Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Кафедра информационных систем и программной инженерии

Курсовой проект

по дисциплине "Распределенные программные системы"

Тема курсового проекта:

Реализация прототипа информационной системы

«Организация предоставления и выполнения бытовых услуг»

Выполнил:

студент гр. ПРИ-120

Парахин К.В.

Приняла:

Проскурина Г.В

Владимир 2023 г.

**АННОТАЦИЯ**

В данном курсовом проекте производилась разработка программной системы для автоматизации работы агенства по организации предоставления и выполнения бытовых услуг.

Проект состоит из 3 основных этапов, включающих в себя основные три этапа, такие как: «Проектирование информационной системы», «Разработка прототипа системы» и «Тестирование системы и подведение итогов».

Первый этап под названием «Проектирование информационной системы», содержащий в себе такие блоки, как: описание предметной области, описание пользователей разрабатываемой подсистемы, описание прецедентов и сущностей системы, динамического моделирования системы, а также моделирования и разработки макета интерфейса отображения информации в системе.

Второй этап под названием «Разработка прототипа системы» состоит из реализации спроектированной системы в виде веб-сервиса с электронным сайтом с набором функционала, реализованном согласно возможностям платформы Java EE и фреймворка Spring Boot.

Третий этап под названием «Тестирование системы и подведение итогов» состоит из модульного (unit) тестирования логики работы системы, нагрузочного тестирования сервиса и выводов по результатам выполнения разработки.

Реализованная система может применяться для таких целей, как, например,

создание и хранение объявлений, продвижение услуг частных мастеров предоставление системы учета и планирования договоров клиентов, сопровождение услуг, осуществление взаимодействия пользователей.

Реализованная система спланирована для использования различными видами пользователей, имеющих различный функционал согласно их ролям и возможностям в системе.

Курсовой проект представлен на 58 страницах, основных рисунков - 14, использованных источников – 7, приложений – 2, иллюстрационный материал на 2 листах формата А1.

**ABSTRACT**

In this course project, a software system was developed to automate the work of the agency for the organization of the provision and performance of household services.

The project consists of 3 main stages, including the main three stages, such as: "Information system design", "System prototype development" and "System testing and summing up".

The first stage is called "Information system Design", which contains such blocks as: a description of the subject area, a description of the users of the subsystem being developed, a description of the precedents and entities of the system, dynamic modeling of the system, as well as modeling and development of the layout of the interface for displaying information in the system.

The second stage, called "System Prototype development", consists of the implementation of the designed system in the form of a web service with an electronic website with a set of functionality implemented according to the capabilities of the Java EE platform and the Spring Boot framework.

The third stage, called "System testing and summing up", consists of modular (unit) testing of the logic of the system, load testing of the service and conclusions based on the results of the development.

The implemented system can be used for such purposes as, for example,

the creation and storage of ads, the promotion of services of private masters, the provision of a system for accounting and planning customer contracts, the maintenance of services, the implementation of user interaction.

The implemented system is planned for use by different types of users with different functionality according to their roles and capabilities in the system.

The course project is presented on 58 pages, the main figures - 14, the sources used – 7, appendices – 2, illustrative material on 2 sheets of A1 format.

Содержание

[Заключение 28](#_Toc132927939)

[Список использованных источников 29](#_Toc132927940)

[ПРИЛОЖЕНИЕ А. Диаграмма прецедентов 30](#_Toc132927941)

ВВЕДЕНИЕ

Для начала выполнения разработки прототипа информационной системы «Огранизация предоставления и выполнения бытовых услуг» требуется выполнить следующие шаги по проектированию:

- описать предметную область

- выделить ее основные функциональные требования, с помощью основных UML-диаграмм описать составы данных и категории рассматриваемой системы

- провести описание основных прецедентов пользователей, сущности системы

- провести динамическое моделирование системы

- осуществить моделирование основных бизнесс-процессов в методологии IDEF0 (as-is)

- разработать макет интерфейса системы

- выбрать средства разработки и основные фреймворки, требующиеся для реализации функционала

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Начнем с описания предметной области и основных ее особенностей.

Предметная область была кратко названа : «Предоставление и выполнение бытовых услуг». Что же значит это определение на практике?

Представим ситуацию, когда какому-то человеку A требуется выполнить какую-то работу (например, сделать ремонт, перевезти вещи, заменить проводку в квартире и т.д.). Он может сделать это сам, а может нанять специалиста, который сделает все это гораздо быстрее и качественнее за определенную цену.   
Казалось бы, это достаточно распространенная практика в наше время.

Но что если автоматизировать данный процесс? Для удобства обоих сторон (исполнителя и нанимателя) создать электронную систему, благодаря которой будет возможно очень быстро организовать подачу заявок, обратную связь и отклик потенциальных «работников».

Благодаря этому, клиенты системы могут быстро связываться с друг другом, договариваться по всем вопросам, выезжать к месту выполнения работы или оказания услуги.

Основные особенности:

- исполнители, как и заказчики являются полноценными пользователями системы, могут в зависимости от цели пользования системой, менять свои роли (то есть им доступен весь базовый функциональ системы)

- система представляет собой автоматизированной приложение, имеет свой электронный сайт

- существует разделение ролей по значимости, кроме обычных клиентов, присутствуют администраторы, отвечающие за рассмотрение подозрительных заявок, выполнение регистрации и аутентификации некоторых пользователей, за расположение рекламы на сайте, а также модерацию раздела комментариев и отзывов пользователей

Описание набора функций системы, определяющего границы предметной области.

Основные функции разрабатываемой системы:

- зарегистрироваться в системе, создать личный кабинет;

- создать заявку на выполнение работы (или оказания услуги), то есть иначе говоря разместить объявление на электронном сайте (в некоторых подозрительных случаях требуется проверка заявки администратором);

- просмотреть каталог заявок на сайте, отфильтровать каталог, выбрать желаемую к выполнению услугу;

- связаться с заказчиком через чат сайта (или получить доступ к телефонному номеру или адресу социальной сети);

- оставить отзыв о проделанной работе (он отобразится в личном кабинете исполнителя), сформировать рейтинг заказчика;

- пожаловаться на работу сайта / команды модерации / исполнителя

- рекламно продвинуть свою заявку в списке остальных

Описание категорий пользователей системы и прецедентов

Как и было сказано ранее, система включает в себя разделение ролей пользователей по значимости:

- обычный пользователь (клиент системы)

- модератор системы

- администратор системы

Обычные пользователи в свою очередь делятся на категории:

- новые пользователи, не зарегистрированные в системе

- пользователи с личным кабинетом и зарегистрированные в системе

По характеру пользования системой клиенты также делятся на такие категории, как:

- заказчики (наниматели), размещающие объявления на сайте;

- исполнители услуг (которые могут откликаться на объявления)

Модераторы системы занимаются деятельностью, связанной с редактированием объявлений, размещаемых клиентами, а также их отзывов.

Администраторы системы занимаются выдачей ролей зарегистрированным пользователям, а также созданием и редактированием организаций, услуги которых предоставляют исполнители, а также редактированием профилей пользователей (напримех, их рейтингов).

Функциональная декомпозиция системы – выделение и анализ прецедентов в системе

Основные прецеденты:

- Управлять своим аккаунтом (клиент системы)

- Зарегистрироваться в личном кабинете

- Авторизоваться

- Редактировать профиль

- Разместить объявление (заказчик)

- Выбрать тип объявления и основные параметры

- Описать стоимость услуги и требования к ее выполнению

- Осуществить отбор объявления на сайте (исполнитель)

- Пользоваться поиском на сайте

- Произвести фильтрацию объявлений

- Произвести сортировку объявлений

- Выполнять интерактивные действия с объявлением

- Просмотреть объявление

- Откликнуться на объявление

- Создать «комнату» услуги (клиенты системы – заказчик и исполнитель)

- Пользоваться пользовательским чатом в комнате

- Получать уведомления о действиях коллеги

- Написать отзыв о частном исполнителе (заказчик)

- Администрировать сайт (администратор системы)

- Выдать роль зарегистрированному клиенту

- Добавлять и актуализировать огранизации

- Осуществлять модерацию сайта (модератор системы)

- Редактировать объявления пользователей

- Редактировать (удалять) отзывы пользователей

Диаграмма прецедентов в нотации UML изображена в приложении А к данной пояснительной записке (рисунок 1).

Описание прецедентов

Прецедент «Разместить объявление»

Предусловие: Пользователь решил разместить объявление и нажал на соответствующую кнопку

Действующее лицо: Заказчик

Основной поток: Клиент открывает в браузере электронный сайт (представление веб-приложения, затем нажимает на кнопку «Разместить объявление», открывается окно для создания объявления, пользователь вводит необходимые данные и нажимает кнопку «создать», объявление проходит модерацию и выводится на поле «Список объявлений».

Альтернативный поток: Пользователь создаёт объявление, однако оно не проходит модерацию, тогда, через некоторое время пользователю приходит уведомление о том, что объявление не прошло модерацию и необходимо пересоздать объявление.

Прецедент «Выбрать тип объявления»

Предусловие: Пользователь решил откликнуться на объявление(Принять заказ) и нажал соответствующую кнопку.

Действующее лицо: Исполнитель

Основной поток: Клиент открывает в браузере электронный сайт (представление веб-приложения), затем в меню фильтрации выбирает требуемый тип объявления.

Постусловие: при отстутствии объявлений по выбранному типу объявлений происходит переадресация на страницу с более общим типом обьявлений.

