

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, математики и электроники
Факультет информатики
Кафедра технической кибернетики

Отчет по курсу
Дисциплина: «Технологии сетевого программирования»

Выполнил:
Варданян Д. А.

Группа:
6301-010302D

Самара 2025

1 Общая архитектура приложения

1.1 Описание приложения

Данное веб-приложение представляет собой систему управления задачами (Task Manager), предназначенную для личного использования или небольших команд. Пользователь может регистрироваться и авторизовываться, после чего получает доступ к функционалу по созданию, просмотру, редактированию и удалению задач. Каждая задача может содержать заголовок, описание и статус (например, «К выполнению», «В процессе», «Выполнено»). Также реализована возможность смены пароля и обновления access-токена через refresh-механизм.

1.2 Состав приложения

Приложение состоит из следующих функциональных частей:

1. Серверная часть приложения.
2. Клиентская часть приложения.
3. База данных.

1.3 Описание функциональных частей

1.3.1 Серверная часть

Серверная часть приложения реализована на языке Java с использованием фреймворка Spring Boot. Безопасность системы обеспечивается с помощью Spring Security, который реализует авторизацию и аутентификацию пользователей на основе JWT-токенов с поддержкой механизма их обновления. Обмен данными между клиентской и серверной частями осуществляется через REST API, построенное с использованием Spring Web. Для взаимодействия с базой данных PostgreSQL используется Spring Data JPA, что позволяет удобно управлять сущностями и выполнять операции сохранения, чтения, обновления и удаления. Сборка проекта и управление зависимостями осуществляется с помощью Maven, что обеспечивает автоматизацию процесса компиляции и подключения необходимых библиотек.

1.3.2 Клиентская часть

Клиентская часть разработана с использованием языка JavaScript и фреймворка React 19. Для организации маршрутизации по страницам приложения применяется библиотека React Router DOM. Обмен данными с серверной частью осуществляется посредством библиотеки Axios, которая позволяет удобно выполнять HTTP-запросы к REST API. Интерфейс приложения стилизован с помощью HTML и CSS, в том числе с применением модульной структуры стилей. Состояние приложения хранится и управляется в компонентных состояниях с использованием встроенного хука useState, а также побочных эффектов через useEffect. Реализация авторизации, добавления задач и обновления токенов JWT также обрабатывается на клиентской стороне.

1.3.3 База данных

В качестве СУБД используется PostgreSQL. А для работы с ней применяются ORM модели(см. Приложение 4).

1.4 Система авторизации и безопасности

Система авторизации реализована через JWT (JSON Web Token). Используются access и refresh токены. Access используется для краткосрочного доступа (время его действия принято устанавливать не больше 2 часов), а refresh для обновления access токена (время действия от 7 до 30 дней) (см. Приложение 5).

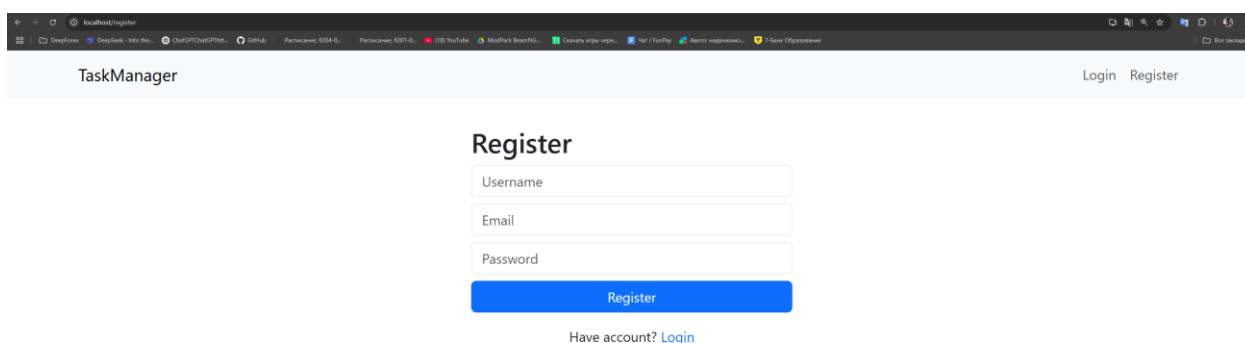
1.5 Функционирование приложения

Приложение собрано и запускается с помощью файла docker-compose.yml, находящегося в корне проекта и который объединяет 3 контейнера (бэкенд, фронтенд и базу данных (см. Приложение 6).

