Федеральное государственное автономное

образовательное учреждение

Высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт космических и информационных технологий |
| институт |
| Программная инженерия |
| кафедра |

**ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ**

|  |
| --- |
| Итоговая |
| тема |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Преподаватель | |  |  |  | Д. В. Грузенкин |
|  | |  | подпись, дата |  | инициалы, фамилия |
| Студент | КИ21-17/1Б, 032156940 |  |  |  | Н. А. Самарин |
|  | номер группы, зачётной книжки |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2024

**СОДЕРЖАНИЕ**

1 Задание............................................................................................................... 3

2 Ход работы........................................................................................................ 3

2.1 Выбор программной системы для проектирования................................. 3

2.2 Работа с требованиями к ПО...................................................................... 4

2.2.1 Контекст.................................................................................................. 4

2.2.2 Получение требований........................................................................... 4

2.2.3 Составленные требования..................................................................... 6

2.2.3.1 Бизнес-требования............................................................................. 6

2.2.3.2 Пользовательские требования.......................................................... 6

2.2.3.3 Функциональные требования........................................................... 7

2.2.3.4 Нефункциональные требования....................................................... 8

2.3 Работа с диаграммой контекста C4............................................................ 9

2.4 Организация компонентов.......................................................................... 9

2.5 Межмодульное взаимодействие................................................................. 10

2.6 Реализация принципов SOLID................................................................... 11

2.7 Реализация паттернов проектирования..................................................... 11

2.8 Диаграмма классов...................................................................................... 11

2.9 Применение архитектурных шаблонов..................................................... 12

3 Вывод................................................................................................................. 12

ПРИЛОЖЕНИЕ А................................................................................................. 13

**1 Задание**

Объединить результаты всех предыдущих работы в едином отчёте с  
целью предоставления наиболее полной и структурированной информации по  
проектируемой системе. Также в приложениях к отчету необходимо привести  
написанный вами код. который на верхнем уровне отражает архитектуру  
одного из модулей вашей системы (результаты работ 5, 7, 9).

**2 Ход работы**

**2.1 Выбор программной системы для проектирования**

Название системы: RecruitHub

Назначение и краткое описание: RecruitHub – это онлайн платформа для  
эффективного поиска работы для соискателей и работников на вакансии для  
работодателей. RecruitHub помогает работодателям сократить время и затраты  
на подбор персонала, повысить эффективность рекрутинговых процессов и  
найти лучших кандидатов для вакансий. Соискателям же, платформа помогает  
повысить эффективность поиска работы, путём упрощения создания и  
размещения резюме, поиска вакансий и связи с работодателями.

Работодатели:

- Работодатели могут регистрироваться, авторизоваться и заполнять  
информацию о себе;

- Работодатели могут создавать и размещать вакансии, указывая  
требуемую должность, а также образование, опыт работы и навыки, требуемые  
от соискателей;

- Работодатели могут связываться с соискателями, откликнувшимися на  
вакансию через встроенный чат.

Соискатели:

- Соискатели могут регистрироваться, авторизоваться и заполнять  
информацию о себе;

- Соискатели могут создавать и размещать резюме, указывая желаемую  
должность, а также образование, опыт работы и навыки;

- Соискатели могут откликаться на вакансии, размещённые  
работодателями;

- Соискатели могут общаться с работодателями, заинтересованными в  
найме, через встроенный чат.

Администраторы:

- Администраторы могут просматривать все вакансии, резюме и данные  
пользователей;

- Администраторы могут удалять вакансии, резюме и пользователей.

Характеристики системы

- Клиентская часть будет реализована как мобильное приложение для  
Android и iOS;

- Данные пользователей будут храниться в базе данных PostgreSQL;

- Данные в базе данных будут защищены путём шифрования данных.

