Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет по курсовой работе
по дисциплине «Программное обеспечение распределенных вычислительных систем»

Система управления проектами

Выполнил:
студент группы 23541/3
Раскин А.Р.

Преподаватель:
Стручков И. В.

Санкт-Петербург
2019 г.

Содержание

1 Введение

В рамках курса было необходимо разработать приложение, позволяющее продемонстрировать применение основных принципов разработки программного обеспечения. В частности, в приложении необходимо было выделить следующие компоненты:

- слой бизнесс-логики;
- слой хранения данных;
- слой представления.

Было решено разработать приложение, основное назначение которого — упростить процесс разработки программных проектов для какой-либо группы разработчиков.

В ней определены следующие объекты:

- **Проект** У каждого проекта есть определенная команда разработчиков, тестировщиков и один менеджер. Также к проекту может быть привязан тимлидер. У проекта определены различные майлстоуны. К каждому проекту могут быть привязаны сообщения об ошибках.
- **Майлстоун** Одна из итераций цикла разработки проекта. Привязан к определенным датам. К майлстоунам привязаны определенные тикеты (задания). Майлстоун имеет определенный статус: открыт, активен или закрыт. Майлстоун может быть закрыт только когда все его тикеты выполнены. В каждый момент времени у проекта может быть только один майлстоун.
- **Тикет** Определенное задание для разработчиков. Может быть выдано определенной группе разработчиков. Привязан к определенному проекту и майлстоуну. Имеет статус: новый, принятый, в процессе выполнения, выполнен.
- **Сообщение об ошибке** Отчет о найденной ошибке в проекте. Привязан к определенному проекту. Имеет статус: новый, исправленный, протестированный, закрытый.

2 Роли

В системе определены следующие роли для пользователей:

- менеджер;
- тимлидер;
- разработчик;
- тестировщик.

Для каждого проекта у пользователя определена своя роль (если он участвует в разработке данного проекта).

У всех пользователей системы есть возможность:

- зарегистрироваться;
- просмотреть все проекты в которых они участвуют;
- посмотреть список заданий, который был им выдан;

- посмотреть список отчетов об ошибках, которые ему надо исправить;
- создать новый проект.

Функции менеджера проекта:

- Управление пользователями:
  - назначение тимлидера
  - добавление разработчика к проекту
  - добавление тестировщика к проекту
- Управление майлстоунами
  - создание нового майлстоуна
  - изменение статуса майлстоуна
- Управление тикетами
  - создание нового тикета
  - привязка разработчика к тикету
  - проверка выполнения тикета

Функции тимлидера:

- Управление тикетами
  - создание нового тикета
  - привязка разработчика к тикету
  - проверка выполнения тикета
- Выполнение тикетов

Функции разработчика:

- Выполнение тикетов
- Создание сообщений об ошибках
- Исправление сообщений об ошибках

Функции тестировщика:

- Тестирование проекта
- Создание сообщений об ошибках
- проверка исправления сообщений об ошибках

3 Описание бизнес процессов

3.1 Разработка проекта.

- Участники:
  - менеджер;
  - тимлидер;
  - разработчик;
  - тестировщик.
- Сущности:
  - прокет;
  - milestone;
  - тикет;
  - сообщение об ошибке.
- Этапы:
  - создание проекта;
  - определение команды разработчиков;
  - назначение тимлидера;
  - итеративный процесс разработки проекта по milestone;
  - параллельно исправление всех появляющихся ошибок;
  - завершение проекта.

3.2 Выполнение тикета

- Участники:
  - менеджер;
  - тимлидер;
  - разработчик.
- Сущности:
  - прокет;
  - milestone;
  - тикет.
- Этапы:
  - создание тикета;
  - определение исполнителей;
  - подтверждение получения тикета исполнителями;
  - выполнение задание из тикета;
  - проверка выполнения задания и закрытие тикета.

3.3 Обработка отчетов об ошибках

- Участники:
  - разработчик;
  - тестировщик.
- Сущности:
  - прокет;
  - сообщение об ошибке.
- Этапы:
  - создание отчета об ошибке
  - исправление ошибки разработчиком
  - проверка исправления тестировщиком.

4 Варианты использования

Все варианты использования, определенные в системе, приведены на рисунке 1.

4.1 Диаграмма вариантов использования

4.2 Описание вариантов использования

4.2.1 Запись пациента на прием

Разработка проекта:

- менеджер создает новый проект;
- менеджер добавляет разработчиков в проект;
- менеджер добавляет тестировщиков в проект;
- менеджер определяет тимлидера;
- менеджер определяет майлстоуны проекта и назначает даты;
- менеджер выдает тикеты разработчикам;
- разработчики выполняют тикеты ближайшего майлстоуна;
- предыдущие три шага итеративно повторяются до завершения проекта;
- проект завершается.

Выполнение тикета:

- менеджер/тимлидер создает новый тикет;
- менеджер/тимлидер определяет разработчиков-исполнителей. Тикет получает статус "новый";
- разработчик получает уведомление о новом тикете и меняет его статус на "принят";

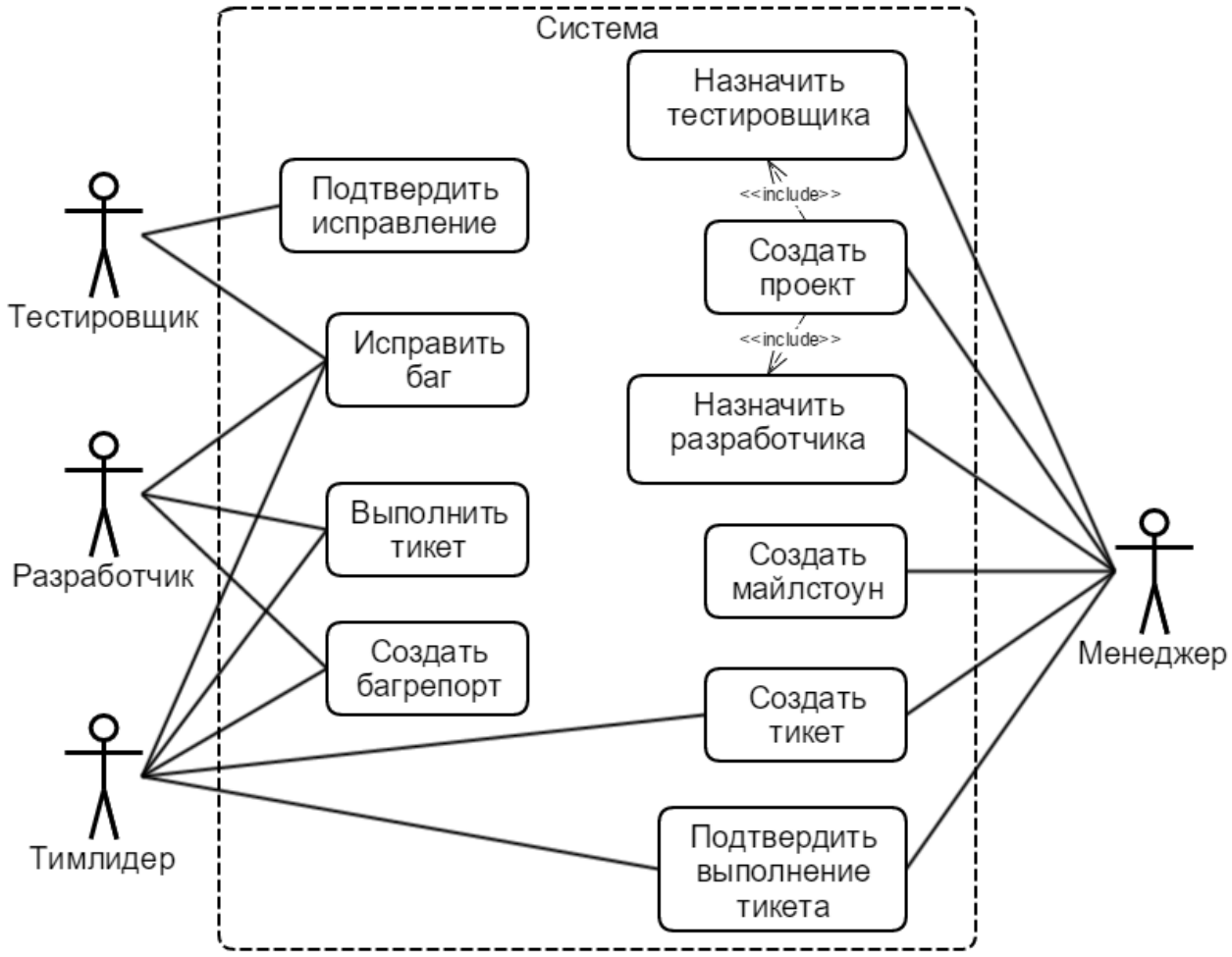

Рис. 1: Варианты использования

- разработчик приступает к выполнению задания. Тикет получает статус "в процессе выполнения";
- разработчик выполняет задание и меняет статус тикета на "выполнен";
- менеджер/тимлидер проверяет и подтверждает выполнение задания. Тикет получает задание "закрыт".

Обработка ошибки:

- разработчик/тестировщик создает отчет с описанием ошибки. Отчет получает статус "новый";
- разработчик принимает уведомление и меняет статус отчета на "активен";
- разработчик исправляет баг и меняет статус на "исправлен";
- тестировщик проверяет исправление и закрывает отчета указав ему статус "закрыт".

5 Диаграммы последовательностей

Диаграмма последовательности бизнес процесса разработки проекта приведена на рисунке 2.