Прецедент «Зарегистрироваться»

Предусловие: Пользователь решил зайти на электронный сайт и воспользоваться каким-то функционалом системы (то есть не только просматрировать каталог).

Действующее лицо: Пользователи: заказчик и исполнитель

Основной поток: Клиент открывает в браузере электронный сайт, на главной странице в правом верхем углу видит раздел для регистрации. Затем проходит регистрацию, указывая свои учетные данные.

Либо нажимает, например, кнопку «Разместить обявление», и его автоматические перенаправляет в раздел авторизации и аутентификации.

Альтернативный поток: Пользователь уже зарегистрирован в системе. Тогда при попытке зарегистрироваться с уже сохраненными в системе учетными данными, он будет уведомлен о наличии зарегистрированной учетной записи с введенными данными.

Постусловие: При наличии зарегистрированной учетной записи, пользователь будет перенаправлен на страницу авторизации.

Прецедент «Авторизоваться»

Предусловие: Пользователь решил воспользоваться функционалом системы при уже имеющейся учетной записи, то есть пользователь уже зарегистрирован в системе.

Действующее лицо: Пользователи (исполнитель и заказчик).

Основной поток: пользователь открывает в браузере электронный сайт, на главной странице видит раздел для авторизации, затем вводит свои данные и получает доступ к своей личной учетной записи и к личному кабинету.

Альтернативный поток: Пользователь ещё не зарегистрирован в системе. Тогда при попытке авторизоваться с некоторыми данными, он будет уведомлен об отсутствии требуемой учетной записи.

Постусловие: При отсутствии зарегистрированной учетной записи, пользователь будет перенаправлен на страницу регистрации.

Прецедент «Создать комнату услуги»

Предусловие: Пользователь решил начать выполнять работу / либо найти исполнителя для выполнения своей выполненной услуги.

Действующее лицо: Пользователи (исполнитель и заказчик).

Основной поток:

- Исполнитель: прочитав условия предложения , исполнитель решается принять заказ, он нажимает кнопку «Откликнуться» и оджидает подтверждения заказчика для начала работы.

- Заказчик: зайдя в личный кабинет, видит, что на его объявление откликнулся исполнитель и рассматривает его кадидатуру. Он или принимает, или отклоняет предложение исполнителя выполнить заказ, с помощью кнопок «Принять» и «Отклонить».

Прецедент «Осуществить модерацию сайта»

Предусловие: Информация на сайте требудет изменения из-за нарушения правил сообщества

Действующее лицо: Модератор

Основной поток: Определённый пользователь заходит свой аккаунт под учётной записью модератора. Затем он заходит на определённое объявление и изменяет значения в полях, проверяет активность пользователей и может заблокировать пользователя или разблокировать его.

Прецедент «Администрировать сайт»

Предусловие: Информация на сайте требует изменения для усовершенствования сайта.

Основной поток: Определённый пользователь заходит свой аккаунт под учётной записью администратора. По необходимости он изменяет роли, изменяет возможности пользовтелей и видимость элементов.

Описание состава данных, основных документов и их атрибутов

Выделим следующие абстракции данных (сущности системы):

- User (пользователь с некоторыми учетными данными. Он может быть как заказчиком, так и исполнителем – причем одновременно для разных событий)

- Organization (некоторая частная организация, к которой прикреплен пользователь. Требуется для учета прибыли частных исполнителей)

- Role (Системная роль пользователя: администратор, модератор, обычный пользователь).

- Task (Задача, поставленная на исполнение заказчиком и выполняемая исполнителем)

- Action (Некоторое интерактивное действие, связанное с определенным пользователем и объектом на сайте – то есть в данном случае объявлением)

- Event (Событие, связанное с объявлением и действием пользователя на сайте. По его началу создается «комната» объявления, к которой имеют доступ два пользователя – заказчик, создавший объявление и подтвердивший создание комнаты и исполнитель – откликнувшийся на объявление)

- Notification (Уведомление, приходящее заказчику при отклике исполнителя, а также исполнителю – при подтверждении создания «комнаты» объявлению заказчиком).

- ChatMessage (сообщение в пользовательском чате, связанное с «комнатой» конкретного объявления)

- Review (Отзыв с оценкой о работе исполнителя, публикуемый по окончании события заказчиком).

Диаграмма классов системы в нотации UML изображена в приложении Б к данной пояснительной записке (рисунок 2).

Построим ER – диаграмму базы данных системы физического уровня (старая упрощенная версия, планировавшаяся к реализации в СУБД MySQL):

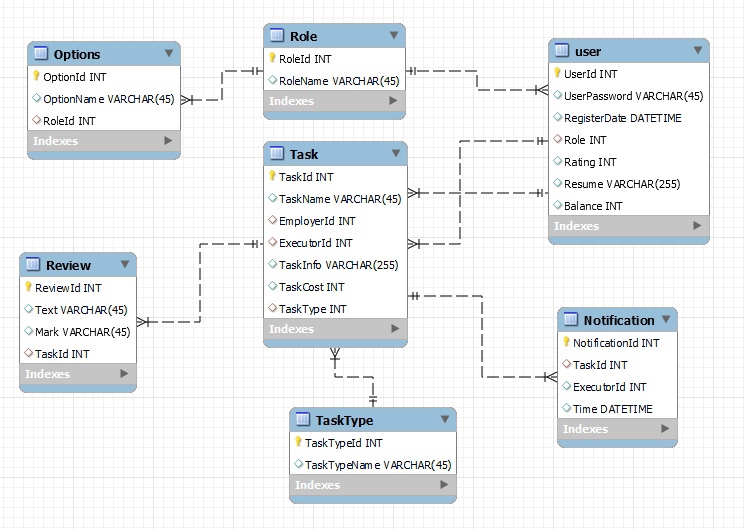


Рисунок 3. ER-диаграмма БД физического уровня

Динамическое моделирование системы

Диаграмма состояний UML (UML Statechart)

Составить диаграмму состояний для сущности «Объявление» (Employer Task – задача, которую работодатель размещает и ждет готовых к её выполнению работников):

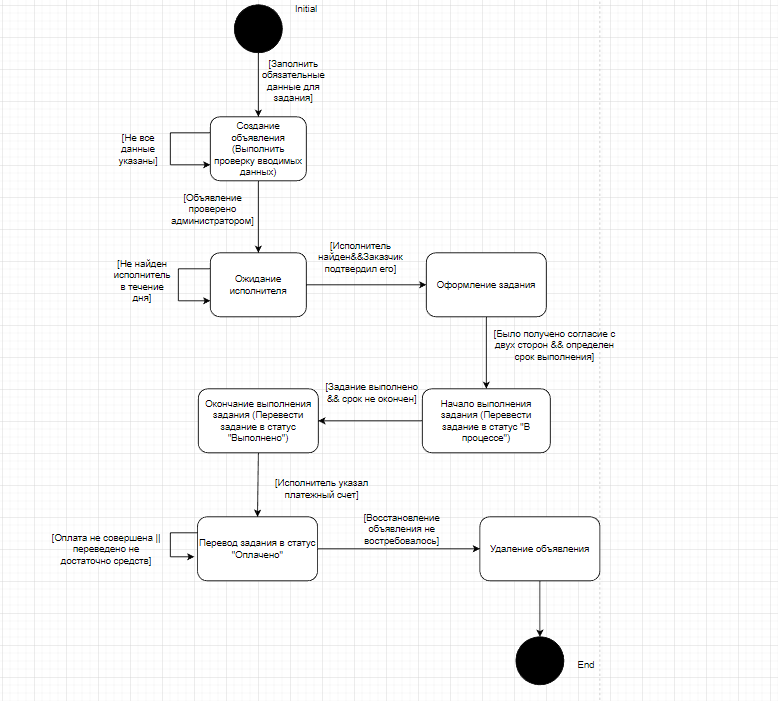


Рисунок 4. Диаграмма состояний для объекта «Объявление»

Диаграмма последовательностей (UML Sequence)

Составить диаграмму последовательностей для последовательности, по которой работодатель работает с системой, создавая объявление (задачу – Task), ожидая и затем по готовности подтверждая ее выполнения.

