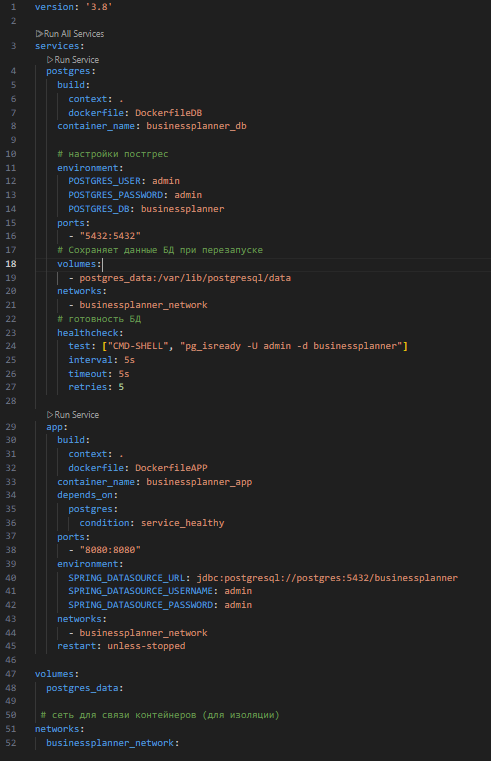


Рисунок 1 - docker-compose.yml

1. Описание API

В таблицах 1 – 5 представлены основные эндпоинты API.

Таблица 1 – Описание API AuthController

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | URL | Описание | Доступ |
| POST | /api/auth/login | Аутентификация пользователя | Все |

Таблица 2 – Описание API TagController

|  |  |  |  |
| --- | --- | --- | --- |
| POST | /api/tags | Создание нового тега | Аутентиф. пользователи |
| PUT | /api/tags/{id} | Обновление тега по ID | Аутентиф. пользователи |
| GET | /api/tags/{id} | Получение тега по ID | Аутентиф. пользователи |
| DELETE | /api/tags/{id} | Удаление тега по ID | ADMIN |
| GET | /api/tags | Получение всех тегов | ADMIN |

Таблица 3 – Описание API TaskController

|  |  |  |  |
| --- | --- | --- | --- |
| POST | /api/tasks | Создание новой задачи | Аутентиф. пользователи |
| GET | /api/tasks/{id} | Получение задачи по ID | Аутентиф. пользователи |
| GET | /api/tasks/creator/{creatorId} | Получение задач по ID создателя | ADMIN |
| PUT | /api/tasks{id} | Обновление задачи по ID | Аутентиф. пользователи |
| DELETE | /api/tasks{id} | Удаление задачи по ID | ADMIN |
| GET | /api/tasks | Получение всех задач | ADMIN |
| GET | /api/tasks/user/{email}/tag/{tagId} | Фильтрация задач по email и tag ID | ADMIN |
| DELETE | /api/tasks/user/{email}/tag/{tagName} | Удаление задач по email и названию тега | ADMIN |

Таблица 4 – Описание API TaskTagController

|  |  |  |  |
| --- | --- | --- | --- |
| POST | /api/task-tags | Создание связи задачи и тега | Аутентиф. пользователи |
| GET | /api/task-tags/{id} | Получение связи по ID | ADMIN |
| GET | /api/task-tags | Получение всех связей | ADMIN |
| PUT | /api/task-tags/{id} | Обновление связи по ID | Аутентиф. пользователи |
| DELETE | /api/task-tags/{id} | Удаление связи по ID | Аутентиф. пользователи |

Таблица 5 – Описание API UserController

|  |  |  |  |
| --- | --- | --- | --- |
| POST | /api/users | Регистрация нового пользователя | Все |
| GET | /api/users/{id} | Получение пользователя по ID | ADMIN |
| GET | /api/users/email/{email} | Получение пользователя по email | ADMIN |
| DELETE | /api/users/{id} | Удаление пользователя по ID | ADMIN |
| DELETE | /api/users/email/{email} | Удаление пользователя по email | ADMIN |
| GET | /api/users | Получение всех пользователей | ADMIN |

1. Архитектура приложения

Приложение разработано с использованием многослойной архитектуры, что обеспечивает четкое разделение ответственности между компонентами.

* 1. Схема взаимодействия компонентов

Пользователь отправляет запросы к API (например, вход в систему, создание задачи). Контроллер валидирует входные данные, вызывает соответствующий сервис. Сервис выполняет бизнес-логику, работает с репозиториями. Репозитории, в свою очередь, формируют SQL-запросы, работают с базой данных через Hibernate. Ответы возвращаются через слои в обратном порядке.

