Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук
Кафедра программной инженерии

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Руководитель ООП
д-р физ.-мат. наук, профессор
_____ О.А. Змеев
«_30_» _05_ 2019 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

РАЗРАБОТКА ПРОГРАММНОЙ СИСТЕМЫ ДЛЯ ОЦЕНКИ ПЕРСОНАЛА ПО МЕТОДУ «360 ГРАДУСОВ»

по основной образовательной программе подготовки магистров
«Управление проектами по разработке программного обеспечения»
направление подготовки
02.04.02 Фундаментальная информатика и информационные технологии

Ильченко Егор Ильич

Научный руководитель ВКР
д-р физ.-мат. наук, доцент
_____ А. Н. Моисеев
«_23_» _мая_ 2019 г.

Автор работы
студент группы № 931709
_____ Е. И. Ильченко

Томск – 2019

Реферат

Магистерская диссертация 66 с., 41 рис., 2 приложения, 15 источников.

ОЦЕНКА 360 ГРАДУСОВ, SCALA, PLAY, ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ, АНАЛИЗ ТРЕБОВАНИЙ

Цель работы — разработать программную систему для оценки персонала по методу «360 градусов».

Результат работы — разработана и внедрена система для оценки персонала по методу «360 градусов»

Содержание

Введение

В сознании большинства людей каждый человек обладает некоторым набором качеств, которые и определяют его индивидуальность. Сейчас распространен другой подход: здесь более важным считается не обладание качеством, а профессиональная эффективность. Важно, чтобы, помимо обладания качествами, способностями, умениями и навыками, человек умел направить их на решение задач, стоящих перед компанией. Именно в этом контексте используются все современные способы оценки персонала.

Оценить навыки можно множеством разных способов: тестовыми методиками, интервью, деловыми играми, наблюдением за работой человека в течение нескольких рабочих дней.

Помимо этого существует метод 360 градусов, который был предложен Питером Уордом в 1987 году. Первое определение, которое он дал этому методу: «Оценка 360 градусов» — это систематический сбор информации о работе индивидуума (или группы), получаемой от некоторого числа лиц, заинтересованных в его работе, и обратная связь по ней.

Изначально процесс оценки был построен на бумажных анкетах, составляемых вручную и раздаваемых сотрудникам, но в настоящее время существует более удобный и быстрый способ проведения оценки, с помощью автоматизированных компьютерных систем.

Этот способ пользуется популярностью в настоящее время, более 90% компаний из Fortune Global 500[1] пользуются этим методом. Но его используют не только крупнейшие компании, фирмы малого и среднего размера также заинтересованы в данном способе оценивания.

Одна из местных фирм решила внедрить эту методику в свой рабочий процесс. Существующие программные решения не подошли либо из-за

[1]Fortune 500 — рейтинг 500 крупнейших мировых компаний, критерием составления которого служит выручка компании

отсутствия необходимого функционала, либо из-за высокой стоимости проведения исследования. В связи с этим было решено разрабатывать свою систему оценки, которая должна полностью удовлетворить все нужды и быть достаточно гибкой, для того чтобы её можно было использовать и в других организациях.

В связи с этим можно сформулировать цель работы — разработать программную систему для оценки персонала по методу «360 градусов». Из цели вытекают следующие задачи:

- составить диаграмму предметной области;
- проанализировать варианты использования;
- спроектировать систему;
- реализовать систему;
- подготовить документацию;
- настроить непрерывную интеграцию и доставку.

Глоссарий

Компетенция — умение, качество или способность человека, существенно влияющее на его эффективность в работе.

Пользователь — это участник системы, которому доступны события оценки, способный оценивать других пользователей в рамках события оценки, и являющийся объектом оценки для других.

Администратор — это пользователь, наделенный правами для управления системой.

Группа — это логическое объединение пользователей по какому-либо признаку.

Форма — это список вопросов.

Событие оценки — это множество проектов, назначенных для оценки на определенный период времени, в котором участники (пользователи) дают ответы на вопросы о сотрудниках и в общем о событиях в компании.

Проект — это совокупность объединений типа «оценивающая группа — форма вопросов — оцениваемая группа». Каждое такое объединение — это отношение проекта.

Отношение проекта — это смысловая основа события оценки. Оно определяет, кто кого по какой форме вопросов оценивает.

1 Определение и фиксация требований

Данная глава посвящена сбору и формализации требований. Так как разрабатываемая система имеет конкретного заказчика, то работы по сбору требований велись именно с ним.

Представленные ниже требования разбиты на функциональные и нефункциональные. Изначально требования представляются в том виде, в котором их указал заказчик. Далее же требования были обработаны, переформулированы и формализованы в виде диаграмм и сценариев вариантов использования.

1.1 Нефункциональные требования

- Реализовать систему в виде веб приложения.
- Реализовать серверную часть системы с использованием языка программирования Scala.
- Построить приложение с использованием Play Framework в качестве каркаса.
- Использовать PostgreSQL в качестве СУБД.
- Для аутентификации использовать учетную запись Google с помощью OAuth2[2].
- Производить развертывание используя Docker[3].
- Иметь документацию API.
- Покрыть систему модульными тестами.
- Реализовать систему в общем виде, без привязки к конкретным организационным ограничениям, для возможности внедрения в других компаниях.

[2]OAuth — открытый протокол авторизации

[3]Docker — программное обеспечение для автоматизации развёртывания и управления приложениями

1.2 Функциональные требования

В данном разделе представлены функциональные требования к разрабатываемой системе, разбитые по актерам.

Первым делом рассмотрим функциональные возможности актера Пользователь. Пользователь — это участник системы являющийся объектом оценивания, либо выполняющий оценку других пользователей в качестве субъекта. Система должна поддерживать следующий функционал для него:

- возможность внести информацию о себе;
- возможность прикрепить своё изображение;
- возможность просмотреть список событий оценки, прошедшие, будущие и текущие, в которых Пользователь является субъектом оценивания;
- возможность произвести оценку коллег по предложенным формам в текущем событии;
- возможность посмотреть свои ответы в прошедших событиях;
- возможность увидеть список всех пользователей в системе.

Следующий актер — это Администратор. Администратор — это пользователь, наделенный дополнительными правами для управления сущностями в системе. Функционал выглядит следующим образом:

- возможность пригласить Пользователя в систему;
- возможность читать, создавать, обновлять и удалять следующие типы объектов:
    - пользователь;
    - группа;
    - проект;
    - форма;
    - прочее.

Следующий актер взаимодействующий с системой — это Аудитор. Аудитор — это лицо, получающее доступ к результатам оценки, для их анализа и совершения необходимых организационных действий. Возможности аудитора:

- просмотреть результат оценки в агрегированном виде;
- просмотреть результат оценки в детальном виде.

Для этого результаты оценивания должны экспортироваться в формате электронной таблицы и быть загружены на Google Drive. Доступ к этим файлам для аудитора должен быть предоставлен автоматически.

1.3 Формализация и анализ требований

После того, как требования, предоставленные заказчиком, были собраны и обработаны, была проведена работа по их формализации, приведению к стандарту представления — вариантам использования.

На рисунке 1.3.1 представлена диаграмма вариантов использования актёра пользователь. Основным вариантом использования является «Произвести оценку коллег».

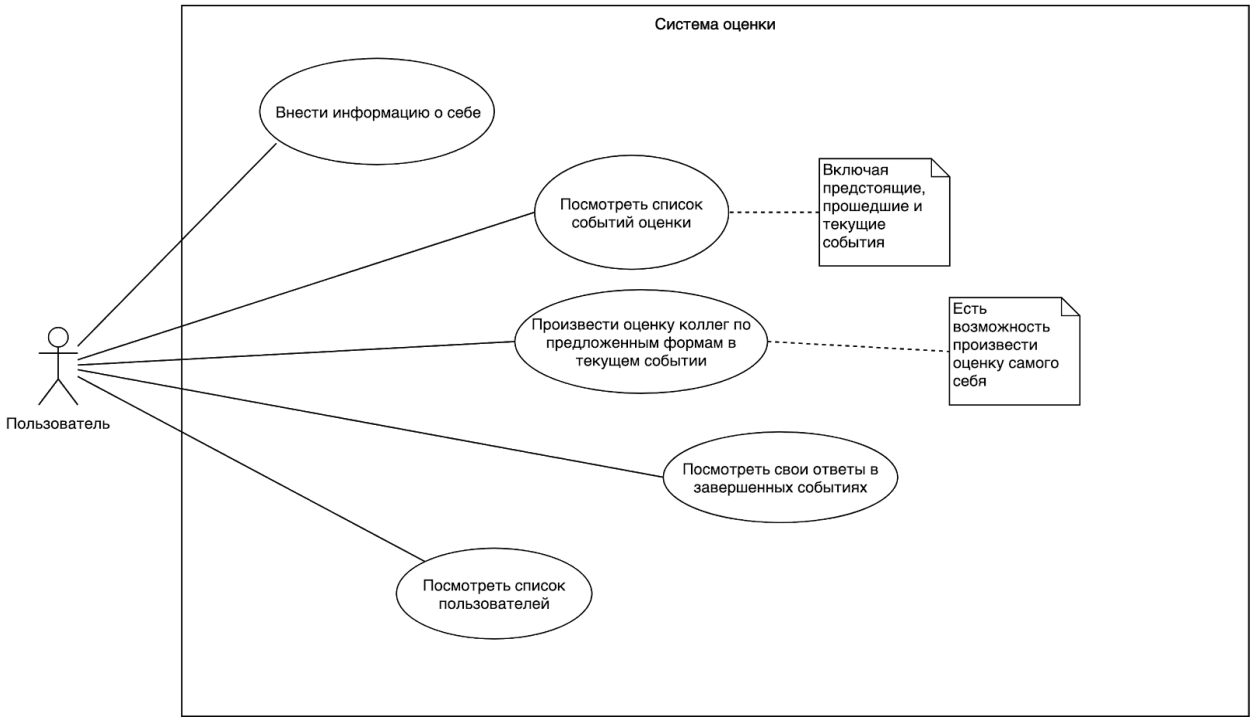


Рисунок 1.3.1 — Диаграмма вариантов использования актера Пользователь

Далее будут рассмотрены сценарии архитектурно значимых вариантов использования для актёра Пользователь в форме текстовой спецификации Алистера Кобёрна.

Таблица 1.3.1 — Описание и сценарий варианта использования «Внести информацию о себе»

Название ВИ	Внести информацию о себе
Цель	Заполнить профиль Пользователя информацией
Актер	Пользователь
Предусловия	1. Пользователь авторизован. 2. Пользователь находится на странице деталей своего профиля.
Сценарий	1. Пользователь нажимает на кнопку «Редактировать». 2. Система открывает страницу редактирования профиля. 3. Пользователь вносит изменения: a. имя; b. email; c. пол; d. часовой пояс. 4. Пользователь нажимает на кнопку «Загрузить изображение профиля» 5. Система открывает диалоговое окно для загрузки изображения с компьютера. 6. Пользователь выбирает новый файл и подтверждает действие. 7. Система закрывает диалоговое окно. 8. Пользователь нажимает на кнопку «Сохранить изменения» 9. Система выполняет верификацию данных. 10. Система сохраняет информацию в БД.
Расширения	8.1. Введенные пользователем данные не прошли валидацию. 8.1.1 Система отмечает, какие поля введены некорректно или были пропущены. 8.1.2 Пользователь исправляет введенные данные.

Таблица 1.3.2 — Описание и сценарий варианта использования «Произвести оценку коллег»

Название ВИ	Произвести оценку коллег
Цель	Отправить свою оценку в систему
Актер	Пользователь
Предусловия	1. Пользователь авторизован. 2. Пользователь состоит хотя бы в одном из событий оценки, которое находится в статусе «В процессе». 3. Пользователь находится в разделе «События оценки».
Сценарий	1. Пользователь нажал на кнопку «Начать оценку» напротив выбранного события оценки. 2. Пользователь выбирает пользователя в списке оцениваемых. 3. Пользователь выбирает вкладку с проектом. 4. Система открывает список пользователей требующих оценки. 5. Пользователь выбирает пользователя в списке оцениваемых. 6. Система открывает форму с вопросами. 7. Пользователь вводит все необходимые ответы. 8. Пользователь нажимает на кнопку «Сохранить и перейти к следующей форме». 9. Система открывает следующую форму для пользователя.
Расширения	7.1. Пользователь на заполнил одно из обязательных полей. 7.1.1. Система подсвечивает поле красным, кнопка «Сохранить и перейти к следующей форме» неактивна. 8.2. Пользователь находится на последней форме проекта. 8.2.1 Система открывает первую форму для первого пользователя в следующем проекте. 8.3 Пользователь находится на последней форме последнего проекта. 8.3.1 Система отображает кнопку «Сохранить и завершить оценку». 8.3.2 Пользователь нажимает на кнопку. 8.3.3 Система открывает раздел «События оценки»

Рассмотрим варианты использования актёра Администратор.

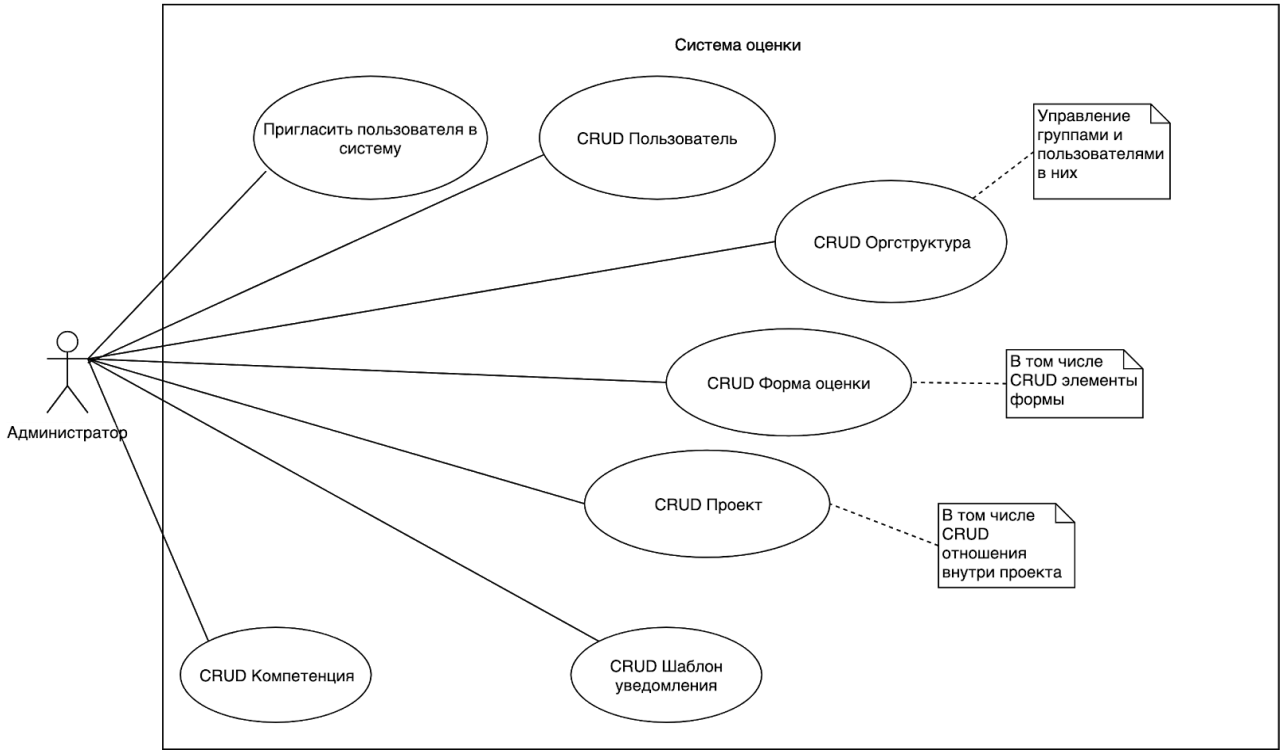


Рисунок 1.3.2 — Диаграмма вариантов использования актера Администратор

Администратор может пользоваться всеми вариантам использования Пользователя, но помимо этого имеет доступ к управлению объектами в системе. Диаграмма изображена на рисунке 1.3.2. В таблице 1.3.3 приведен сценарий варианта использования «Пригласить пользователя в систему».

Таблица 1.3.3 — Сценарий варианта использования «Пригласить Пользователя в систему»

Название ВИ	Пригласить пользователя в систему
Цель	Пригласить пользователя в систему
Актер	Администратор
Предусловия	1. Пользователь с ролью «Администратор» авторизован в системе. 2. Администратор находится на странице «Приглашения»
Сценарий	1. Администратор нажимает на кнопку «Пригласить пользователей». 2. Система открывает страницу «Отправить приглашения».

	3. Администратор заполняет поля
	  a. email;
	  b. группы.
	4. Администратор нажимает на кнопку «Отправить приглашения».
	5. Система отправляет приглашение на введенный адрес.
	6. Система открывает страницу «Приглашения»

Альтернативный сценарий 1	1. Администратор переходит на страницу деталей группы. 2. Администратор нажимает на кнопку «Пригласить Пользователей» в секции «Пользователи, состоящие в группе» 3. Система открывает страницу «Отправить приглашения» 4. Администратор заполняет поле email. 5. Администратор нажимает на кнопку «Отправить приглашения». 6. Система отправляет приглашение на введенный адрес. 7. Система открывает страницу деталей группы.

На рисунке 1.3.3 представлена диаграмма вариантов использования актёра Аудитор. Аудитор не взаимодействует с системой напрямую, просмотр отчета производится на стороннем сервисе, в который система должна предварительно загрузить результат.