2 Работа приложения

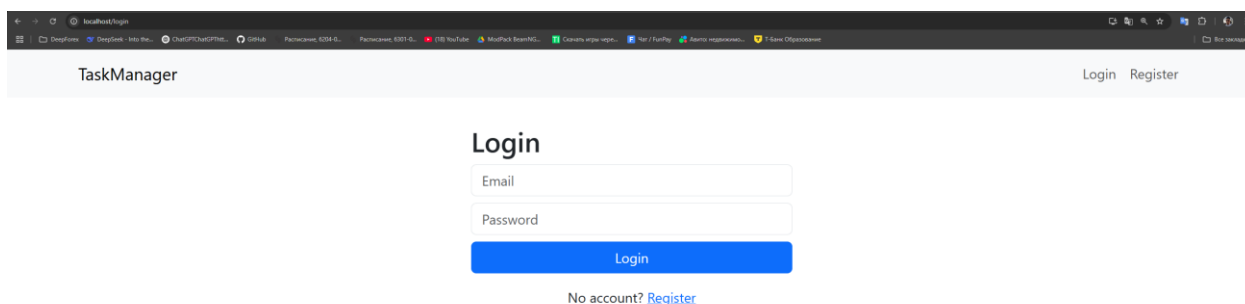
2.1 Регистрация и авторизация

На рисунке 1 представлена страница регистрации, где пользователь вводит свои данные (имя, фамилию, электронную почту, пароль, номер телефона и адрес). Зарегистрировать пользователя с уже имеющейся почтой в БД не выйдет. После успешной регистрации, пользователь может войти в магазин на странице входа, показанной на рисунке 2.



The screenshot shows a web browser window with the address bar displaying 'localhost/register'. The page has a header with 'TaskManager' on the left and 'Login Register' on the right. The main content area is titled 'Register' and contains three input fields: 'Username', 'Email', and 'Password'. Below these fields is a blue 'Register' button. At the bottom of the form, there is a link: 'Have account? [Login](#)'.

Рисунок 1 –Страница регистрации



The screenshot shows a web browser window with the address bar displaying 'localhost/login'. The page has a header with 'TaskManager' on the left and 'Login Register' on the right. The main content area is titled 'Login' and contains two input fields: 'Email' and 'Password'. Below these fields is a blue 'Login' button. At the bottom of the form, there is a link: 'No account? [Register](#)'.

Рисунок 2– Страница входа

Далее открывается главная страница приложения, изображённая на рисунке, на которой отображается список задач, принадлежащих текущему авторизованному пользователю. В верхней части интерфейса размещена панель навигации, содержащая логотип приложения, а также ссылки для выхода из аккаунта и перехода к форме смены пароля. Ниже представлена форма добавления новой задачи, включающая поля для ввода заголовка, описания и выбора статуса из выпадающего списка. Под формой отображается список уже созданных задач, где каждая задача сопровождается возможностью изменить её статус и удалить при необходимости. Все действия пользователя автоматически синхронизируются с сервером через REST API.

The screenshot displays the 'TaskManager' application interface. At the top, there is a header bar with the application name 'TaskManager' on the left and links for 'Change Password' and 'Logout' on the right. Below the header, the main section is titled 'Tasks' and indicates the user is 'Logged in as: n'. A form for adding a new task is present, featuring input fields for 'Title' and 'Content', a dropdown menu for status (currently set to 'Новая'), and a green 'Add Task' button. Below the form, a list of tasks is shown. Each task entry includes a status dropdown (e.g., 'В работе', 'На проверке'), a description field, and a red 'Delete' button. The tasks listed are 'Моя задача' (twice) and '345'.