База данных (PostgreSQL) хранит пользователей, задачи, теги, статусы задач и обеспечивает связь между ними.

У пользователя есть возможность создания и удаления своих задач, а также их сортировки по тегам, приоритету и статусу (рисунок 2).

Представление ответа пользователю находится в виде шаблона Thymeleaf (например, allTasks.html, createTask.html).

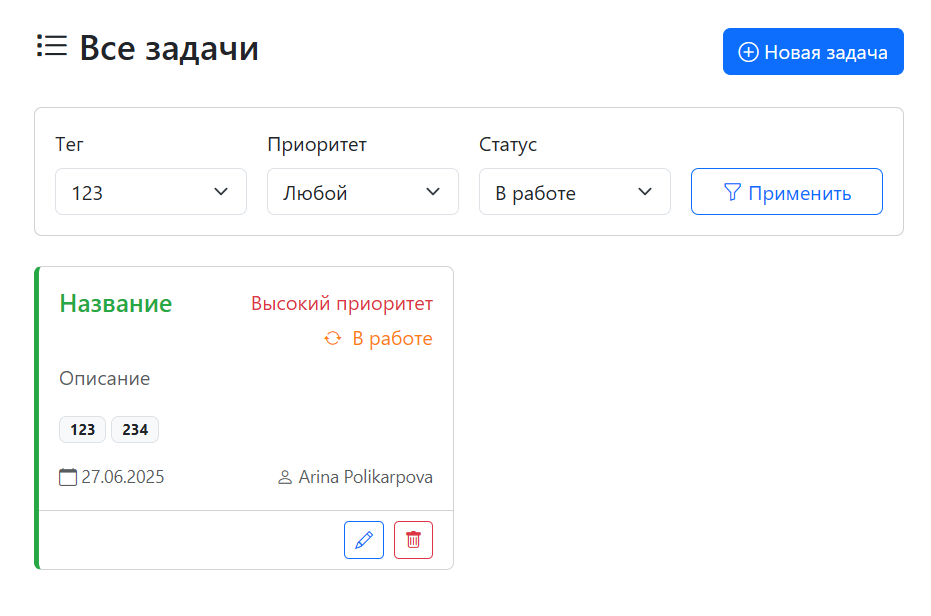


Рисунок 2 – Фильтрация задач пользователя

* 1. Авторизация и аутентификация пользователей

Веб-приложение использует механизм JWT для управления доступом пользователей. Аутентификация и авторизация реализованы на стороне backend-приложения с использованием Spring Security.

* + 1. Аутентификация

Приложение использует стандартный механизм аутентификации на основе сессий. Работа включает три основных этапа:

1. Идентификация пользователя. Клиент отправляет запрос на адрес auth/login → передаются учетные данные (email и пароль);
2. Проверка подлинности. Система находит пользователя в базе данных по email → сравнивает полученный пароль с сохраненным хешем (для сравнения используется алгоритм BCrypt);
3. Создание сессии. После успешной проверки создается уникальная сессия → клиенту возвращается идентификатор сессии и все последующие запросы содержат этот идентификатор.
   * 1. Авторизация

Каждому пользователю при регистрации присваивается роль:

* USER — пользователь, который может создавать и редактировать только свои задачи;
* ADMIN — администратор, имеющий полный доступ ко всем функциям, также может управлять другими пользователями.

На основе роли пользователя backend фильтрует доступ к маршрутам, используя аннотации и конфигурации Spring Security. Основная настройка осуществляется в классе SecurityConfig, где определяются правила доступа к API-эндпоинтам с использованием механизма URL-фильтрации и ограничения по ролям.

* 1. Схема базы данных

Логическая схема БД изображена на рисунке 3.