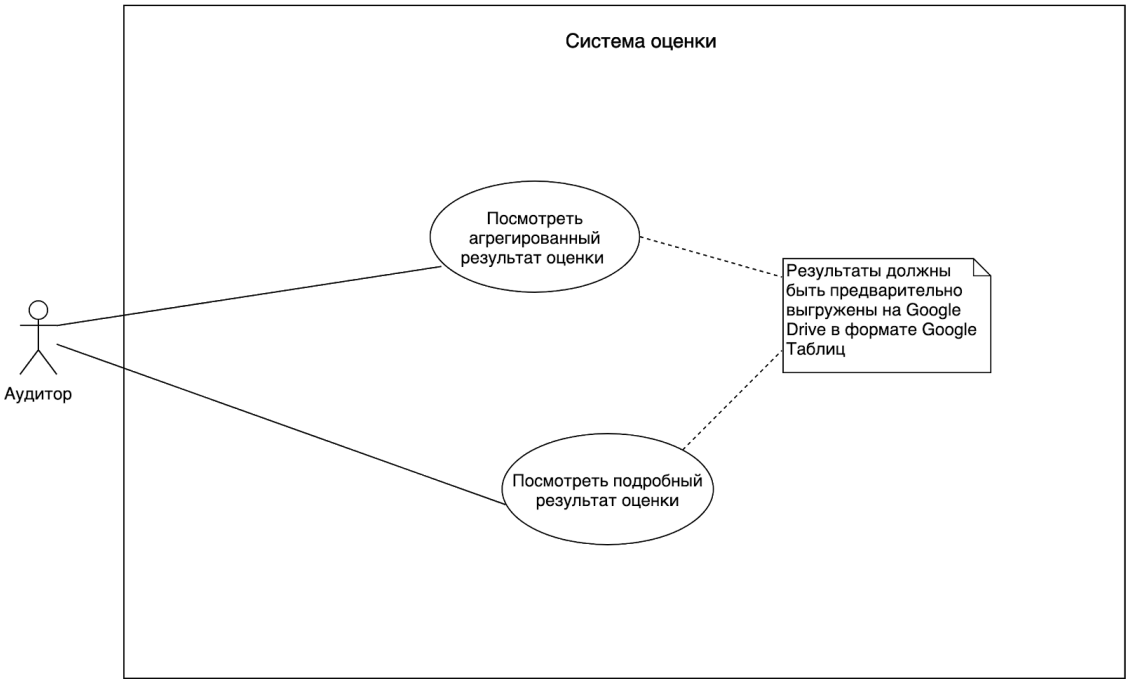


Рисунок 1.3.3 — Диаграмма вариантов использования актера Аудитор

2 Проектирование

2.1 Диаграмма предметной области

На основании построенных диаграмм вариантов использования была спроектирована и зафиксирована диаграмма предметной области в форме диаграммы классов. Данная диаграмма фиксирует предметную область разрабатываемой системы и не привязана к классам реализации.

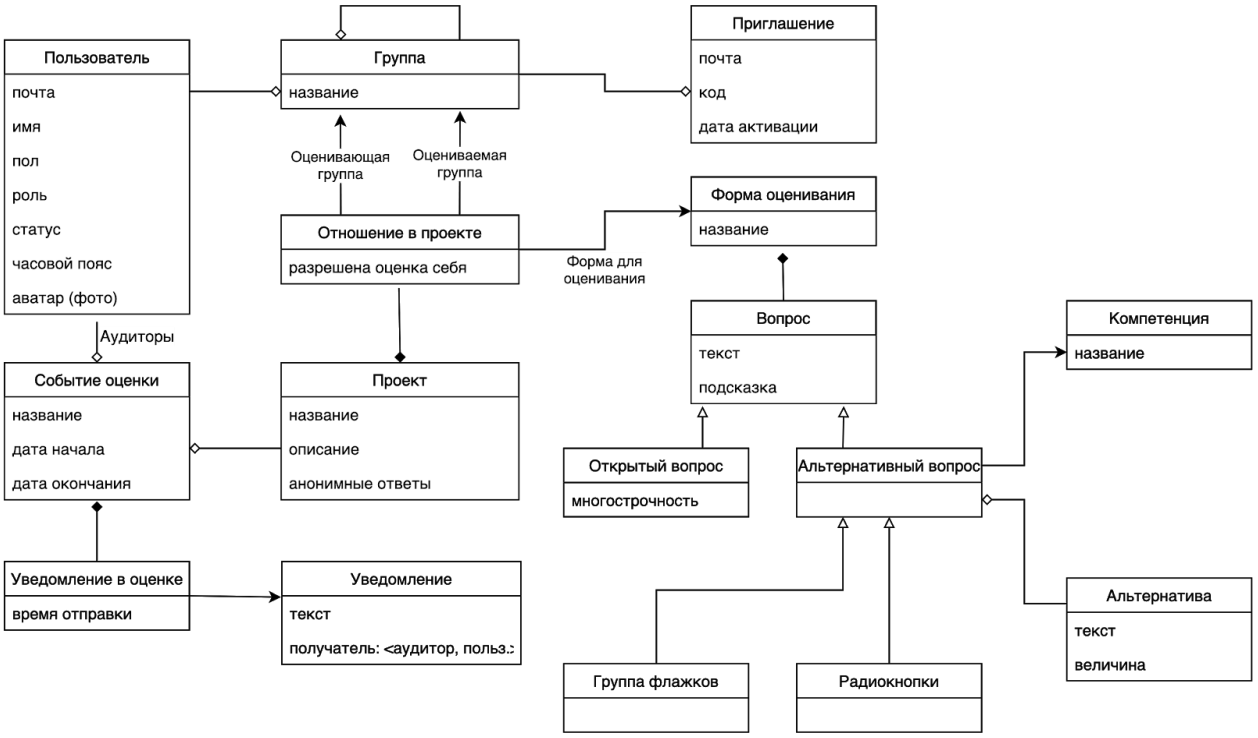


Рисунок 2.1.1 — Диаграмма классов модели предметной области

Рассмотрим рисунок 2.1.1 подробнее. Класс Пользователь хранит информацию о зарегистрированных в системе пользователях. Пользователи являются как субъектами, так и объектами оценивания. Класс имеет следующие атрибуты:

- почта — email, используемый для отправки уведомлений;
- имя — ФИО, отображаемое в процессе оценки и в отчетах;
- пол — мужской, женский;
- роль — Пользователь, Администратор;

- статус — Новый, Активный. Новых пользователей нельзя добавить в событие оценки, требуется предварительная активация;
- часовой пояс — используется для отображения дат и времени в локальном часовом поясе пользователя;
- аватар — идентификатор загруженного файла с изображением.

Пользователи могут быть включены в одну или несколько Групп. Группа — это логическое объединение пользователей по какому-либо признаку. Группы имеют название и могут вкладываться друг в друга, образуя древовидную иерархию. Событие оценки определяется названием и датой/времени начала и окончания. Пока Событие оценки активно, пользователи имеют возможность отвечать на вопросы о коллегах. В Событие оценки должны быть включены Проекты. Проект — это набор информации о том, по каким формам группы оценивают друг друга, и разрешена ли самооценка пользователя, в случае если он находится и в оцениваемой, и в оценивающей группах. Атрибут «анонимные ответы» у Проекта определяет, есть ли у пользователя в этом проекте возможность произвести оценку анонимно.

К событию оценки могут быть привязаны уведомления, отправляемые в определенное время, содержащие текст и получателя (Пользователь или Аудитор).

Форма — это набор вопросов. Она имеет название и состоит из различных типов вопросов (Открытый, Альтернативный). Каждый вопрос имеет текст и подсказку, используемую как пояснение для отвечающего. Открытый вопрос имеет атрибут «многострочность», используемый для определения внешнего вида текстового поля. Альтернативный вопрос может иметь вид Радиокнопок или Флажков. Помимо этого для Альтернативного вопроса можно задать Компетенцию и набор Альтернатив, каждая из которых содержит текст и числовое значение Компетенции. Это значение используется

для расчета среднего значения по каждой Компетенции при генерации отчета для пользователя.

Приглашение хранит в себе информацию о почте, на которую оно было отправлено, а также код, содержащийся в ссылке в письме, по которому регистрируемый пользователь будет идентифицирован. Каждое приглашение может быть связано с несколькими группами, пользователь, который активировал приглашение, будет добавлен в них автоматически. После активации текущая дата должна быть записана в атрибут «дата активации».

Для описания семантики сущностей на рисунке 2.1.2 приведена диаграмма объектов.

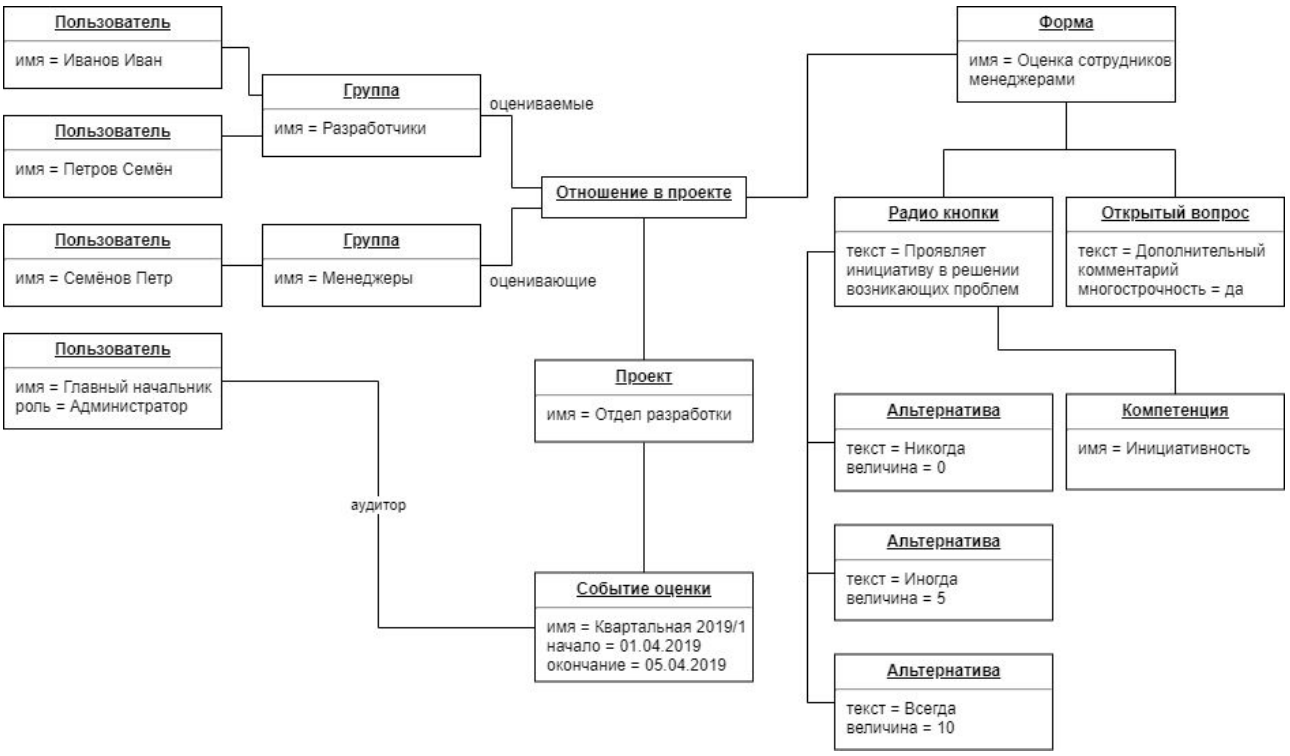


Рисунок 2.1.2 — Пример диаграммы объектов

На рисунке 2.1.2 представлены пользователи, трое из которых состоят в проекте под названием «Отдел разработки». Двое из них являются разработчиками, один — менеджер. Помимо этого в системе присутствует пользователь с ролью «Администратор», имеющие права на создание групп, проектов, событий оценки. Администратор создает создает событие оценки, в которое включает проект «Отдел разработки». В проекте заранее было создано отношение «Менеджеры» оценивают «Разработчиков» по форме «Оценка сотрудников менеджерами». Форма состоит из двух вопросов, первый подразумевает выбор одного варианта из предложенных, ответ на второй является простым текстом, вводимым в текстовое поле. К первому вопросу привязана компетенция «Инициативность», а, соответственно, к альтернативам — численные значения этой компетенции.

2.2 Контекст системы

Разрабатываемая программа должна взаимодействовать с двумя внешними системами:

- сервис e-mail рассылок — для отправки писем;
- Google Диск/Таблицы — для загрузки и хранения отчетов.

Согласно требованиям существуют три Актера:

- Пользователь;
- Администратор;
- Аудитор.

Для определения того, как актеры взаимодействуют с системой, и как разрабатываемый инструмент взаимодействует с внешними системами на рисунке 2.2.1 приведена диаграмма, демонстрирующая контекст.

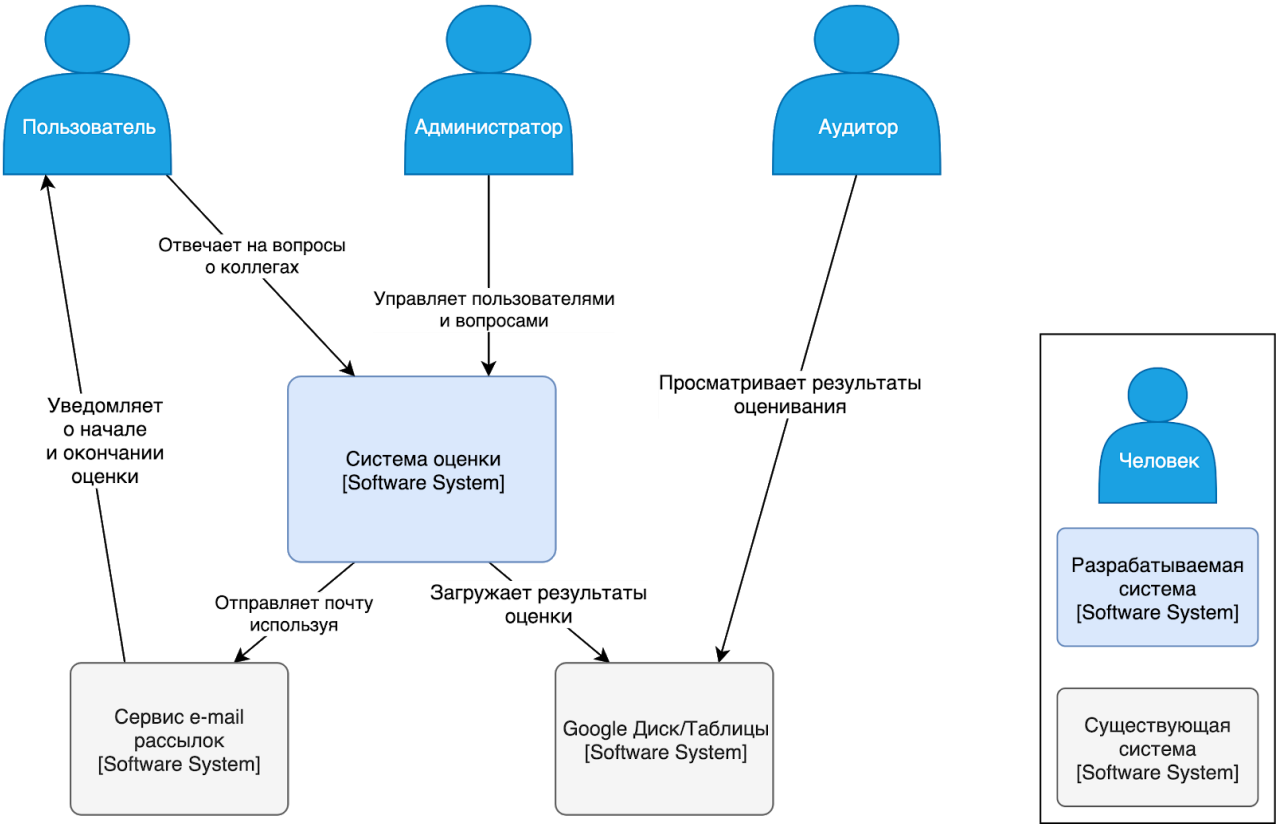


Рисунок 2.2.1 — Диаграмма контекста системы

2.3 Разбиение на подсистемы

Рассмотрим составляющие системы оценки (см. рисунок 2.2.1) подробнее. Для реализации была выбрана архитектура с одностраничным приложением, предоставляющим пользовательский интерфейс, API приложением содержащим в себе бизнес логику и имеющим REST[4] интерфейс, который используется одностраничным приложением. Помимо этого выделена отдельная подсистема, которая выполняет фоновые задачи при наступлении определенных событий. Для взаимодействия между приложениями используется шина сообщений, для сохранения состояния используется реляционная база данных. На рисунке 2.3.1 приведена диаграмма, показывающая выделенные подсистемы, вместе с языками и технологиями, применяемыми для реализации. Помимо этого там представлены виды взаимодействий между подсистемами, с указанием протокола. Контекст разрабатываемой системы выделен пунктирным прямоугольником. Одностраничное приложение разрабатывалось другой командой, и его реализация не рассматривается в данной работе.