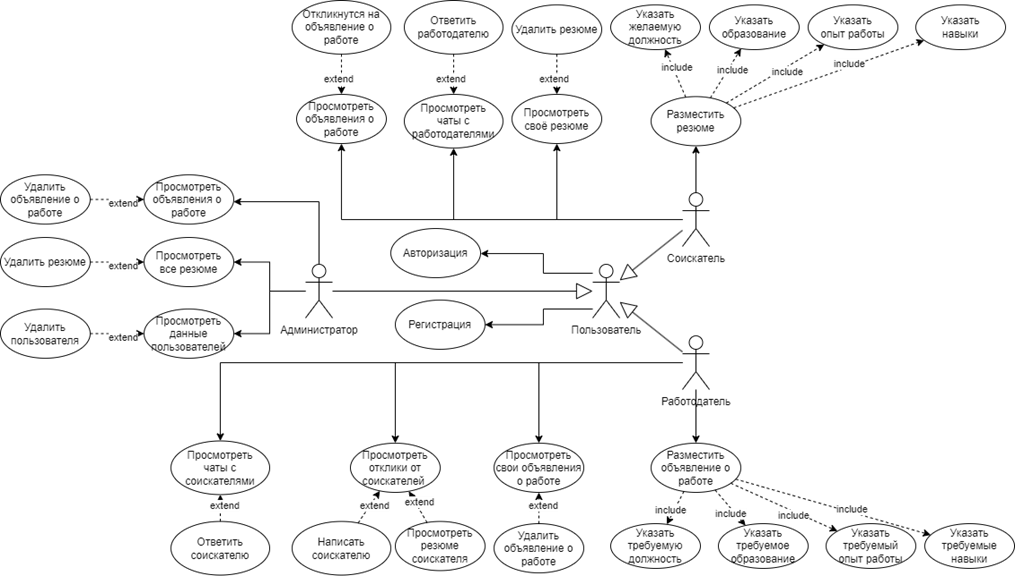


Рисунок 1 – UML–диаграмма вариантов использования

**2.2 Работа с требованиями к ПО**

**2.2.1 Контекст**

В связи с повышением цифровизации большинства современных рабочих  
сфер и предпочтением большинства современных соискателей искать работу  
онлайн, существует потребность в существовании онлайн платформа для  
эффективного поиска работы для соискателей и работников на вакансии для  
работодателей. На данный момент уже существует ряд подобных платформ, но  
существует проблема в том, что они ориентированы на предоставление услуг  
крупным организациям, способным оплачивать высокие тарифы платформ  
онлайн-рекрутинга. Так, например, платформа “Зарплата.ру” за плату в размере  
5400 рублей позволяет работодателю разместить только одну вакансию сроком  
на 1 месяц. И подобные тарифы существуют на всех популярных платформах  
онлайн-рекрутинга. Такие условия не подходят большинству малых бизнесов.  
Отсюда исходит идея создания онлайн-платформы рекрутинга, финансово  
доступной для малого бизнеса, но предоставляющего функционал схожий с  
популярными платформами.

Описанная выше идея принадлежит небольшому стартапу, который  
собрал деньги от инвесторов на реализацию данного проекта, в размере 500000  
рублей. Стартап хочет заказать на эти деньги разработку описанной  
платформы, для того чтобы ввести её в эксплуатацию, окупить вложения  
инвесторов и начать зарабатывать.

**2.2.2 Получение требований**

Q: Чего вы хотите достичь с помощью реализации своего проекта? A:  
Наша цель - сократить время и затраты на поиск и подбор персонала, улучшить  
качество подбора сотрудников, а также создать удобный и эффективный

инструмент для работы с кандидатами. Q: В каком виде вы представляете  
разрабатываемую платформу? A: Мы хотим, чтобы это было мобильное  
приложение, запускаемое на телефонах с ОС Android и iOS современных  
версий. Q: Какие основные функции вы бы хотели видеть у системы? A:  
Пользователи могут регистрироваться и авторизоваться в качестве  
работодателей, соискателей и администраторов. Работодатели могут создавать  
и размещать вакансии, и связываться с соискателями, откликнувшимися на  
вакансию через встроенный чат. Соискатели могут создавать и размещать  
резюме, откликаться на вакансии, размещённые работодателями и общаться с  
работодателями, через встроенный чат. Администраторы могут модерировать и  
удалять вакансии, резюме и пользователей. Q: Есть ли у вас какие-либо  
предпочтения в интерфейсе? A: Интерфейс должен быть адаптивным,  
корректно отображаясь на всех популярных стандартах экранов современных  
мобильных устройств. Также хотелось бы, чтобы в приложении было боковое  
меню, через которое пользователи могли бы добраться до всех разделов  
приложения. Q: Какие дополнительные особенности вы хотели бы чтобы мы  
учли при разработке? A: Мы хотели бы чтобы все личные данные  
пользователей были зашифрованы. Также нам важно быстродействие системы,  
время отклика на любое действие пользователя не должно превышать  
половины секунды. Q: Как вы планируете монетизировать систему? A: Мы  
планируем размещать рекламные баннеры в интерфейсе, но также  
приветствуем предложения, но важно чтобы работодателям не приходилось  
платить за размещение вакансий, это противоречит нашей бизнес-идее. Q:  
Какие критерии вы установили для оценки успеха вашего проекта и какие цели  
вы планируете достичь после ввода системы в эксплуатацию? A: Мы считаем,  
что успех проекта можно измерить по количеству пользователей приложения и  
полученному нами доходу. Наши цели - Повысить посещаемость приложения  
до 1 миллиона в месяц в течение первых 18 месяцев с момента ввода системы в  
эксплуатацию. Зафиксировать минимум 100,000 откликов соискателей на  
вакансии за первые 18 месяцев. Окупить вложения инвесторов в размере  
500000 рублей в течение первого года. И заработать не менее 2,000,000 рублей  
за первые 18 месяцев.

В качестве проектов конкурентов для анализа были выбраны платформы  
hh.ru и зарплата.ру.