Рисунок 3 – Главная страница

2.1 Изменение статуса задачи

Чтобы изменить статус задачи, пользователь может воспользоваться выпадающим списком, расположенным рядом с заголовком каждой задачи. На рисунке показано, как отображается список доступных статусов: «Новая», «В работе», «На проверке», «Завершена», «Отложена». После выбора нового статуса он автоматически отправляется на сервер, где происходит обновление соответствующей записи в базе данных. Обновлённый список задач мгновенно отображается на клиентской стороне без необходимости перезагрузки страницы.

Моя задача	В работе	Delete
Описание задачи	Новая	
Моя задача	В работе	Delete
Описание задачи	На проверке	
	Завершена	
	Отложена	
345	На проверке	Delete
345		

Рисунок 4 – Главная страница, товары

2.2 Изменение пароля пользователя

На странице профиля есть возможность поменять пароль. Для изменения пароля необходимо нажать на соответствующую кнопку, написать актуальный пароль и новый нужно написать 2 раза, при несоответствии паролей, выведет ошибка.

TaskManager

Change Password Logout

Change Password

Current password

New password

Save

Рисунок 5– Профиль пользователя

Для изменения пароля надо нажать на кнопку смены пароля, рядом с кнопкой выхода, как на рисунке 5.

После сохранения данных выходит сообщение об успешности изменений. Поле можно увидеть на рисунке 6.

Change Password

Password updated — please login again

Current password

New password

Save

Рисунок 6 — Успешное изменение пароля пользователя

2.3 Выход из профиля

При нажатии на "Logout", стираются все данные пользователя (срабатывает функция, стирающая данные из localStorage: userId, access/refresh токены) и перед ним открывается страница входа. Если человек захочет переместиться на другую страницу приложения через адресную строку, его обратно перебросят на "вход".

ЗАКЛЮЧЕНИЕ

В ходе работы были освоены ключевые технологии: Spring Security, JWT, ReactRouter, ORM. Стала более понятна работа с RESTAPI, контейнеризацией и работа с такими инструментами как Postman, DBeaver.

Были выполнены все основные требования к курсу:

1. Проектирование приложения.
2. Разработка базы данных.
3. Разработка API.
4. Авторизация.
5. Разработка клиентской части
6. Финализация приложения и упаковка в Docker

Что можно улучшить:

1. Добавить тесты (JUnit, Jest).
2. Реализовать ролевую модель (админ/пользователь).

Приложение 1

Файл WebSecurityConfig.java

```
package com.taskmanager.backend.security;

import com.taskmanager.backend.security.jwt.AuthEntryPointJwt;
import com.taskmanager.backend.security.jwt.AuthTokenFilter;
import com.taskmanager.backend.security.service.UserDetailsServiceImpl;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.access.AccessDeniedHandler;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
```

@EnableWebSecurity

@EnableMethodSecurity

@RequiredArgsConstructor

```
public class WebSecurityConfig {

    private final UserDetailsServiceImpl userDetailsService;
    private final AuthTokenFilter authTokenFilter;
    private final AuthEntryPointJwt unauthorizedHandler;

    /* ----- DAO provider ----- */
    @Bean
    public DaoAuthenticationProvider daoAuthenticationProvider() {
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setUserDetailsService(userDetailsService);
        provider.setPasswordEncoder(passwordEncoder());
        return provider;
    }

    /* ----- Password encoder ----- */
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    /* ----- AccessDenied → 401 ----- */
    @Bean
    public AccessDeniedHandler accessDeniedHandler() {
        return (req, res, ex) ->
            res.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Error: Un-
authorized");
    }
}
```

```

}

/* ----- AuthenticationManager ----- */
@Bean
public AuthenticationManager authenticationManager(
    AuthenticationConfiguration authConfig) throws Exception {
    return authConfig.getAuthenticationManager();
}

/* ----- Security filter chain ----- */
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Excep-
tion {
    http
        // 1) CSRF отключён (stateless REST)
        .csrf(csrf -> csrf.disable())

        // 2) Stateless сессии
        .sessionManagement(sm ->
            sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        )

        // 3) Обработка ошибок: и аутентификация, и отказ в доступе → 401
        .exceptionHandling(ex -> ex
            .authenticationEntryPoint(unauthorizedHandler)
            .accessDeniedHandler(accessDeniedHandler())
        )

        // 4) Публичные эндпоинты: всё под /api/auth/**
        // и остальное — только с валидным JWT

