Пояснительная записка к проекту «**Система управлениями проектами»**

**Описание задачи:**

Основная цель: разработать систему управления проектами.

Легенда:

Есть некая компания, в которой ведутся свои проекты. Компания нуждается в планировании этих проектов, их управлении, реализации и дальнейшей поддержке.

Не будем заострять внимание на различных этапах и сложностях процесса внутри данной компании. Основная задача сделать некоторый движок, который бы позволял управлять проектами, задачами внутри проекта, командами, которые работают над проектом.

Основные блоки и функции, которые необходимо будет реализовать в данной системе:

* Управление сотрудниками компании.
* Управление проектами и их жизненным циклом.
* Управление задачами внутри проекта и их жизненным циклом.
* Управление командами внутри проекта.
* Аутентификация в системе

**Описание архитектуры**

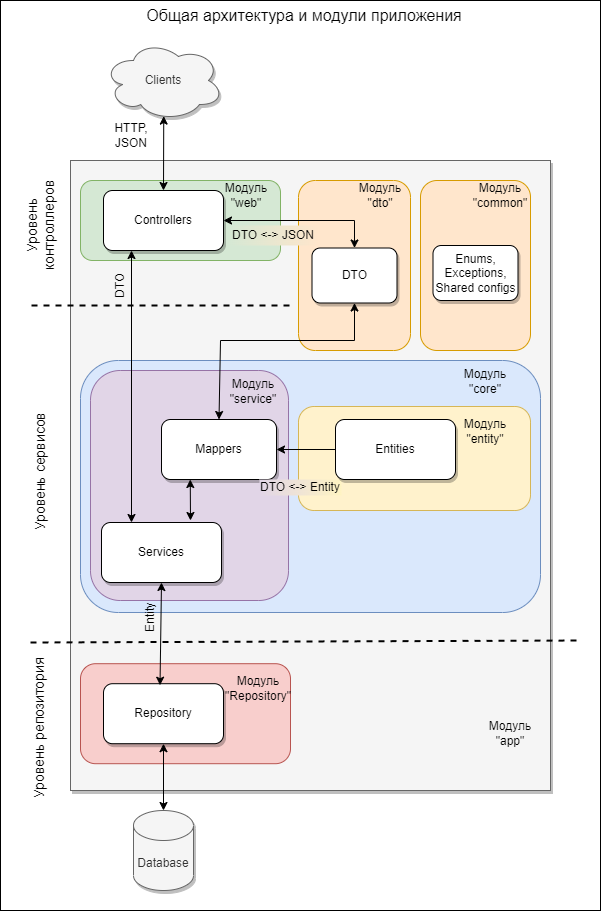
В приложении реализовано 3 основных уровня:

1. Уровень контроллеров - уровень представления информации, на котором происходит получение запросов от пользователей и подготовка ответов;
2. Уровень сервисов - уровень бизнес-логики, на котором производится работа с полученными или отсылаемыми данными;
3. Уровень репозитория - уровень управления, обработки и хранения информации.

Уровни будут взаимодействовать между собой последовательно: уровень контроллеров с уровнем сервисов, сервисы с репозиторием и аналогично в обратном порядке.

Помимо логического разделения на уровни, приложение будет разделяться на модули, выполняющие конкретную функцию.В отдельные модули выделены блоки приложения которые обеспечивают взаимодействие между уровнями приложения (DTO) и которые содержат объекты необходимые на разных уровнях (Common).

Схема проекта



**Описание модулей**

1. Модуль “app”. Базовый модуль, точка входа в приложение.
2. Модуль “web”. Модуль отвечает за прием и обработку HTTP запросов, подготовку и отправку ответов клиентам. Полученные из запроса данные передаются в модуль “service” модуля “core” для их дальнейшей обработки. Данные для подготовки ответа клиенту также поступают от модуля “service”. Обмен данными с модулем “service” используются DTO объекты. Для представления данных в виде DTO объекта модуль “web” взаимодействует с модулем “dto”.
3. Модуль “dto”. Данный модуль служит для представления различной информации (из HTTP запросов, из базы данных) в виде специализированных DTO объектов, с помощью которых осуществляется передача информации между некоторыми модулями. Не относится непосредственно к какому-либо уровню приложения, т.к. в основном обеспечивает взаимодействие между уровнями контроллеров и сервисов.
4. Модуль “common”. Содержит объекты которые могут использоваться разными модулями и частями приложения. Это объекты типа Enum, Exception, различные конфигурации.
5. Модуль “core”. Модуль состоит из нескольких подмодулей и объединяет в себе основные сущности и бизнес-логику приложения.