Платформа hh.ru является одним из крупнейших сервисов  
онлайн-рекрутинга в России и странах СНГ. 18 млн посетителей ежемесячно.  
Представляет собой сайт и мобильное приложение.

Отличительные особенности:

1. Разнообразные инструменты поиска и отбора: платформа  
предоставляет широкий выбор инструментов для эффективного поиска  
кандидатов, включая фильтры по профессиональным навыкам, опыту работы,  
образованию и другим критериям.

2. Эффективная система рекомендаций: платформа использует алгоритмы  
и технологии машинного обучения для предложения компаниям наиболее  
подходящих кандидатов согласно их потребностям и требованиям.

3. Уникальные функции и технологии для улучшения процесса  
рекрутинга: такие как тестирования кандидатов на профессиональные навыки,  
онлайн-собеседования и другие инструменты.

Способы монетизации:

1. Для размещения вакансий работодателям нужно платить.

2. Услуги платного продвижения резюме и вакансий в рекомендациях.

Зарплата.ру. Онлайн-ресурс по трудоустройству. Входит в топ-5 порталов  
поиска работы по России. Портал ежедневно посещает 500 000 уникальных  
пользователей. Представляет собой сайт и мобильное приложение.

Отличительные особенности:

1. Система фильтров позволяет быстро находить подходящие вакансии по  
требуемой зарплате, региону, опыту работы и другим критериям.

2. Платформа предоставляет информацию о средних заработных платах  
по различным профессиям и регионам, что помогает кандидатам и  
работодателям ориентироваться на рынке труда.

Способы монетизации:

1. Для размещения вакансий работодателям нужно платить.

2. Услуги платного продвижения резюме в рекомендациях.

3. На сайте представлена сторонняя реклама в виде баннеров.

**2.2.3 Составленные требования**

**2.2.3.1 Бизнес-требования**

- В течение года после ввода системы в эксплуатацию, платформа должна  
принести заказчикам доход в размере 500,000 рублей, для того чтобы окупить  
вложения инвесторов, после этого система должна выйти на самоокупаемость;

- Достижение общего дохода в размере не менее 2,000,000 рублей за  
первые 18 месяцев с момента ввода системы в эксплуатацию;

- Повысить посещаемость приложения до 1 миллиона в месяц в течение  
первых 18 месяцев с момента ввода системы в эксплуатацию;

- Зафиксировать минимум 100,000 откликов соискателей на вакансии за  
первые 18 месяцев с момента ввода системы в эксплуатацию.

**2.2.3.2 Пользовательские требования**

Регистрация и авторизация:

- Пользователь должен иметь возможность зарегистрироваться в системе  
как работодатель, соискатель или администратор, используя электронную  
почту и пароль;

- Зарегистрированный пользователь должен иметь возможность войти в  
систему, используя электронную почту и пароль;

- Пользователь должен иметь возможность восстановить пароль  
используя электронную почту.

Размещение вакансий и резюме:

- Соискатель должен иметь возможность создавать и размещать резюме,  
указывая желаемую должность, а также образование, опыт работы и навыки;

- Работодатель должен иметь возможность создавать и размещать  
вакансии, указывая требуемую должность, а также образование, опыт работы и  
навыки, требуемые от соискателей;

- Соискатель должен иметь возможность удалять свои размещённые  
резюме;

- Работодатель должен иметь возможность удалять свои размещённые  
вакансии.

Взаимодействие работодателей и соискателей:

- Соискатель должен иметь возможность просматривать и фильтровать по  
требуемым параметрам (должность, образование и требуемые навыки)  
вакансии, размещённые работодателями;

- Соискатель должен иметь возможность откликаться на вакансии,  
размещённые работодателями;

- Работодатель и соискатель должен иметь возможность общаться друг с  
другом, если соискатель откликнулся на вакансию работодателя, через  
встроенный чат;

- Работодатель и соискатель должен иметь возможность общаться друг с  
другом, через встроенный чат;

- Работодатель должен иметь возможность просматривать резюме  
откликнувшихся соискателей и начинать с ними чат.

Функции администраторов:

- Администратор должен иметь возможность просматривать и удалять  
вакансии, резюме и пользователей.

Работодатель должен иметь возможность поднимать свои вакансии в  
выдаче, выдаваемой соискателям, за дополнительную плату.

**2.2.3.3 Функциональные требования**

Пользователь должен иметь возможность зарегистрироваться в системе  
как работодатель, соискатель или администратор, используя электронную  
почту и пароль (URQ): FRQ 1: Ввод электронной почты; FRQ 2: Ввод пароля;  
FRQ 3: Подтверждение пароля; FRQ 4: Подтверждение регистрации.

Зарегистрированный пользователь должен иметь возможность войти в  
систему, используя электронную почту и пароль (URQ): FRQ 1: Ввод  
электронной почты; FRQ 2: Ввод пароля; FRQ 3: Подтверждение входа;

Пользователь должен иметь возможность восстановить пароль используя  
электронную почту (URQ): FRQ 1: Запрос на восстановление пароля; FRQ 2:  
Проверка подлинности пользователя с помощью электронной почты; FRQ 3:  
Установка нового пароля.