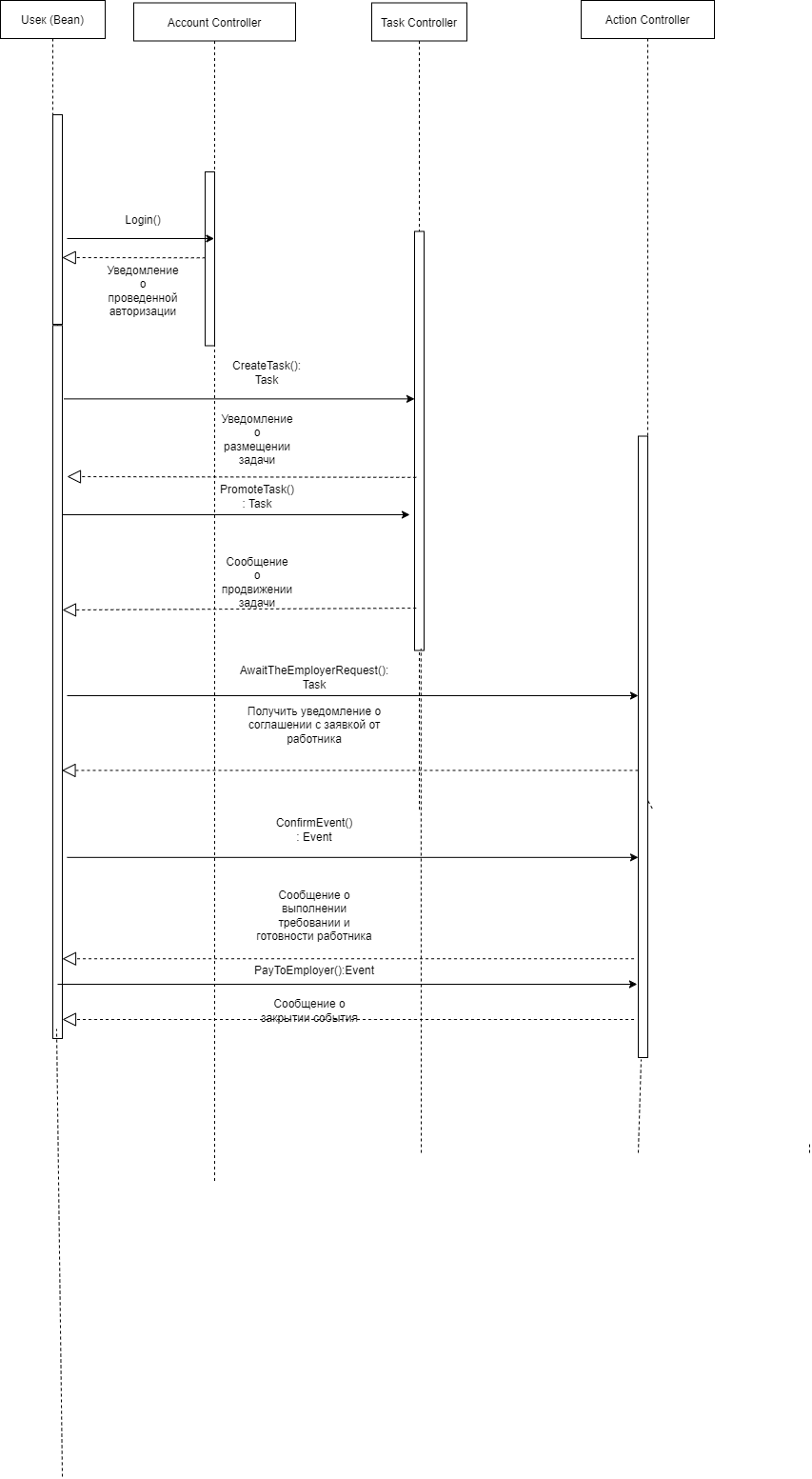


Рисунок 5. Диаграмма последовательностей

Построение диаграммы бизнесс-процесса модели системы в нотации IDEF0 трех уровней.

Бизнесс-процесс – работа сервиса по организации предоставления и выполнения бытовых услуг.

Модель as-is данного автоматизируемого бизнесс-процесса представлена ниже на рисунках 6.1 - 6.3:

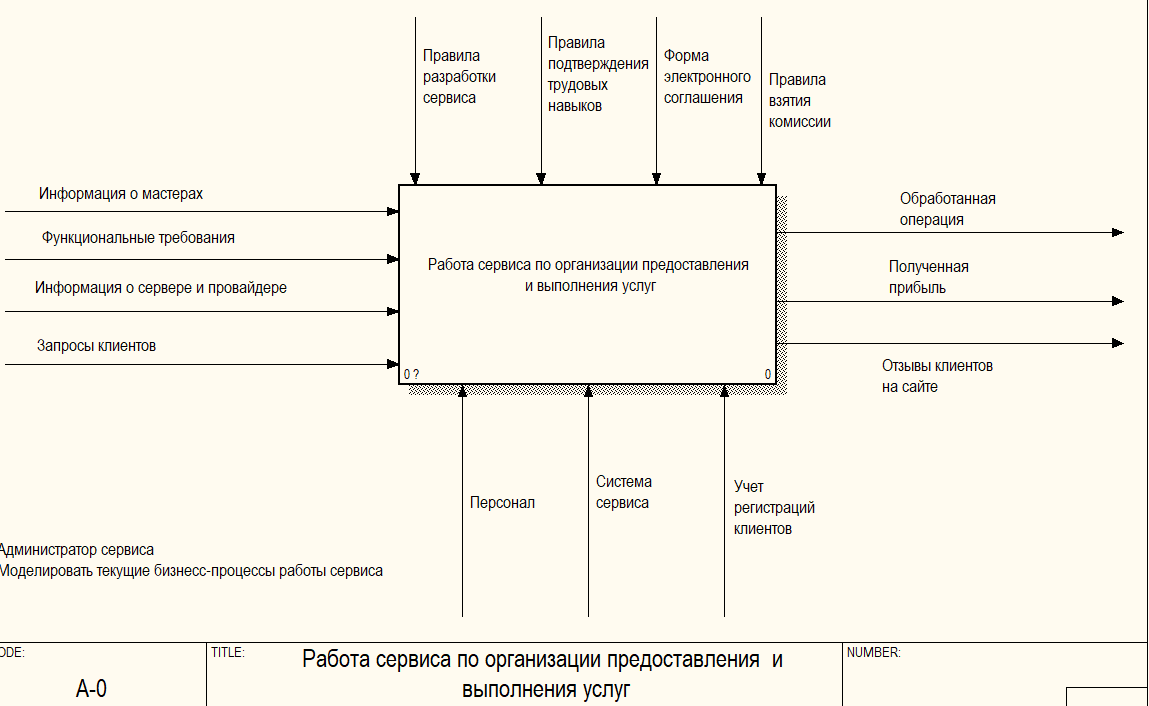


Рисунок 6.1. Диаграмма модели IDEF0

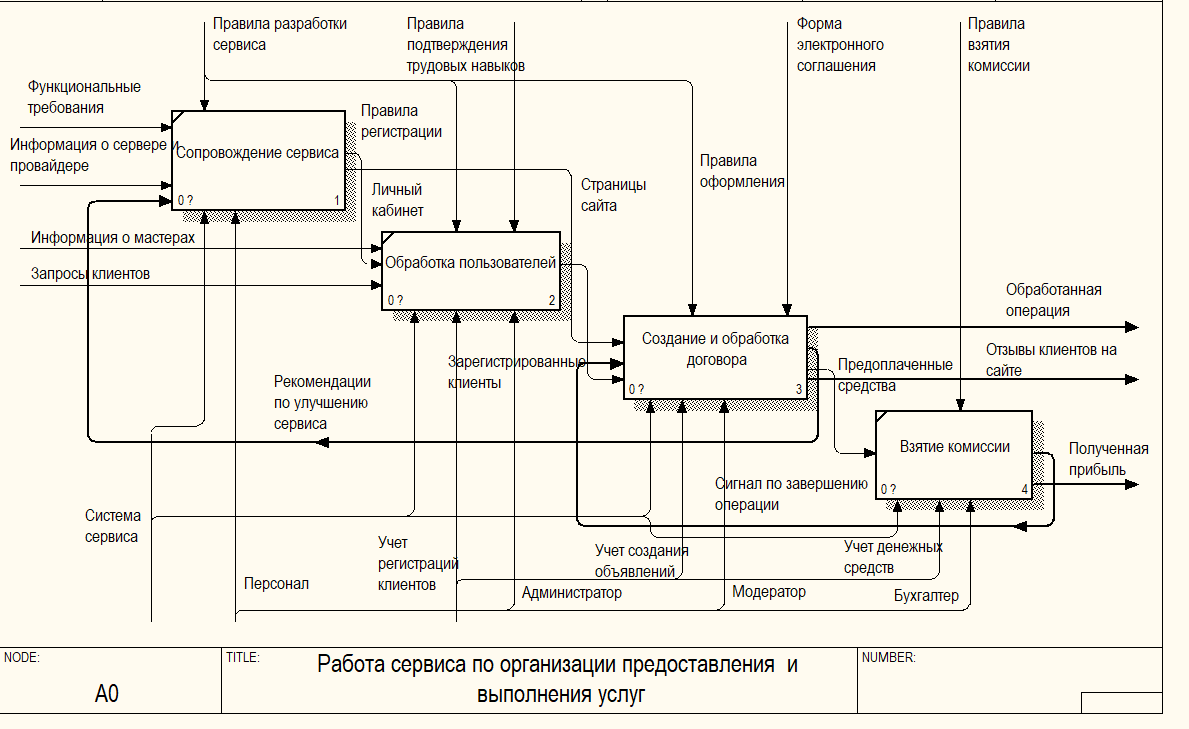


Рисунок 6.2. Диаграмма модели IDEF0 первого уровня

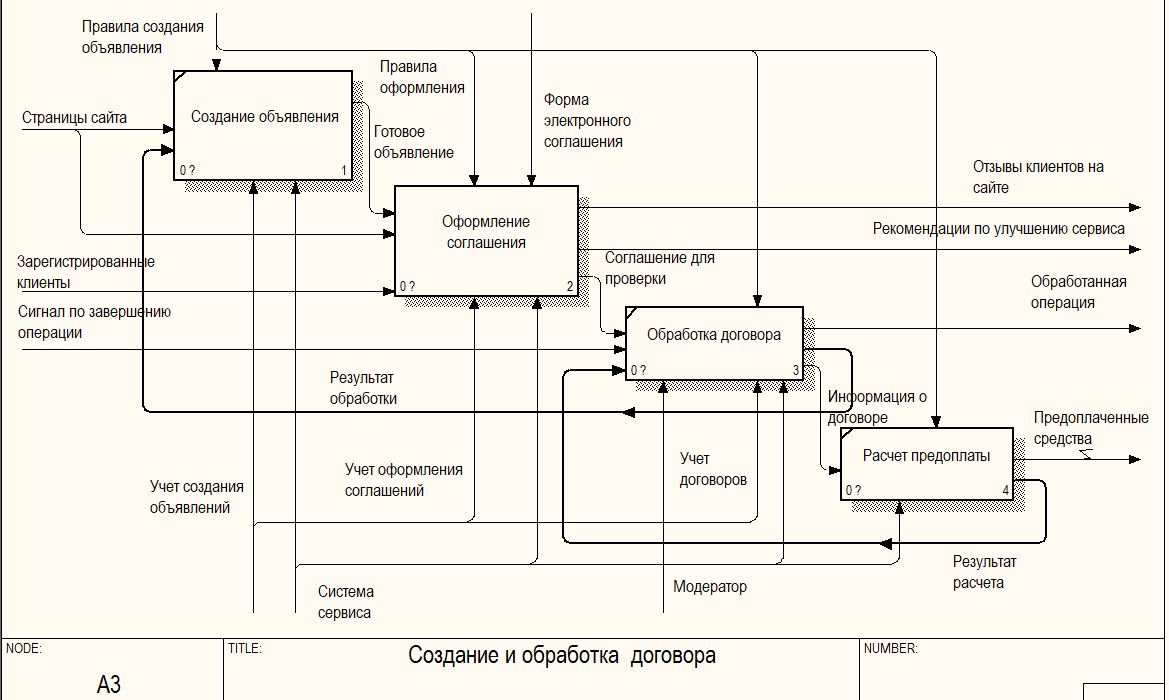


Рисунок 6.3. Диаграмма модели IDEF0 второго уровная

Список требований к разрабатываемой системе

Выделим основные категории требований, которые необходимо предъявить к анализируемой выше разрабатываемой системе:

- Функциональные требования (то есть необходимые минимальные функции для прототипа):

- Возможность использовать авторизацию и аутентификацию пользователей

- Возможность регистрации пользователя, выбора роли и создания личного кабинета

- Возможность добавлять аватар и картинку для объявления

- Предоставить возможность разместить объявление

- Предоставить возможность выбрать объявление и пользоваться поиском, фильтрацией и сортировкой в каталоге

- Выбирать и просматривать постраничный (пагинированный) список объявлений на сайте

- Предоставить возможность откликнуться на объявление

- Предоставить возможность пользоваться чатом для осуществления взаимодействия пользователей

- Организовать систему уведомлений о действиях пользователей

- Реализовать оповещения пользователей

- Возможность оставлять отзывы

- Модерировать объявления и сообщения (обладая функционалом модератора)

- Выдавать пользователям роли (будучи администратором)

- Создавать и редактировать организации (будучи администратором)

- Нефункциональные требования (то есть требования, определяющие свойства, которые система должна демонстрировать, или ограничения, которые она должна соблюдать, не относящиеся к поведению системы):

- Трафик запросов (не менее 1000 одновременных запросов от разных пользователей в небольшой промежуток времени)

- Надёжность (система работает 24 на 7 и предоставляет возможность для работы с ней)

- Кроссплатформенность (возможность запускать приложение на операционных системах семейства Windows)

- Адаптивность (корректное отображение интерфейса в различных браузерах, например, в Google Chrome и Mozilla Firefox)

- Масштабируемость системы (расширяемость под большее кол-во функционала, сервисных обращений пользователей)

- Использование обозначенных прикладных средств разработки:

- Платформа Java EE

- Фреймворк SpringBoot

- База данных PostgreSQL

- Наличие удобного пользовательского интерфейса­­­ (написанного на языке гипертекстовой разметки HTML)

- Выполнение утонченного дизайна интерфейса сайта (с использованием CSS)

Прототипирование макета интерфейса сайта

Ниже (на рисунке 7) представлена схема главного окна прототипа сайта системы:

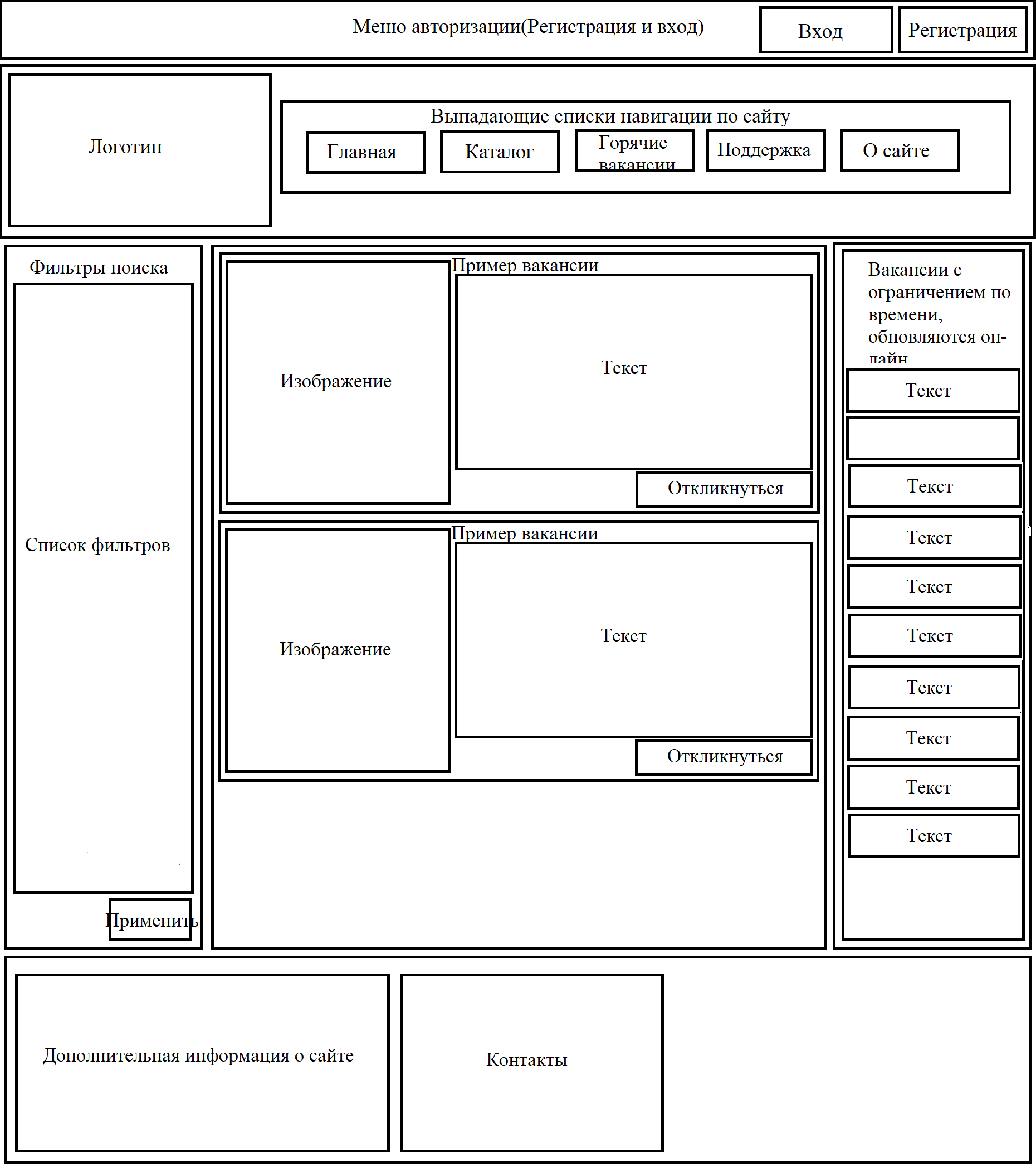


Рисунок 7. Схема главного окна сайта

РАЗРАБОТКА ПРОТОТИПА СИСТЕМЫ

После проектирования системы по организации предоставления и выполнения бытовых услуг техники требуется выполнить реализацию основных компонентов, таких как:

- База данных системы, в которой будет храниться информация о всех пользователях, зарегистрированных в системе, их объявлениях и действиях на сайте

- Сервис системы, представляющий набор контроллеров и зависимостей логических компонентов, благодаря взаимодействию которых происходит работа системы.

- Электронный сайт для системы, который будет открываться в веб-браузере и доступен для просмотра пользователям системы. Он будет состоять из набора представлений, сгенерированных на основе созданных моделей объектов системы, а также стандартной HTML – верстки (с использованием CSS и стандартных скриптов библиотеки Bootstrap).

В процесс реализации я буду использовать такую среду программирования, как IntelliJ IDEA 2022 и платформу Java EE. Разработка будет вестись с использованием фреймворка Spring Boot 3.

Основой для решения будет являться создание проекта MVC с веб-архитектурой, представляющего сервис системы.

Дополнительно в директории проекта будут созданы папки, в которые будут добавляться такие классы, как:

- Модели системы, на основе которых будет создаваться база данных

- Перечисления (на основе сопоставления которых будут типизироваться экземпляры создаваемых моделей в системе)

- Логика системы, к которой обращаются контроллеры (сервисы)

- Репозитории моделей системы, к которым обращаются методы сервисо- Unit-тесты к компонентам системы, благодаря которым можно проверять функциональность логики системы, а также разрабатывать и прорабатывать всякие возможные сценарии их работы, то есть использовать подход TDD к проектированию логики.