[4]REST — архитектурный стиль взаимодействия компонентов распределенного приложения в сети

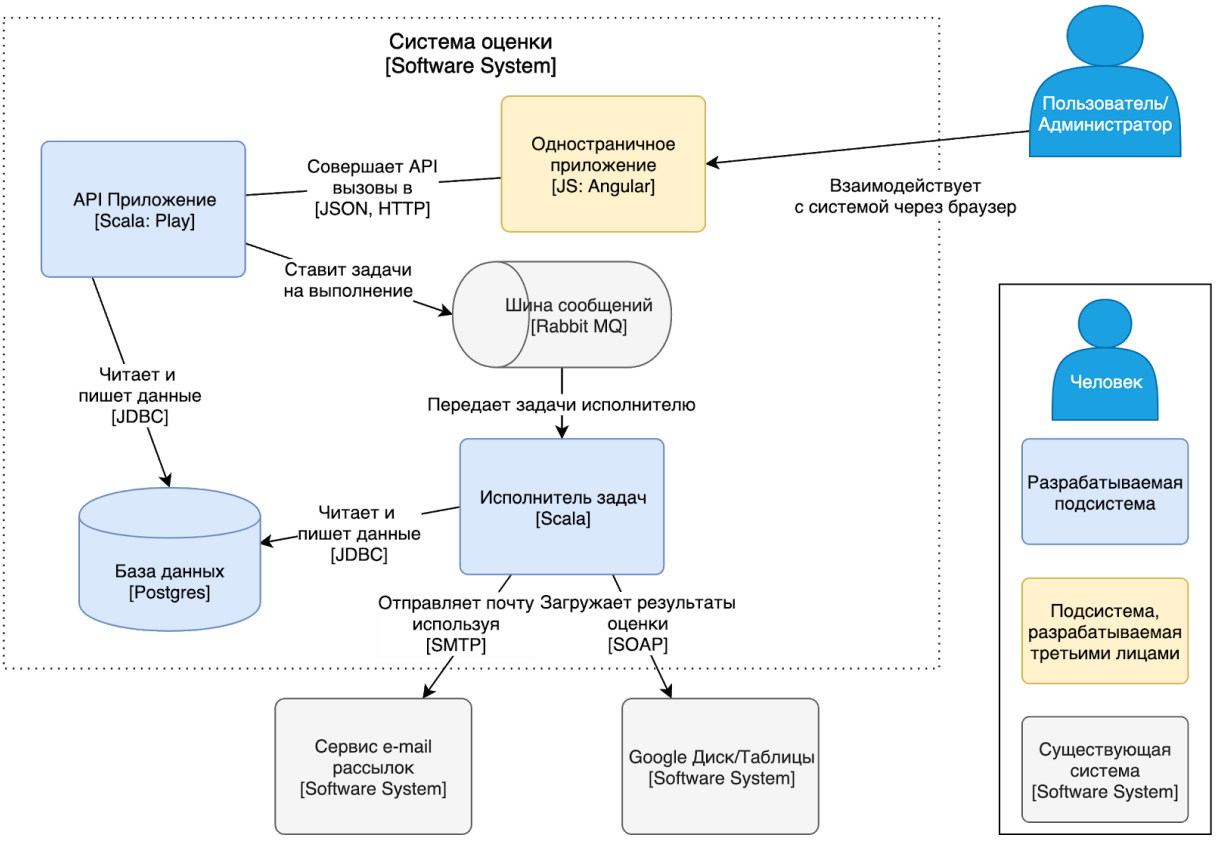


Рисунок 2.3.1 — Диаграмма подсистем

2.4 Схема база данных

Схема базы данных создавалась на основе модели предметной области. Использовались шаблоны проектирования «Single Table Inheritance» и «Association Table Mapping» для отображения классов в таблицы.

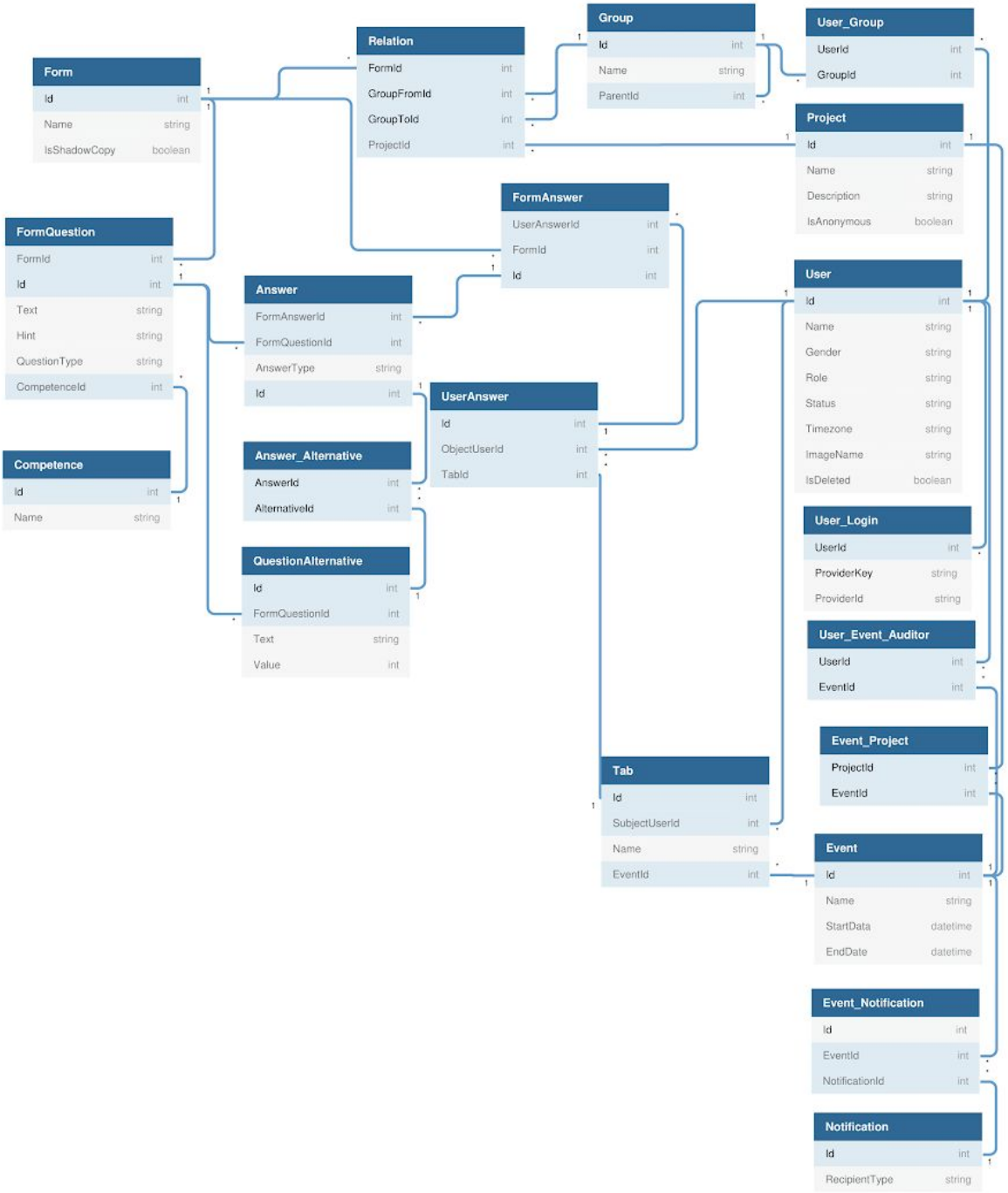


Рисунок 2.4.1 — Схема базы данных

2.5 API-Приложение

Перейдем к более подробному рассмотрению устройства API-Приложения. Для начала определимся с его областью ответственности и функционалом. API-Приложение должно:

- Предоставлять REST API для использования одностраничным приложением;
- Аутентифицировать и авторизовывать пользователей;
- Манипулировать данными в СУБД, соблюдая все ограничения;
- Ставить задачи для отложенного выполнения;
- Отправлять задачу на выполнение при наступлении определенного времени.

Диаграмма компонентов приведена на рисунке 2.5.1.

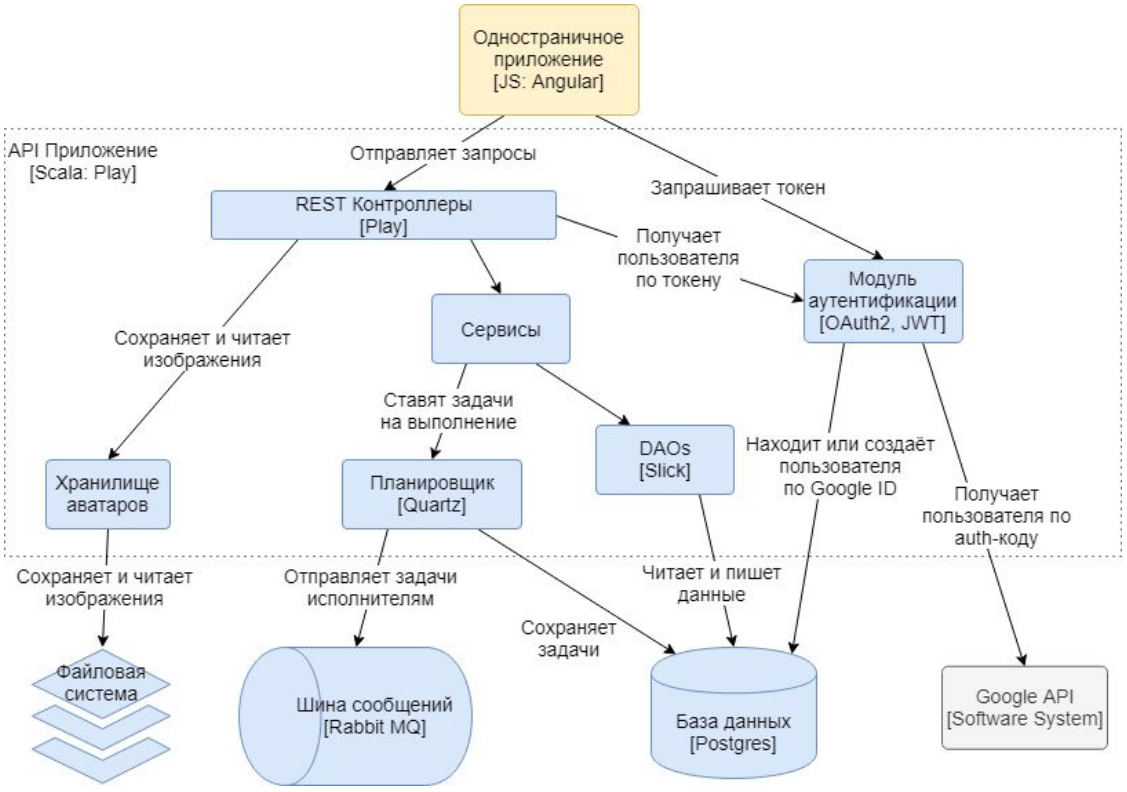


Рисунок 2.5.1 — Компоненты API-Приложения

REST Контроллеры представляют собой пакет с классами, содержащими публичные API методы, диаграмма классов которого представлена на рисунке 2.5.2.

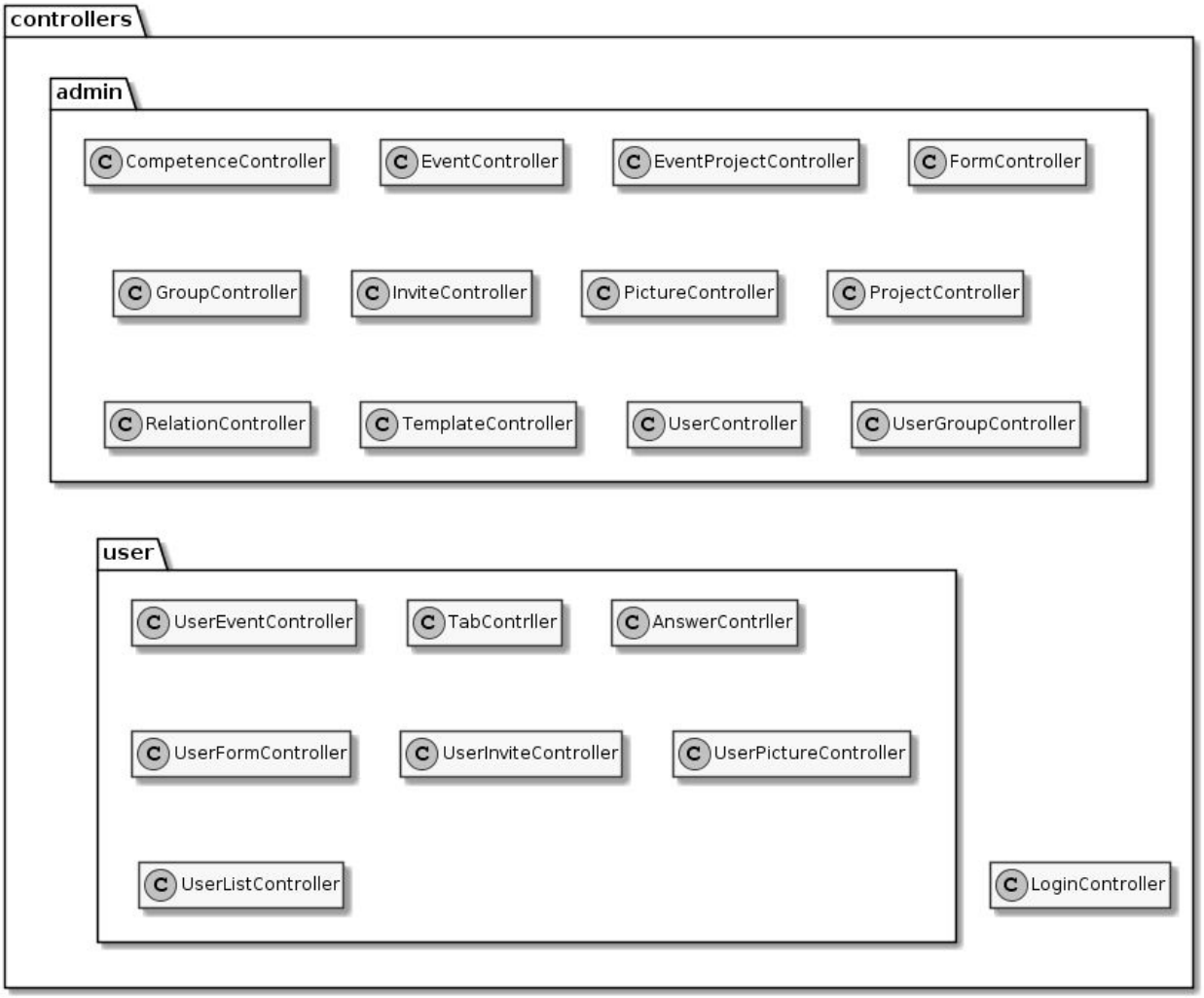


Рисунок 2.5.2 — Диаграмма классов REST контроллеров

Исходя из нефункциональных требований для реализации REST интерфейса был использован фреймворк для веб приложений Play. Play Framework — это каркас разработки с открытым кодом, написанный на Scala и Java, использует паттерн проектирования Model-View-Controller (MVC), подходит для создания как классических веб-сайтов, так и для создания REST API. Рассмотрим составляющие фреймворка, которые использовались в этой работе.

Маршрутизация. Play Framework поддерживает обработку входящих HTTP запросов и вызов методов контроллеров, в зависимости от пути в запросе. Соответствия между путём и методом контроллера описываются в специальном формате, пример приведен ниже.

```
PUT /admin/users/:id controllers.admin.UserController.update(id: Long)
```

Это означает, что PUT запрос по пути `/admin/users/:id` должен вызывать метод `update` у класса `UserController` лежащего в пакете `controllers.admin`

Сериализация и десериализация JSON. Каркас имеет встроенный функционал для автоматической конвертации объектов в JSON и конвертации JSON в объекты. Это необходимо потому что, в REST API для передачи состояния объектов используется именно формат JSON.

Внедрение зависимостей. Поддерживается внедрение зависимостей по стандарту JSR 330[5]. В качестве реализации используется библиотека Google Guice.

Продолжим разбор функционала, рассмотрим реализацию аутентификации и авторизации пользователей. Необходимым требованием к приложению является поддержка входа через учетную запись Google. Так как ввод логина и пароля напрямую на странице сервиса не соответствует принятым нормам безопасности, было решено использовать протокол OAuth2 для авторизации получения доступа к основной информации о Google аккаунте пользователя. Далее, с помощью вызова в Google API получается уникальный идентификатор пользователя, который и используется для аутентификации.

После успешной аутентификации API Приложение генерирует токен, который надо передавать с каждым последующим запросом. Для генерации токена был выбран стандарт JWT. Json Web Token — это открытый спецификаци для аутентификации в веб приложениях. Токены создаются сервером, подписываются секретным ключом и передаются клиенту, который в

[5] JSR-330 — спецификация внедрения зависимостей для языка Java

дальнейшем использует данный токен для подтверждения своей личности. Система включает в токен идентификатор пользователя Google, время жизни токена и подпись, сгенерированную с использованием секретного ключа. Это позволяет отказаться от хранилища сессий и от состояния, что, в дальнейшем, облегчит горизонтальное масштабирование путем увеличения количества запущенных приложений.