Соискатель должен иметь возможность создавать и размещать резюме,  
указывая желаемую должность, а также образование, опыт работы и навыки  
(URQ): FRQ 1: Создание резюме; FRQ 2: Указание желаемой должности; FRQ  
3: Указание образования; FRQ 4: Указание опыта работы; FRQ 5: Указание  
навыков;

Работодатель должен иметь возможность создавать и размещать  
вакансии, указывая требуемую должность, а также образование, опыт работы и  
навыки, требуемые от соискателей (URQ): FRQ 1: Создание вакансии; FRQ 2:  
Указание должности; FRQ 3: Указание требуемого образования; FRQ 4:  
Указание требуемого опыта работы; FRQ 5: Указание требуемых навыков;

Соискатель должен иметь возможность удалять свои размещённые  
резюме (URQ): FRQ 1: Просмотр своих резюме; FRQ 2: Удаление резюме;

Работодатель должен иметь возможность удалять свои размещённые  
вакансии (URQ): FRQ 1: Просмотр своих вакансий; FRQ 2: Удаление вакансии;

Соискатель должен иметь возможность просматривать и фильтровать по  
требуемым параметрам (должность, образование и требуемые навыки)  
вакансии, размещённые работодателями (URQ): FRQ 1: Доступ в выдачу  
вакансий; FRQ 2: Фильтрация по должности; FRQ 3: Фильтрация по  
образованию; FRQ 4: Фильтрация по навыкам;

Соискатель должен иметь возможность откликаться на вакансии,  
размещённые работодателями (URQ): FRQ 1: Отклик на вакансию в выдаче с  
выбором одного из своих резюме;

Работодатель должен иметь возможность просматривать резюме  
откликнувшихся соискателей и начинать с ними чат (URQ): FRQ 1: Просмотр  
резюме откликнувшихся пользователей; FRQ 2: Начала чата работодателем с  
откликнувшимся соискателем;

Работодатель и соискатель должен иметь возможность общаться друг с  
другом, через встроенный чат (URQ): FRQ 1: Отправка сообщений.

Администратор должен иметь возможность просматривать и удалять  
вакансии, резюме и аккаунты пользователей (URQ): FRQ 1: Доступ в меню  
модерации; FRQ 2: Выбор вакансии, резюме или аккаунта; FRQ 3: Удаление  
выбранной вакансии, резюме или аккаунта из системы.

Работодатель должен иметь возможность поднимать свои вакансии в  
выдаче, выдаваемой соискателям, за дополнительную плату (URQ): FRQ 1:  
Запрос на поднятие вакансии в выдаче; FRQ 2: Оплата услуги.

**2.2.3.4 Нефункциональные требования**

- Приложение должно запускаться и стабильно работать на телефонах с  
операционными системах Android, версии от 7.0 и выше, и iOS, версии от 8 и  
выше, и характеристиками от 2 ГБ ОЗУ;

- Все личные данные пользователей должны быть зашифрованы в  
соответствии с принятыми в индустрии стандартами безопасности (например,  
AES с ключом не менее 256 бит);

- Время отклика на любое действие пользователя при работе с  
приложением не должно превышать 0.5 секунды;

- Интерфейс приложения должен быть адаптирован под различные  
форматы и разрешения экранов, обеспечивая удобное использование на всех  
современных мобильных устройствах;

- Все разделы приложения должны быть доступны для быстрого доступа  
через боковое меню.

**2.3 Работа с диаграммой контекста C4**

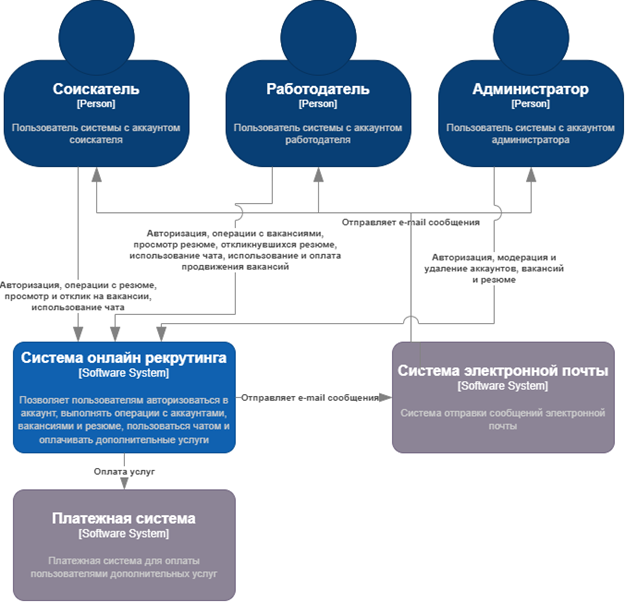


Рисунок 2 – Диаграмма контекста C4

**2.4 Организация компонентов**

На основе ранее составленной диаграммы контекста C4 была составлена  
диаграмма контейнеров C4. В ней соблюдаются: принцип ацикличности  
зависимостей, принцип устойчивых зависимостей, принцип согласованного  
изменения.