- Unit-тесты к компонентам системы, благодаря которым можно проверять функциональность логики системы, а также разрабатывать и прорабатывать всякие возможные сценарии их работы, то есть использовать подход TDD к проектированию логики.

- Ресурсы веб-приложения, содержащие набор представлений (views – в формате .HTML), а также базовых скриптов для создания и начальных миграций базы данных системы.

Разработка backend – части системы:

Начнем с создания базы данных. В процессе разработки использовался такой фрэймфорк, как Hibernate (а конкретно – Java Persitense API), а также Postgre SQL Server – СУБД для созданной БД.

Использованный подход – Code First (то есть базы данных сгенерировалась после кодирования основных зависимостей её компонентов) – на рисунке 6.1. представлены созданные таблицы в базе данных (показанные в программе pgAdmin):

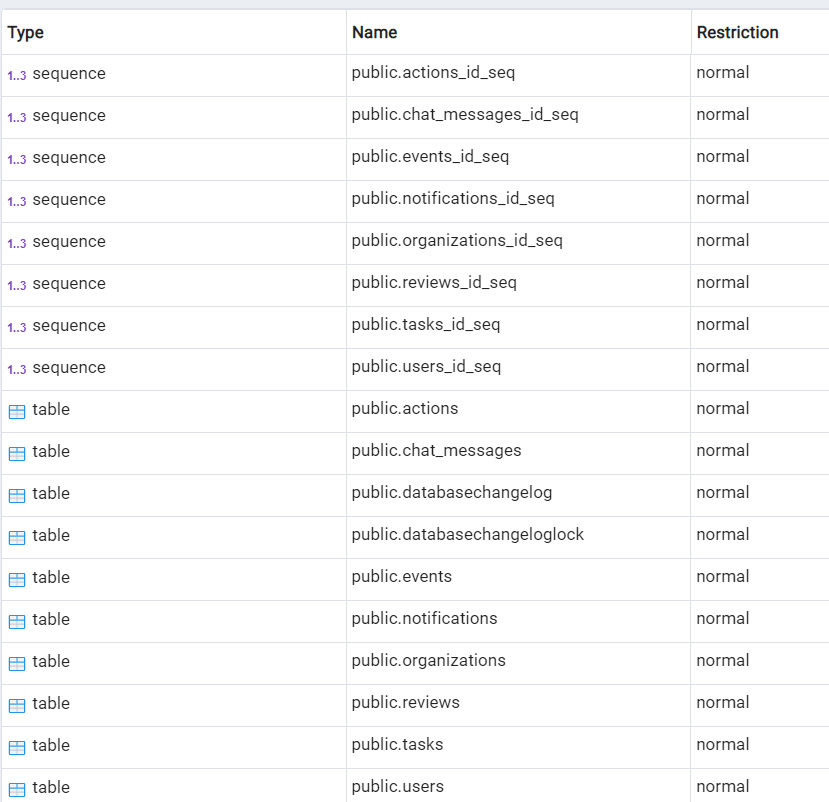


Рисунок 8.1. Основные таблицы и зависимости в pgAdmin

В процессе разработки было принято решение оставить перечисления на стороне сервиса и не создавать такие таблицы, как RoleType, TaskType, ActionType, EventStatus и т.д. в базе данных. Это возможно благодаря сопоставлению JPA (аннотации – @Enumerated(value = EnumType.STRING)).

Также в ресурсы веб-приложения был добавлен файл import.sql, который содержит данные, которые будут добавлены в БД после её создания – рисунок 8.2:

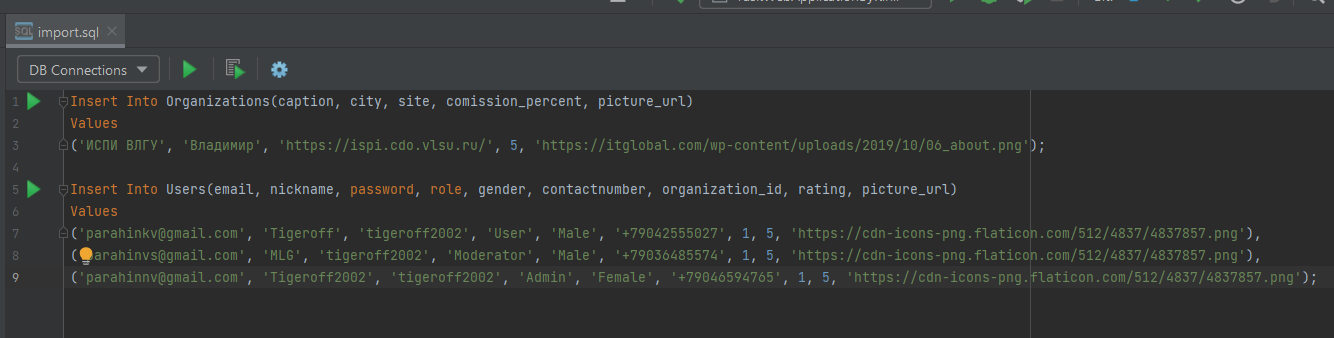


Рисунок 8.2. Первоначальная миграция

Все настройки конфигурации были произведены в файле application.properties, расположенном в ресурсах приложения.

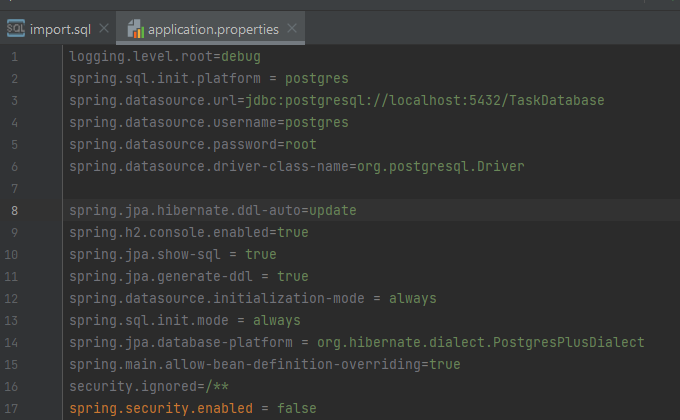


Рисунок 8.3. Конфигурационный файл веб-приложения Spring

Разработка веб-сервиса

Далее перейдем к разработке сервиса системы. Как я уже говорил, я использовал шаблон MVC. Модели были мною уже написаны и находятся в отдельной директории (Beans) и благодаря внедрению зависимостей я могу получить к ним доступ из любого места проекта согласно принципу SOLID.

Дополнительно использовались еще модели в директории ExtraBeans (модели, наследующие классы из папки beans – обладающие вспомогательные полями и флагами, помогающие в отобращении важной дополнительной информации на представлениях, например, лайков, значков просмотров, а также параметров фильтрации, сортировки и пагинации).

В отдельной директории enums были созданы основные перечисления, на основе которых сопоставляются и типизируются экземпляры объектов в базе данных и сервисе системы.

Список всех основных моделей - сущностей и перечислений (показаны ниже на рисунках 8.4 и 8.5, соответственно).

Beans:

* User
* Organization
* Task
* Action
* Event
* Notification
* Review
* ChatMessage

Enums:

* RoleType
* TaskType
* TaskStatus
* ActionType
* EventStatus
* NotificationType
* FilterBy
* SortBy

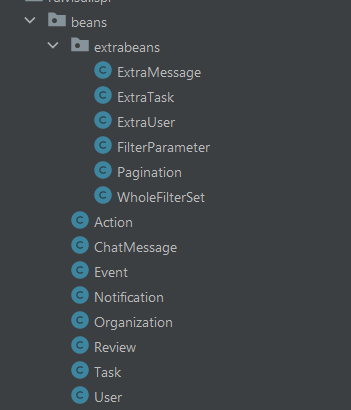


Рисунок 8.4. Файлы с классами доменных моделей

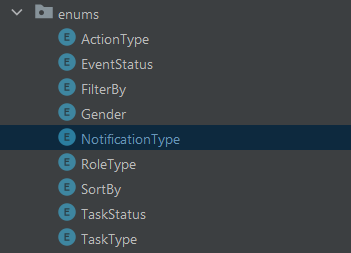


Рисунок 8.5. Файлы с перечислениями

Далее я написал несколько основных контроллеров (которые представлены ниже ни рисунке 8.6):

* HomeController – это API-Controller, находящийся по адресу (маршруту) «home» и автоматически запускаемый при старте веб-приложения (а конкретно запускается его HTTP-GET метод DefaultIndex – который представляет собой показ каталога товаров на сайте). По умолчанию, и авторизованные, и неавторизованные пользователь могут просматривать каталог, а также проходить по страницам с более подробной информацией о товарах.

Некоторые элементы, такие как ссылки на страницы для менеджеров и администраторов, скрыты от остальных пользователей.

* AccountController – это API - Controller, находящийся по адресу (маршруту) « account». Его методы используются при регистрации/авторизации пользователей на сайте, а также в процессе аутентификации (то есть определения роли пользователя в систему и, соответсвенно, их функционала на электронном сайте).

По умолчанию пользователю не предлагается регистрироваться или пройти авторизацию, но при попытках выполнить действия «Создать новое объявление» - пользователь будет автоматически перенаправлен на страницу авторизации. Сделано это для оптимизации работы системы, чтобы не приходилось создавать записи в базе данных и асинхронно использовать ресурсы backend – части системы.