```

```

        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/api/auth/**").permitAll()
            .anyRequest().authenticated()
        )

        // 5) DAO-провайдер и JWT-фильтр
        .authenticationProvider(daoAuthenticationProvider())
        .addFilterBefore(authTokenFilter, UsernamePasswordAuthenticationFilter.class)
    ;

    return http.build();
}
}

```

Приложение 2

Файл AuthController.java

```

// AuthController.java

package com.taskmanager.backend.controller.auth;

import com.taskmanager.backend.dto.*;
import com.taskmanager.backend.security.jwt.JwtUtils;
import com.taskmanager.backend.security.service.*;
import com.taskmanager.backend.service.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.*;
import org.springframework.security.authentication.*;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.*;

```

```

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired private AuthenticationManager authenticationManager;
    @Autowired private UserService userService;
    @Autowired private JwtUtils jwtUtils;
    @Autowired private RefreshTokenService refreshTokenService;
    @Autowired private UserDetailsServiceImpl userDetailsServiceImpl;

    // LOGIN → access + refresh (persisted)
    @PostMapping("/login")
    public ResponseEntity<JwtResponse> authenticateUser(@RequestBody
LoginRequest req) {
        Authentication auth = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(req.getEmail(),
req.getPassword()))
        );
        SecurityContextHolder.getContext().setAuthentication(auth);
        UserDetailsImpl ud = (UserDetailsImpl) auth.getPrincipal();

        String accessToken = jwtUtils.generateJwtToken(ud.getUsername());
        // сохраняем и возвращаем refresh
        var refreshToken = refreshTokenService.createRefreshToken(ud.getId());

        return ResponseEntity.ok(new JwtResponse(
            accessToken,
            refreshToken.getToken(),
            ud.getId(),

```

```

        ud.getName(),
        ud.getEmail()
    ));
}

//                                     REGISTER
@PostMapping("/register")
public ResponseEntity<String> registerUser(@RequestBody SignupRequest req)
{
    try {
        userService.registerUser(req.getUsername(), req.getEmail(), req.getPassword());
        return ResponseEntity.ok("User registered successfully");
    } catch (IllegalArgumentException e) {
        // 400 Bad Request с понятным сообщением
        return ResponseEntity
            .status(HttpStatus.BAD_REQUEST)
            .body(e.getMessage());
    }
}

//                                     CHANGE                                     PASSWORD
@PostMapping("/change-password")
public ResponseEntity<String> changePassword(@RequestBody ChangePasswordDTO dto) {
    String email = SecurityContextHolder.getContext().getAuthentication().getName();
    userService.changePassword(email, dto.getOldPass(), dto.getNewPass());
    return ResponseEntity.ok("Password changed successfully");
}

```

```

//                                REFRESH                                ACCESS                                TOKEN
@PostMapping("/refresh-token")
public ResponseEntity<TokenRefreshResponse> refreshToken(@RequestBody
TokenRefreshRequest                                request)                                {
    String                                rt                                =                                request.refreshToken();
    return                                refreshTokenService.findByToken(rt)
        .map(refreshTokenService::verifyExpiration)
        .map(validToken                                ->                                {
            String                                username                                =                                validToken.getUser().getEmail();
            String                                newAccess                                =                                jwtUtils.generateJwtToken(username);
            return ResponseEntity.ok(new TokenRefreshResponse(newAccess,
validToken.getToken()));
        })
        .orElseGet(()                                ->                                {
            ResponseEntity
                .status(HttpStatus.UNAUTHORIZED)
                // Возвращаем «пустой» объект и ошибочный статус
                .body(new TokenRefreshResponse("", ""))
        });
    }
}