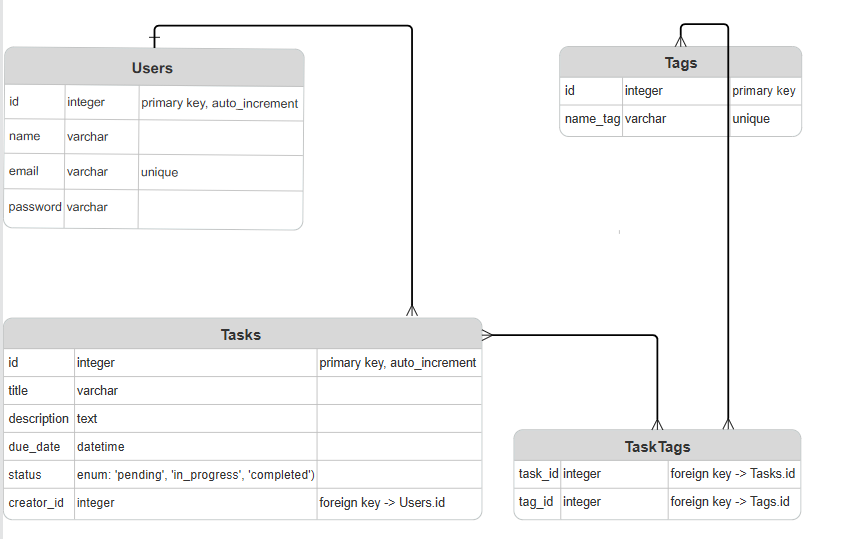


Рисунок 3 – Схема базы данных

* 1. Безопасность

В базе данных пароли хранятся с использованием хеширования BCrypt. Это критически важная мера, так как, например, при утечке данных восстановить исходные пароли невозможно.

Также реализованы следующие меры:

* Валидация входных данных;
* Проверка уникальности идентификаторов пользователя (email) при регистрации;
* Управление доступом происходит за счет ролевой модели с двумя уровнями привилегий.

1. Демонстрация работы контейнеров и приложения

Для запуска всего приложения необходимо выполнить команду для сборки образов контейнеров ***docker-compose up –build***.

* Контейнер с базой данных PostgreSQL разворачивается с заданными параметрами;
* Контейнер бэка автоматически подключается к базе и запускает Spring Boot приложение;
* Доступ к приложению осуществляется по адресу http://localhost:8080.

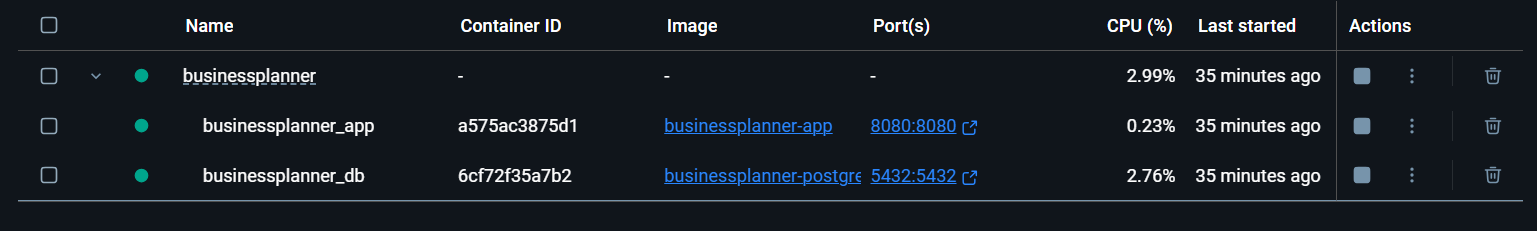


Рисунок 4 – Работа контейнеров

Для входа в систему пользователю необходимо аутентифицироваться в системе (рисунок 5).

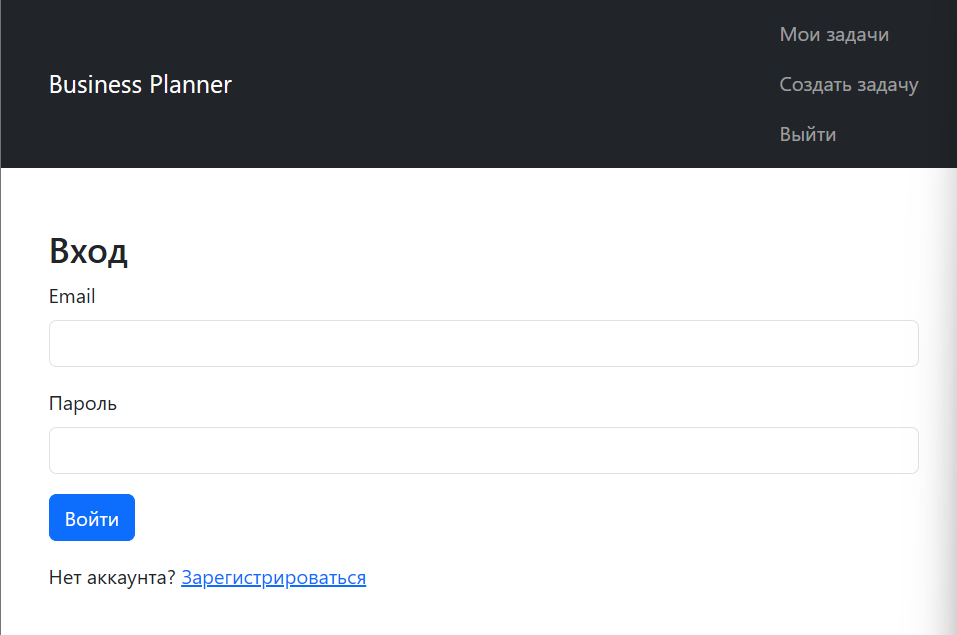


Рисунок 5 – Страница входа

В случае, если пользователь не зарегистрирован в системе, необходимо пройти процедуру регистрации (рисунок 6).