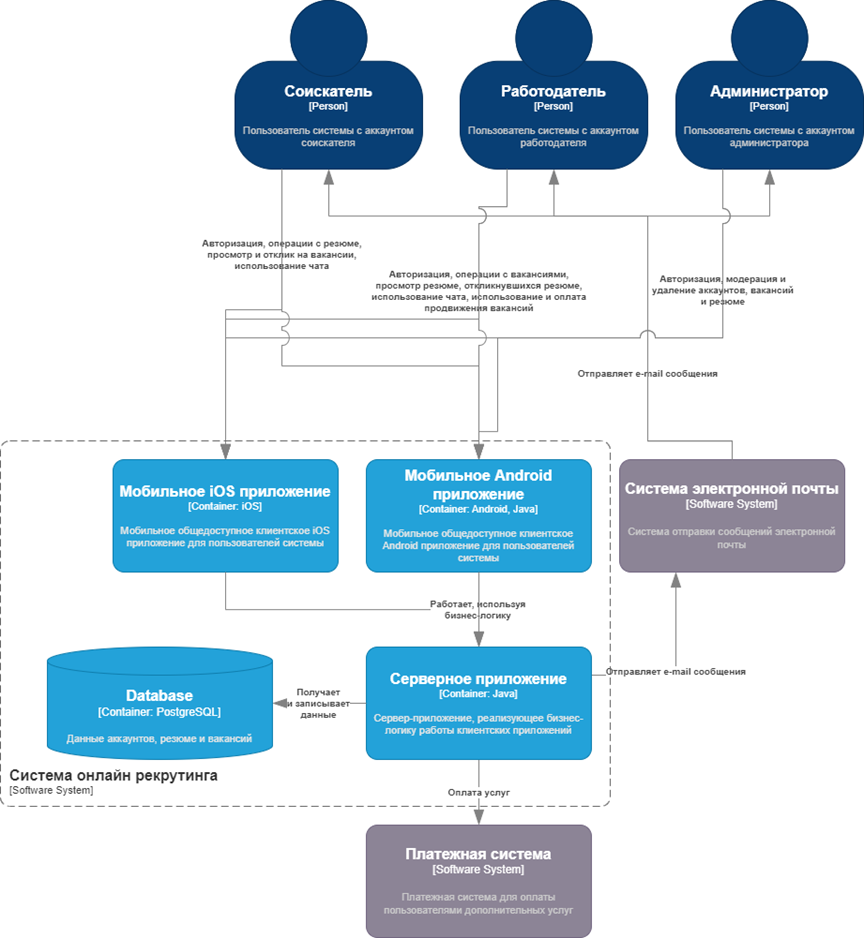


Рисунок 3 – Диаграмма контейнеров C4

**2.5 Межмодульное взаимодействие**

На основе ранее составленной диаграммы контейнеров C4 была  
составлена диаграмма компонентов C4. В ней отображены: Принцип  
единственной ответственности, принцип инверсии зависимостей, принцип  
разделения интерфейсов.