- TaskController – это контроллер, который используется при всех действиях пользователя, связанных с созданием, редактированием и любыми активностями с объявлением.

- Actions Controller – это контролер, который используется для обработки интерактивного взаимодействия пользователя с системой.

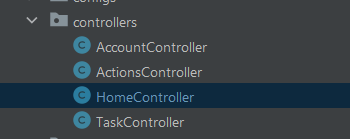


Рисунок 8.6. Файлы с контроллерами

Далее была разработана логическая система, отвечающая за основные действия пользователя, создание объявлений и операции над ними, обработка интерактивного взаимодействия пользователя с системой, создание «комнаты» объявления, добавление сообщений в чат, создание и отображение уведомлений у пользователей в личном кабинете и так далее.

Для этих целей был создана директория logic, в которой располагаются файлы-сервисы (services) – представлены ниже на рисунке 8.7.

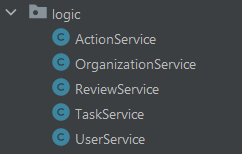


Рисунок 8.7. Файлы с сервисами

Для работы с запросами к данными и для создания более удобной и масштабируемой системы, также были созданы такие классы, как репозитории – ниже на рисунке 8.8 представлены файлы с логикой репозиториев:

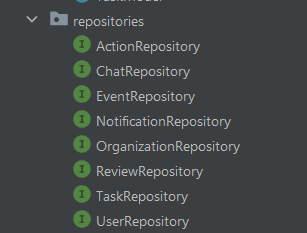


Рисунок 8.8. Файлы с классами репозиториев

Стоит также упомянуть про используемые средства для создания системы аутентификации и авторизации. При разработке я применил стандартный подход к обеспечению безопасности веб-приложения – Spring Security с использованием класса WebSecurityConfig (представлен в приложении В среди показательного кода программной системе). Дополнительно я использовал класс HttpSession, благодаря я добавлял в сессию браузера идентификатор текущего пользователя, а также информации по пагинации, фильтрации и сортировке объявлений на странице. Сделано это было для улучшения структуризации страницы и удобства разработки.

Разработка frontend – части системы:

Кроме этого еще велась разработка внешнего вида системы, а именно создания электронного сайта, который состоит из отдельных страниц (или views – представлений). Для разработки веб-страниц использовался фреймворк Thymeleaf, дающий возможность написания представлений, объектно-ориентированных на атрибуты моделей и написанных на чистом HTML.

На рисунках 9.1 – 9.18 показаны скриншоты электронного сайта:

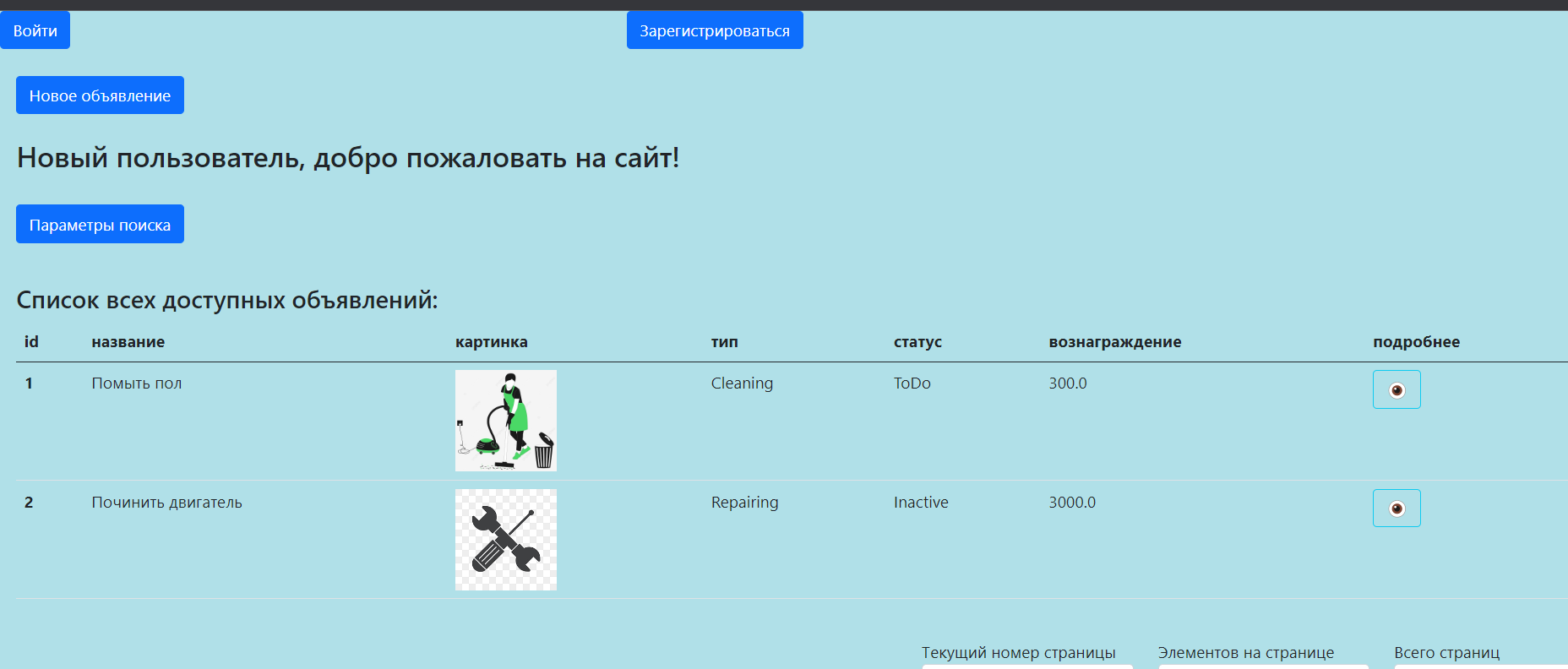


Рисунок 9.1.1. Каталог сайта (нового пользователя)

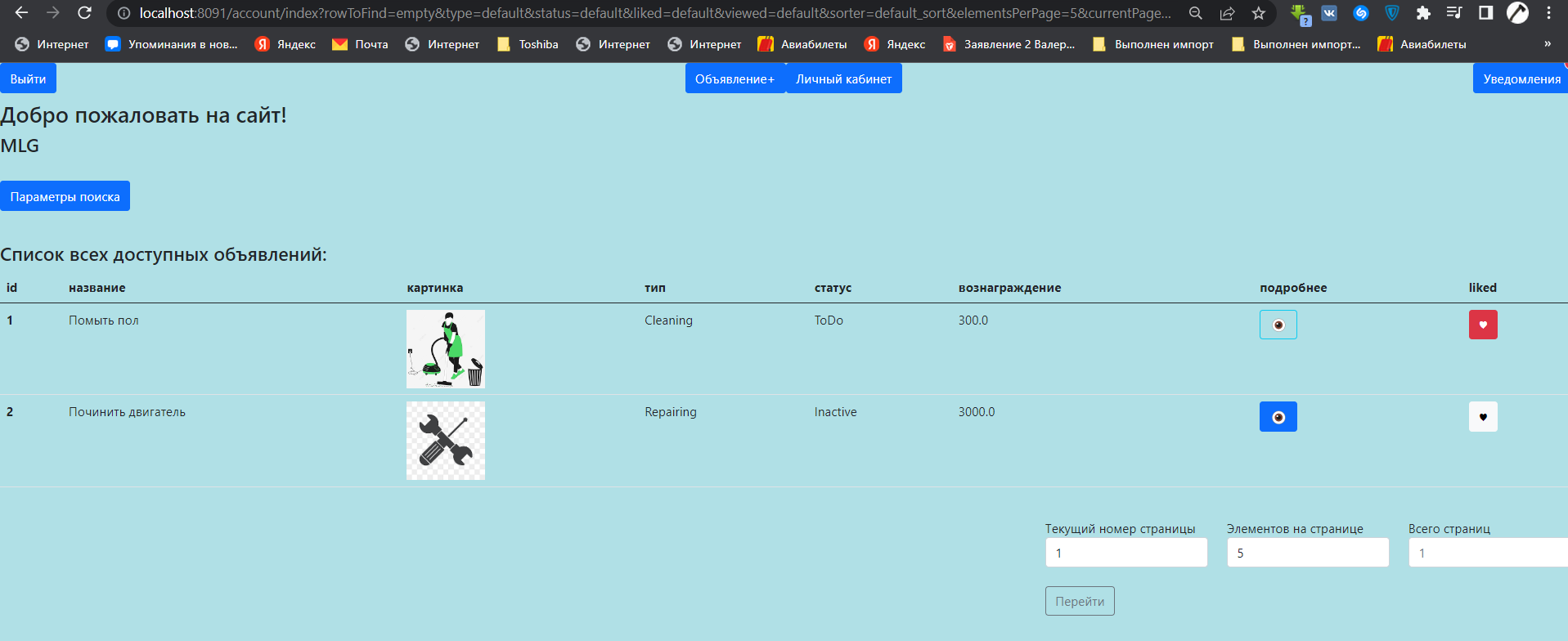


Рисунок 9.1.2. Каталог сайта (зарегистрированного пользователя)