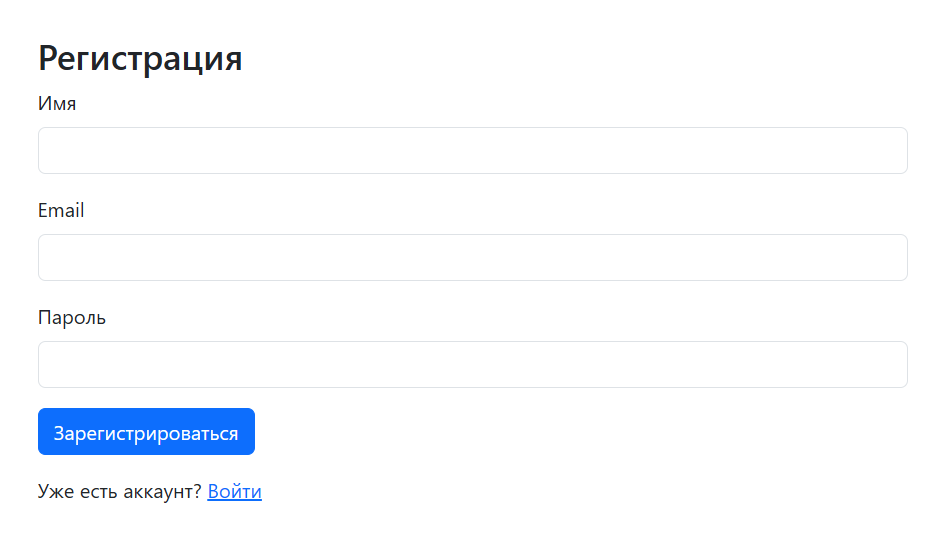


Рисунок 6 – Страница регистрации

После успешной регистрации пользователь перенаправляется на страницу входа, откуда уже может войти в систему и просмотреть все существующие задачи и создать новые (рисунок 7, 8).