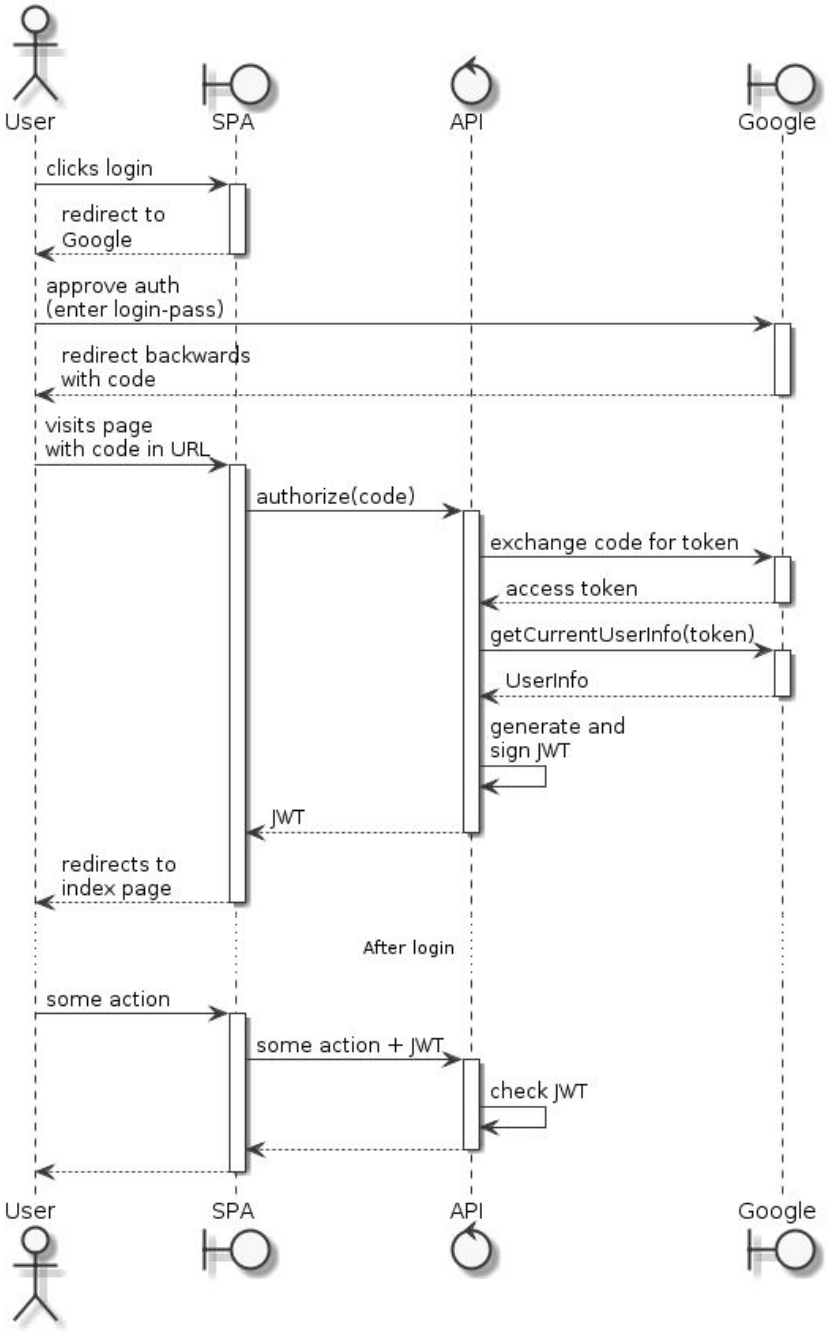


Рисунок 2.5.3 — Аутентификация пользователя

Авторизация в приложении реализована по ролям. Существует две роли — администратор и пользователь.

Для авторизации, аутентификации, генерации и проверки токена была использована библиотека Silhouette.

Следующий компонент подсистемы — это манипулирование данными в БД. Для этого был использован паттерн Data Access Object. На каждую сущность было создано по классу, ответственному за чтение, создание, обновление и удаления этой сущности в БД.

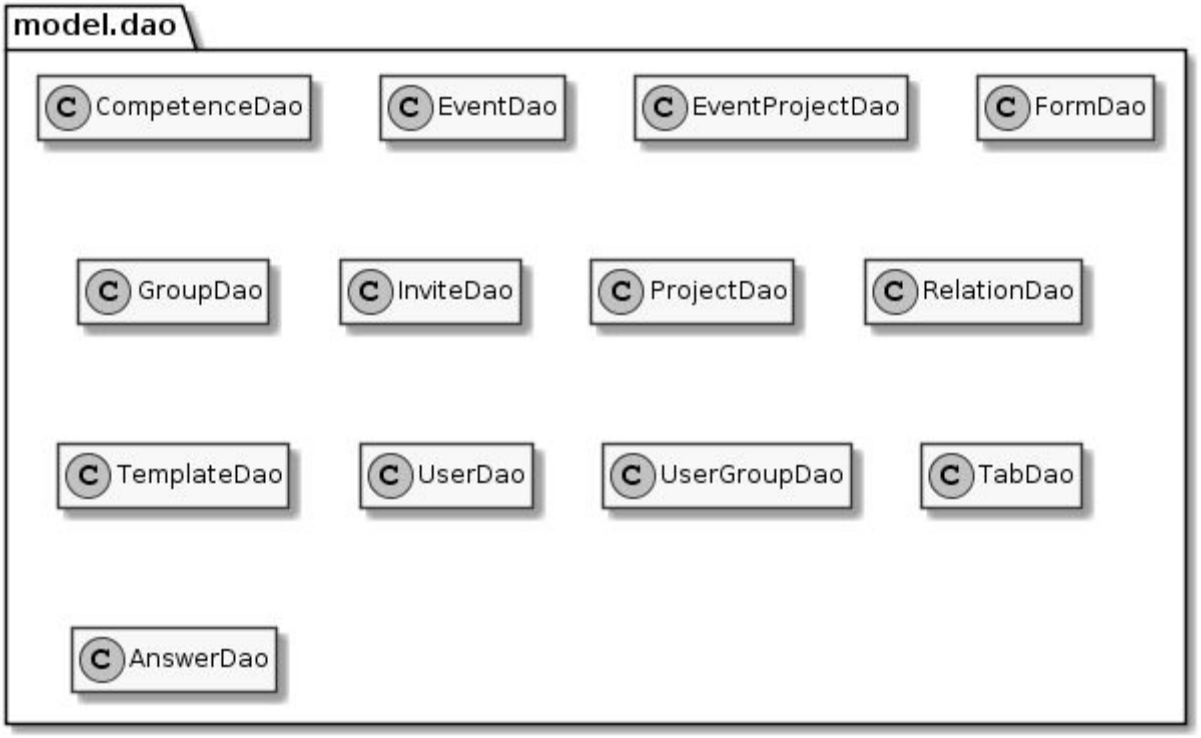


Рисунок 2.5.4 — Классы в пакете model.dao

Написание SQL запросов напрямую в коде не является подходящем решением, так как такие запросы не проверяются компилятором и могут стать причиной ошибки. Поэтому для написания запросов был использован Slick. Slick — библиотека для работы с БД для языка Scala, позволяющая писать запросы так, будто работа ведется с Scala коллекциями, расположенными в

памяти. Помимо этого Slick поддерживает отображение результата запроса в объекты.

Для использования Slick необходимо:

1. Указать параметры подключения в файле конфигурации
2. Объявить схему. Выглядит это следующим образом:

```
case class Competence(id: Long, name: String)

class CompetenceTable(tag: Tag) extends Table[Competence](tag, "competence") {
    def id = column[Long]("id", O.PrimaryKey, O.AutoInc)
    def name = column[String]("name")

    def * = (id, name) <> ((Competence.apply _).tupled, Competence.unapply)
  }
}
```

   Здесь объявляется класс `Competence` и отображение `CompetenceTable`, в котором содержится информация о названии таблицы и именах и типах столбцов.

3. Написать запрос.

   Запросы на вставку пишутся так:

```
val competence = ... // doesn't matter how object created
db.run(Competencies += competence)
```

   На получение данных так:

```
db.run(Competencies.filter(_.id === id).result.headOption)
```

   Подобные запросы автоматически преобразуются в SQL, и посылаются в СУБД с использованием JDBC[6]

Перейдем к планированию и выполнению отложенных задач. В системе существуют действия, которые необходимо выполнять в фоне в определенное время, например отправка писем-уведомлений, генерация отчетов после события оценки. Непосредственно выполнением этих задач занимается отдельная подсистема, которая будет рассмотрена отдельно. Сейчас же

[6]JDBC — платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД

разберемся в деталях того, как именно планируется выполнение задач, и как они вызываются в нужно время.

Для этого используется Quartz — библиотека с открытым исходным кодом, позволяющая создавать задания, которые необходимо выполнить в определенный момент времени.

Перед использованием необходимо сконфигурировать хранилище для заданий, в данном случае задания будут храниться в основной БД. Все задания должны реализовывать интерфейс Job. Также в каждое задание передается контекст, содержащий в себе идентификатор события оценки и прочую необходимую информацию.

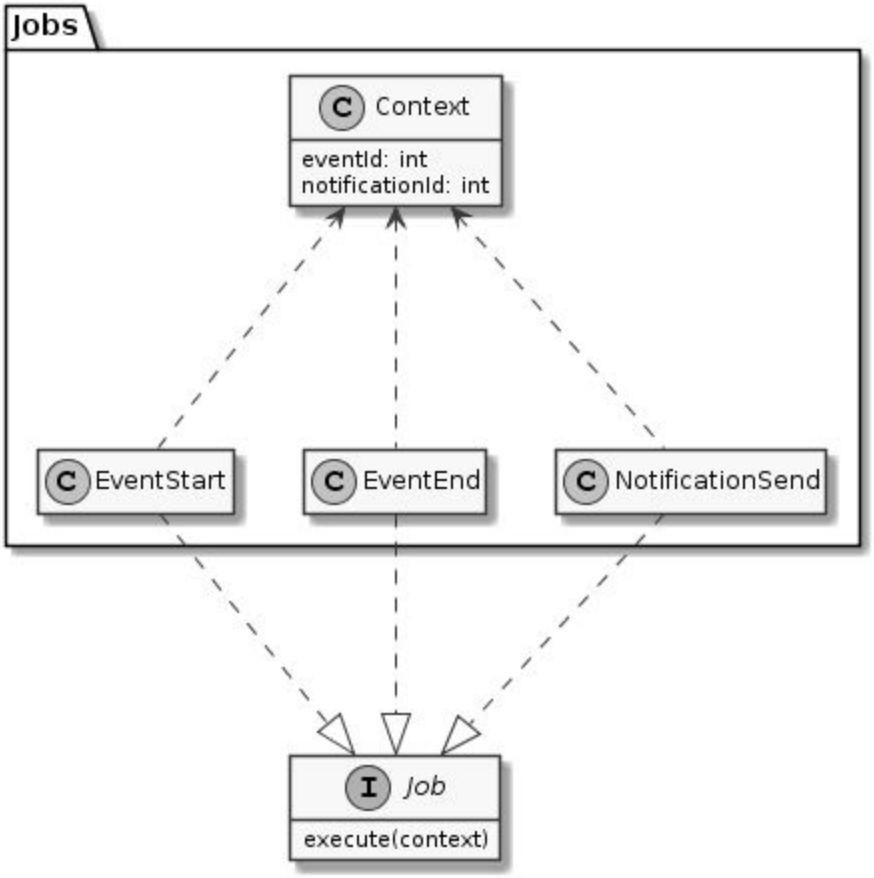


Рисунок 2.5.5 — Диаграмма классов задач

При срабатывании задача отправляет сообщение в очередь, которое будет прочитано Executor`ом и выполнено.

2.6 Исполнитель задач

Для функционирования системы необходимо выполнять ряд фоновых задач, таких как подготовка события, отправка уведомлений, формирование отчета. Так как выполнение задач — это фоновый процесс, которые может занимать некоторое количество времени, было решено выделить их выполнение в отдельное приложение. Это позволяет системе быть более гибкой, устойчивой к перезапуску и отказам, позволяет, в случае надобности, выполнять горизонтальное масштабирование, путем увеличения количества запущенных исполнителей.

Основные компоненты приведены ниже:

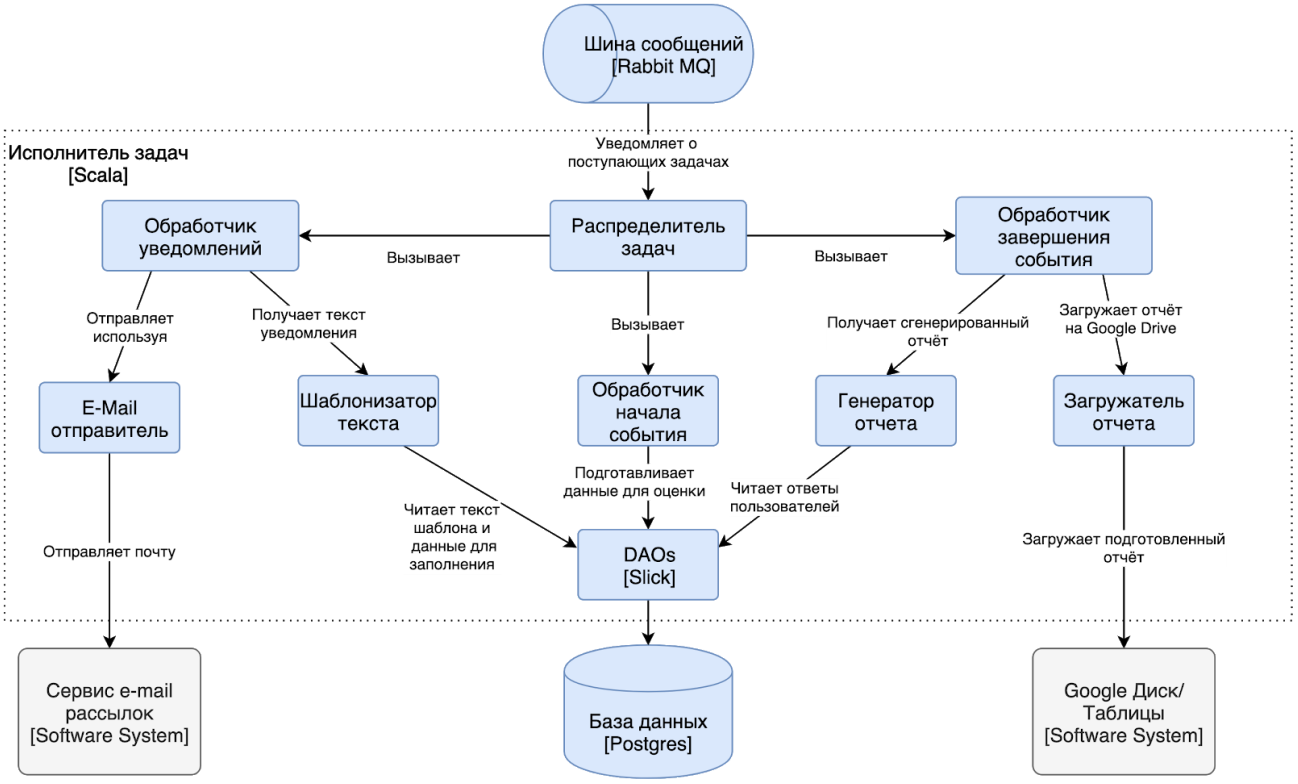


Рисунок 2.6.1 — Компоненты исполнителя задач

Приложение получает команды на выполнение через RabbitMQ. RabbitMQ — это программный брокер сообщений на основе стандарта AMQP[7]. Создан на основе системы Open Telecom Platform, написан на языке Erlang, в

[7]AMQP — (Advanced Message Queuing Protocol) открытый протокол для передачи сообщений между компонентами системы

качестве системы управления базой данных для хранения сообщений использует Mnesia. Использование RabbitMQ в проекте даёт следующие преимущества:

- В случае некорректного завершения работы сервера, данные в очереди не теряются. При последующем запуске обработка продолжается с того места, где был обрыв;
- Если результат обработки не удовлетворяет, задачу можно послать в очередь повторно, например, при сбое или ошибке выполнения;
- Количество хранимых в очереди сообщений не ограничено;
- API Приложение и Исполнитель задач могут быть развернуты на различных серверах;
- В случае, если один запущенный экземпляр приложения не справляется с нагрузкой, есть возможность запустить несколько экземпляров. Задачи будут равномерно распределяться между ними.

В RabbitMQ, а также обмене сообщениями в целом, используется следующая терминология:

- Producer (поставщик) — программа, отправляющая сообщения;
- Exchange (точка обмена) — получает сообщения от поставщика и отправляет эти сообщения в очередь;
- Queue (очередь) — буфер, хранящий сообщение;
- Consumer (подписчик) — программа, принимающая сообщения из очереди.

Поставщик никогда не отправляет сообщения напрямую в очередь. Фактически, довольно часто поставщик не знает, дошло ли его сообщение до конкретной очереди. Вместо этого поставщик отправляет сообщение в точку обмена. Точки обмена бывают нескольких типов, в данном случае используется direct — все присылаемые сообщения перенаправляются в одну очередь, определяемую значением routing key у сообщения. Routing key есть у каждого

сообщения, он задается поставщиком. Итоговая схема представлена на рисунке 2.6.2

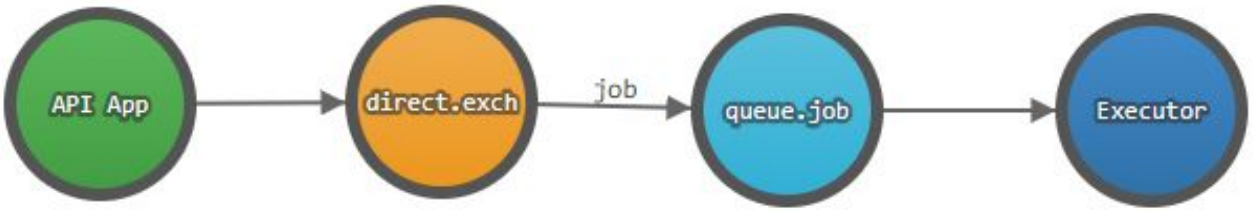


Рисунок 2.6.2 — Конфигурация обменников и очередей

В случае, если Исполнитель по какой-то причине не может выполнить задачу (например, отсутствует доступ к интернету) задача возвращается обратно в очередь, и может быть прочитана снова, после перезапуска Исполнителя. Также задача автоматически возвращается в очередь если приложение завершилось во время выполнения. И чтобы мы могли быть уверены в отсутствии потерянных сообщений, RabbitMQ поддерживает подтверждение сообщений. Подтверждение (ack) отправляется подписчиком для информирования RabbitMQ о том, что полученное сообщение было обработано и RabbitMQ может его удалить. Если подписчик прекратил работу и не отправил подтверждение, RabbitMQ поймет, что сообщение не было обработано.

Для обработки сообщений отсутствует тайм-аут. RabbitMQ вернет сообщение обратно в очередь только если соединение с подписчиком будет закрыто, поэтому нет никаких ограничений на время обработки сообщения. Помимо этого подписчик может вернуть сообщение в очередь, отправив в RabbitMQ отказ (reject).

