Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

По дисциплине «Рефакторинг баз данных и приложений»
Итерация №1

Студент:
Дениченко Александр P3412
Беляев Михаил P3412
Практик:
Логинов Иван Павлович

Санкт-Петербург
2025 г.

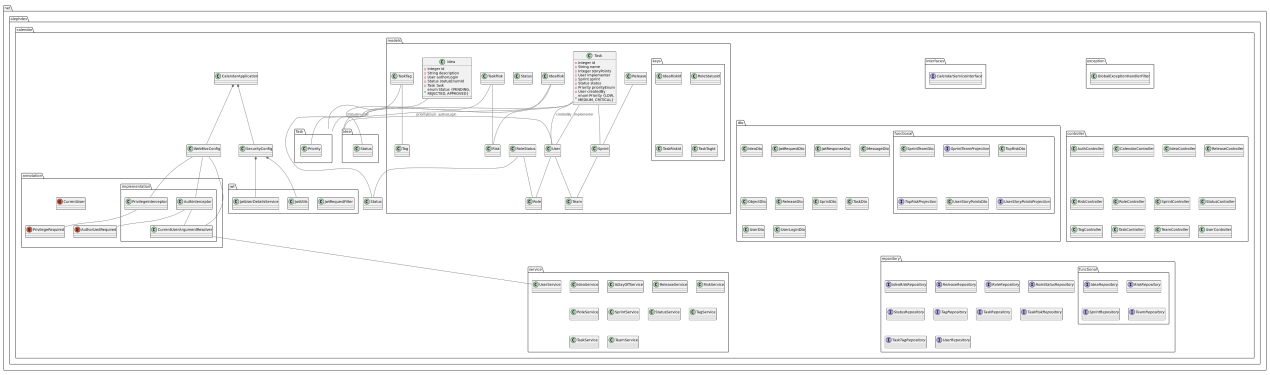
Цель

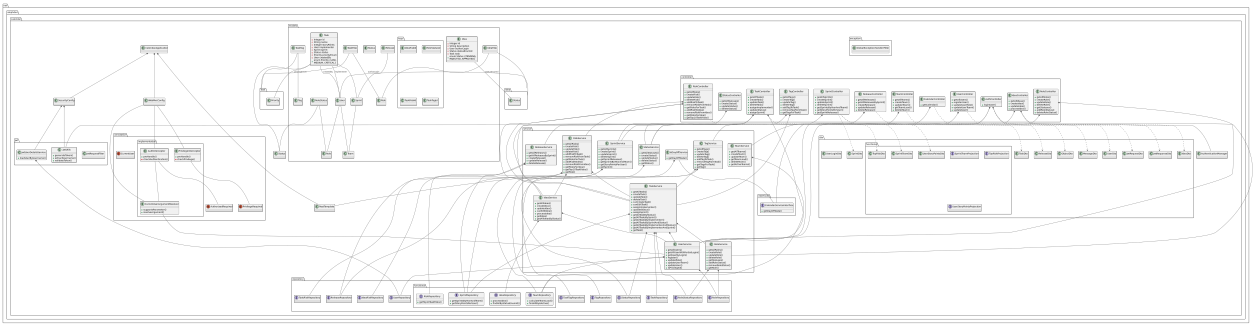
Данный отчёт описывает комплексный рефакторинг проекта системы управления календарём и задачами. В рамках рефакторинга были применены практики разработки программного обеспечения, направленные на повышение качества кода, улучшение архитектуры и упрощение развертывания системы.

1 Вводная часть

Репозиторий: github.com/Alex-de-bug/is_kurs
Ветка: iteration/1
Период: коммиты с id от ‘d3a49cfce558150da979b59822d139bccfb22d93‘ до ‘7f48e326894c5f440d1e9394f58fa0f50776461f‘

2 Архитектура





3 Рефакторинг инфраструктуры и развертывания

Коммит: `d3a49cf` — «рефакторинг автоматического разворота»

3.0.1 Описание практики

Выполнен рефакторинг процесса контейнеризации и автоматического развертывания приложения с использованием Docker и Docker Compose.