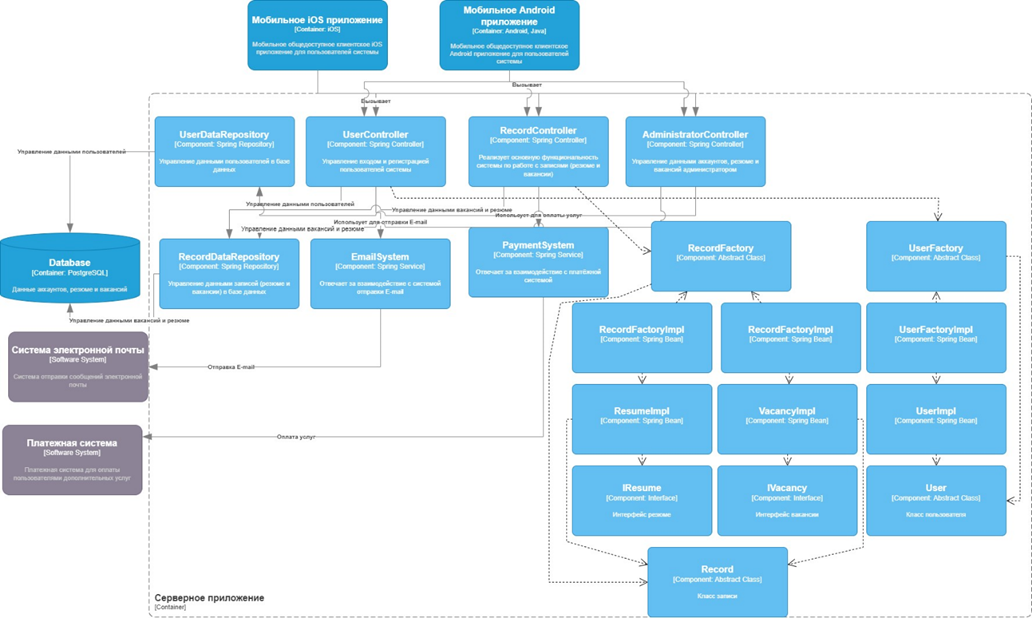


Рисунок 4 – Диаграмма компонентов C4

**2.6 Реализация принципов SOLID**

На языке программирования Java были реализованы принципы SOLID на  
примере компонентов из диаграммы из предыдущей работы. Реализованы  
принципы: Принцип единственной ответственности, принцип инверсии  
зависимостей, принцип разделения интерфейсов, принцип подстановки  
Барбары Лисков. Листинги представлены в приложении А.

**2.7 Реализация паттернов проектирования**

На языке программирования Java были реализованы паттерны  
проектирования: Фабричный метод, Фасад, Шаблонный метод. Листинги  
представлены в приложении А.

**2.8 Диаграмма классов**

Была построена UML диаграмма классов, были реализованы следующие  
паттерны:

- Фабричный метод - Используется чтобы упростить создание  
ResumeImpl и VacancyImpl;

- Фасад - Используется чтобы упростить отправку email в различных  
методах;

- Шаблонный метод - Используется так как создание ResumeImpl и  
VacancyImpl различается только частично.

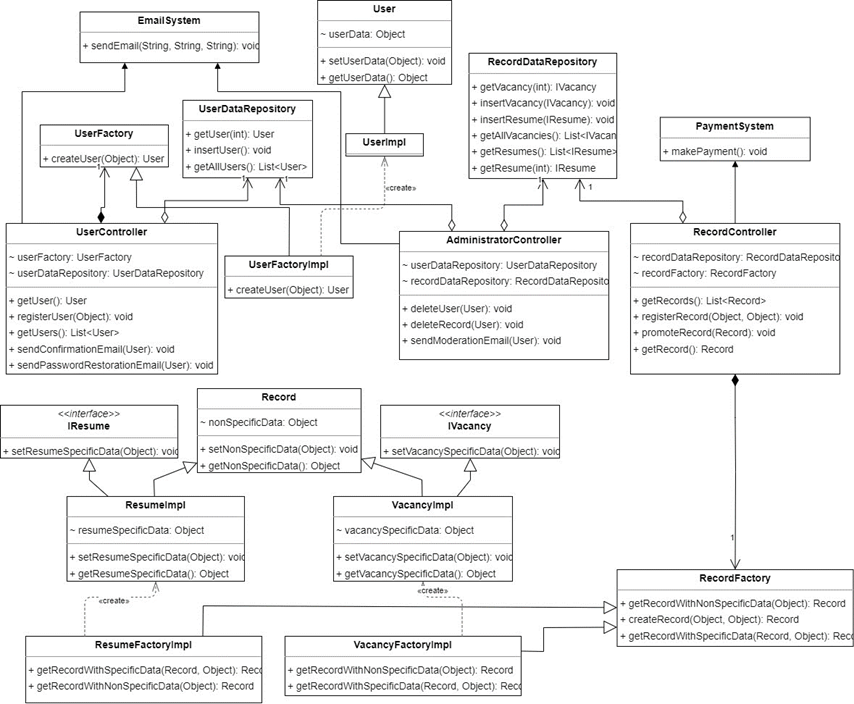


Рисунок 5 – Диаграмма классов

**2.9 Применение архитектурных шаблонов**

Был реализован шаблон проектирования MVC. Использование шаблона  
обосновано тем, что данный компонент представляет собой сервер  
взаимодействующий с несколькими мобильными приложениями через REST  
запросы для управления данными пользователей и записей. MVC позволяет  
разделить логику пользовательского интерфейса и бизнес-логику.  
Разработанный код представляет собой Spring RESTful сервер, реализующий  
MVC. В случае Spring RESTful сервера, контроллеры обрабатывают  
HTTP-запросы, модели предоставляют данные, а представления представлены в  
формате данных JSON или XML, передаваемых клиентским приложениям  
сервером с помощью HTTP. Листинги представлены в приложении А.

**3 Вывод**

В ходе работы за семестр была спроектирована задуманная программная  
система.