Далее рассмотрим основные компоненты Исполнителя. Первым делом задачи на выполнение поступают в распределитель. Задачи передаются в очередь в сериализованном виде, и распределитель должен произвести десериализацию, а затем передать ее подходящему обработчику. Для того, чтобы распределителю не нужно было знать об обработчиках и типах

сообщений, которые они поддерживают, был применен подход, представленный на рисунке 2.6.3. Его суть заключается в следующем: каждый обработчик должен реализовывать интерфейс `JobHandler`, имеющий два метода `supports` и `handle`. Метод `supports` должен возвращать истину в том случае, если обработчик поддерживает задачу. Метод `handle` непосредственно выполняет действия, необходимые для обработки сообщения конкретного типа.

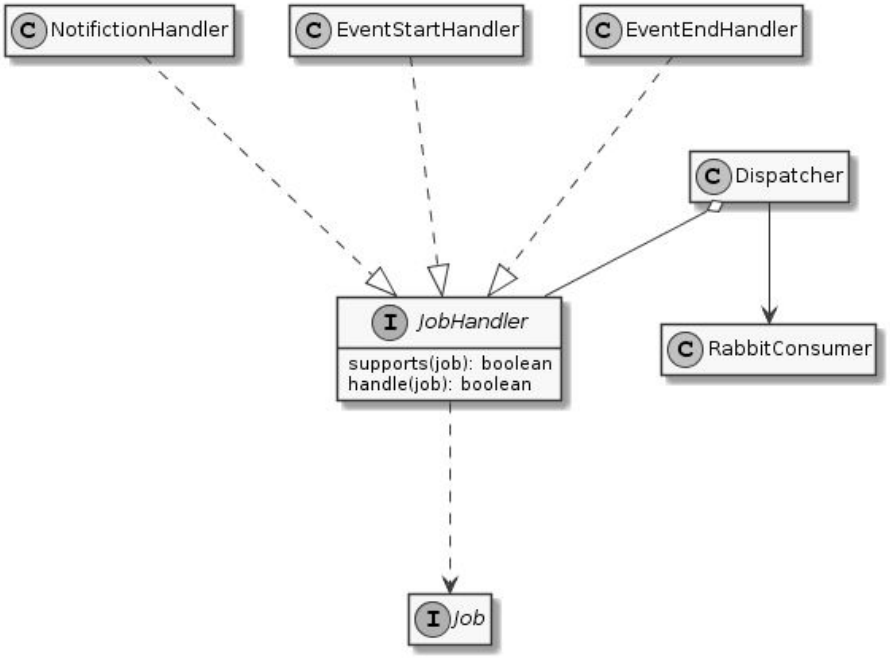


Рисунок 2.6.3 — Диаграмма классов исполнителей задач

На рисунке 2.6.4 приведена диаграмма последовательности обработки поступающих сообщений. При создании экземпляра класса `Dispatcher` происходит регистрация функции обратного вызовы (callback) в `RabbitConsumer`, которая является входной точкой для обработки сообщений. `Dispatcher` содержит ссылку на коллекцию экземпляров `JobHandler`. При поступлении сообщения для каждого из них производится проверка, поддерживает ли обработчик сообщение. Если такой обработчик найден, то сообщение передается ему, иначе сообщение отправляется обратно в очередь. Также сообщение будет возвращено в очередь в случае возникновения исключения (exception).

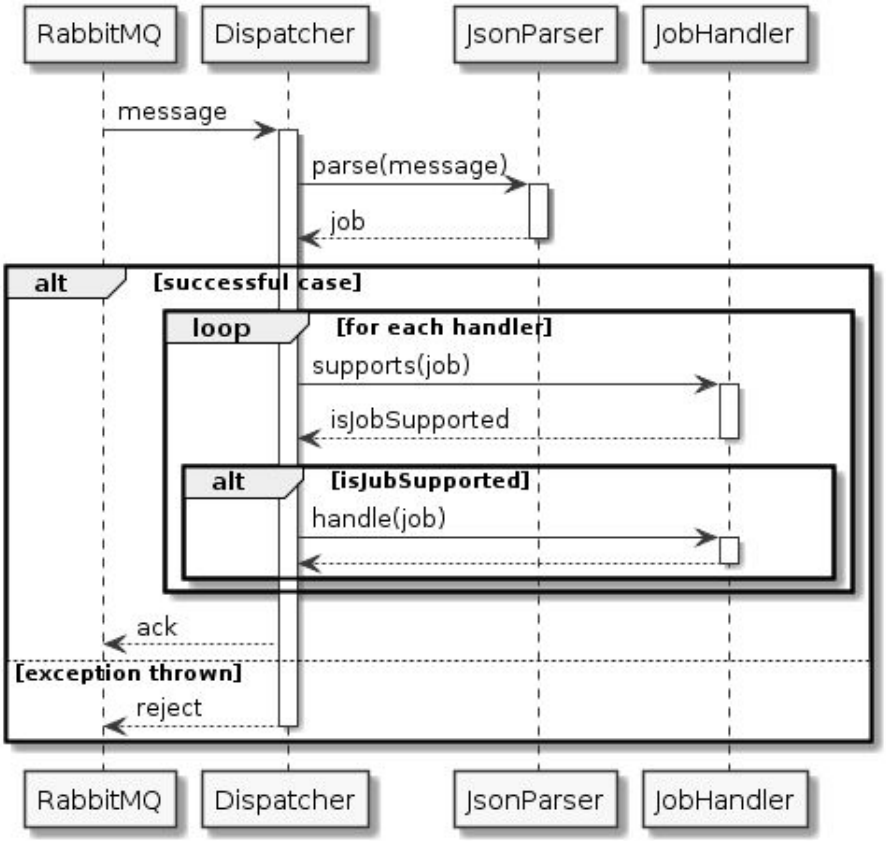


Рисунок 2.6.4 — Последовательность обработки сообщения

Перейдем к рассмотрению генерации уведомлений. Одним из требований к системе является отправка уведомлений на почту оценивающим пользователям и аудиторам. Для того, чтобы подготовленный текст можно было переиспользовать многократно, но при этом письма содержали в себе актуальную информацию, было решено применить систему шаблонов. Так как редактирование шаблонов уведомлений доступно администратору, и должно быть ему понятно, был использован шаблонизатор с минимальным количеством логики и с простейшим синтаксисом — Mustache. Шаблонизатор способен генерировать итоговый текст из шаблона, используя некий контекст. Контекст — это набор переменных, значения которых будут использованы для генерации. Шаблон представляет собой текст, содержащий в себе специальные синтаксические конструкции, позволяющие вставлять в него значения переменных, а также включать/исключать некоторые части в зависимости от

значения переменных. В качестве реализации использована библиотека Scalate. Пример шаблона приведен ниже.

```
Добрый день, {{user.full_name}}

Вы принимаете участие в оценке "{{event.name}}", которая
продлится с {{event.start}} до {{event.end}}

Вам необходимо оценить своих коллег в следующих проектах:
{{#projects}}
 * Проект "{{name}}"
{{/projects}}

Для участия перейдите по ссылке:
https://assessment-system-url.com/events/{{event.id}}
```

Отправка писем реализована по протоколу SMTP, в качестве посредника выступает Amazon SES[8]. Этот сервис необходим, так как иначе рассылка большого количества писем ведет к их попаданию в спам.

Следующий компонент Исполнителя задач — это Обработчик начала события оценки. Он необходим для создания всех записей, используемых для отображения вопросов пользователю и для сохранения ответов.

[8]Amazon Simple Email Service — платформа отправки и получения электронной почты для использования в бизнесе и разработке ПО

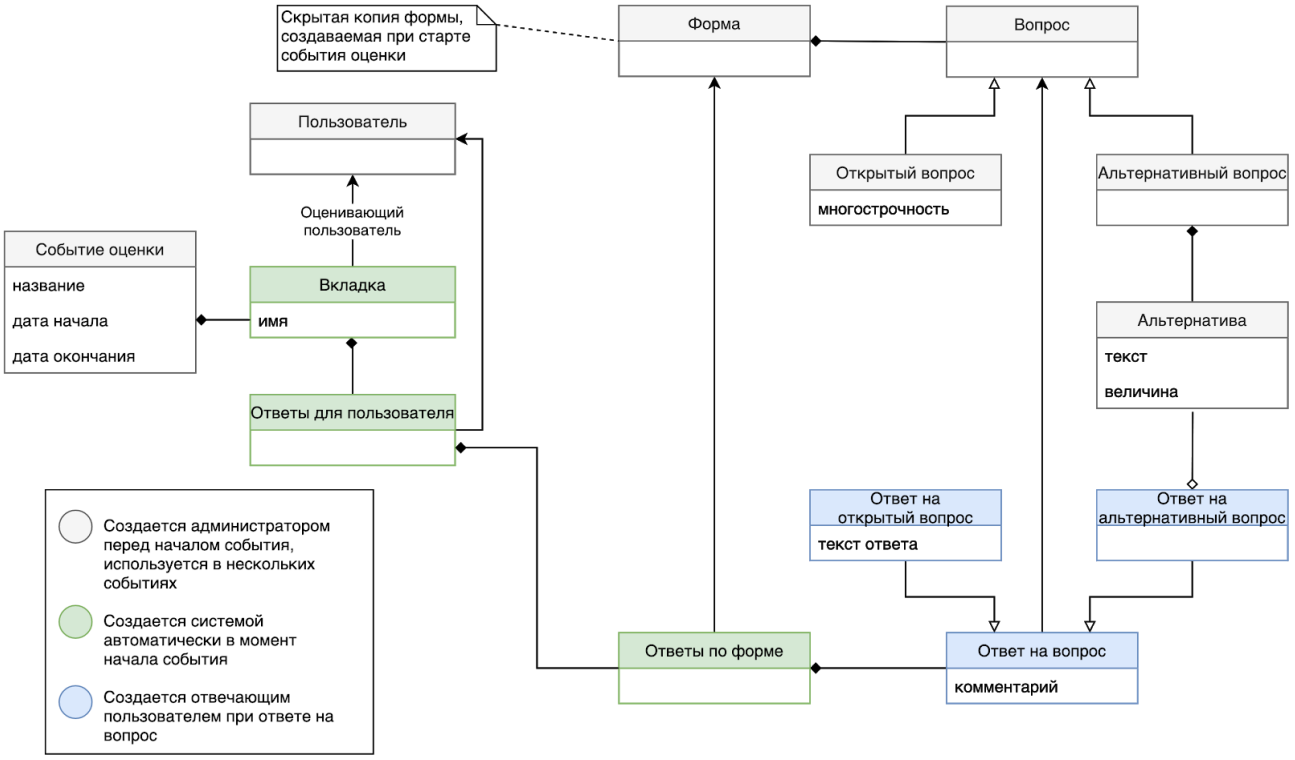


Рисунок 2.6.5 — Сущности для хранения ответов

На рисунке 2.6.5 приведена информация о моменте времени, в который создаются экземпляры классов, необходимых для оценки. В этом обработчике создаются экземпляры следующих классов:

- Вкладка;
- Ответы для пользователя;
- Ответы по форме.

Для каждого проекта, в котором пользователь является оценивающим, создается вкладка. Для всех пользователей из всех отношений проекта создаются ответы для пользователя, содержащие в себе ответы по форме. На рисунке 2.6.6 приведен пример того, каким образом это может быть представлено интерфейсе.

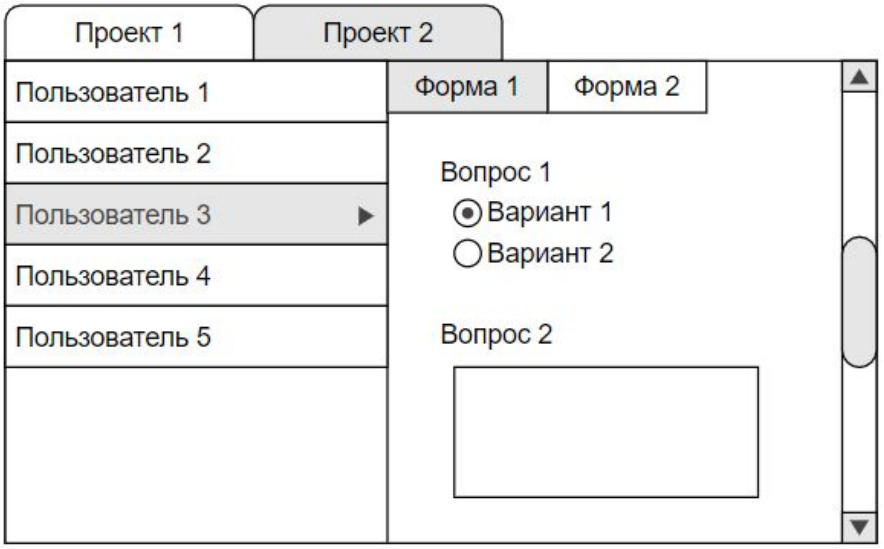


Рисунок 2.6.6 — Пример интерфейса для ответа на вопросы в событии

Такой подход позволяет отвязать текущие и завершенные события от изменений в оргструктуре(в группах, проектах, отношениях). То есть изменения этих сущностей не повлияет на набор пользователей и вопросы в прошедших и текущих событиях оценки.

Перейдем к процессу формирования отчетов. Одной из важнейших функций системы является просмотр результатов оценки аудитором. Результат оценивания должен содержать всю информацию об ответах, как в детальном, так и агрегированном виде. Это необходимо для того, чтобы аудитор мог надлежащим образом проанализировать текущую обстановку в организации, особенности взаимоотношений между сотрудниками, сильные и слабые места каждого из них. Согласно требованиям, отчет должен формироваться в виде электронной таблицы и загружаться на сервис Google Drive. Доступ к файлам для аудиторов должен предоставляться автоматически. Таким образом, задача состоит в том, чтобы каким-то образом формировать файл электронной таблицы с результатами и загружать его на Drive.

Для каждого события оценки система должна создавать отдельную директорию, содержащую по одному файлу с результатом на каждого оцениваемого, каждый файл должен содержать следующие элементы:

- отчет по компетенциям — усредненное значение каждой компетенции, используемой в вопросах формы, отдельно необходимо считать значения самооценки, если эта опция включена, и оценок, полученных от других людей, результат представлен как в виде таблице, так и в виде круговой диаграммы;
- коэффициент согласованности ответов — величина, показывающая уровень однообразности мнений о человеке;
- история изменений компетенций, в виде таблицы и графика;
- агрегированные ответы на вопросы о человеке, сколько голосов было отдано за тот или иной варианты ответа;
- детальный ответы на вопросы, содержащие в себе информацию по каждому отвечающему.

Для оценки согласованности ответов используется коэффициент конкордации Кендалла *W*. Значение коэффициента конкордации может находится в диапазоне от 0 до 1. Если *W*=0, считается, что мнения экспертов не согласованы. Если *W*=1, то оценки экспертов полностью согласованы. На рисунке 2.6.7 приведен вывод формулы, по которой рассчитывается это значение. Где:

- **n** — число компетенций;
- **k** — число оценивающих;
- $\mathbf{r_{ij}}$ — ранг i–ой компетенции определённая j–ым оценивающим;
- $\mathbf{d_i}$ — сумма рангов i–ой компетенции по всем оценивающим;
- **W** — коэффициент конкордации Кендалла.

$$W = \frac{12}{n^3 - n} \sum_{i=1}^{n} \left(\frac{1}{k} \sum_{j=1}^{k} r_{ij} - \frac{n+1}{2} \right)^2 \Leftrightarrow$$

$$\Leftrightarrow W = \frac{12}{n^3 - n} \sum_{i=1}^{n} \left(\overline{r}_i - \frac{n+1}{2} \right)^2, \ \overline{r}_i = \frac{1}{k} \sum_{j=1}^{k} r_{ij} \Leftrightarrow$$

$$\Leftrightarrow W = \frac{12}{k^2(n^3 - n)} \sum_{i=1}^{n} \left(\sum_{j=1}^{k} r_{ij} - \frac{k(n+1)}{2} \right)^2 \Leftrightarrow$$

$$\Leftrightarrow W = \frac{12}{k^2(n^3 - n)} \sum_{i=1}^{n} \left((d_i - \overline{d})^2 \right), \ d_i = \sum_{j=1}^{k} r_{ij}, \ \overline{d} = \frac{k(n+1)}{2}$$

Рисунок 2.6.7 — Вывод формулы для расчета коэффициента конкордации Кендалла

Расчет коэффициента производился с помощью библиотеки Apache Commons Math. Для большей наглядности, перед отображением значение коэффициента умножается на 10.

Рассмотрим возможные способы работы с электронными таблицами из языка программирования Scala. Существует много библиотек для выполнения этой задачи, обратим подробное внимание на две из них, кардинально отличающиеся подходом. Первая из них — это Apache POI. Apache POI является самой популярной библиотекой для работы с файлами MS Office в экосистеме Java. Она предоставляет интерфейс для чтения и записи офисных файлов. Рассмотрим подробнее формирование файлов электронных таблиц с помощью этой библиотеки. Подход является полностью императивным, программисту доступны самые низкоуровневые API для манипулирования элементами в документе. Например, для того, чтобы записать значение в первую ячейку, программист должен сделать следующее

```
Workbook book = new HSSFWorkbook();
Sheet sheet = book.createSheet("Название листа");
Row row = sheet.createRow(0);
Cell name = row.createCell(0);
name.setCellValue("Значение ячейки A1");
```

Можно видеть, что в распоряжении разработчика есть самые базовые методы. Реализация формирования сложных документов с таким подходом неудобна, а абсолютные индексы могут стать причиной ошибки при рефакторинге. Помимо этого, смотря на код, крайне сложно понять, каким образом будет выглядеть итоговый документ. По этим причинам библиотека не подходит, необходим инструмент с более высоким уровнем абстракций для формирования необходимых структур в документе.

Следующий инструмент — это Jxls. Jxls внутри использует Apache POI для формирования отчетов, но разработчик освобожден от низкоуровневой работы с документом. Для создания электронной таблицы с данными необходимы две вещи — шаблон, сам являющийся электронной таблицей и контекст с данными. По своей идее данная библиотека похожа на шаблонизаторы для языка HTML, но работает с электронными таблицами. Шаблон содержит в себе управляющие теги в ячейках, и при формировании отчета библиотека подставляет вместо тегов соответствующие значения из контекста. Пример шаблона представлен на рисунке 2.6.8.