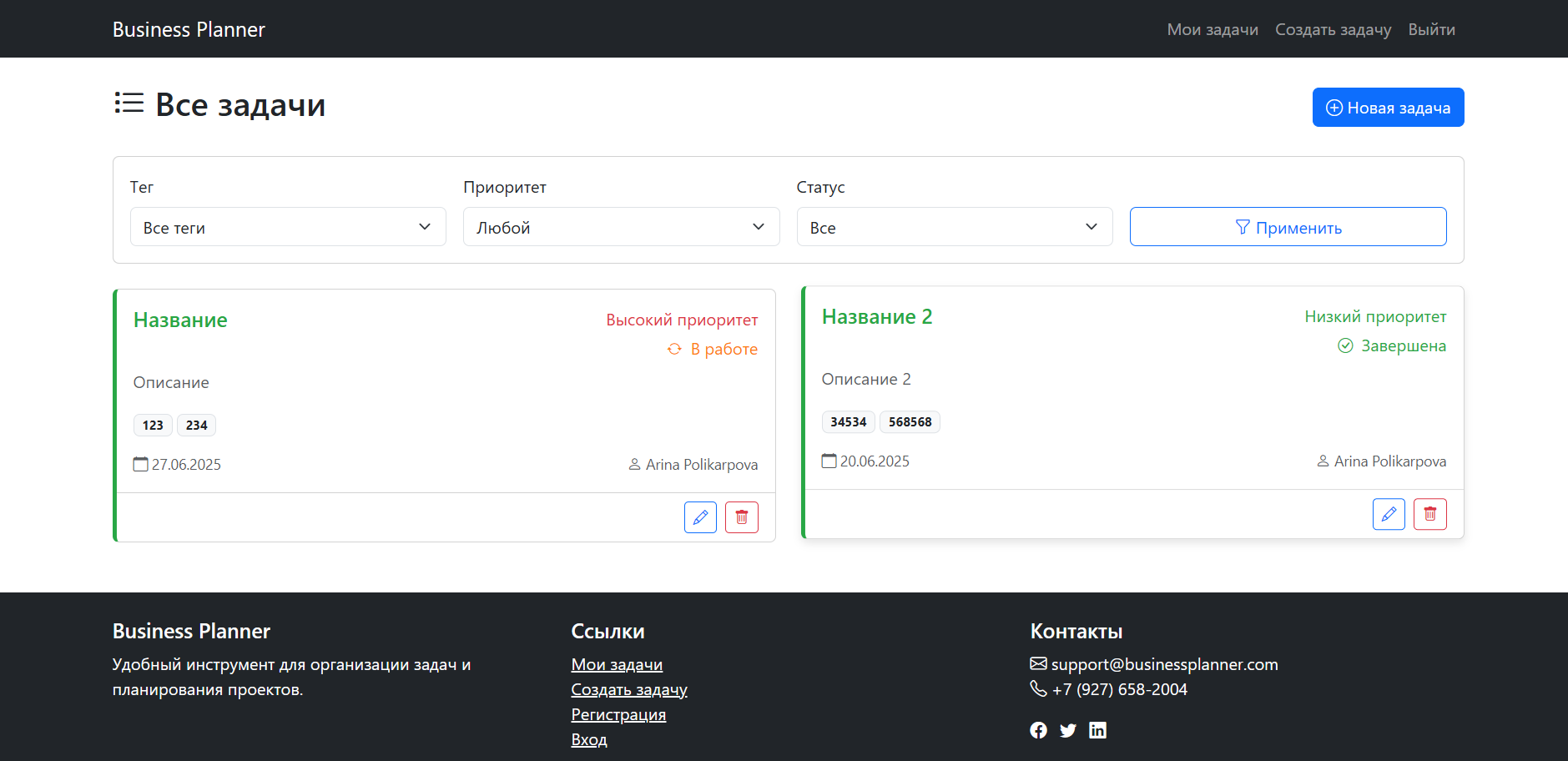


Рисунок 7 – Главная страница с задачами пользователя

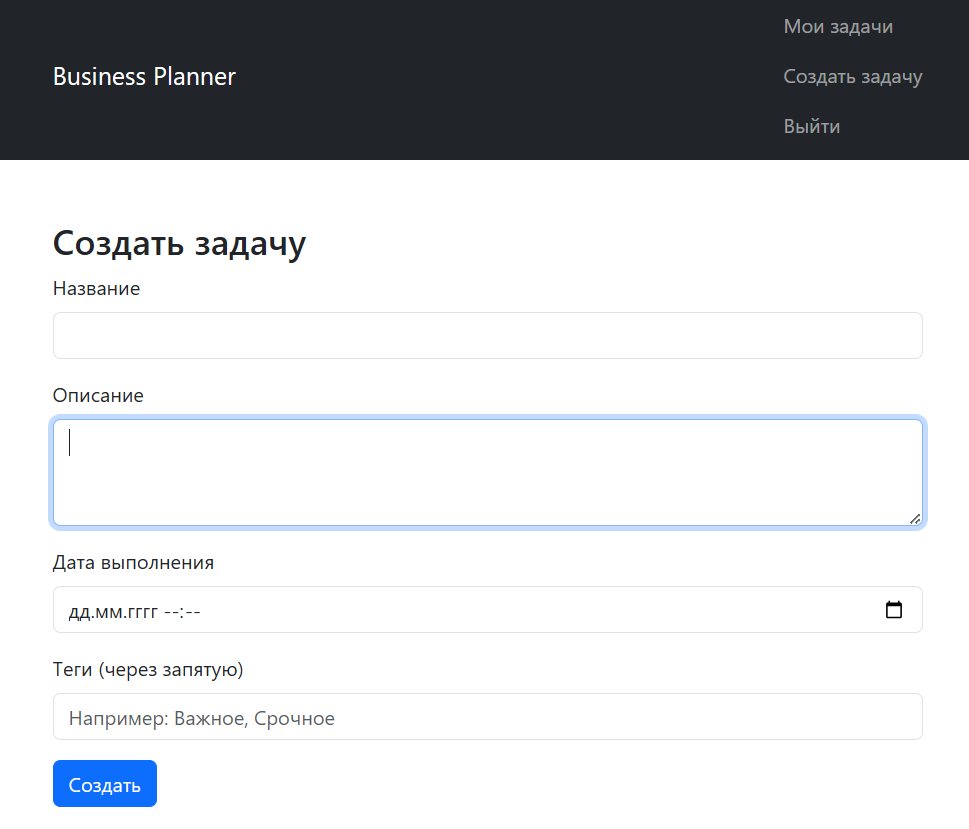


Рисунок 8 – Страница создания задачи

Приложение

Ссылка на GitHub с полной версией проекта:

https://github.com/yuliyaloganova/BusinessPlanner.git