- Модуль “service”. Модуль содержит бизнес-логику приложения. Взаимодействует с модулем “web” для получения входящих и отправки обработанных данных. Взаимодействует с модулем “entity” для преобразования DTO объектов в сущности (Entities) и наоборот. Модуль выполняет функции управления информацией путем взаимодействия с модулем “repository”.

- Модуль “entity”. Содержит основные сущности участвующие в бизнес-логике приложения.

1. Модуль “repository”. Взаимодействует непосредственно с базой данных для управления сущностями. Получает сущности из модуля “service”.

**Основные объекты управления и операции над ними**

**Сотрудник (Employee)**

**Требования:**

Профиль пользователя, который обладает следующими атрибутами:

* Фамилия (обязательное поле)
* Имя (обязательное поле)
* Отчество (не обязательное)
* Должность (не обязательное)
* Учетная запись (не обязательное), но уникальное значение среди активных сотрудников (у двух активных профилей не может быть одинаковой учетной записи, если оно не пустое).
* Адрес электронной почты (не обязательное).
* Статус сотрудника - (обязательное поле) фиксированный набор значений (Активный, Удаленный)

**Реализация:**

Сотрудник представлен классом Employee. Статус сотрудника вынесен в отдельный Enum EmployeeStatus.

|  |  |
| --- | --- |
| Операция | Реализация логики в EmployeeService |
| Создание профиля сотрудника. При создании сотрудника должна создаваться карточка сотрудника с перечисленным выше набором атрибутов, также статус сотрудника становится Активный. | При поступлении запроса в EmployeeService выполняются проверки и вызывается метод save, который определяет операцию создания или обновления Employee. При создании автоматически устанавливается статус ACTIVE |
| Изменение сотрудника. При редактировании сотрудника должны редактировать поля профиля сотрудника. Удаленного сотрудника изменить нельзя. | При поступлении запроса в EmployeeService выполняются проверки и вызывается метод save, который определяет операцию создания или обновления Employee. При попытке изменения Employee со статусом Deleted выбрасывается исключение. |
| Удаление сотрудника. При удалении сотрудника, сотрудник переводится в статус Удаленный. | При поступлении запроса в EmployeeService выполняются проверки и вызывается метод deleteById, который переводит Employee в статус DELETED. При попытке изменения удаления Employee со статусом Deleted выбрасывается исключение. |
| Поиск сотрудников. Поиск осуществляется по текстовому значению, которое проверяется по атрибутам Фамилия, Имя, Отчество, учетной записи, адресу электронной почты и только среди активных сотрудников. | Реализован поиск по условиям. Условие задается через DTO-объект EmployeeFilter. Метод findAllByFilter |
| Получение карточки сотрудника либо по идентификатору профиля, либо по УЗ с | Реализованы методы получения сотрудника по id, получения всех сотрудников и получения сотрудника по названию учетной записи. Методы findById, findAll, findAllByFilter |

**Проект (Project)**

**Требования:**

Это сущность, которая обладает следующим набором атрибутов:

* Код проекта - некоторое уникальное имя проекта. Является обязательным и уникальным среди всех проектов.
* Наименование - текстовое значение содержащее короткое наименование проекта.. Обязательное поле.
* Описание - текстовое значение содержащее более детальную информацию о проекте. Не обязательное поле.
* Статус проекта - текстовое значение, обозначающее состояние проекта. Список статусов фиксированный (Черновик, В разработке, В тестировании, Завершен). Обязательное поле.

Реализация

Проект представлен классом Project, статус вынесен в отдельный Enum ProjectStatus.

|  |  |
| --- | --- |
| Операция | Реализация логики в ProjectService |
| Создание проекта. При создании проекта должны быть заполнены все обязательные поля, и выполнена проверка на уникальность кода проекта. Проект создается в статусе Черновик. | При поступлении запроса в ProjectService выполняются проверки и вызывается метод save, который определяет операцию создания или обновления Project. При создании автоматически устанавливается статус DRAFT |
| Изменение проекта. При изменении проекта должны изменяться поля карточки проекта. | При поступлении запроса в ProjectService выполняются проверки и вызывается метод save, который определяет операцию создания или обновления Project. |
| Поиск проектов. Поиск должен осуществляться по текстовому значению (по полям Наименование проекта, Код проекта) и с применением фильтров по Статусу проекта. Т.е. на вход передается некоторое текстовое значение и список статусов. | Реализован поиск по условиям. Условие задается через DTO-объект ProjectFilter. Метод findAllByFilter. |
| Перевод в другое состояние проекта. Можно перевести в другой статус проект, согласно диаграмме ниже о возможных изменениях статуса. | Реализован метод изменения статуса. Изменение возможно только на следующий статус по цепочке, нельзя выставить предыдущий статус или перескочить через один. Метод updateStatus. |

**Команда проекта (Team)**

**Требования:**

Это группа сотрудников объединенных общим проектом.