**ПРИЛОЖЕНИЕ А**

Листинг А.1 – RecordController

package org.example;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RestController;  
import java.util.List;  
@RestController  
public class RecordController {  
 RecordFactory recordFactory;  
 RecordDataRepository recordDataRepository;  
 @Autowired  
 public RecordController(RecordDataRepository recordDataRepository, RecordType   
 type) {  
 switch (type) {  
 case VACANCY -> recordFactory = new VacancyFactoryImpl();  
 case RESUME -> recordFactory = new ResumeFactoryImpl();  
 }  
 this.recordDataRepository = recordDataRepository;  
 }  
 @PostMapping  
 public void registerRecord(Object nonSpecificRecordData, Object specificRecord  
 Data) {  
 Record record = recordFactory.createRecord(nonSpecificRecordData, specificRe  
 cordData);  
 // Работа с recordDataRepository  
 }  
 @GetMapping  
 public Record getRecord(Long id) {  
 // Работа с recordDataRepository  
 return null;  
 }  
 @GetMapping  
 public List<Record> getRecords(List<Long> ids) {  
 // Работа с recordDataRepository  
 return null;  
 }  
 @PostMapping  
 public void promoteRecord(Record record){  
 // Работа с paymentSystem  
 PaymentSystem.makePayment();  
 }  
}

Листинг А.2 – AdministratorController

package org.example;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
@RestController  
public class AdministratorController {  
 UserDataRepository userDataRepository;  
 RecordDataRepository recordDataRepository;  
 @Autowired  
 public AdministratorController(UserDataRepository userDataRepository, RecordDa  
 taRepository recordDataRepository) {  
 this.userDataRepository = userDataRepository;

Окончание листинга А.2

this.recordDataRepository = recordDataRepository;  
 }  
 @PostMapping  
 public void sendModerationEmail(User user){  
 //работа с EmailSystem  
 //EmailSystem.sendEmail();  
 }  
 @DeleteMapping  
 public void deleteUser(Long id){}  
 @DeleteMapping("/records/{id}")  
 public void deleteRecord(@PathVariable Long id){}  
}

Листинг А.3 – UserController

package org.example;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RestController;  
import java.util.List;  
@RestController  
public class UserController {  
 UserFactory userFactory;  
 UserDataRepository userDataRepository;  
 @Autowired  
 public UserController(UserFactory userFactory, UserDataRepository userDataRepo  
 sitory) {  
 this.userFactory = userFactory;  
 this.userDataRepository = userDataRepository;  
 }  
 @PostMapping  
 public void registerUser(Object userData) {  
 User user = userFactory.createUser(userData);  
 // Работа с userDataRepository  
 }  
 @GetMapping  
 public User getUser(Long id) {  
 // Работа с userDataRepository  
 return null;  
 }  
 @GetMapping  
 public List<User> getUsers(List<Long> ids) {  
 // Работа с userDataRepository  
 return null;  
 }  
 @PostMapping  
 public void sendConfirmationEmail(User user){  
 //работа с EmailSystem  
 //EmailSystem.sendEmail();  
 }  
 @PostMapping  
 public void sendPasswordRestorationEmail(User user){  
 //работа с EmailSystem  
 }  
}

Листинг А.4 – EmailSystem

package org.example;  
import org.springframework.stereotype.Service;  
import javax.mail.Message;  
import javax.mail.MessagingException;  
import javax.mail.Session;  
import javax.mail.Transport;  
import javax.mail.internet.InternetAddress;  
import javax.mail.internet.MimeMessage;  
import java.util.Properties;  
@Service  
public class EmailSystem {  
 public static void sendEmail(String to, String text, String subject){  
 String from = "sender@abc.com"; // sender email  
 String host = "127.0.0.1"; // mail server host  
 Properties properties = System.getProperties();  
 properties.setProperty("mail.smtp.host", host);  
 Session session = Session.getDefaultInstance(properties); // default session  
 try {  
 MimeMessage message = new MimeMessage(session); // email message  
 message.setFrom(new InternetAddress(from)); // setting header fields  
 message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));  
 message.setSubject(subject); // subject line  
 message.setText(text);  
 Transport.send(message);  
 } catch (MessagingException mex){ mex.printStackTrace(); }  
 }  
}

Листинг А.5 – PaymentSystem

package org.example;  
import org.springframework.stereotype.Service;  
@Service  
public class PaymentSystem {  
 public static void makePayment(){  
 //работа с внешней системой оплаты  
 }  
}

Листинг А.6 – Record

package org.example;  
import jakarta.persistence.Entity;  
import jakarta.persistence.Id;  
@Entity  
public abstract class Record {  
 Object nonSpecificData;  
 private Long id;  
 public void setNonSpecificData(Object nonSpecificData) {  
 this.nonSpecificData = nonSpecificData;  
 }  
 public Object getNonSpecificData() {  
 return nonSpecificData;  
 }  
 public void setId(Long id) {  
 this.id = id;  
 }  
 @Id

Окончание листинга А.6

public Long getId() {  
 return id;  
 }  
}

Листинг А.7 – RecordFactory

package org.example;  
public abstract class RecordFactory {  
 public Record createRecord (Object nonSpecificData, Object specificData){  
 Record record = getRecordWithNonSpecificData(nonSpecificData);  
 record = getRecordWithSpecificData(record, specificData);  
 return record;  
 }  
 public abstract Record getRecordWithNonSpecificData(Object nonSpecificData);  
 public abstract Record getRecordWithSpecificData(Record record, Object Specifi  
 cData);  
}