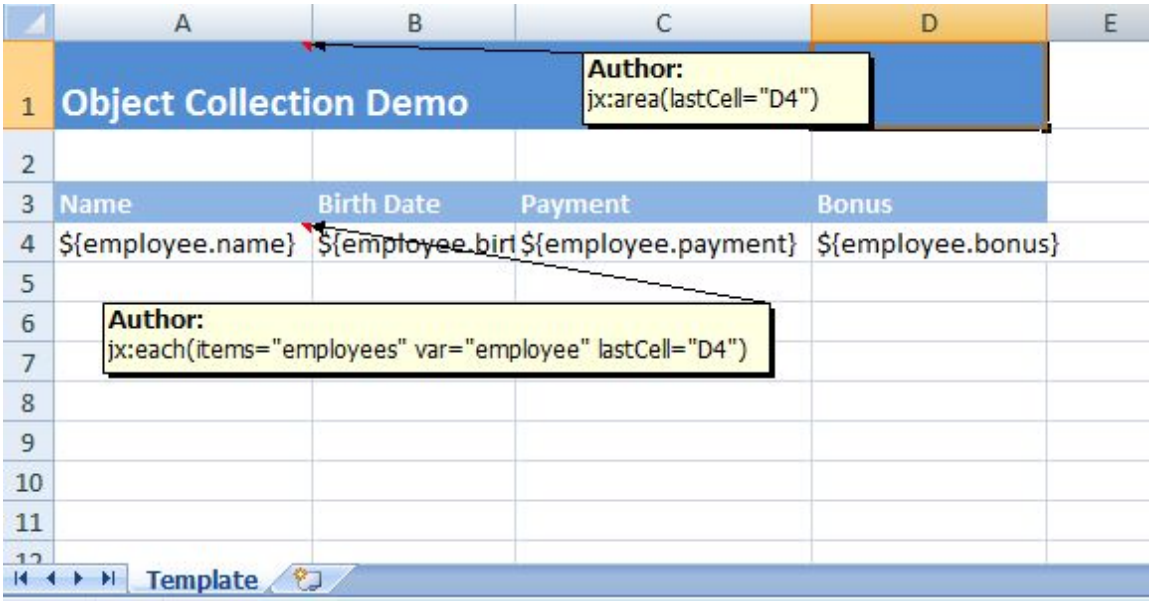


Рисунок 2.6.8 — Создание шаблона для jxls

Шаблоны можно редактировать в специализированных офисных программах, таких как Microsoft Excel или Libre Office Calc. Для того, чтобы сгенерировать результат, библиотеке нужно передать сам шаблон и контекст. Контекст — это объект, содержащий в себе поля, которые будут использоваться для отображения. Поддерживается неограниченная вложенность объектов, списки, пары ключ-значение (Map).

```
List<Employee> employees = getData();
InputStream template = getTemplate();
OutputStream result = new FileOutputStream("result.xls");
Context context = new Context();
context.putVar("employees", employees);
JxlsHelper.getInstance().processTemplate(is, os, context);
```

Такой подход работы с таблицами крайне удобен, но имеет один существенный недостаток. Шаблоны хранятся в системе контроля версий в виде двоичных файлов, из чего вытекает невозможность генерации списка изменений при коммите и невозможность разрешения конфликтов в случае их возникновения. Кроме того, отсутствует возможность проверять теги внутри шаблона на корректность. В случае рефакторинга, изменения названия переменных, проект будет собран успешно, но в процессе работы документы будут создаваться некорректно. Тесты также не могут помочь, так как в библиотеке отсутствует функционал валидации тегов, в случае неправильного названия или опечатки соответствующий блок просто не будет выведен в результирующий документ. Помимо этого в процессе работы были встречены различные проблемы с отображением полученных документов в программах, отличных от Microsoft Excel. Так как было необходимо отображать документы онлайн с помощью Google Sheets[9], это стало критическим недостатком.

Так как существующие библиотеки по тем или иным причинам не подходили, было решено реализовывать собственную. Основные функциональные возможности, которые хотелось бы поддерживать, это:

[9]Google Sheets — бесплатный онлайн сервис для просмотра и редактирования электронных таблиц

- декларативный подход, позволяющий программисту меньше уделять внимания низкоуровневым деталям и уменьшающий вероятность ошибок;
- набор высокоуровневых абстракций, таких как таблица, список;
- поддержка стилей, таких как цвет и толщина границ, стиль шрифта, фоновый цвет ячейки;
- независимость от реализации электронных таблиц. Библиотека должна поддерживать различные модули, занимающиеся непосредственно генерацией таблиц. То есть используя один и тот же код структуры документа должна быть возможность формировать как Excel файлы, так и вызовы в Google Sheets API.

Принимая во внимания необходимые функциональные возможности, была представлена разработана концепция, удовлетворяющая требованиям. На рисунке 2.6.9 представлено, из каких частей состоит состоит библиотека и каким образом эти части взаимодействуют. Прямоугольник со скругленными углами — это данные, передаваемые в некоем формате. Простые прямоугольники — обработчики. Стрелки — потоки данных между обработчиками.

Рисунок 2.6.9 — Потоки данных библиотеки

Такой подход позволяет разделить шаблон, данные и конкретный генератор, используемый для формирования. По своей идее это напоминает шаблон проектирования двухэтапное представление (Two Step View).

Рассмотрим способ формирования шаблона. Для этого был разработан DSL (или предметно-ориентированный язык), с поддержкой всех базовых конструкций.

```
Document[ExportDTO] result = Document.create(doc => {
    doc.page("Пользователи", page => {
        page.label(l => {
            l.text("Список пользователей")
            l.style(s => s.align(Alignment.CENTER))
        })
        page.table(_.getUsers, usersTable => {
            usersTable.column(_.getFirstName)
            usersTable.column(_.getLastName)
        })
    })
})
```

DSL основан на использовании лямбда-выражений. Лямбда выражение — это специальный синтаксис для определения функциональных объектов. В языке Scala запись лямбда выражений выглядит так: `(аргументы)=>выражение`. Существует особый синтаксис для лямбда-выражений, содержащих в себе вызов метода у объекта. Такую запись: `user=>user.getFirstName` можно кратко записать как `_.getFirstName`.

Диаграмма классов интерфейсов, доступных для пользователя этой библиотеки представлена на рисунке 2.6.10.

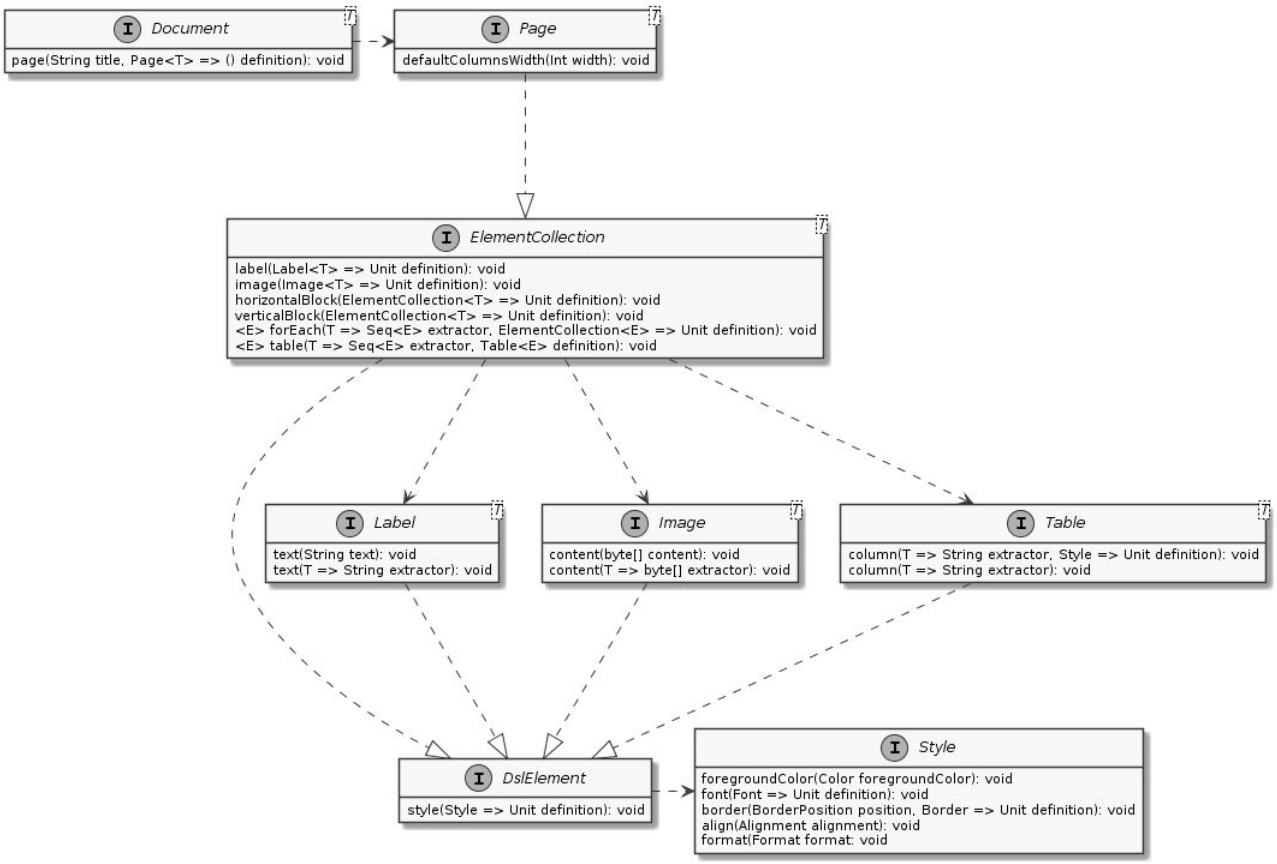


Рисунок 2.6.10 — Диаграмма классов для создания DSL

Каждый из этих интерфейсов имеет свою реализацию, и каждая эта реализация унаследована от базового интерфейса `VisitableDslElement`, имеющего один метод — `visit(DslElementVisitor visitor)`. Такой подход позволяет легко обрабатывать созданное дерево объектов, создавать новые классы таких объектов и различные обработчики. Перейдем к преобразованию шаблона в промежуточную структуру. Для этого было введено две абстракции — контейнер и ячейка. Контейнер может содержать ячейки и другие контейнеры, а также имеет направление, в котором добавляются новые элементы. Фактически это является реализацией паттерна проектирования «Компоновщик».

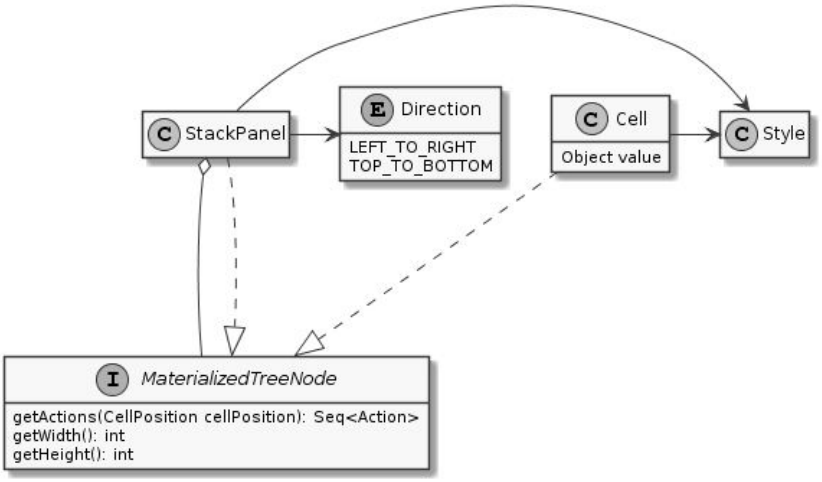


Рисунок 2.6.10 — Диаграмма классов для промежуточной структуры

После этого структуру необходимо преобразовать в список действий, для этого необходимо вызвать метод `getActions` у корневого элемента, передав начальную позицию отрисовки в качестве параметра. Метод будет последовательно вызван у всех элементов контейнера и вернет полный список действий, необходимых для отрисовки таблицы. Диаграммы последовательности для классов `Cell` и `StackPanel` приведены на рисунках 2.6.11, 2.6.12.

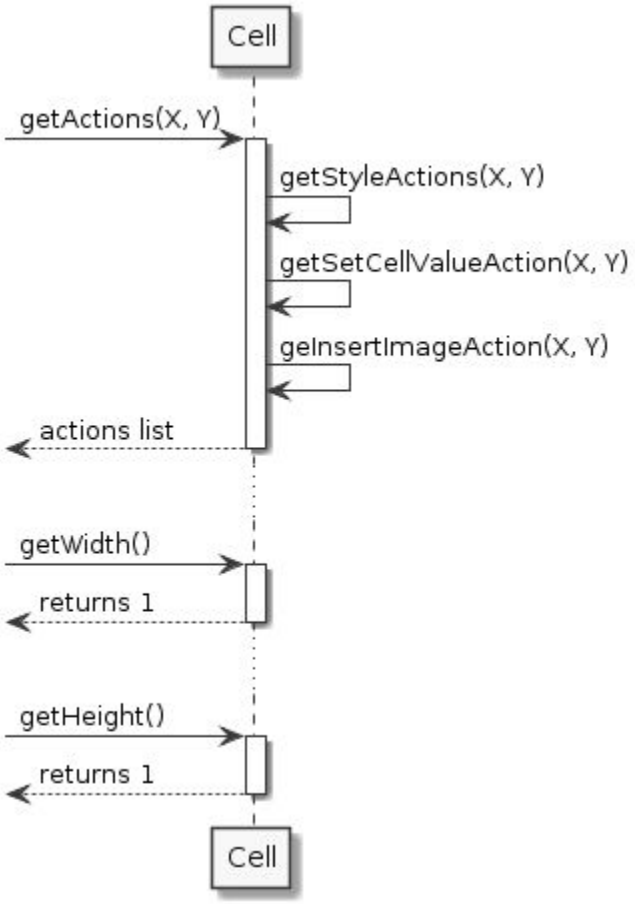


Рисунок 2.6.11 — Диаграмма последовательности класса Cell

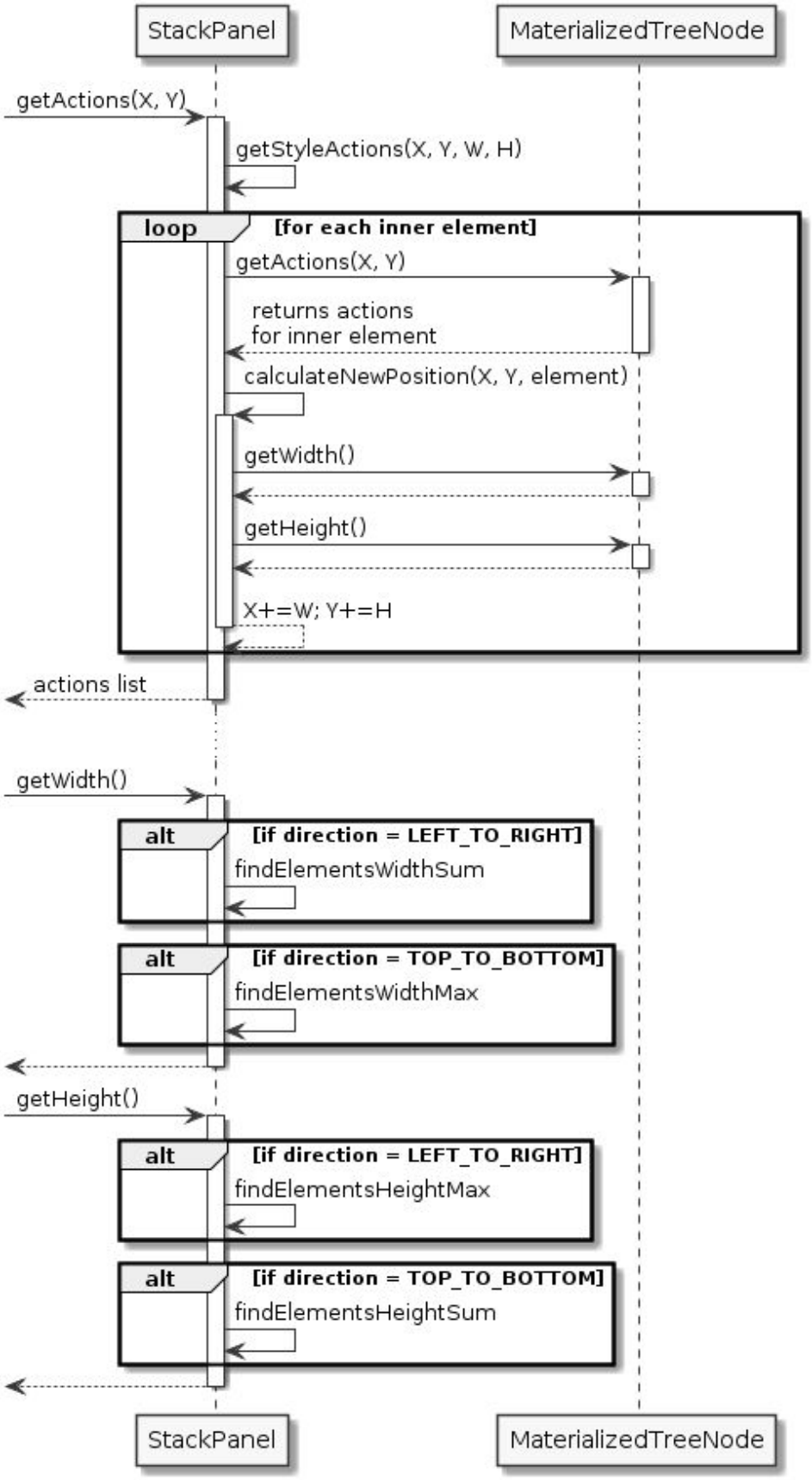


Рисунок 2.6.12 — Диаграмма последовательности класса StackPanel

Диаграмма классов для действий приведена на рисунке 2.6.13.