```

Приложение 3

Файл RefreshTokenRepository.java

```

package                                com.taskmanager.backend.repository;

import                                com.taskmanager.backend.model.RefreshToken;
import                                com.taskmanager.backend.model.User;
import                                org.springframework.data.jpa.repository.JpaRepository;
import                                java.util.Optional;

```

```

public interface RefreshTokenRepository extends JpaRepository<RefreshToken,
Long>
{
    Optional<RefreshToken> findByToken(String token);
    void deleteByUser(User user);
}

```

Приложение 4

Файл App.js

```

< import React from 'react';
import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom';
import Login from './pages/Login';
import Register from './pages/Register';
import TasksList from './pages/TasksList';

```

```

function PrivateRoute({ children }) {
    const token = localStorage.getItem('accessToken');
    return token ? children : <Navigate to="/login" />;
}

```

```

function App() {
    return (
        <BrowserRouter>
        <Routes>
            <Route path="/login" element={<Login/>} />
            <Route path="/register" element={<Register/>} />
            <Route
                path="/tasks"
                element={
                    <PrivateRoute>
                        <TasksList/>

```



```

        </PrivateRoute>
    }
/>
    <Route path="*" element={<Navigate to="/tasks" replace />} />
</Routes>
</BrowserRouter>
);
}

```

export default App;

Приложение 5

Перехватчик ответов в axios.js

```

// src/api/axios.js

import axios from 'axios';
import { authService } from '../services/authService';

const axiosInstance = axios.create({
  baseURL: '/api', // проху на localhost:8080
  headers: { 'Content-Type': 'application/json' }
});

/* ----- запрос: подставляем токен ----- */
axiosInstance.interceptors.request.use(config => {
  const token = localStorage.getItem('accessToken');
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});

/* ----- helpers для очереди запросов, ожидающих refresh ----- */
let isRefreshing = false;

```

```

let subscribers = [];

const subscribe = cb => subscribers.push(cb);
const onRefreshed = token => {
  subscribers.forEach(cb => cb(token));
  subscribers = [];
};

/* ----- ответ: если 401 → пробуем refresh ----- */
axiosInstance.interceptors.response.use(
  res => res,
  async error => {
    const { config, response } = error;
    if (!response || response.status !== 401 || config.__isRetryRequest) {
      // не 401 или уже повторяли — отдаём ошибку дальше
      return Promise.reject(error);
    }

    /* маркируем запрос, чтобы не зациклиться */
    config.__isRetryRequest = true;

    /* ---- запускаем refresh один раз ---- */
    if (!isRefreshing) {
      isRefreshing = true;
      try {
        const newToken = await authService.refresh(); // POST /auth/refresh-token
        isRefreshing = false;
        onRefreshed(newToken); // разбудим «ожидających»
      } catch (refreshErr) {
        isRefreshing = false;

```

```

    authService.logout(); // refresh недействителен
    window.location.href = '/login';
    return Promise.reject(refreshErr);
  }
}

/* ---- ставим запрос в очередь до завершения refresh ---- */
return new Promise(resolve => {
  subscribe(token => {
    config.headers.Authorization = `Bearer ${token}`;
    resolve(axiosInstance(config)); // повторяем запрос
  });
});
}
);

```

```
export default axiosInstance;
```

Приложение 6

Настройки в application.yml

```

# Database configuration
spring.datasource.url=jdbc:postgresql://localhost:5432/taskmanager
spring.datasource.username=postgres
spring.datasource.password=postgres

# Hibernate configuration
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

# Flyway configuration