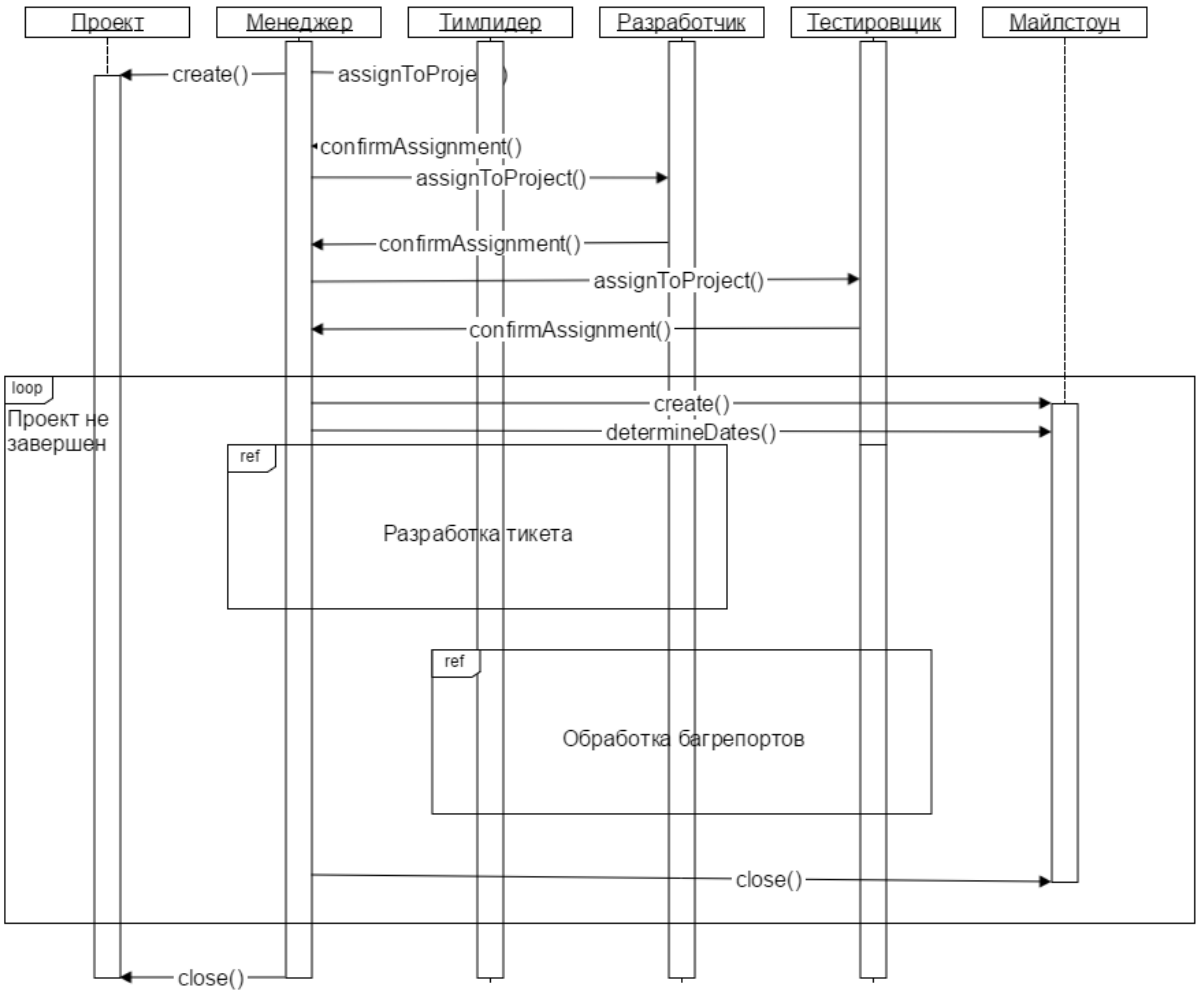

Рис. 2: Разработка проекта

Диаграмма последовательности бизнес процесса выполнения тикета приведена на рисунке 3.

Диаграмма последовательности бизнес процесса исправления ошибки приведена на рисунке 4.

6 Слой бизнес-логики

Слой бизнес логики был реализован с помощью использования шаблона "Модель предметной области". Все классы, реализующий слой бизнес-логики разделены на два пакета:

- `entity` — в данном пакете описаны все сущности, определенные в системе;
- `logic` — в данном пакете описаны все классы, реализующие бизнес-логику системы.

6.1 Пакет `entity`

В данном пакете описаны все сущности, над которыми оперирует бизнес-логика. Диаграмма классов пакета `entity` приведена на рисунке 5.

Все классы данного пакета являются сущностными, т.е. они лишь хранят необходимые данные и имеют методы для получения/записи этих данных. Рассмотрим классы данного пакета более подробно.

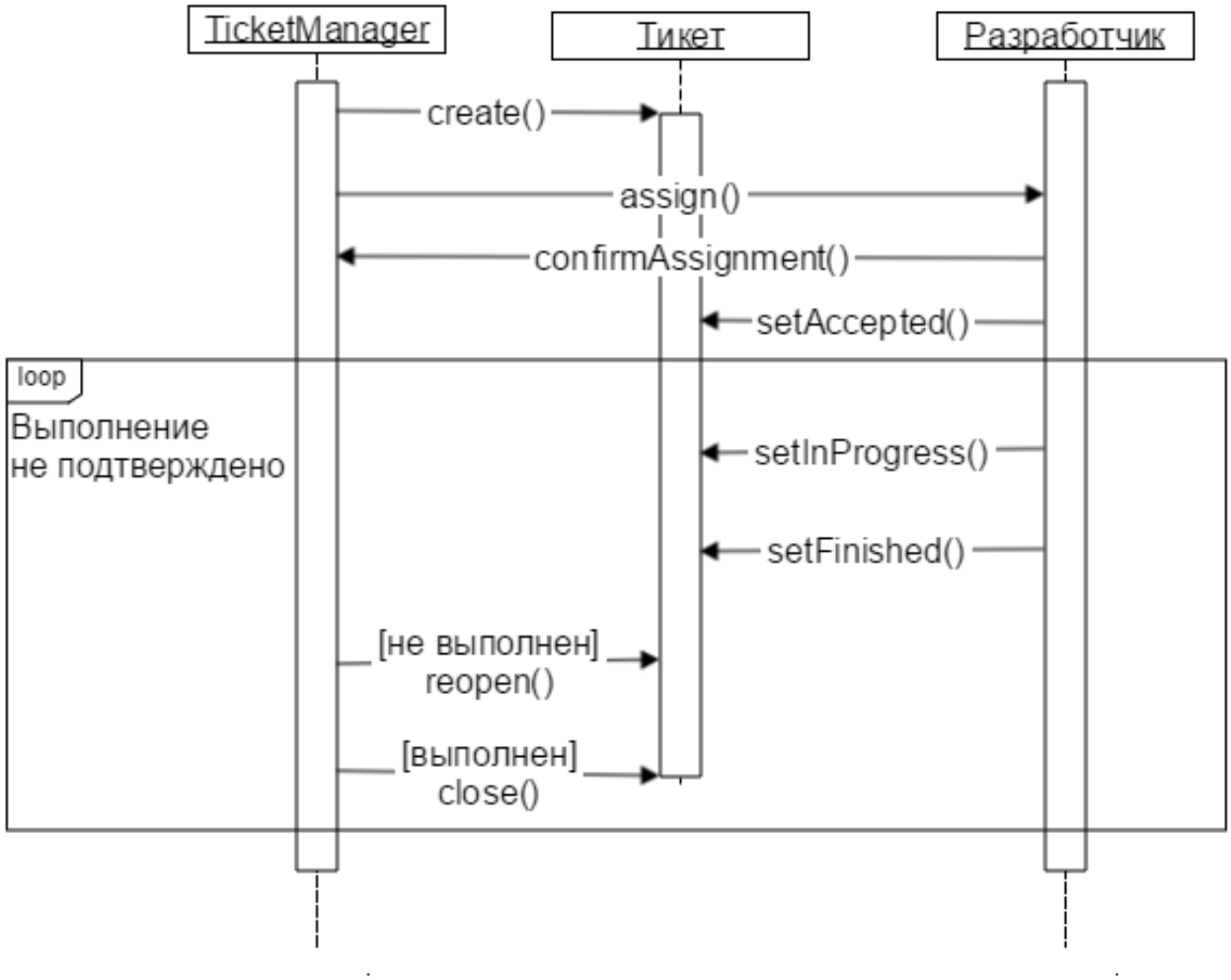

Рис. 3: Выполнение тикета

- `User` — класс, описывающий пользователя. Данный класс хранит: уникальный идентификатор пользователя; уникальный логин пользователя; имя; пароль и список уведомлений, полученных данным пользователем.

- `Project` — класс, описывающий проект. Данный класс хранит: уникальный идентификатор пользователя; уникальное имя проекта; указатель на менеджера проекта; указатель на тимлидера проекта (если он определен); список разработчиков; список тестировщиков; список майлстоунов; список сообщений об ошибках.

- `BugReport` — класс, описывающий сообщение об ошибке. Данный класс хранит: уникальный идентификатор отчета; указатель на проект (к которому он привязан); указатель на автора отчета об ошибке; указатель на разработчика, который исправляет данную ошибку (если он определен); описание ошибки; список комментариев к данной ошибке.

- `Milestone` — класс, описывающий майлстоун. Класс хранит: уникальный идентификатор; проект, к которому привязан; текущий статус; предполагаемое время начало майлстоуна; время, когда майлстоун действительно был начат; предполагаемое время завершения майлстоуна; время, когда майлстоун был дейтсвительно завершен; список тикетов, привязанных к данному майлстоуну.