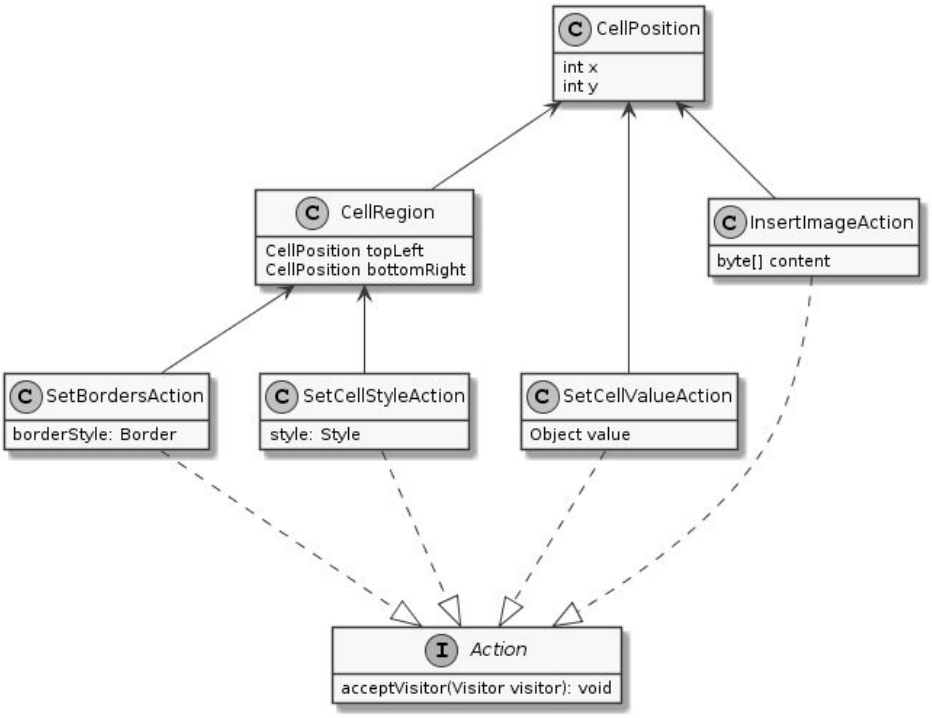


Рисунок 2.6.13 — Диаграмма классов действий

После того, как список действий получен, можно преобразовывать каждое из них в низкоуровневые вызовы в сторонние библиотеки, например, с помощью Apache POI. В данном случае преобразование производится напрямую в вызовы Google API для модификации данных в таблице.

Помимо текста и таблиц было необходимо добавить поддержку вывода диаграмм и изображений. Диаграммы можно отрисовывать двумя путями:

- использовать инструменты встроенные в средство редактирования и просмотра электронных таблиц;
- генерировать их заранее и вставлять в виде обычного изображения.

В данной работе был выбран второй подход, так как он позволяет сделать внешний вид диаграмм независимым от формата экспорта, дает возможность использовать типы диаграмм, неподдерживаемые офисным приложением. Этот подход имеет и минус, существенный в общем случае, но являющийся неважным в этом приложении — диаграммы не будут обновляться при

обновлении данных в таблице. Это не является проблемой, так как редактирование полученных отчетов не является необходимостью.

Существует большое количество библиотек генерации диаграмм для языков Java и Scala. Из них была выбрана библиотека XChart, так как она имеет все необходимые функции, бесплатна, с открытыми исходными кодами и активно поддерживается разработчиками. Чтобы вывести график разработчику необходимо выполнить следующие действия:

```
RadarChart chart = new RadarChartBuilder().width(800).height(600).title("График
компетенций").build
chart.setVariableLabels(Array("Инициативность", "Интерес к работе",
"Исполнительность", "Командная работа"))
chart.addSeries("Среднее", Array(0.8, 0.7, 0.4, 0.5))
chart.addSeries("Самооценка", Array(0.6, 0.9, 0.6, 0.7))
val byteArrayOutputStream = new ByteArrayOutputStream()
BitmapEncoder.saveBitmap(chart, byteArrayOutputStream,
BitmapEncoder.BitmapFormat.PNG)
val bytes = byteArrayOutputStream.toByteArray
```

Этот код создает новую круговую диаграмму в формате PNG, результат сохраняется в массив байт для дальнейшего использования.

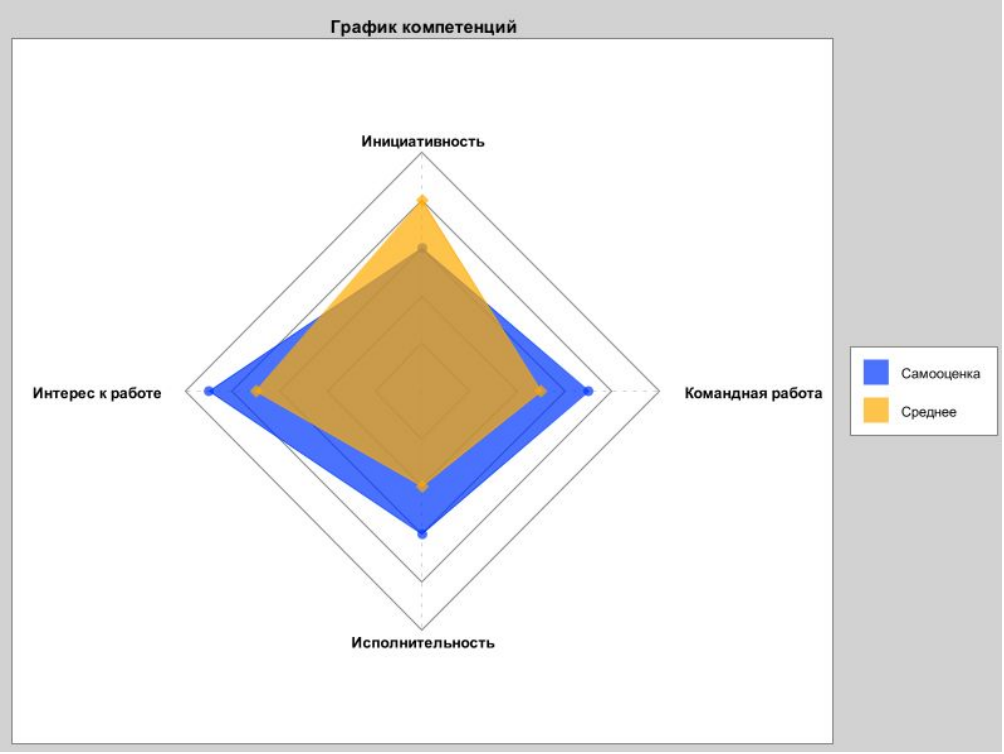


Рисунок 2.6.14 — Пример круговой диаграммы

3 Процесс разработки

3.1 Документирование API

Так как в процессе разработки участвовало две команды, было необходимо каким-то образом документировать REST API, для возможности его использования разработчиками одностраничного приложения. Для решения этой задачи существуют несколько подходов:

- составление описания методов и моделей в текстовом виде вручную;
- использование одной из спецификаций:
    - Swagger(OpenAPI);
    - RAML;
    - WADL.

Первый подход не требует изучения спецификаций и установки дополнительного ПО, но это ведет к отсутствию возможности автоматической генерации интерактивной документации. Второй подход требует знания инструментов и языков, но дает преимущества, такие как генерация документации и кода по спецификации.

В качестве инструмента для документирования API в проекте был выбран Swagger, так как в его состав входят библиотеки для генерации спецификации, интерактивного отображения и отправки запросов.

Основной частью документирования API с помощью Swagger является схема. Она представляет собой текстовый файл, содержащий описание всех HTTP ресурсов, включая доступные пути, методы, модели запроса и ответа. Текст может иметь формат JSON или YAML, набор доступных полей определяется спецификацией OpenAPI. Из особенностей стоит отметить, что спецификация не зависит от языка программирования и имеет декларативный характер, поэтому позволяет клиентам использовать её, не вникая в детали реализации сервера.

Спецификация может создаваться тремя способами:

- ручной, когда API полностью описывается человеком;
- автоматический, когда спецификация генерируется на основе кода;
- полуавтоматический, когда часть описания генерируется на основе кода, а часть, например, названия методов и комментарии, заполняются человеком.

В системе был применен полуавтоматический способ генерации, так как он имеет разумный баланс между трудоемкостью создания документации и ее понятностью для человека. Для этого была использована библиотека play-swagger. Она генерирует файл спецификации на основе файла с маршрутами (это компонент Play Framework). Каждая запись из этого файла превращается в один метод в документации, при необходимости разработчик может добавить дополнительную информацию вручную, в виде комментария к записи с маршрутом. Комментарий содержит в себе текст в формате YAML, он должен соответствовать спецификации OpenAPI. Так как этот файл хранится в системе контроля версий в текстовом формате, это позволяет без труда отслеживать вносимые изменения. Выглядит это следующим образом:

```
###
# summary: Returns logged in user
# tags:
#   - User
###
GET /users/current controllers.user.UserController.me
```

Для отображения сгенерированной схемы используется специальный компонент — Swagger UI. Swagger UI — это веб приложение, которое можно развернуть самостоятельно в локальной сети, визуализирующее файл для схемы в наглядном виде, позволяющее отправлять запросы на сервер прямо из документации. Его интерфейс представлен на рисунке 3.1.1.

Рисунок 3.1.1 — Интерфейс Swagger UI

В итоге мы получили страницу, всегда содержащую актуальную информацию о доступных методах сервера, позволяющая UI разработчикам более эффективно разбираться в структуре API, без привлечения разработчика серверной части.

3.2 Особенности тестирования

Одним из нефункциональных требований являлось покрытие всего кода юнит-тестами. Юнит-тестирование или модульное тестирование — это процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки. Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже протестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Для того, чтобы покрытие было более полным, использовался подход, называемый property-based testing или тесты, основанный на свойствах. Первое, что стоит сказать property-based тесты используют подход черного ящика и запускаются многократно, соответственно отлично подходят для использования в модульных тестах. Суть заключается в следующем, для каждой тестируемой функции необходимо описать допустимое множество входных значений, при которых выполняются заданные условия. Причём, в отличии от классических юнит-тестов, тестовые данные не задаются явно, а генерируются автоматически. Это дает следующие преимущества:

- код тестов становится короче;
- так как входные данные не контролируются, покрытие кода тестами обычно получается выше чем в классическом подходе;
- в сгенерированные данные всегда включаются специальные случаи, которые разработчик мог упустить из вида — пустые списки, максимальные значения чисел, строки из иероглифов;
- принято считать, что property-based тесты легче читаются человеком;
- благодаря процессу, называемому shrinking(сокращение) тестовый фреймворк подбирает минимальный набор данных, вызывающий падение тестов.

Тестовый фреймворк делает несколько прогонов теста с различными сгенерированными данными (по умолчанию 100). Суть shrinking заключается в том, что в случае падения теста фреймворк не сразу выдает результат с данными его вызвавшими, а тратит некоторое количество попыток, чтобы найти максимально простые(минимальные) параметры, вызвавшие падение.

В качестве фреймворка для модульного тестирования использовался ScalaTest, для поддержки property-base тестирования была использована библиотека ScalaCheck. ScalaCheck имеет в своём составе генераторы для всех базовых типов данных: числа, строки, коллекции, даты. Помимо этого есть

возможность описывать генераторы для собственных типов. Для тестирования DAO классов была использована СУБД H2, позволяющая быстро создавать тестовые БД в памяти. Разберем тест, проверяющий корректность сохранения формы (вместе со всеми элементами) в базе.

```
"formDao.create" should {
  "create form in database" in {
    forAll { (form: Form) =>
      dao.create(form)
      val formFromDb =dao.findById(created.id)
      formFromDb mustBe form
    }
  }
}
```

Здесь автоматически генерируются различные экземпляры класса Form. Далее они сохраняются и затем сразу получаются из БД, после чего производится проверка, что полученная из БД форма соответствует сохраняемой.

3.3 Непрерывная интеграция и доставка

Помимо проектирования и разработки, было необходимо обеспечить автоматическое тестирование новых версий, развертывание на окружении для разработки, а в дальнейшем и развертывание на рабочее окружение, для проведения опросов среди сотрудников.

Непрерывная интеграция – это практика разработки программного обеспечения, при которой разработчики регулярно объединяют изменения программного кода в центральном репозитории, после чего автоматически выполняется сборка, тестирование и запуск. Понятие непрерывной интеграции чаще всего применяется к стадии сборки или интеграции процесса выпуска ПО и включает в себя как компонент автоматизации, так и компонент культуры разработки. Главная задача непрерывной интеграции – быстрее находить и исправлять ошибки, улучшать качество ПО и сокращать временные затраты на проверку и выпуск новых обновлений ПО.

Непрерывная доставка – это, в свою очередь, практика разработки программного обеспечения, когда при любых изменениях в программном коде выполняется автоматическая сборка, тестирование и подготовка к окончательному выпуску. Непрерывная доставка является одним из основополагающих принципов разработки современных приложений, поскольку расширяет практику непрерывной интеграции за счет того, что все изменения кода после стадии сборки развертываются в тестовой и/или в рабочей среде. Отличие непрерывной доставки от непрерывного развертывания заключается в том, что при непрерывной доставке для развертывания обновлений в рабочей среде требуется подтверждение вручную. При непрерывном развертывании это происходит автоматически без специального подтверждения.

Для реализации всех этих подходов использовался компонент GitLab, под названием GitLab CI. GitLab — это платформа управления Git-репозиториями, анализа кода, отслеживания ошибок, тестирования, деплоя, ведения каналов и вики-страниц.

Так как сам GitLab уже был запущен в локальной сети предприятия, его установка и настройка рассматриваться не будет. Для того, чтобы автоматически выполнялись активности по тестированию и развертыванию сборок необходимы две вещи — конфигурация CI/CD, представляющая собой YAML файл `.gitlab-ci.yaml` в корне репозитория и runner (исполнитель) — программа, запущенная в каком-либо окружении и, собственно, выполняющая команды, прописанные в конфигурационном файле. В конфигурационном файле есть возможность определять различные стадии. Стадия — это набор действий, выполняемый при определенных условиях. Например, можно создать стадию `test`, запускающую модульные тесты при каждой фиксации изменений в репозитории и стадию `deploy_develop`, автоматически выполняемую при

загрузке изменений в develop ветку. Также существует термин pipeline — это набор стадий, выполняемых для конкретного действия в репозитории.

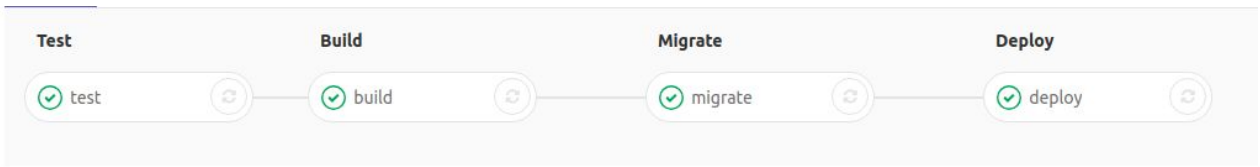


Рисунок 3.3.1 — Pipeline из 4 стадий

В приложении были созданы следующие стадии:

- `test` — запуск модульных тестов;
- `build` — создание исполняемых артефактов(jar файлов) и сборка Docker образов;
- `migrate` — запуск утилиты, выполняющей миграцию схемы и данных в БД;
- `deploy` — развертывание приложения на тестовом окружении;
- `deploy_master` — развертывание окружения на рабочем окружении, запускается только вручную.

Стоит рассказать про миграцию базы данных. Для выполнения этой задачи была использована библиотека Flyway. Flyway позволяет разработчику описывать изменения в БД на языке SQL и автоматически применять эти изменения к базе данных по команде. В случае возникновения ошибки изменения будут отменены.

Чтобы развертывать приложение используется Docker. Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесен на любую систему, поддерживающую запуск контейнеров, а также предоставляет среду по управлению контейнерами.

Модель Docker состоит из следующих базовых элементов:

- Dockerfile;

- образ;
- контейнер.

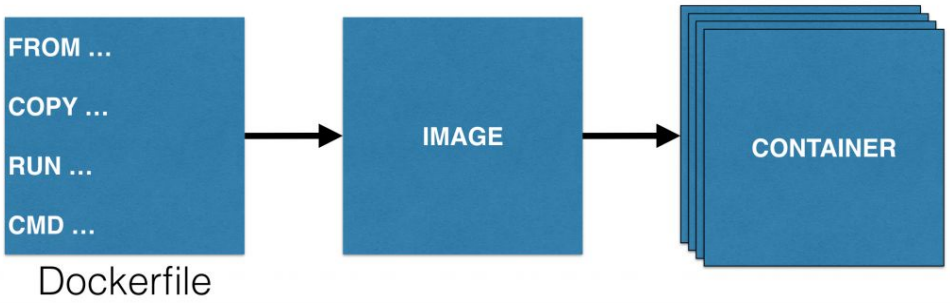


Рисунок 3.3.2 — Взаимосвязи между элементами системы Docker

Dockerfile автоматизирует процесс создания образов. Dockerfile описывает среду и используется для сборки образа. Образ — это шаблон для создания контейнера, содержащий в себе исполняемые файлы со всеми необходимыми зависимостями. Контейнер — это запущенный образ, исполняемая единица, которой можно управлять независимо — перезапускать, останавливать, удалять, конфигурировать.

Каждая подсистема разворачивается в отдельном контейнере, также в Docker запущены СУБД и брокер сообщений. Такая схема развертывания применяется на всех окружениях: локальном, тестовом и рабочем. В случае необходимости система может быть развернута на новом окружении одной командой.