```

```
spring.flyway.enabled=true
spring.flyway.locations=classpath:db/migration
spring.flyway.baseline-on-migrate=true
```

```
#                                JWT                                settings
#  ??????? 32  ???????,  ????? 40 (UTF-8  ???  ==  ??????? ASCII)
jwt.secret=ThisIsASuperSecretKeyWith32PlusChars12345
jwt.expirationMs=60000
#      ?????      ?????      refresh      token      (?????????,      30      ????)
jwt.refreshExpirationMs=2592000000
```

Файл RefreshTokenService.java

```
package                                com.taskmanager.backend.service;

import                                com.taskmanager.backend.model.RefreshToken;
import                                com.taskmanager.backend.model.User;
import                                com.taskmanager.backend.repository.RefreshTokenRepository;
import                                com.taskmanager.backend.repository.UserRepository;
import                                org.springframework.beans.factory.annotation.Value;
import                                org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional; // <— добав-
лено

import                                java.time.Instant;
import                                java.util.Optional;
import                                java.util.UUID;

@Service

public                                class                                RefreshTokenService                                {

    @Value("${jwt.refreshExpirationMs}")
```

```

private                                Long                                refreshTokenDurationMs;

private    final    RefreshTokenRepository    refreshTokenRepository;
private    final    UserRepository            userRepository;

public RefreshTokenService(RefreshTokenRepository refreshTokenRepository,
                           UserRepository            userRepository)    {
    this.refreshTokenRepository    =    refreshTokenRepository;
    this.userRepository            =    userRepository;
}

/**
 * Создаёт новый refresh-токен, предварительно удалив старый (если был).
 */
@Transactional
public RefreshToken createRefreshToken(Long    userId)    {
    User    user    =    userRepository.findById(userId)
        .orElseThrow(() -> new IllegalArgumentException("User not found"));

    // удаляем старый, если он существует
    refreshTokenRepository.deleteByUser(user);

    RefreshToken    rt    =    new RefreshToken();
    rt.setUser(user);
    rt.setToken(UUID.randomUUID().toString());
    rt.setExpiryDate(Instant.now().plusMillis(refreshTokenDurationMs));
    return    refreshTokenRepository.save(rt);
}

/**

```

```

    *      Ищет refresh-токен по строковому значению.
    */

    public Optional<RefreshToken> findByToken(String token) {
        return refreshTokenRepository.findByToken(token);
    }

    /**
     * Проверяет, не истёк ли срок токена; если истёк — удаляет его и кидает
    исключение.
    */

    @Transactional
    public RefreshToken verifyExpiration(RefreshToken token) {
        if (token.getExpiryDate().isBefore(Instant.now())) {
            refreshTokenRepository.delete(token);
            throw new RuntimeException("Refresh token expired");
        }
        return token;
    }

    /**
     * Удаляет все refresh-токены, связанные с данным userId.
     */

    @Transactional
    public void deleteByUserId(Long userId) {
        userRepository.findById(userId)
            .ifPresent(refreshTokenRepository::deleteByUser);
    }
}

```

Приложение 9

Файл `docker-compose.yml`

```
version: "3.8"

services:
# ----- PostgreSQL
# -----
  db:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: taskmanager
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 5s
      timeout: 5s
      retries: 5

# ----- Back-end (Spring Boot)
# -----
  backend:
    build: ./backend
    ports:
      - "8080:8080"
    environment:
```

```

    SPRING_DATASOURCE_URL:      jdbc:postgresql://db:5432/taskmanager
    SPRING_DATASOURCE_USERNAME:      postgres
    SPRING_DATASOURCE_PASSWORD:      postgres

depends_on:
  db:
    condition:      service_healthy

healthcheck:
  test:      ["CMD", "curl", "-f", "http://localhost:8080/actuator/health"]
  interval:      10s
  timeout:      5s
  retries:      5

# ----- Front-end (React + Nginx)
-----

frontend:
  build:      ./frontend
  ports:
    -      "80:80"
  depends_on:
    backend:
      condition:      service_healthy
  healthcheck:
    test:      ["CMD", "curl", "-f", "http://localhost"]
    interval:      10s
    timeout:      5s
    retries:      5

volumes:
  postgres_data:

```