В разных командах один сотрудник может принимать различные роли, но только одну роль внутри проекта.

Т.е. после создания проекта, можно наполнить проект сотрудниками, которые в нем участвуют и выполняют определенную роль.

Список ролей фиксированный: Руководитель проекта, Аналитик, Разработчик, Тестировщик.

Итого команда проекта состоит из списка связей: проект - сотрудник - роль в проекте.

**Реализация:**

Команда представлена классом Team. Участник команды представлен отдельным классом Teammember, который имеет ссылку на определенного Employee, команду и роль этого Employee в команде. Роль вынесена в отдельный Enum TeammemberProjectRole.

Основные операции:

|  |  |
| --- | --- |
| Операция | Реализация логики в TeamService и TeamMemberService |
| Добавить участника проекта. Т.е. добавить сотрудника в команду проекта с определенной ролью. | TeamService метод save позволяет добавлять и изменять команды. TeamMemberService метод save позволяет добавлять и изменять участников команды. При этом выполняется ряд проверок. Для команды - не назначена ли уже команда на проект для которого создается или меняется команда. Для участника - не состоит ли он уже в этой команде. |
| Удалить участника проекта. Т.е. исключить из команды по проекту сотрудника. | TeamMemberService метод deleteById удаляет участника. |
| Получить всех участников проекта. | TeamMemberService методы findAllByTeamId и findAllByProjectId ищут всехх сотрудников по указанному id команды или проекта |

**Задача (Task)**

**Требования:**

Задача назначает на сотрудника внутри проекта. Структура задач линейная (никаких деревьев и т.п.).

Задача обладает следующим набором атрибутов:

* Наименование задачи - текстовое значение, отражающее краткую информацию о задачи (обязательное поле).
* Описание задачи - текстовое значение, содержащее детальное описание задачи. (не обязательное поле)
* Исполнитель задачи - сотрудник, которому необходимо исполнить задачу. (не обязательное поле). Можно выбрать исполнителя только участника проекта ( сотрудник добавленный в команду проекта). Назначить исполнителя можно только сотрудника в статусе Активный.
* Трудозатраты - оценка, сколько в часах необходимо на ее исполнение. (обязательное поле)
* Крайний срок - дата, когда задача должна быть исполнена. Нельзя выбрать дату если дата меньше, чем дата создания + трудозатраты. Обязательное поле.
* Статус задачи - фиксированный список состояний задачи (Новая, В работе, Выполнена, Закрыта).
* Автор задачи - заполняется автоматически, тем кто создавал задачу. Автором задачи может являться только участник проекта.
* Дата создания - дата когда задача была создана.
* Дата последнего изменения задачи - дата последнего редактирования задачи (но не изменение статуса задачи).

**Реализация:**

Задача представлена классом Task. Статус задачи вынесен в отдельный Enum TaskStatus.

|  |  |
| --- | --- |
| Операция | Реализация логики в TaskService |
| Создание задачи. При создании задачи должны заполняться все обязательные поля, и проходить соответствующие проверки. Также задача автоматически создается в статусе Новая. | При поступлении запроса в TaskService выполняются проверки и вызывается метод save, который определяет операцию создания или обновления Task. При создании автоматически устанавливается статус New. Автоматически задается дата создания и автором устанавливается текущий авторизованный пользователь. |
| Изменение задачи. Могут редактироваться поля задачи, но не может меняться поле статус и генерируемые значения. Автор задачи в таком случае должен быть изменен. | При поступлении запроса в TaskService выполняются проверки и вызывается метод save, который определяет операцию создания или обновления Task. При изменении невозможно изменить статус и дату создания. Автоматически автором устанавливается текущий авторизованный пользователь. |
| Поиск задач - задачи должны искать по текстовому значению (по полям Наименование задачи) и с применением фильтров (по статусам задачи, по исполнителю, по автору задачи, по периоду крайнего срока задачи, по периоду создания задачи). Фильтры все не обязательны, как и текстовое поле. Результат должен быть отсортирован по дате создания задачи в обратном порядке (сначала свежие задачи). | Реализован поиск по условиям. Условие задается через DTO-объект TaskFilter. Метод findAllByFilter. Результат сортируется по дате создания от новых к старым. В случае если все условия фильтра будут пусты, то будут найдены все задачи. |
| Изменение статуса задачи. При этом действии задача переводится в новый статус, который поступил на вход. Переход по статусам возможен согласно диаграмме ниже. | Реализован метод изменения статуса. Изменение возможно только на следующий статус по цепочке, нельзя выставить предыдущий статус или перескочить через один. Метод updateStatus. |

**Аутентификация в системе.**

**Требования:**

В системе может быть аутентифицирован сотрудник только со статусом Активный.