3.0.2 Выполненные изменения

- Добавлен файл `.dockerignore` для исключения ненужных файлов из Docker-образа
- Улучшен `Dockerfile` — оптимизирована многоэтапная сборка
- Переработан `docker-compose.yml` — улучшена конфигурация сервисов

3.0.3 Применённые практики

- **Multi-stage builds** — использование многоэтапной сборки Docker для уменьшения размера образа
- Последовательность команд для эффективного кеширования
- Параметризация настроек приложения

3.0.4 Важность для проекта

1. Автоматизированный процесс сборки и запуска сокращает время развертывания
2. Гарантируется идентичность окружений разработки, тестирования и продакшена
3. Оптимизация Docker-образов снижает нагрузку на инфраструктуру и ускоряет доставку
4. Новые разработчики могут поднять проект одной командой

4 Рефакторинг структуры базы данных

Коммит: `613271a` — «Отделение sql для инита»

4.0.1 Описание практики

Проведена модульная реорганизация SQL-скриптов инициализации базы данных с разделением на логические компоненты.

4.0.2 Выполненные изменения

- Создана отдельная директория `postgres/` для всех SQL-скриптов
- Монолитный файл разделён на модульные компоненты:
  - `init-tables.sql` — создание таблиц
  - `init-constraints.sql` — определение ограничений
  - `init-index.sql` — создание индексов
  - `init-func.sql` — функции базы данных
  - `init-procedure.sql` — хранимые процедуры
  - `init-triggers.sql` — триггеры
  - `init-inserts.sql` — начальные данные
  - `init-pass.sql` — инициализация паролей
- Реорганизована структура проекта (перемещены диаграммы классов в отдельную папку)
- Обновлён `application.yaml` для работы с новой структурой

4.0.3 Применённые практики

- **Separation of Concerns (SoC)** — разделение ответственности на уровне SQL-скриптов
- **Single Responsibility Principle (SRP)** — каждый файл отвечает за одну область БД

4.0.4 Важность для проекта

1. Легко найти и модифицировать конкретный аспект схемы
2. Точечные изменения в git-истории, простота code review
3. Изменения в одном компоненте не влияют на другие
4. Разные разработчики могут работать с разными файлами без конфликтов

5 Внедрение стандартов качества кода

Коммит: `e7c6774` — «Добавлена проверка при сборке проекта на соответствие стилю кода»

5.0.1 Описание практики

Интегрирована автоматическая проверка соответствия кода стандартам Google Java Style Guide на этапе сборки проекта.

5.0.2 Выполненные изменения

- Добавлена зависимость `maven-checkstyle-plugin`
- Интегрированы конфигурационные файлы:
  - `tools/checkstyle/google_checks.xml` — правила проверки кода
  - `tools/checkstyle/eclipse-java-google-style.xml` — конфигурация для IDE
- Настроена автоматическая проверка при выполнении сборки

5.0.3 Применённые практики

- **Static Code Analysis** — статический анализ кода без его выполнения
- **Coding Standards** — следование индустриальным стандартам (Google Style Guide)
- **Shift Left Testing** — раннее обнаружение проблем на этапе сборки
- **Continuous Integration** — автоматизация проверок качества кода

5.0.4 Важность для проекта

1. Весь код проекта следует единому стилю, независимо от автора
2. Часть проверок выполняется автоматически, экономя время команды
3. Многие потенциальные проблемы обнаруживаются до попадания в репозиторий
4. Стандартизированный код легче читать и понимать
5. Проблемы выявляются и устраняются сразу, не накапливаясь

6 Рефакторинг кодовой базы

6.1 Коммиты

- `8722ded` — «Рефакторинг кода на стиль»
- `403fa28` — «устранение проблем в коде для соответствия google style»

6.1.1 Описание практики

Проведён масштабный рефакторинг всей кодовой базы для приведения к стандартам Google Java Style Guide и применения лучших практик ООП.