- `Ticket` — класс, описывающий тикет. Класс хранит: уникальный идентификатор; указатель на майлстоун, к которому привязан; указатель на создателя тикета; теку-

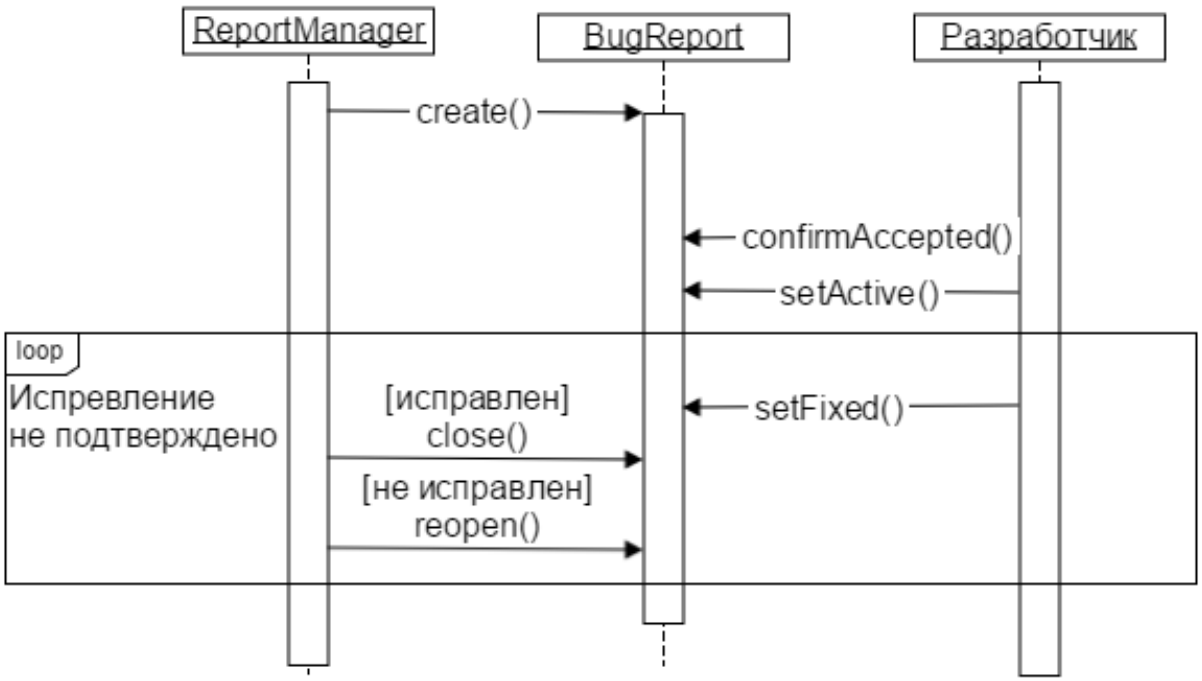

Рис. 4: Исправление ошибки

щий статус; список разработчиков, назначенных на выполнение тикета; дату создания тикета; описание задания; список комментариев к данному тикету.

- `Comment` — класс, описывающий комментарий к тикету или отчету об ошибке. Класс хранит: уникальный идентификатор; дату создания комментария; указатель на пользователя, который оставил комментарий; текст комментария.

- `Message` — класс, описывающий уведомления пользователя. Класс хранит: уникальный идентификатор; дату создания уведомления; указатель на пользователя, которому отправлено уведомление; текст уведомления.

6.2 Пакет `logic`

В данном пакете описана вся бизнес-логика системы. Диаграмма классов пакета `logic` приведена на рисунке 6.

Бизнес логика в данном пакете описывается на уровне интерфейсов, а классы соответствуют ролям и реализуют соответствующие интерфейсы. Рассмотрим их более подробно.

- `UserInterface` — базовый интерфейс пользователя. Имеет методы:
  - `getUser()` — получить пользователя, к которому привязан текущий объект.
  - `addMessage(String message)` — добавить новое уведомление данному пользователю.

- `UserImpl` — класс, который реализует базовый интерфейс пользователя.

- `UserManager` — интерфейс, в котором реализованы методы управления пользователями проекта. Расширяет интерфейс `UserInterface`. Методы могут выбросить два

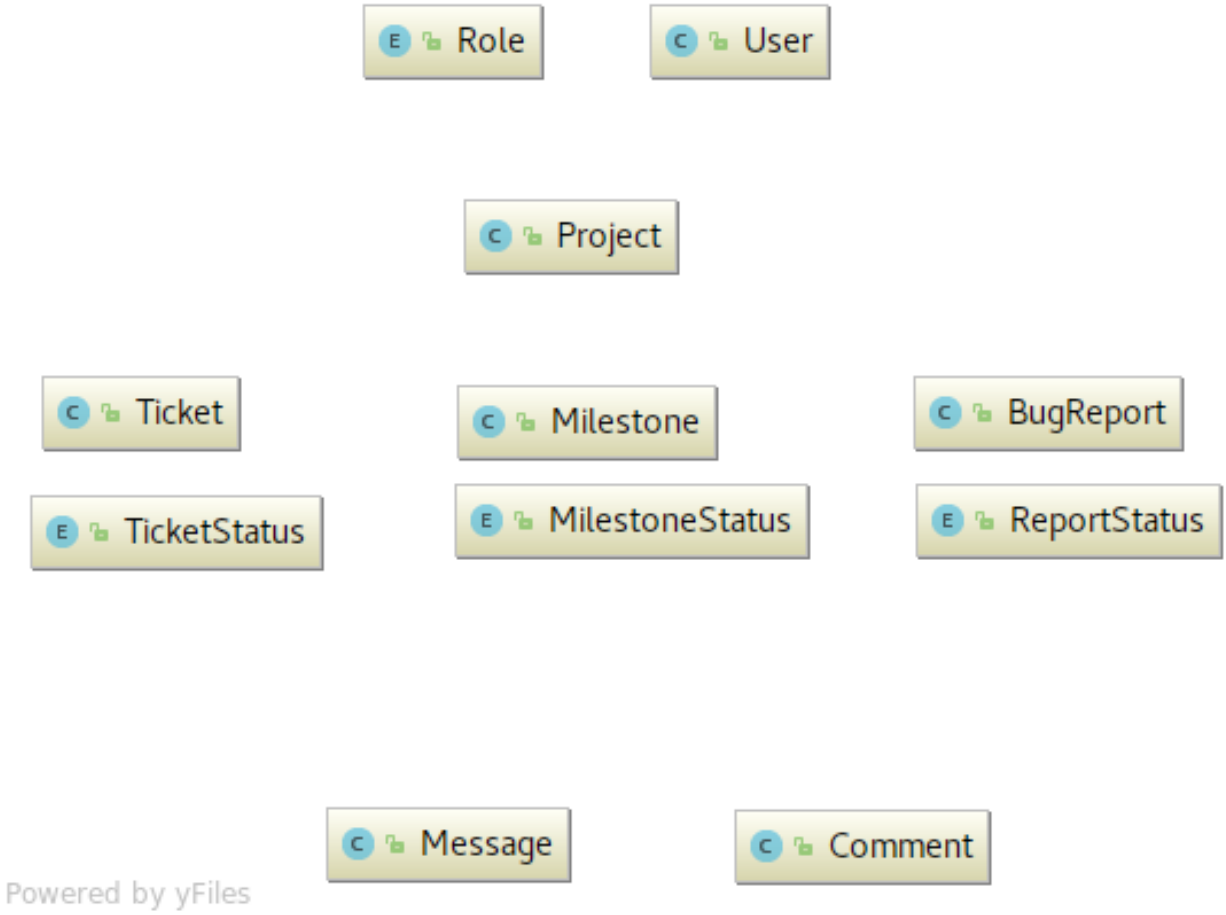

Рис. 5: Диаграмма классов пакета `entity`

исключения: `MultipleRoleException` если какому-то пользователю пытаются присвоить две роли в проекте и `NoRightsException` если у текущего пользователя нет прав на управление пользователями проекта. Имеет методы:

  - `setTeamLeader(Project project, User teamLeader)` — добавить нового тимлидера к проекту.
  - `addDeveloper(Project project, User developer)` — добавить нового разработчика в проект.
  - `addTester(Project project, User tester)` — добавить нового тестировщика в проект.

- `MilestoneManager` — интерфейс, в котором реализованы методы управления майлстоуном. Расширяет интерфейс `UserInterface`. Имеет методы:

  - `createMilestone(Project project, Date start, Date end)` — добавить новый майлстоун в проект.
  - `activateMilestone(Milestone milestone)` — сделать майлстоун активным.
  - `closeMilestone(Milestone milestone)` — закрыть майлстоун.

  Выбрасывает следующие исключения:

  - `NoRightsException` — если у текущего пользователя нет прав на управление проектами.

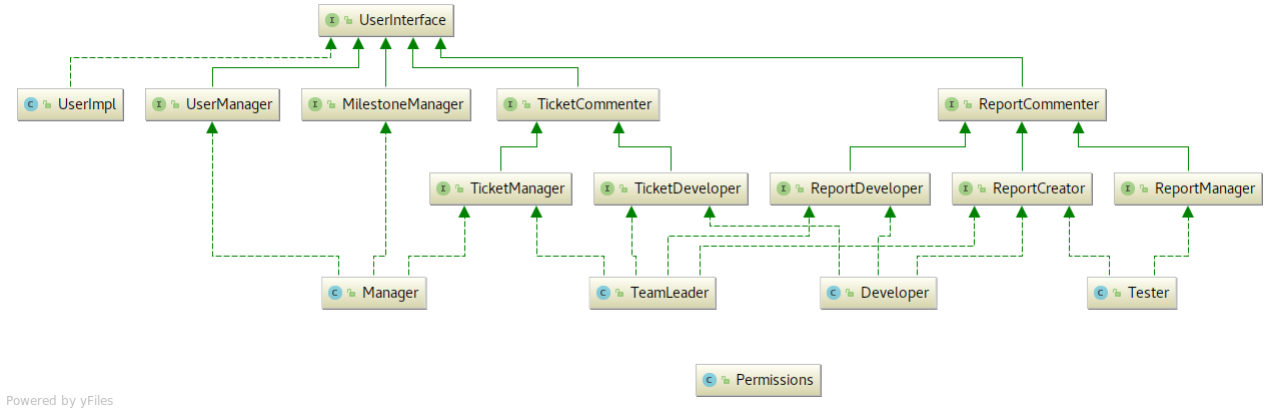

Рис. 6: Диаграмма классов пакета `logic`

  - `TwoActiveMilestonesException` — если пытаемся сделать одновременно два майлстоуна активными.
  - `WrongStatusException` — если пытаемся установить майлстоуну неправильный статус.
  - `MilestoneTicketNotClosedException` — если пытаемся закрыть майлстоун, у которого закрыты не все тикеты.