**Реализация:**

Создана сущность Account которая имеет основные поля Username и Password на основе которых происходит регистрация и аутентификация в системе. Т.к. указано требование, что может быть аутентифицирован сотрудник только со статусом Активный, то для Account автоматически создается Employee (для этого при регистрации помимо Username и Password необходимо передать Firstname и Lastname, т.к. они обязательны для Employee). Если Employee, связанный с Account будет переведен в статус Deleted, то аутентификация станет невозможна.

Username и Password для Account хранятся в БД, Password хранится в зашифрованном виде (BCrypt).

Также подключена Swagger аутентификация (Http Basic) для тестирования endpoint’ов.

**Технические требования.**

|  |  |
| --- | --- |
| Проект должен быть реализован на JDK 18 | При разработке использована OpenJDK 18 |
| Проект должен собираться с использованием maven | Реализована сборка проекта с помощью maven и spring-boot-maven-plugin в единый jar-файл |
| Проект должен быть покрыт логами | Использован log4j2, добавлено логирование операций создания, изменения и удаления основных сущностей - account, employee, project, task. А также при обработке кастомных исключений, отправке почты и загрузке файлов. |
| API должно быть документировано с использованием OpenAPI | Создана документация с помощью OpenApi v3, /swagger-ui/index.html#/ |
| Основные функции должны быть покрыты тестами. | Написаны unit и интеграционные тесты для основных сервисов и контроллеров.  Для тестирования «руками»: в папке проекта, в папке docs/sql-test-scripts есть файл 03-test-insert.sql. С его помощью можно вставить данные в бд и дальше их тестировать |
| Проект должен запускаться в docker-compose. | Разработаны dockerfile и docker-compose. Проект запускается в compose c профилем “docker”. В директории проекта расположена папка docker-compose с необходимыми файлами. Для запуска проект также можно использовать скрипт build-image.sh в папке проекта, он запустит compose. Проброшен порт 8080, можно отправлять запросы на localhost.  Также можно запустить проект локально с профилем “default”, предварительно подняв docker контейнеры postgres и rabbitmq. В application properties заданы настройки для подключения к контейнерам |
| Структура БД должны инициализироваться с помощью liquibase. | Для инициализации таблиц используется liquibase yml changelog. Миграции находятся в resources в пакете repository |
| Проект должен быть разработан на основе Spring Boot. | Проект разработан на Spring Boot 3.0.4 |
| Должен быть разработан Docker файл. | Разработан dockerfile (в директории проекта) |
| Проект должен быть мультимодульным. | Проект использует многомодульность. |

**Дополнительные требования.**

|  |  |
| --- | --- |
| Разработать возможность прикрепления файлов к задаче и проекту. И возможность скачать файл по ссылке. | Разработан сервис FileService позволяющий выполнять загрузку и скачивание файлов. Реализован сервисы TaskFileService и TaskAttachmentService которые реализуют возможность прикрепления файлов к задаче.  TaskFileService реализует логику скачивания на основе базового сервиса FileService. TaskAttachmentService управляет сущностью TaskAttachment. Эта сущность хранит информацию о файле - имя, путь, ссылку на задачу, к которой относится файл.  В TaskController добавлены эндпойнты позволяющие добавить файл, скачать файл, получить список файлов прикрепленных к задаче или удалить прикрепленный файл.  При удалении задачи удаляются также все файлы, которые были прикреллены к ней |
| Разработать уведомления на почту при появлении задачи назначенной на данного сотрудника. Реализовать можно через отправку напрямую по SMTP. Но будет плюсом если отправка будет реализована асинхронно через RabbitMQ. | Разработан EmailService который реализуют функции отправки почты асинхронно через RabbitMQ. При создании новой задачи отправляется сообщение с этой задачей. EmailService содержит слушатель который получает сообщение и осуществляет отправку письма исполнителю указанному в задаче. |
| Сделать связанные задачи. Т.е. задача может зависеть от несколько других задач. Т.е. добавить функции линковки задачи к другой задачи. | Добавлены связи между задачами. Реализована следующая логика если задача ссылается на какие-либо задачи то ее нельзя перевести в статус CLOSED пока задачи на которые она ссылается не будут переведены в этот статус. Другими словами нельзя завершить задачу пока не завершены задачи от которых она зависит. Нельзя сделать ссылку на задачу, которая уже ссылается на исходную задачу |