6.1.2 Выполненные изменения

Первый этап (8722ded) — Общий рефакторинг:

- Затронутые области:
  - **Конфигурация безопасности** (`SecurityConfig.java`, `WebMvcConfig.java`, `WebSocketConfig.java`)
  - **Контроллеры** — все 13 контроллеров (Auth, Calendar, Idea, Release, Risk, Role, Sprint, Status, Tag, Task, Team, User)
  - **Модели данных** — все 15 entity-классов
  - **DTO** — 12 классов передачи данных
  - **Сервисы** — все 10 сервисных классов
  - **Репозитории** — функциональные репозитории и спецификации
  - **Обработка исключений** (`GlobalExceptionHandlerFilter`)
  - **JWT-аутентификация** (`JwtRequestFilter`, `JwtUtils`, `JwtUserDetailsService`)

Второй этап (403fa28) — Устранение проблем:

- Устранение оставшихся нарушений стиля в 30 файлах
- Исправление предупреждений Checkstyle, которые не правятся автоматически

6.1.3 Применённые практики

- **Consistent Indentation** — единообразные отступы (2 пробела)
- **Line Length Limit** — ограничение длины строки (100 символов)
- **Import Organization** — правильная организация импортов
- **Naming Conventions** — соблюдение конвенций именования (camelCase, PascalCase)
- **DRY (Don't Repeat Yourself)** — устранение дублирования кода
- **Clean Code** — применение принципов чистого кода

7 Документирование API через OpenAPI

7.1 Коммиты

- `0935443` — «openapi config»
- `9134f3a` — «functional dtos schemas»
- `6887adf` — «general dtos schemas»
- `f703de0` — «all controllers schemas»

7.1.1 Описание практики

Внедрена полная документация REST API с использованием спецификации OpenAPI 3.0 (Swagger), включающая аннотирование всех контроллеров, методов и моделей данных.

7.1.2 Выполненные изменения

Этап 1 — Конфигурация OpenAPI (0935443)
Этап 2 — Документирование Functional DTOs (9134f3a):
Добавлены аннотации `@Schema` для специализированных DTO:

- `SprintTeamDto` — данные о команде спринта
- `TopRiskDto` — информация о главных рисках
- `UserStoryPointsDto` — статистика story points пользователей

Этап 3 — Документирование General DTOs (6887adf):

- `IdeaDto`, `ReleaseDto`, `SprintDto`, `TaskDto`, `UserDto` — основные сущности
- `JwtRequestDto`, `JwtResponseDto` — аутентификация
- `MessageDto` — системные сообщения

Этап 4 — Документирование контроллеров (f703de0):

- **Аннотации контроллеров:** `@Tag` для группировки endpoints
- **Аннотации методов:** `@Operation` с описанием каждой операции
- **Аннотации параметров:** `@Parameter` для query, path и body параметров
- **Аннотации ответов:** `@ApiResponse` для всех возможных HTTP-статусов
- **Схемы безопасности:** `@SecurityRequirement` для защищённых endpoints

7.1.3 Применённые практики

- **OpenAPI Specification** — использование индустриального стандарта документирования API
- **Self-Documenting Code** — код документирует сам себя через аннотации
- **API-First Design** — явное описание контракта API

7.1.4 Важность для проекта

Для разработки:

1. **Contract-First Development** — чёткий контракт между frontend и backend
2. **Reduced Communication Overhead** — меньше вопросов между командами
3. **API Testing** — возможность тестирования API прямо из браузера
4. **Mock Generation** — автоматическая генерация моков для тестирования

Для команды:

5. **Onboarding** — новые разработчики быстро понимают структуру API
6. **Knowledge Sharing** — документация доступна всем участникам команды
7. **Consistency** — единообразное понимание API всеми разработчиками

Для проекта:

8. **Professional Standard** — использование индустриальных стандартов
9. **Integration Ready** — легкая интеграция с внешними системами
10. **API Discovery** — возможность исследования API через Swagger UI
11. **Reduced Documentation Debt** — документация поддерживается автоматически вместе с кодом
12. **Quality Assurance** — явное описание ожидаемого поведения упрощает тестирование

8 Итоги и результаты

При каждой фазе рефакторинга проект стновилось легче и легче просматривать, развивать. Данная работа помогла структурировать практически мертвый для разворота и разработки проект.