Для конфигурации контейнеров используется утилита Docker Compose. Это инструмент для описания и запуска систем, состоящих из нескольких контейнеров. Утилита использует YAML файл с конфигурацией контейнеров (открытые порты, настройки томов) и способна запускать контейнеры используя эту конфигурацию. По сути, это является автоматизированным аналогом ручного вызова команд Docker с помощью терминала.

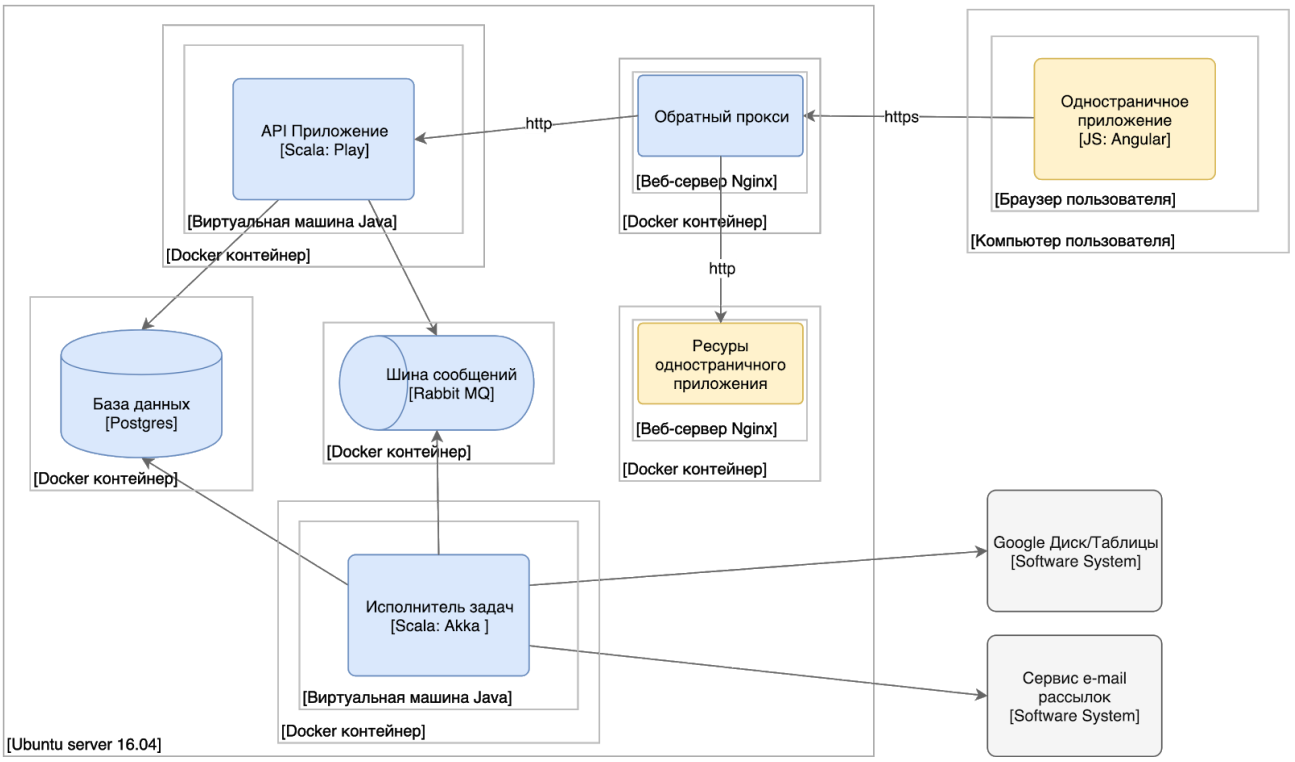


Рисунок 3.3.3 — Диаграмма развертывания системы

На рисунке 3.3.3 представлена диаграмма развертывания системы. Обратный прокси здесь — это веб сервер Nginx, который сконфигурирован переадресовывать запросы в той или иной контейнер в зависимости от URL, а также предоставляющий HTTPS интерфейс для запросов извне.

Заключение

В рамках данной работы решены все поставленные задачи:

- собраны и формализованы требования;
- система спроектирована;
- система разработана;
- система протестирована и внедрена.

Таким образом, цель — разработать и внедрить систему для оценки персонала по методу «360 градусов» — достигнута. Система успешно встроена в рабочий процесс компании. Проблем в ходе эксплуатации не выявлено, события оценки проводятся регулярно.

Литература

1. Ларман К. Применение UML и шаблонов проектирования. / К. Ларман – Издательский дом «Вильямс», 2004. 620 с.
2. Уорд П. Метод 360 градусов. М.: ГИППО, 2006. 120 с.
3. Cockburn A. Writing Effective Use Cases. 1st ed. Addison-Wesley Professional, 2000. 304 pp.
4. Edwards, M.R. and Ewen, A.J. 360-Feedback: The Powerful New Model for Employee Assessment & Performance Improvement. AMACOM, New York, 2013
5. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002.
6. Gitlab User Guide [Электронный ресурс] // Gitlab [сайт]. 2019. URL: https://docs.gitlab.com/ (дата обращения: 30 апреля 2019).
7. jXLS documentation [Электронный ресурс] // jXLS [сайт]. 2019. URL: http://jxls.sourceforge.net (дата обращения: 28 апреля 2019).
8. Legendre P. The Kendall Coefficient of Concordance Revisited. Journal of Agricultural, Biological, and Environmental Statistics, 2005, Volume 10, Number 2, Pages 226–245 DOI: 10.1198/108571105X46642.
9. Peiperl M. Getting 360-Degree Feedback Right. Harvard Business Review, January 2001, 79(1):142-7, 177, PMID: 11189458.
10. Play 2.7.x documentation [Электронный ресурс] // Play Framework [сайт]. 2019. URL: https://www.playframework.com/documentation/2.7.x/Home (дата обращения: 15 января 2019).
11. RabbitMQ documentation [Электронный ресурс]. 2019. URL: https://www.rabbitmq.com/documentation.html (дата обращения: 21 марта 2019).
12. Saxena S. Mastering Play Framework for Scala. Packt Publishing, 2015.

13. Swagger documentation [Электронный ресурс] // Swagger [сайт]. 2019. URL: https://swagger.io/docs/ (дата обращения: 29 апреля 2019).
14. Torno W. Introduction to special issue on 360 - degree feedback. Human Resource Management, Summer/Fall 1993, vol. 32, no. 2, pp. 211-219. doi: 10.1002/hrm.3930320202.
15. Turnbull J. The Docker Book: Containerization is the new virtualization. / James Turnbull; 18092 edition, 2014.

Приложение А. Скриншоты интерфейса

Open360 / Профиль / Егор Ильченко

Настройки профиля

Имя * Егор Ильченко

Email * ilchenko_ei@bw-sw.com

Пол * Мужской

Часовой пояс * Asia/Tomsk (UTC+07:00)

Изображение профиля

Загрузить изображение профиля

Сохранить изменения

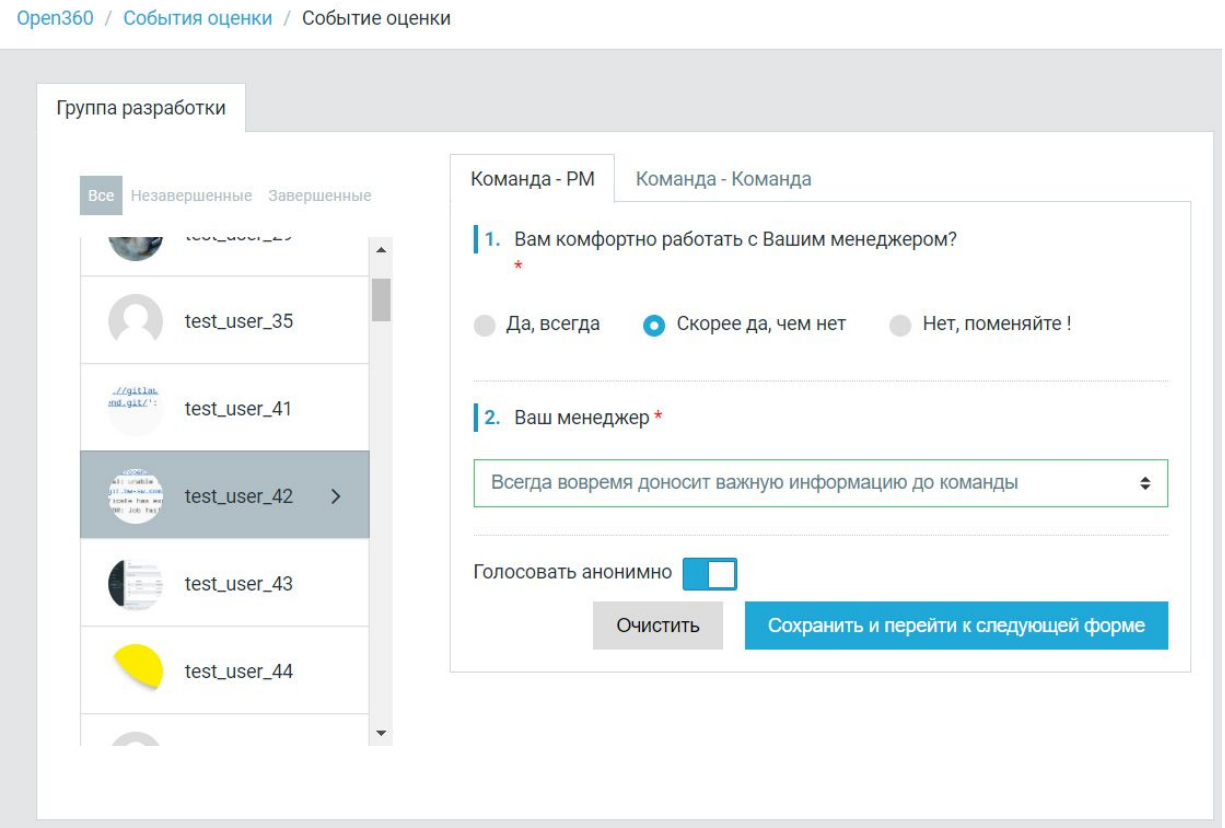
Редактирование профиля

Open360 / События оценки

В процессе | Не начато | Завершено

#	Описание	Дата начала	Дата окончания	Событие завершится	
568	Событие #61	14 мая 2019 г., 22:12:00	14 мая 2019 г., 22:20:00	через 7 минут	Начать оценку

Список активных событий оценки



Процесс ответов на вопросы формы

Приложение Б. Скриншоты отчета

Отчет по компетенциям		
Компетенция	Среднее	Самооценка
Инициативность	8	6
Интерес к работе	7	9
Исполнительность	4	6
Командная работа	5	7
Итого	6	7
Согласованность	5,6675	

Таблица компетенций

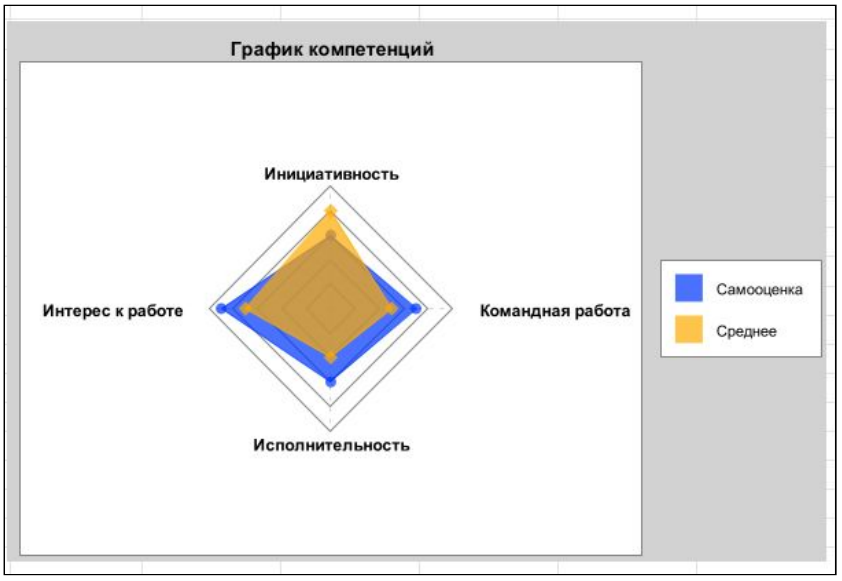


График компетенций

История значений			
Компетенция	Оценка 2018/3	Оценка 2018/4	Оценка 2019/1
Гибкость	7	7	
Инициативность	5	5	8
Интерес к работе	6	7	7
Исполнительность	6	6	4
Командная работа	3	5	5

История значений компетенций

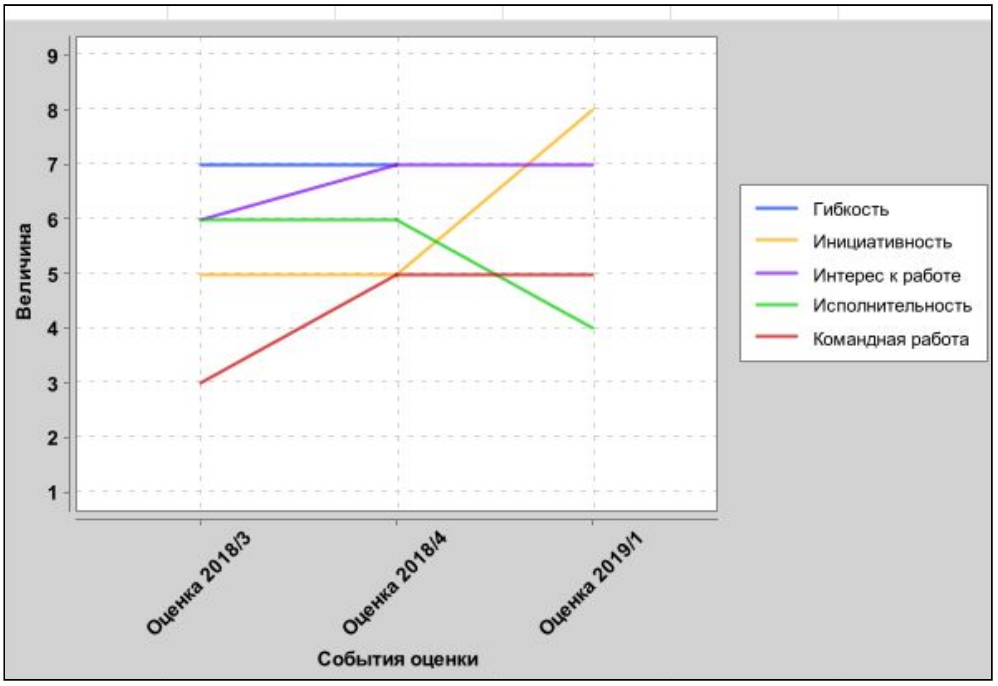


График изменения компетенций

	Оценка Команда - Команда			
Имя	Проявляет инициативу, вносит рационализаторские предложения.	Стремится решить проблему максимально быстро и эффективно	Умеет хорошо концентрироваться на задаче, внимателен к мелочам.	Старается найти общие интересы и общий язык с коллегами в случае решения совместных задач.
Иванов Иван (самооценка)	Иногда	Всегда	Иногда	Часто
Мармазов Измаил	Всегда	Иногда	Иногда	Иногда
Варфоломеева Ольга	Всегда	Иногда	Иногда	Иногда
Бабыкин Андрон	Всегда	Всегда	Всегда	Часто
Салтыкова Эмилия	Всегда	Всегда	Часто	Иногда
Чекмарёва Римма	Часто	Часто	Часто	Редко
Балабанова Елизавета	Всегда	Всегда	Иногда	Иногда
Новолодский Тимофей	Всегда	Всегда	Часто	Часто
Кропанина Каролина	Редко	Иногда	Редко	Никогда
Безукладникова Пелагея	Всегда	Всегда	Всегда	Всегда
Лукьянов Влад	Часто	Часто	Часто	Часто

Детальный отчёт

Оценка Команда - Команда	
Вопрос	Результат
Проявляет инициативу, вносит рационализаторские предложения.	Всегда - 7 Часто - 2 Редко - 1
Стремится решить проблему максимально быстро и эффективно	Всегда - 5 Часто - 2 Иногда - 3
Умеет хорошо концентрироваться на задаче, внимателен к мелочам.	Всегда - 2 Часто - 4 Иногда - 3 Редко - 1
Старается найти общие интересы и общий язык с коллегами в случае решения совместных задач.	Всегда - 1 Часто - 3 Иногда - 4 Редко - 1 Иногда - 1

Агрегированный отчет

АНТИПЛАГИАТ
ТВОРИТЕ СОБСТВЕННЫМ УМОМ

Отчет о проверке на заимствования №1

Автор: Ильченко Егор zzodoo@gmail.com / ID: 2302102
Проверяющий: Ильченко Егор (zzodoo@gmail.com / ID: 2302102)

Отчет предоставлен сервисом «Антиплагиат»- http://users.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 7
Начало загрузки: 14.05.2019 18:31:17
Длительность загрузки: 00:00:04
Имя исходного файла: магистерская
Размер текста: 3435 кБ
Символов в тексте: 58010
Слов в тексте: 7026
Число предложений: 466

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 14.05.2019 18:31:22
Длительность проверки: 00:00:03
Комментарии: не указано
Модули поиска: Модуль поиска Интернет

ЗАИМСТВОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
5,95%	0%	94,05%

Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты; общеупотребительные выражения; фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Заимствования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Доля в тексте	Источник	Ссылка	Актуален на	Модуль поиска	Блоков в отчете	Блоков в тексте
[01]	1,73%	1,73%	Методика оценки персонал...	http://knowledge.allbest.ru	раньше 2011	Модуль поиска Интернет	6	6
[02]	0,91%	0,91%	Модульное тестирование	http://ru.wikipedia.org	09 Апр 2018	Модуль поиска Интернет	4	4
[03]	0,88%	0,88%	RabbitMQ tutorial 2 — Очере...	http://habrahabr.ru	28 Окт 2014	Модуль поиска Интернет	6	6

Еще источников: 8
Еще заимствований: 2,43%