- `TicketCommenter` — интерфейс, который реализует метод комментирования тикетов. Расширяет интерфейс `UserInterface`.

- `TicketManager` — интерфейс, в котором реализованы методы управления тикетом. Расширяет интерфейс `TicketCommenter`. Имеет методы:

  - `checkTicketManagerPermissions(Ticket ticket)` — проверить права текущего пользователя на управление данным тикетом.
  - `createTicket(Milestone milestone, String task)` — создать новый тикет.
  - `addAssignee(Ticket ticket, User developer)` — добавить нового разработчика в тикет.
  - `reopenTicket(Ticket ticket)` — переоткрыть тикет.
  - `closeTicket(Ticket ticket)` — закрыть тикет.

  Выбрасывает следующие исключения:

  - `NoRightsException` — если у текущего пользователя нет прав на управление тикетом.
  - `MilestoneAlreadyClosedException` — если пытаемся создать новый тикет в майлстоуне, который уже закрыт.

- `TicketDeveloper` — интерфейс, в котором реализованы методы управления тикетом. Расширяет интерфейс `TicketCommenter`. Имеет методы:

  - `checkTicketDeveloperPermissions(Ticket ticket)` — проверить права текущего пользователя на разработку данного тикета.
  - `acceptTicket(Ticket ticket)` — поставить тикету статус "принят".
  - `setInProgress(Ticket ticket)` — поставить тикету статус "выполняется".

- `finishTicket(Ticket ticket)` — поставить тикету статус "завершен".

Выбрасывает исключение `NoRightsException` если у текущего пользователя нет прав на разработку тикета.

- `ReportCommenter` — интерфейс, который реализует метод комментирования отчетов об ошибках. Расширяет интерфейс `UserInterface`.

- `ReportCreator` — интерфейс, в котором реализованы методы создания отчетов об ошибках. Расширяет интерфейс `ReportCommenter`. Имеет методы:

  - `checkReportCreatorPermissions(BugReport report)` — проверить права текущего пользователя на создание отчетов об ошибках.
  - `createReport(Project project, String description)` — создать новый отчет об ошибке.

  Выбрасывает исключение `NoRightsException` если у текущего пользователя нет прав на создание отчетов об ошибках.

- `ReportDeveloper` — интерфейс, в котором реализованы методы исправления ошибок проекта. Расширяет интерфейс `ReportCommenter`. Имеет методы:

  - `checkReportDeveloperPermissions(BugReport report)` — проверить права текущего пользователя на исправление ошибок проекта.
  - `acceptReport(BugReport report)` — принять отчет об ошибке на исправление.
  - `fixReport(BugReport report)` — установить статус "исправлен"отчету об ошибке.

  Выбрасывает следующие исключения:

  - `NoRightsException` — если у текущего пользователя нет прав на исправление ошибок проекта.
  - `AlreadyAcceptedException` — если пытаемся принять на исправление отчет, который уже принят другим пользователем.

- `ReportManager` — интерфейс, в котором реализованы методы управления отчетами об ошибках. Расширяет интерфейс `ReportCommenter`. Имеет методы:

  - `checkReportManagerPermissions(BugReport report)` — проверить права текущего пользователя на управление отчетами об ошибках.
  - `reopenReport(BugReport report)` — переоткрыть отчет об ошибке, если она не была исправлена.
  - `closeReport(BugReport report)` — закрыть отчет об ошибке.

  Выбрасывает исключение `NoRightsException` если у текущего пользователя нет прав на управление отчетами об ошибках.

- `Manager` — класс, который описывает роль менеджера проекта. Реализует интерфейсы `MilestoneManager`, `UserManager` и `TicketManager`.

- `TeamLeader` — класс, который описывает роль тимлидера проекта. Реализует интерфейсы `ReportCreator`, `ReportDeveloper`, `TicketManager`, `TicketDeveloper`.

- `TeamLeader` — класс, который описывает роль разработчика проекта. Реализует интерфейсы `ReportCreator`, `ReportDeveloper`, `TicketDeveloper`.
- `TeamLeader` — класс, который описывает роль тестировщика проекта. Реализует интерфейсы `ReportCreator`, `ReportManager`.
- `Permissions` — класс, который хранит права пользователя для какого-то проекта в виде битовой маски.

7 Слой хранения данных

В качестве СУБД была выбрана система MySQL. Для работы с базой данных был использован Java Persistance API, в частности его реализация в библиотеке Hibernate. В базе данных хранятся все сущности, описанные в пакете `entity`. Описание сущностей производится с помощью специальных аннотаций JPA. Пример сущностного класса `User` приведен в листинге 1.

Листинг 1: Описание сущности

```
@Entity
@Table(name = "USERS")
public class User {

@Id
@Column(name = "ID")
@GeneratedValue
private Long id;

@Column(name = "LOGIN", unique = true, nullable = false)
private String login;

@Column(name = "NAME")
private String name;

@Column(name = "PASSWORD")
private String password;

@JsonIgnore
@OneToMany(fetch = FetchType.EAGER, mappedBy = "owner")
private List<Message> messages = new ArrayList<>();
}
```

Также, для того чтобы Hibernate нашел все сущности, необходимо описать все сущностные классы в файле persistance.xml(листинг 2).

Листинг 2: Файл persistance.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">

<persistence-unit name="my-pms">
<class>com.kspt.pms.entity.BugReport</class>
<class>com.kspt.pms.entity.Comment</class>
<class>com.kspt.pms.entity.Message</class>
<class>com.kspt.pms.entity.Milestone</class>
<class>com.kspt.pms.entity.Project</class>
<class>com.kspt.pms.entity.Ticket</class>
<class>com.kspt.pms.entity.User</class>
</persistence-unit>

</persistence>
```

Для каждой сущности был реализован свой репозиторий, в котором определены методы для поиска и сохранения сущностей. Репозитории были реализованы с помощью

библиотеки Spring Data. Данная библиотека позволяет описывать только интерфейс, и по названиям методов этого интерфейса умеет автоматически генерировать его реализацию. Это очень сильно упрощает работу с базой данных. Рассмотрим данные репозитории более подробно.

- `BugReportRepository`
  - `Optional<BugReport> findById(Long id)` — поиск по идентификатору.
  - `Collection<BugReport> findByProjectName(String name)` — поиск всех отчетов, принадлежащих проекту с заданным именем.
  - `Collection<BugReport> findByCreatorLogin(String login)` — поиск всех отчетов, созданных пользователем с заданным логином.
  - `Collection<BugReport> findByDeveloperLogin(String login)` — поиск всех отчетов, исправляемых пользователем с заданным логином.
- `CommentRepository`
  - `Optional<Comment> findById(Long id)` — поиск по идентификатору.
  - `Collection<Comment> findByUserLogin(String login)` — поиск всех комментариев, оставленных пользователем с заданным логином.
- `MessageRepository`
  - `Collection<Message> findByOwnerLogin(String login)` — поиск всех уведомлений, принадлежащих пользователю с заданным логином.
- `MilestoneRepository`
  - `Optional<Milestone> findById(Long id)` — поиск по идентификатору.
  - `Collection<Milestone> findByProjectName(String name)` — поиск всех майлстоуну, принадлежащих проекту с заданным именем.
- `ProjectRepository`
  - `Optional<Project> findByName(String name)` — поиск по имени.
  - `Collection<Project> findByManagerLogin(String login)` — поиск по менеджеру.
  - `Collection<Project> findByTeamLeaderLogin(String login)` — поиск по тимлидеру.
  - `Collection<Project> findByDevelopersContaining(User user)` — поиск по разработчику.
  - `Collection<Project> findByTestersContaining(User user)` — поиск по тестировщику.
- `TicketRepository`
  - `Optional<Ticket> findById(Long id)` — поиск по идентификатору.
  - `Collection<Ticket> findByMilestoneId(Long id)` — поиск майлстоуну.
  - `Collection<Ticket> findByAssigneesContaining(User user)` — поиск по исполнителю.

- `Collection<Ticket> findByCreatorLogin(String login)` — поиск по создателю.

- `UserRepository`

  - `Optional<User> findByLogin(String login)` — поиск по логину.

Для того, чтобы все эти библиотеки правильно проинициализировались, необходимо в классе конфигурации определить источник данных (листинг 3).

Листинг 3: Конфигурация источника данных

```
@Bean
public DataSource dataSource() {
DriverManagerDataSource dataSource = new DriverManagerDataSource();
dataSource.setDriverClassName(env.getRequiredProperty("jdbc.driverClassName"));
dataSource.setUrl(env.getRequiredProperty("jdbc.url"));
dataSource.setUsername(env.getRequiredProperty("jdbc.username"));
dataSource.setPassword(env.getRequiredProperty("jdbc.password"));
return dataSource;
}

@Bean
public JdbcTemplate jdbcTemplate(DataSource dataSource) {
JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
jdbcTemplate.setResultsMapCaseInsensitive(true);
return jdbcTemplate;
}

public Properties additionalProreties() {
Properties properties = new Properties();
//        properties.setProperty("hibernate.show_sql", "true");
return properties;
}

@Bean
public EntityManagerFactory entityManagerFactory() {
HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
vendorAdapter.setGenerateDdl(true);

LocalContainerEntityManagerFactoryBean factory = new
    LocalContainerEntityManagerFactoryBean();
factory.setJpaVendorAdapter(vendorAdapter);
factory.setPackagesToScan("com.kspt.pms");
factory.setDataSource(dataSource());
factory.setPersistenceUnitName("my-pms");
factory.setPersistenceProviderClass(HibernatePersistenceProvider.class);
factory.setJpaProperties(additionalProreties());
factory.afterPropertiesSet();
return factory.getObject();
}