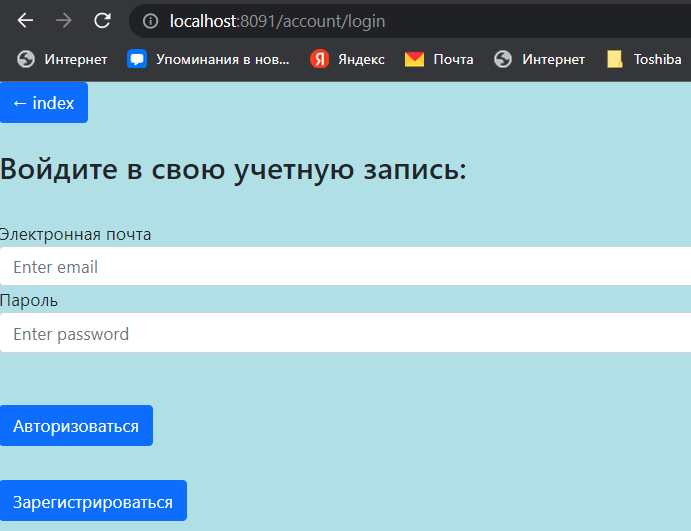


Рисунок 9.2. Форма для авторизации на сайте

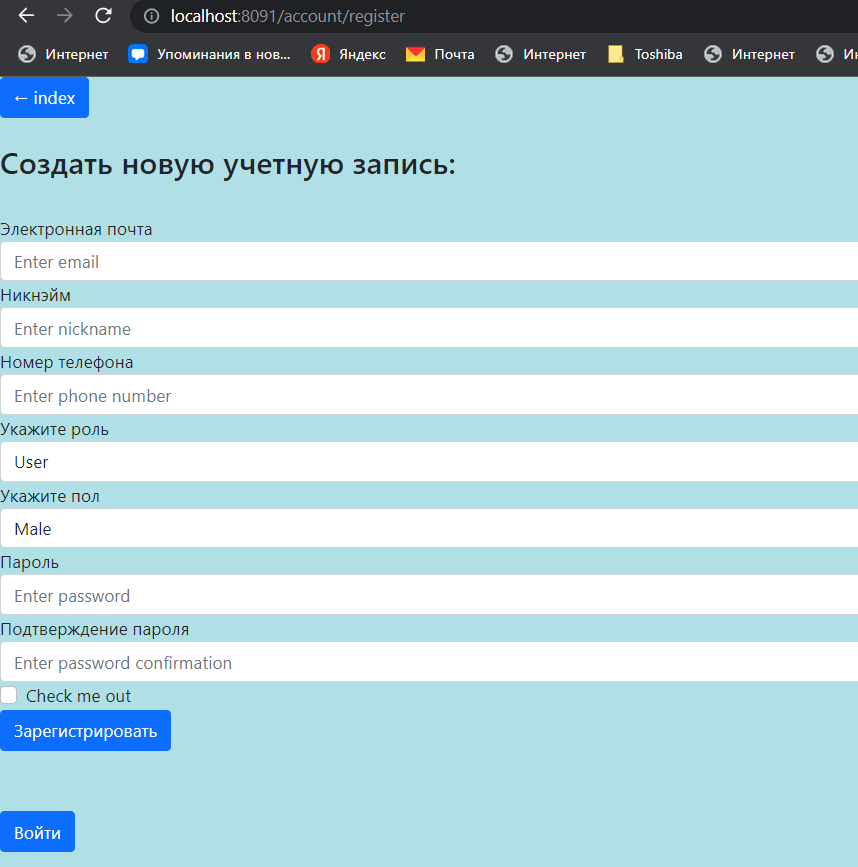


Рисунок 9.3. Форма регистрации на сайте

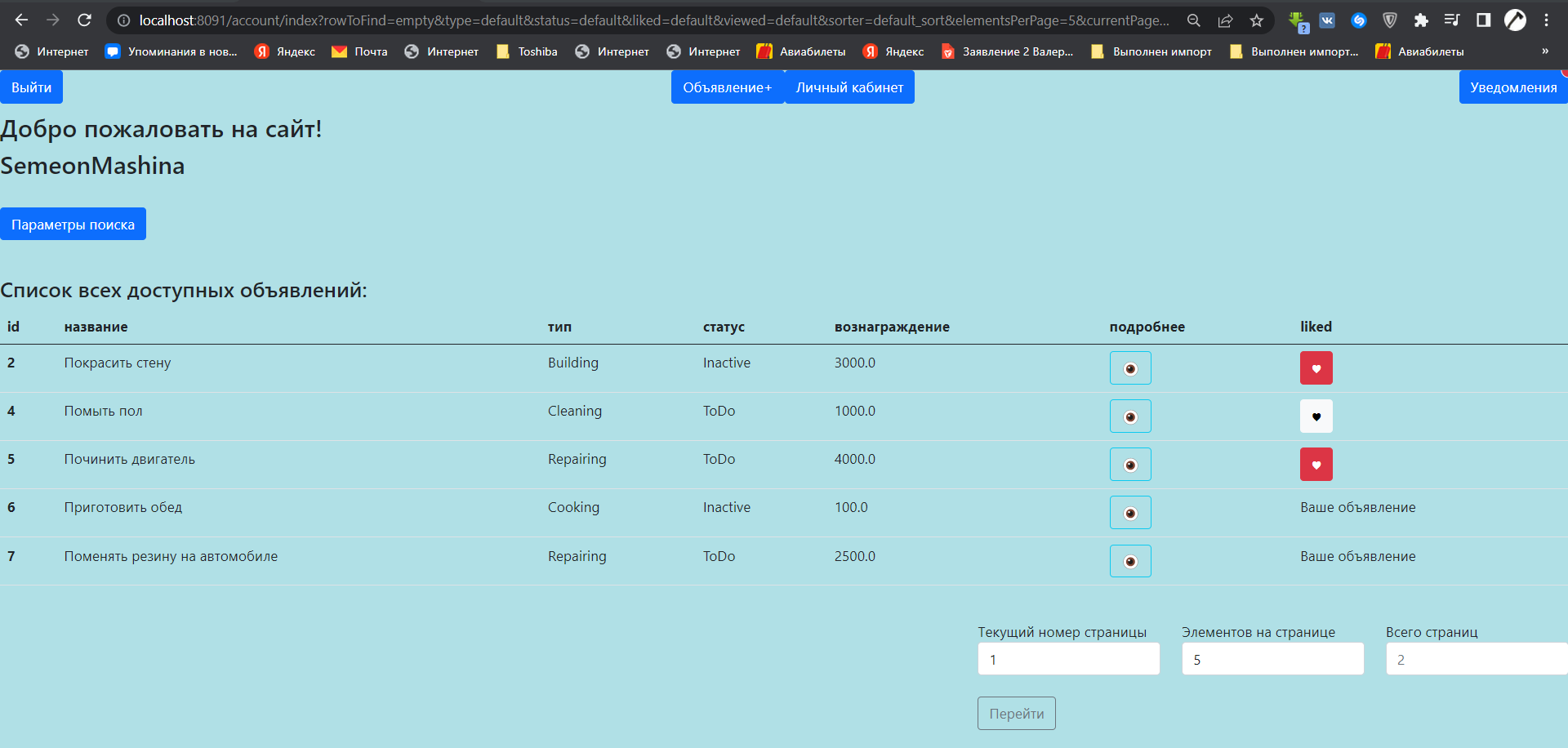


Рисунок 9.4. Каталог авторизованного пользователя

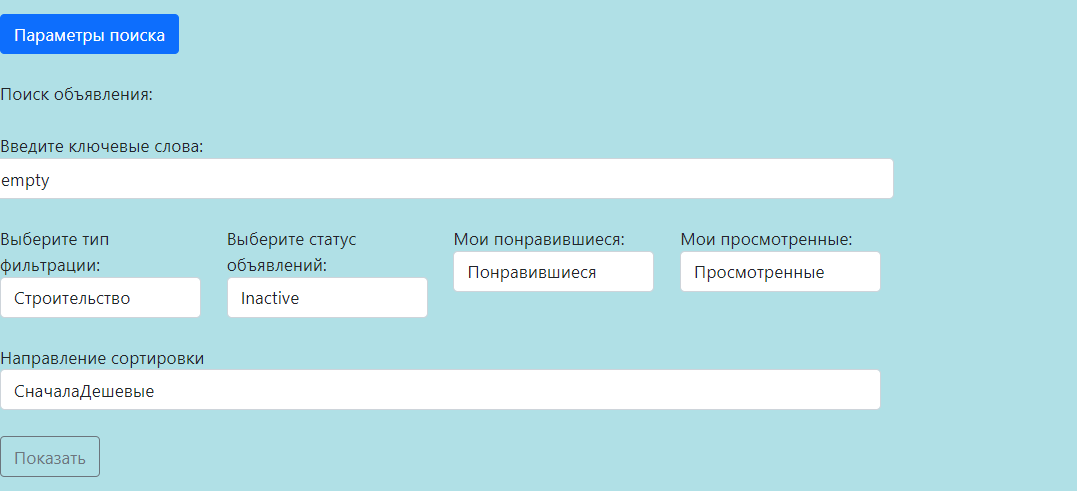


Рисунок 9.5. Параметры поиска, фильтрации и сортировки

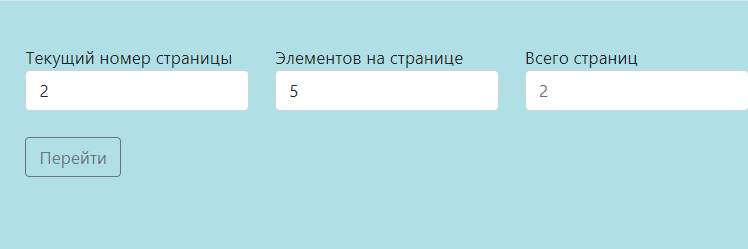