Листинг А.8 – RecordType

package org.example;  
import org.springframework.stereotype.Service;  
@Service  
public enum RecordType {  
 RESUME,  
 VACANCY  
}

Листинг А.9 – RecordDataRepository

package org.example;  
import org.springframework.stereotype.Repository;  
import java.util.List;  
@Repository  
public class RecordDataRepository {  
 public void insertVacancy(IVacancy vacancy){  
 //работа с SQL  
 }  
 public void insertResume(IResume resume){  
 //работа с SQL  
 }  
 public IVacancy getVacancy(int id){  
 //работа с SQL  
 return null;  
 }  
 public IResume getResume(int id){  
 //работа с SQL  
 return null;  
 }  
 public List<IVacancy> getAllVacancies(){  
 //работа с SQL  
 return null;  
 }  
 public List<IResume> getResumes(){  
 //работа с SQL  
 return null;  
 }

Окончание листинга А.9

}

Листинг А.10 – ResumeImpl

package org.example;  
import jakarta.persistence.Entity;  
@Entity  
public class ResumeImpl extends Record implements IResume{  
 Object resumeSpecificData;  
 @Override  
 public void setResumeSpecificData(Object resumeSpecificData) {  
 this.resumeSpecificData = resumeSpecificData;  
 }  
 public Object getResumeSpecificData() {  
 return resumeSpecificData;  
 }  
}

Листинг А.11 – ResumeFactoryImpl

package org.example;  
public class ResumeFactoryImpl extends RecordFactory{  
 @Override  
 public Record getRecordWithNonSpecificData(Object nonSpecificData) {  
 ResumeImpl resume = new ResumeImpl();  
 resume.setNonSpecificData(nonSpecificData);  
 return resume;  
 }  
 @Override  
 public Record getRecordWithSpecificData(Record record, Object resumeSpecificDa  
 ta) {  
 ResumeImpl resume = (ResumeImpl) record;  
 resume.setResumeSpecificData(resumeSpecificData);  
 return resume;  
 }  
}

Листинг А.12 – VacancyImpl

package org.example;  
import jakarta.persistence.Entity;  
@Entity  
public class VacancyImpl extends Record implements IVacancy{  
 Object vacancySpecificData;  
 private Long id;  
 @Override  
 public void setVacancySpecificData(Object vacancySpecificData) {  
 this.vacancySpecificData = vacancySpecificData;  
 }  
 public Object getVacancySpecificData() {  
 return vacancySpecificData;  
 }  
}

Листинг А.13 – VacancyFactoryImpl

package org.example;  
public class VacancyFactoryImpl extends RecordFactory {

Окончание листинга А.13

@Override  
 public Record getRecordWithNonSpecificData(Object nonSpecificData) {  
 VacancyImpl vacancy = new VacancyImpl();  
 vacancy.setNonSpecificData(nonSpecificData);  
 return vacancy;  
 }  
 @Override  
 public Record getRecordWithSpecificData(Record record, Object vacancySpecificD  
 ata) {  
 VacancyImpl vacancy = (VacancyImpl) record;  
 vacancy.setVacancySpecificData(vacancySpecificData);  
 return vacancy;  
 }  
}

Листинг А.14 – User

package org.example;  
import jakarta.persistence.Entity;  
import jakarta.persistence.Id;  
@Entity  
public abstract class User {  
 Object userData;  
 private Long id;  
 public void setUserData(Object userData) {  
 this.userData = userData;  
 }  
 public Object getUserData() {  
 return userData;  
 }  
 public void setId(Long id) {  
 this.id = id;  
 }  
 @Id  
 public Long getId() {  
 return id;  
 }  
}

Листинг А.15 – UserImpl

package org.example;  
import jakarta.persistence.Entity;  
@Entity  
public class UserImpl extends User{  
}

Листинг А.16 – UserFactory

package org.example;  
public abstract class UserFactory {  
 public abstract User createUser (Object userData);  
}

Листинг А.17 – UserFactoryImpl

package org.example;  
import org.springframework.stereotype.Service;

Окончание листинга А.17

@Service  
public class UserFactoryImpl extends UserFactory {  
 @Override  
 public User createUser(Object userData) {  
 User user = new UserImpl();  
 user.setUserData(userData);  
 return user;  
 }  
}

Листинг А.18 – UserDataRepository

package org.example;  
import org.springframework.stereotype.Repository;  
import java.util.List;  
@Repository  
public class UserDataRepository {  
 public void insertUser(){  
 //работа с SQL  
 }  
 public User getUser(int id){  
 //работа с SQL  
 return null;  
 }  
 public List<User> getAllUsers(){  
 //работа с SQL  
 return null;  
 }  
}