@Bean
public PlatformTransactionManager transactionManager() {
JpaTransactionManager txManager = new JpaTransactionManager();
txManager.setEntityManagerFactory(entityManagerFactory());
return txManager;
}
```

8 Слой представления

В качестве фреймворка для слоя представления была выбрана технология Spring MVC. Данная технология позволяет реализовать паттерн Model-View-Controller при помощи слабо связанных компонентов. В качестве модели выступает слой бизнес-логики приложения. Контроллеры реализованы в виде RESTful контроллеров с помощью фреймворка Spring. Представление реализовано с помощью JavaScript фреймворка AngularJS и HTML страниц. Работа происходит по следующему сценарию:

1. RESTful сервис принимает запрос от пользователя;
2. сервис вызывает соответствующие методы бизнес логики для обработки запроса;
3. сервис возвращает ответ на запрос в виде JSON-объекта;
4. AngularJS клиент получает ответ от сервера и рэндерит соответствующую HTML-страницу и показывает его в браузере.

Рассмотрим более подробно REST API, предоставляемый сервером.

- `UserController`
  - URL: `"rest/user/:login"` — получение пользователя с логином `login`
  - URL: `"rest/user/:login"`, метод POST — регистрация нового пользователя
  - URL: `"rest/user/:login/authenticate"` — аутентификация пользователя
  - URL: `"rest/user/:login/messages"` — получить все уведомления для пользователя
  - URL: `"rest/user/:login/projects"` — получить все проекты, в которых участвует пользователя
  - URL: `"rest/user/:login/projects"`, метод POST — добавление нового проекта пользователю
  - URL: `"rest/user/:login/managed_tickets"` — получить все тикеты, которыми руководит пользователь
  - URL: `"rest/user/:login/assigned_tickets"` — получить все тикеты, которые разрабатывает пользователь
  - URL: `"rest/user/:login/managed_reports"` — получить все ошибки, которыми руководит пользователь
  - URL: `"rest/user/:login/assigned_reports"` — получить все ошибки, которые разрабатывает пользователь
- `TicketController`
  - URL: `"rest/ticket/:id"` — получение тикета с идентификатором `id`
  - URL: `"rest/ticket/:id/permissions"` — получение прав пользователя для тикета
  - URL: `"rest/ticket/:id/milestone"` — получение майлстоуна тикета
  - URL: `"rest/ticket/:id"`, метод PUT — установить новый статус тикету
  - URL: `"rest/ticket/:id/assignees"` — получить всех разработчиков тикета
  - URL: `"rest/ticket/:id/assignees"`, метод POST — добавить нового разработчика в тикет
  - URL: `"rest/ticket/:id/comments"` — получить все комментарии тикета
  - URL: `"rest/ticket/:id/comments"`, метод POST — добавить новый комментарий тикету
- `ReportController`
  - URL: `"rest/report/:id"` — получение отчета с идентификатором `id`

  - URL: "rest/report/:id/permissions" — получение прав пользователя для отчета
  - URL: "rest/report/:id/project" — получение проекта отчета
  - URL: "rest/report/:id", метод PUT — установить новый статус отчету
  - URL: "rest/report/:id/comments" — получить все комментарии отчета
  - URL: "rest/report/:id/comments", метод POST — добавить новый комментарий отчету

- ProjectController
  - URL: "rest/project/:name" — получение проекта с именем name
  - URL: "rest/project/:name", метод PUT — обновить проект с именем name
  - URL: "rest/project/:name/permissions" — получение прав для проекта
  - URL: "rest/project/:name/reports" — получение ошибок для проекта
  - URL: "rest/project/:name/reports", метод POST — добавление ошибки в проект
  - URL: "rest/project/:name/milestones" — получение майлстоунов для проекта
  - URL: "rest/project/:name/milestones", метод POST — добавление майлстоуна в проект
  - URL: "rest/project/:name/developers" — получение разработчиков проекта
  - URL: "rest/project/:name/developers", метод POST — добавление разработчика в проект
  - URL: "rest/project/:name/testers" — получение тестировщиков проекта
  - URL: "rest/project/:name/testers", метод POST — добавление тестировщика в проект

- MilestoneController
  - URL: "rest/milestone/:id" — получение майлстоуна с идентификатором id
  - URL: "rest/milestone/:id/permissions" — получение прав пользователя для майлстоуна
  - URL: "rest/milestone/:id/project" — получение проекта майлстоуна
  - URL: "rest/milestone/:id", метод PUT — установить новый статус майлстоуну
  - URL: "rest/milestone/:id/tickets" — получить все тикеты майлстоуна
  - URL: "rest/milestone/:id/tickets", метод POST — добавить новый тикеты майлстоуну

При возникновении неполадок на стороне сервера, выбрасывается соответствующее исключение. Данное исключение перехватывается с помощью специальных инструментов и генерируется HTTP ответ, который содержит код и сообщение с описанием ошибки.

Клиентская сторона посылает запросы к соответствующим RESTful сервисам и затем рэндерит соответствующую HTML страницу. Работа с REST API происходит с помощью библиотеки Angular Resource. Пример angular-контроллера приведен на листинге

Листинг 4: Angular-контроллер

```
/**
* Created by kivi on 18.12.17.
*/

function ReportService($resource) {
return $resource('rest/report/:id', {id: '@id'});
}

function ReportPermService($resource) {
return $resource('rest/report/:id/permissions?user=:login', {id: '@id', login: '@login'});
}

function ReportProjectService($resource) {
return $resource('rest/report/:id/project', {id: '@id'});
}

function ReportCommentService($resource) {
return $resource('rest/report/:id/comments?user=:login', {id: '@id', login: '@login'});
}

function ReportController($scope, $http, $routeParams,
ReportService,
ReportPermService,
ReportCommentService,
ReportProjectService,
InfoShareService) {
function url() {
return {id: $routeParams.id};
}
function url_with_login(login) {
return {id: $routeParams.id, login:login};
}

this.user = InfoShareService.getUser();
this.permissions = ReportPermService.get(url_with_login(this.user.login));
this.project = ReportProjectService.get(url());
this.instance = ReportService.get(url());
this.comments = ReportCommentService.query(url());

this.changeStatus = function (status) {
$http.put('rest/report/' + this.instance.id + '?user=' + this.user.login, status)
.then(function () {
this.update();
}.bind(this), function (error) {
alert(error.data.message);
});
};

this.commentReport = function () {
if (isEmpty($scope.reportComment)) {
alert("Enter comment message");
} else {
var comment = new ReportCommentService();
comment.description = $scope.reportComment;
comment.$save(url_with_login(this.user.login), function () {
$scope.reportComment = "";
this.updateComments();
}.bind(this), function (error) {
alert(error.data.message);
});
}
};

this.update = function () {
this.instance = ReportService.get(url());
};

this.updateComments = function () {
this.comments = ReportCommentService.query(url());
}
}

app
.factory('ReportService', ReportService)
.factory('ReportPermService', ReportPermService)
```

```
  .factory('ReportProjectService', ReportProjectService)
  .factory('ReportCommentService', ReportCommentService)
  .controller('ReportController', ReportController);
```

Скриншоты пользовательского интерфейса приведены на рисунках 7 – 12.

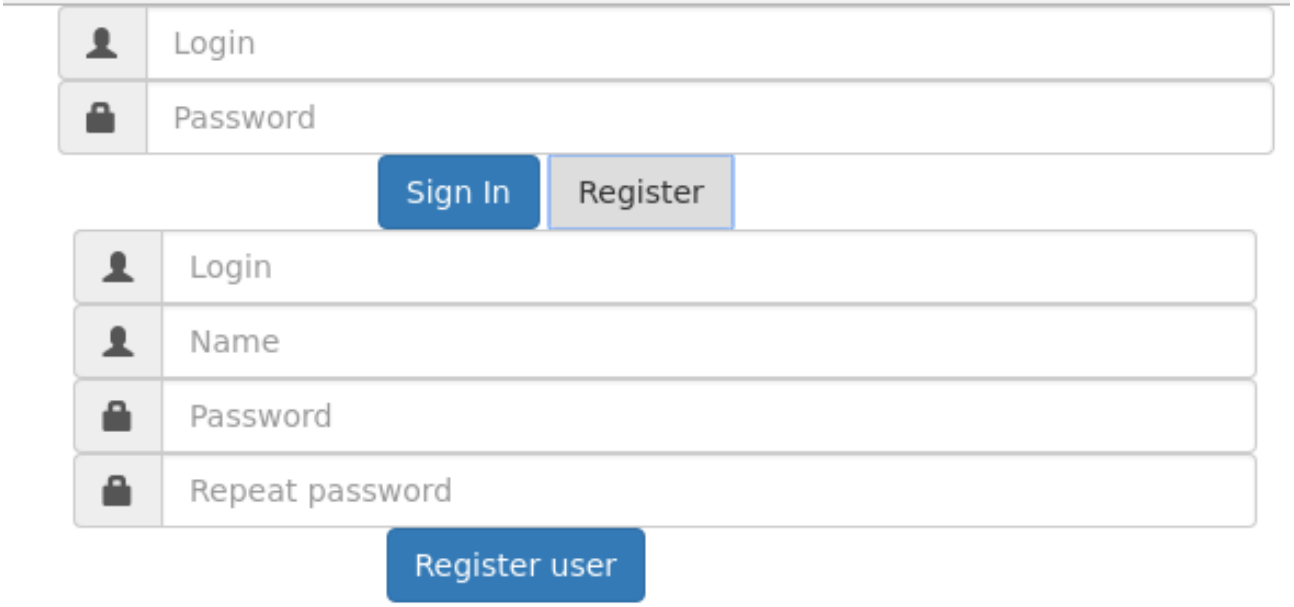

Рис. 7: Страница входа

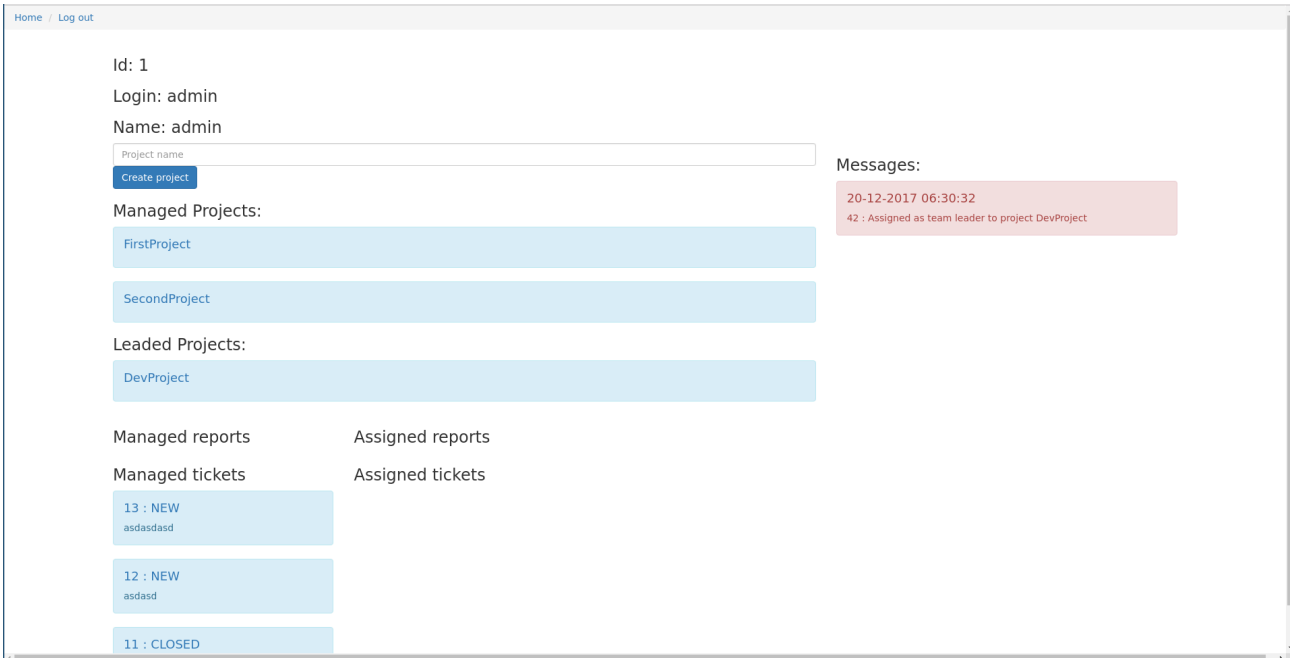

Рис. 8: Страница пользователя

9 Тестирование

Было проведено ручное функциональное тестирование приложения. Были протестированы все бизнес-процессы в приложении, проверена обработка ошибочных ситуаций. Список проводимых тестов:

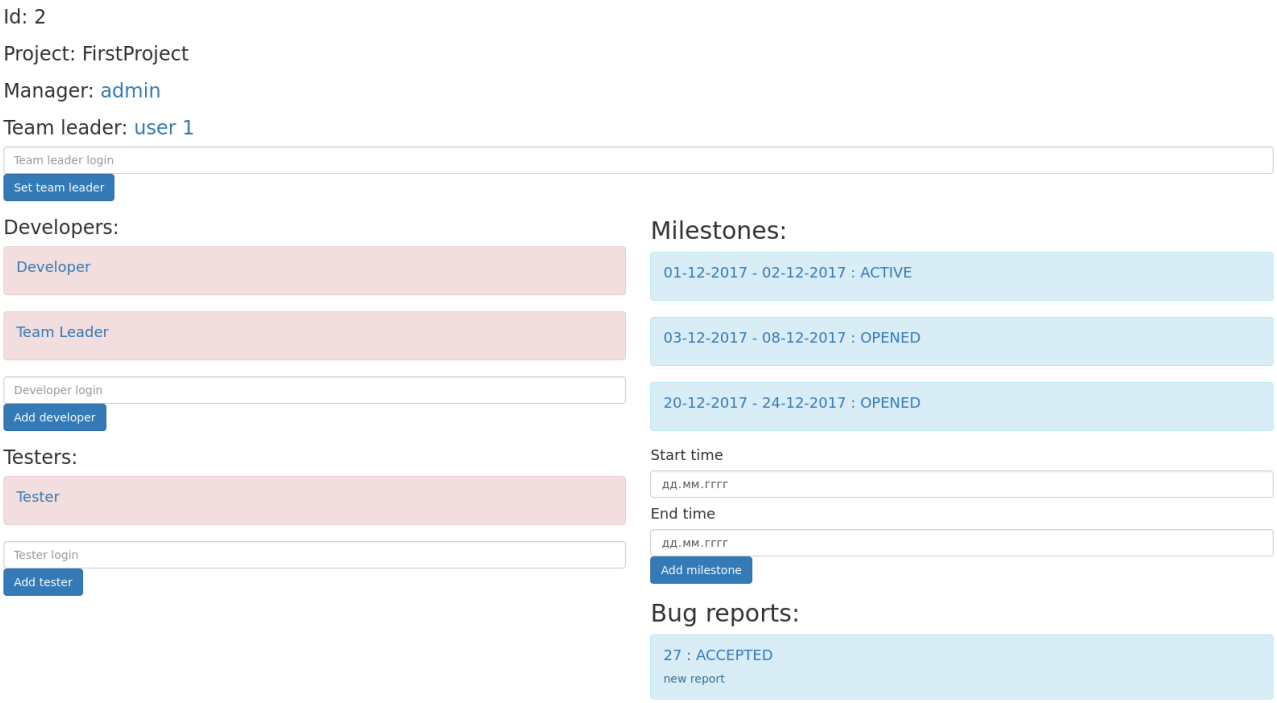

Рис. 9: Страница проекта

- **Попытка добавить пользователя, который уже существует**. Окрываем окно регистрации нового пользователя, вводим данные пользователя. В поле `Login` указываем логин уже зарегистрированного пользователя. Нажимаем кнопку `Register`. В результате появится окно с сообщением «User `login` already exists».

- **Регистрация нового пользователя**. Открыть окно регистрации пользователя. Ввести уникальные данные пользователя. Нажать кнопку `Register`. Регистрация должна пройти успешно. Должно отрыться окно входа в систему.

- **Попытка войти в систему с неправильным логином или паролем**. В окне входа в систему ввести неправильные данные пользователя (логин или пароль). Нажать кнопку `Sign In`. Должно появиться окно с сообщением «Incorrect login or password».

- **Вход в систему**. В окне входа в систему ввести правильные данные пользователя. Нажать кнопку `Sign In`. Должно открыться окно указанного пользователя.

- **Создание проекта с дублирующимся именем**. В окне пользователя в строке добавления нового проекта ввести имя нового проекта, которое полностью совпадает с именем уже существующего проекта. Нажать кнопку `Create project`. Должно появиться окно ошибки с сообщением «Project `name` already exists».

- **Создание нового проекта**. В окне пользователя в строке добавления нового проекта ввести уникальное имя нового проекта. Нажать кнопку `Create project`. В списке «Managed projects» должен появиться новый проект с указанным именем.

- **Установка несуществующего пользователя тимлидером**. Открыть окно нового проекта. Должна быть доступна кнопка `Set team leader`. Ввести несуществующий логин пользователя и нажать на кнопку. Должно появиться окно ошибки с сообщением «User `login` not found».

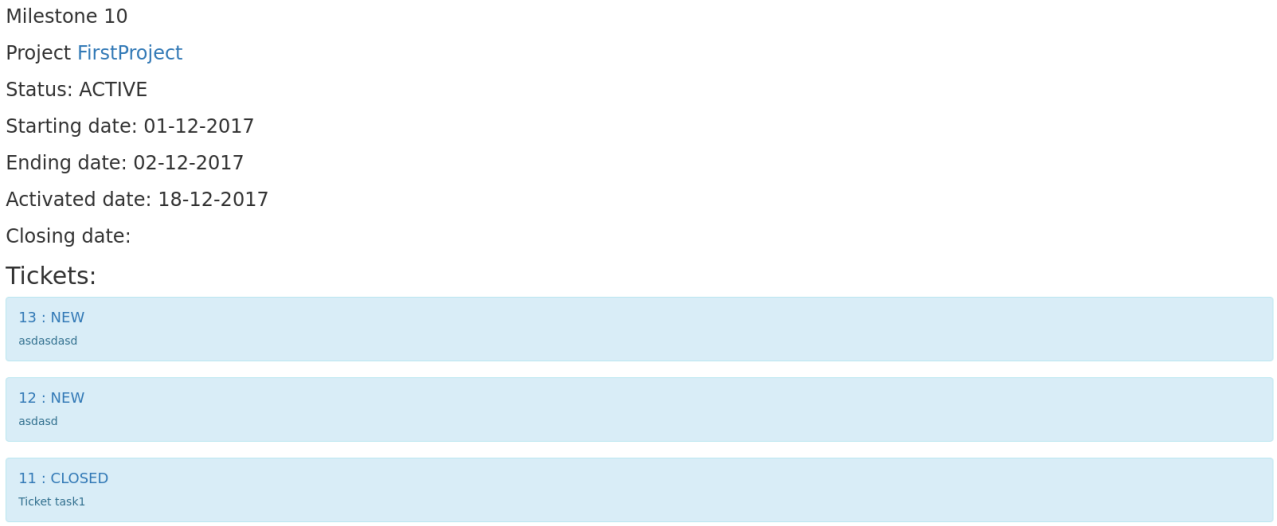

Рис. 10: Страница майлстоуна

Рис. 11: Страница тикета

- **Назначение тимлидера**. Ввести данные существующего пользователя и нажать на кнопку `Set team leader`. Операция должна успешно завершиться. В окне проекта возле пункта `Team leader` должно появиться имя указанного пользователя.
- **Добавление участников в проект**.
  - В окне проекта под списком разрабочиков ввести логин несуществующего пользователя и нажать кнопку `Add developer`. Должно появиться окно ошибки с сообщением «User `login` not found».
  - В окне проекта под списком разрабочиков ввести логин тимлидера проекта и нажать кнопку `Add developer`. Должно появиться окно ошибки с сообщением «User `login` already have a role `role` in project `name`».
  - В окне проекта под списком разрабочиков ввести логин существующего пользователя и нажать кнопку `Add developer`. Операция должна завершиться успеш-

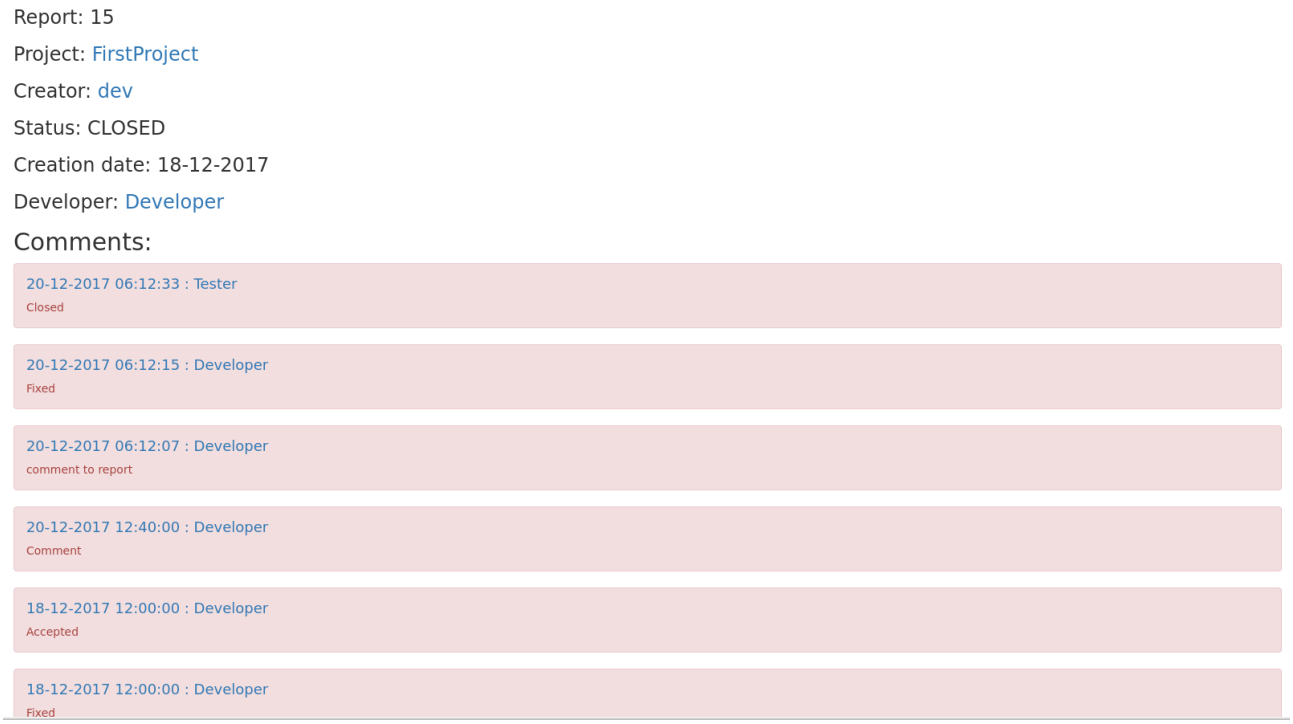

Рис. 12: Страница отчета об ошибке

но. В списке разработчиков должна появиться новая строчка с именем указанного пользователя.

- В окне проекта под списком тестировщиков ввести логин несуществующего пользователя и нажать кнопку `Add tester`. Должно появиться окно ошибки с сообщением «User `login` not found».
- В окне проекта под списком тестировщиков ввести логин тимлидера проекта и нажать кнопку `Add tester`. Должно появиться окно ошибки с сообщением «User `login` already have a role `role` in project `name`».
- В окне проекта под списком тестировщиков ввести логин существующего пользователя и нажать кнопку `Add tester`. Операция должна завершиться успешно. В списке тестировщиков должна появиться новая строчка с именем указанного пользователя.

- **Добавление майлстоуна, у которого дата завершения раньше чем дата начала**. В окне проекта под списком майлстоунов ввести даты начала и завершения майлстоуна так, чтобы дата завершения была перед датой начала. Нажать `Add milestone`. Должно появиться окно ошибки с сообщением «Can't create milestone with end date before start date».

- **Добавление майлстоуна, у которого дата завершения в прошлом**. В окне проекта под списком майлстоунов ввести даты начала и завершения майлстоуна так, чтобы дата завершения была в прошлом. Нажать `Add milestone`. Должно появиться окно ошибки с сообщением «Can't create milestone with end date in the past».

- **Добавление майлстоуна**. В окне проекта под списком майлстоунов ввести корректные даты начала и завершения майлстоуна. Нажать `Add milestone`. В списке майлстоунов проекта должна появиться новая запись, соответствующая созданному майлстоуну. Статус появившегося майлстоуна — «OPENED».

- **Неправильная смена статуса майлстоуна**. Открыть окно нового майлстоуна. Попытаться изменить статус майлстоуна с «OPENED» на «CLOSED» нажав на кнопку `Close`. Должно появиться окно ошибки с сообщением «Cannot change status from OPENED to CLOSED».

- **Добвление тикета**. В окне майлстоуна ввести описание нового тикета и нажать на кнопку `Add ticket`. В списке тикетов майлстоуна должна появиться новая строчка с описанием созданного тикета.

- **Активация майлстоуна**. В окне нового майлстоуна нажать кнопку `Activate`. Операция должна пройти успешно. Статус майлстоуна должен поменяться на «ACTIVE». В окне майлстоуна справа от строчки `Activated date` должна появиться текущая дата.

- **Два активных майлстоуна одновременно**. В окне проекта создать новый майлстоун. Открыть окно нового майлстоуна и нажать кнопку `Activate`. Должно появиться окно ошибки с сообщением «Attempting to activate milestone `id1`, when milestone `id2` is already active».

- **Закрыть майлстоун когда не все его тикеты закрыты**. Окрыть окно активного майлстоуна. Нажать кнопку `Close`. Должно появиться окно ошибки с сообщением «Ticket `id` is not closed».

- **Добавить разработчика в тикет**. Открыть окно нового тикета. Под списком разработчиков вести логин разработчика, являющегося разработчиком в проекте. Нажать `Add assignee`. В списке разработчиков тикета должна появиться новая строчка с именем указанного пользователя.

- **Добавить неправильного пользователя в проект**. В окне тикета ввести логин пользователя, который существует в системе но никак не связан с проектом. Нажать `Add assignee`. Должно появиться окно ошибки с сообщением «User `login` has no permission `permission` for project `name`».

- **Проверка добавления разработчика**. Выйти из аккаунта менеджера. Войти в всистему за пользователя, которого только что назначили разработчиком тикета. В главном окне пользователя быть одна запись в списке разрабатываемых тикетов — созданный ранее тикет.

- **Поменять статус тикета на «ACCEPTED»**. Октрыть окно тикета. Нажать на кнопку `Set accepted`. Статус тикета должен поменяться. В списке комментариев тикета должна появиться соответствующая запись.

- **Поменять статус тикета на «IN_PROGRESS»**. Октрыть окно тикета. Нажать на кнопку `Set in progress`. Статус тикета должен поменяться. В таблице комментариев тикета должна появиться соответствующая запись.

- **Поменять статус тикета на «FINISHED»**. Октрыть окно тикета. Нажать на кнопку `Set finished`. Статус тикета должен поменяться. В таблице комментариев тикета должна появиться соответствующая запись.

- **Закрыть тикет**. Выйти из системы. Войти в систему за пользователя `manager`. Открыть окно тикета. Нажать на кнопку `Set closed`. Статус тикета должен поменяться. В таблице комментариев тикета должна появиться соответствующая запись.

- **Закрыть майлстоун**. Открыть окно майлстоуна, к которому был привязан тикет. Нажать на кнопку `Close`. Операция должна завершиться успешно. Статус майлстоуна должен поменяться на «CLOSED». В окне майлстоуна справа от строчки `Closing date` должна появиться текущая дата.

- **Добавить тикет в закрытый майлстоун**. Открыть окно закрытого майлстоуна. Ввести описание нового тикета и нажать на кнопку `Add ticket`. Должно появиться окно ошибки с сообщением «Milestone `id` of project `name` is already closed».

- **Создать багрепорт**. Выйти из системы. Войти в систему за пользователя `teamleader`. Открыть окно проекта. Ввести описание ошибки и нажать на кнопку `Add report`. В списке ошибок должна появиться новая строчка, соответствующая созданному ошибок.

- **Принять багрепорт**. Выйти из системы. Войти в систему за пользователя `developer`. Открыть окно багрепорта. Поменять статус на «ACCEPTED». Статус багрепорта должен поменяться. В таблице комментариев должен появиться соответствующий комментарий. На странице багрепорта справа от пункта `Developer` должно появиться имя текущего пользователя.

- **Принять принятый багрепорт**. Выйти из системы. Войти в систему за пользователя `teamleader`. Открыть окно багрепорта. Поменять статус на «ACCEPTED». Должно появиться окно ошибки с сообщением, что другой пользователь уже принял этот багрепорт.

- **Багрепорты разработчика**. Выйти из системы. Войти в систему за пользователя `developer`. В списке разрабатываемых багрепортов на странице пользователя должна появиться запись, соответствующая принятому багрепорту.

- **Исправить багрепорт**. Открыть окно багрепорта. Поменять статус на «FIXED». Статус багрепорта должен поменяться. В таблице комментариев должен появиться соответствующий комментарий.

- **Переоткрыть багрепорт**. Выйти из системы. Войти в систему за пользователя `tester`. Открыть окно багрепорта. Поменять статус на «OPENED». Статус багрепорта должен поменяться. В таблице комментариев должен появиться соответствующий комментарий.

- **Снова исправить багрепорт**. Выйти из системы. Войти в систему за пользователя `developer`. Открыть окно багрепорта. Поменять статус на «FIXED». В появившемся окне ввести комментарий. Статус багрепорта должен поменяться. В таблице комментариев должен появиться соответствующий комментарий.

- **Закрыть багрепорт**. Выйти из системы. Войти в систему за пользователя `tester`. Открыть окно багрепорта. Поменять статус на «CLOSED». В появившемся окне ввести комментарий. Статус багрепорта должен поменяться. В таблице комментариев должен появиться соответствующий комментарий.

Все описанные тесты успешно выполняются. Все полученные результаты совпажают с ожидаемыми. Функциональное тастирование позволило понять, что созданной приложение работает корректно и выполняет свои функции.

10 Инструкция системного администратора

1. Скачать и распаковать сервер Apache Tomcat 7.
2. Установить СУБД MySQL.
3. Настроить СУБД:
   - создать новую базу данных;
   - создать нового пользователя;
   - дать полный доступ к созданной базе данных.
4. Скачать проект из репозитория `https://github.com/AbdullinAM/distributed_systems`
5. Прописать параметры подключения к БД (название БД, имя пользователя и пароль) в файле `distributed_systems/src/main/resources/application.properties`
6. Перейти в корневую папку проекта. Собрать war файл при помощи команды `mvn compilewar:war`.
7. Удалить предыдущие версии версии проекта из папки веб приложений сервера Tomcat с помощью команды `rm -rf $TOMCAT_WEBAPPS/pms-1.0-SNAPSHOT`.
8. Скопировать новый war файл в папку веб-приложений сервера: `cp pms-1.0-SNAPSHOT $TOMCAT_WEBAPPS`.
9. Перезапустить сервер Tomcat.

11 Инструкция пользователя

При переходе на главную страницу сайта будет выведена форма для авторизации в системе. Если у пользователя нет аккаунта, он может зарегистрироваться на этой же странице. После успешной авторизации в системе, пользователь будет перенаправлен на свою страницу пользователя.

11.1 Страница пользователя

Внешний вид страницы пользователя приведен на рисунке 8. В верхней части страницы находится панель навигации, с помощью которой можно перейти на домашнюю страницу (страница авторизованного пользователя) или выйти из системы.

Под панелью навигации выводится общая информация о пользователе: идентификатор, логин, имя. Далее на данной странице предоставляется следующая информация:

- список проектов, в которых данный пользователь является менеджером;
- список проектов, в которых данный пользователь является тимлидером;
- список проектов, в которых данный пользователь является разработчиков;
- список проектов, в которых данный пользователь является тестировщиком;
- список ошибок, которые были созданы данным пользователем;

- список ошибок, которые исправляются данным пользователем;
- список тикетов, которыми руководит данный пользователь;
- список тикетов, которые выполняет данный пользователь;
- список уведомлений, полученных данным пользователем.

Все приведенные в списке элементы (за исключением уведомлений) являются активными, и при нажатии на них пользователь будет перенаправлен на страницу соответствующего объекта.

Если пользователь находится на своей домашней странице (т.е. на странице пользователя, за которого он авторизован), ему также будет доступна возможность создания нового проекта. Для этого в соответствующее текстовое поле необходимо ввести имя нового проекта и нажать на кнопку `Create project`. В случае успеха список проектов, в которых данный пользователь является менеджером, обновится. Иначе появится информационное сообщение с описанием ошибки.

11.2 Страница проекта

Внешний вид страницы проекта приведен на рисунке 9. В верхней части страницы находится панель навигации, с помощью которой можно перейти на домашнюю страницу (страница авторизованного пользователя) или выйти из системы.

Под панелью навигации выводится общая информация о проекте: идентификатор, имя, менеджер, тимлидер.

Для того, чтобы сделать пользователя тимлидером данного проекта, необходимо ввести логин пользователя в соответствующее поле и нажать на кнопку `Set team leader`. В случае успеха информация о проекте обновится. Иначе выведется информационное сообщение с описанием ошибки.

Далее на странице показано две колонки:

- Первая колонка показывает пользователей, привязанных к проекту. Для того, чтобы добавить нового пользователя в проект необходимо ввести его логин в соответствующее поле и нажать на кнопку добавления. Если операция выполнится успешно, информация на странице обновится. Иначе выведется информационное сообщение с описанием ошибки.

- Во второй колонке показаны составляющие проекта: майлстоуны и отчеты об ошибках. Для того, чтобы добавить новый майлстоун необходимо выбрать в соответствующих полях даты его начала и завершения и нажать на кнопку `Add milestone`. Если операция выполнится успешно, информация на странице обновится. Иначе выведется информационное сообщение с описанием ошибки.

  Для того, чтобы добавить новый отчет об ошибке, необходимо ввести в соответствующее текстовое поле описание ошибки и нажать на кнопку `Add report`. Если операция выполнится успешно, информация на странице обновится. Иначе выведется информационное сообщение с описанием ошибки.

Все элементы на странице являются активными, т.е. при нажатии на них пользователь будет перенаправлен на соответствующую данному объекту страницу.

Элементы управления (кнопки добавления пользователей и т.д.) отображаются только в том случае, если у пользователя есть соответствующие права.

11.3 Страница майлстоуна

Внешний вид страницы майлстоуна приведен на рисунке 10. В верхней части страницы находится панель навигации, с помощью которой можно перейти на домашнюю страницу (страница авторизованного пользователя) или выйти из системы.

Под панелью навигации выводится общая информация о майлстоуне: идентификатор, проект, предполагаемая дата начала, фактическая дата начала (если она есть), предполагаемая дата завершения, фактическая дата завершения (если она есть).

Под общей информацией находятся кнопки управления майлстоуном: кнопка активации майлстоуна и кнопка закрытия майлстоуна. Они отображаются пользователю только в том случае, если у него есть права на управление майлстоуном. Для того, чтобы поменять статус майлстоуна необходимо нажать на соответствующую кнопку. Если операция выполнится успешно, информация на странице обновится. Иначе выведется информационное сообщение с описанием ошибки.

Ниже на странице отображается список всех тикетов, привязанных к данному майлстоуну. При нажатии на тикет пользователь будет перенаправлен на страницу данного тикета.

Для того, чтобы добавить новый тикет к майлстоуну необходимо ввести описание задачи в соответствующее поле и нажать кнопку `Create ticket`. Если операция выполнится успешно, информация на странице обновится. Иначе выведется информационное сообщение с описанием ошибки.

11.4 Страница тикета

Внешний вид страницы тикета приведен на рисунке 11. В верхней части страницы находится панель навигации, с помощью которой можно перейти на домашнюю страницу (страница авторизованного пользователя) или выйти из системы.

Под панелью навигации выводится общая информация о тикете: идентификатор, майлстоун, дата создания, создатель, статус, описание задачи.

Под общей информацией располагаются кнопки управления статусом тикета. Они доступны пользователю только в том случае, если он обладает какими-либо правами в данном проекте. Для того, чтобы изменить статус тикета, необходимо нажать соответствующую кнопку. Если операция выполнится успешно, информация на странице обновится. Иначе выведется информационное сообщение с описанием ошибки.

Ниже на странице показывается две колонки:

- Список разработчиков тикета. Для того, чтобы добавить нового разработчика к тикету, необходимо ввести его логин в соответствующее поле и нажать на кнопку `Add assignee`. Если операция выполнится успешно, список разработчиков обновится. Иначе выведется информационное сообщение с описанием ошибки. Данные кнопки доступны пользователю только при наличии соответствующих прав.

- Список комментариев к тикету. Для того, чтобы добавить новый комментарий к тикету, необходимо ввести содержание комментария в соответствующее поле и нажать на кнопку `comment`. В случае успеха в списке комментариев появится добавленный комментарий. Иначе выведется информационное сообщение с описанием ошибки. Данные кнопки отображаются пользователю только в том случае, если он обладает правами на комментирование тикета.

11.5 Странице отчета об ошибке

Внешний вид страницы отчета приведен на рисунке 12. В верхней части страницы находится панель навигации, с помощью которой можно перейти на домашнюю страницу (страница авторизованного пользователя) или выйти из системы.

Под панелью навигации выводится общая информация об отчете: идентификатор, проект, дата создания, создатель, статус, описание задачи, разработчик (если назначен).

Далее на странице показывается список комментариев к отчету. Для того, чтобы добавить новый комментарий к отчету, необходимо ввести содержание комментария в соответствующее поле и нажать на кнопку `comment`. В случае успеха в списке комментариев появится добавленный комментарий. Иначе выведется информационное сообщение с описанием ошибки. Данные кнопки отображаются пользователю только в том случае, если он обладает правами на комментирование отчета.

12 Выводы

В рамках данной работы были изучены принципы работы с ORM Hibernate, принципы работы с технологиями Spring Data и Spring MVC, создание распределенных веб-приложений на языке Java. Также были изучены основы создания RESTful-клиентов с использованием фреймворка AngularJS. Поставленные в рамках работы задачи были выполнены. Однако, полученное приложение можно далее развивать в нескольких направлениях:

- улучшение интерфейса;
- расширение функциональности текущих ролей;
- добавление новых ролей и вариантов использования.

Использование библиотек ORM и Spring Data позволило значительно облегчить разработку слоя источников данных. ORM позволяет сделать многие вещи, связанные с хранением данных, прозрачными для программиста, а Spring Data дает возможность автоматической генерации всех необходимых методов обращения к слою хранения. Благодаря архитектуре созданного приложения, оно должно быть легко масштабируемым, однако данное свойство не было проверено в рамках курсовой работы.