Рисунок 9.6. Панель пагинации

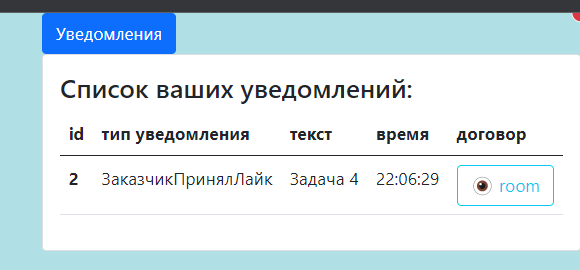


Рисунок 9.7. Панель просмотра уведомлений

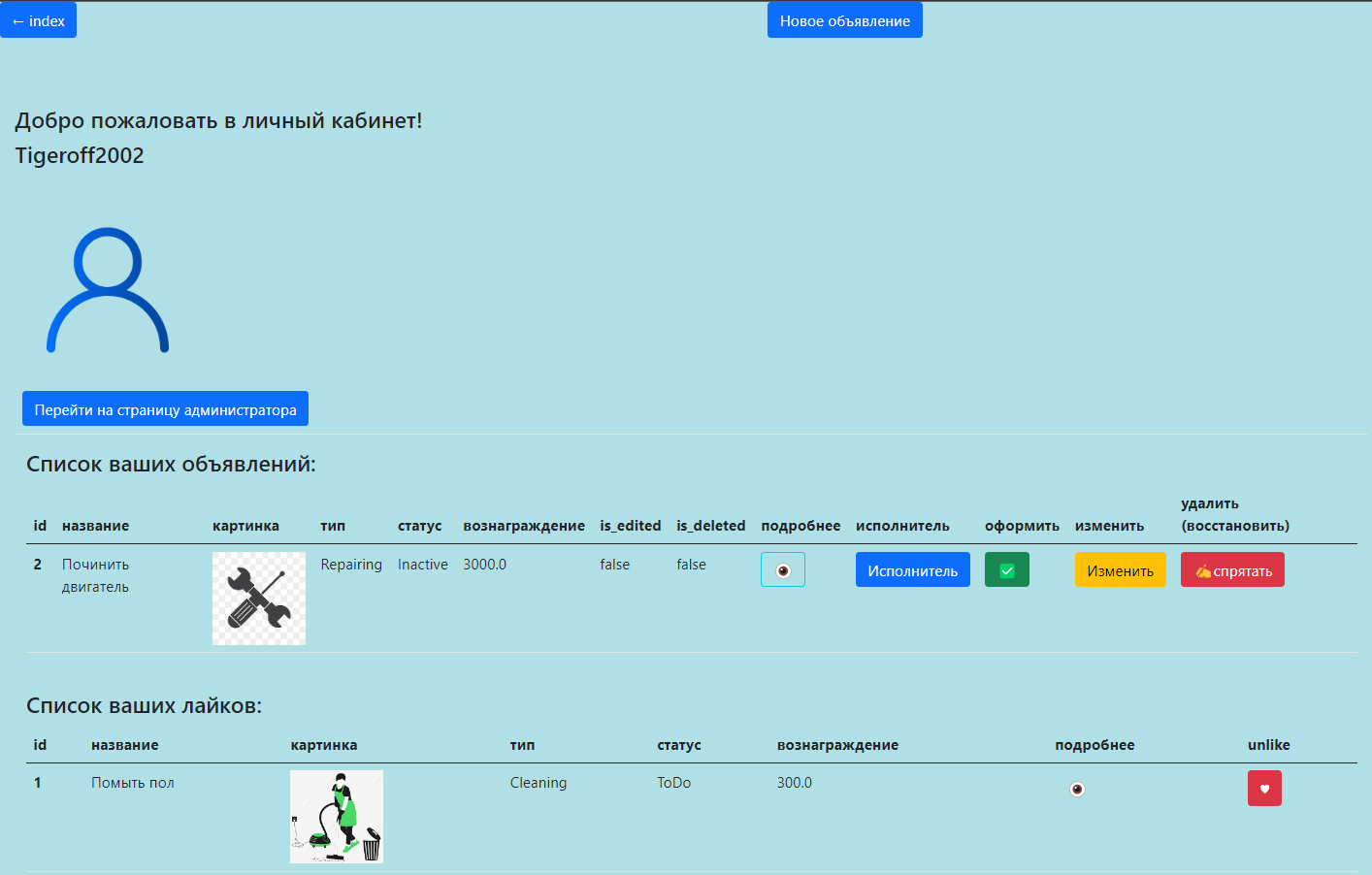


Рисунок 9.8. Личный кабинет пользователя

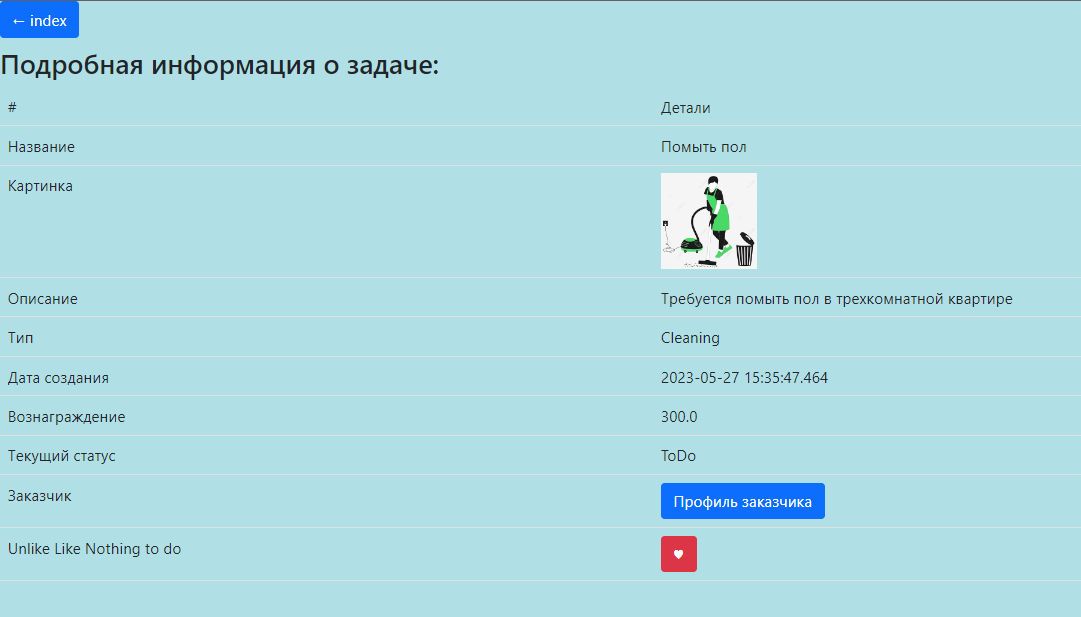


Рисунок 9.9. Подробный просмотр объявления

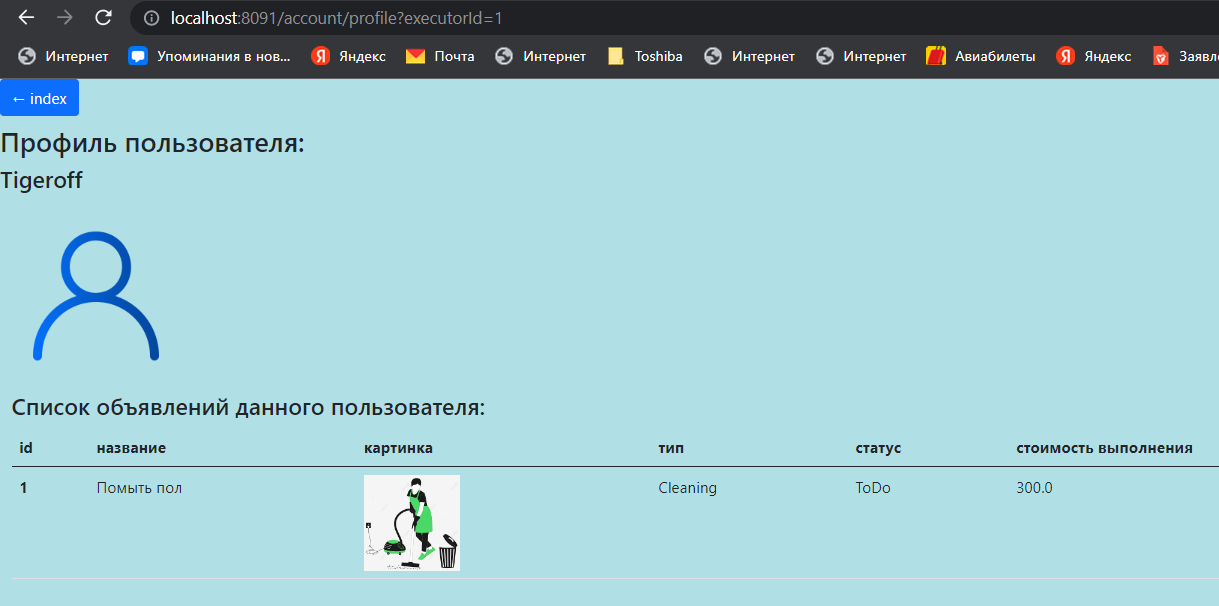


Рисунок 9.10. Просмотр профиля заказчика

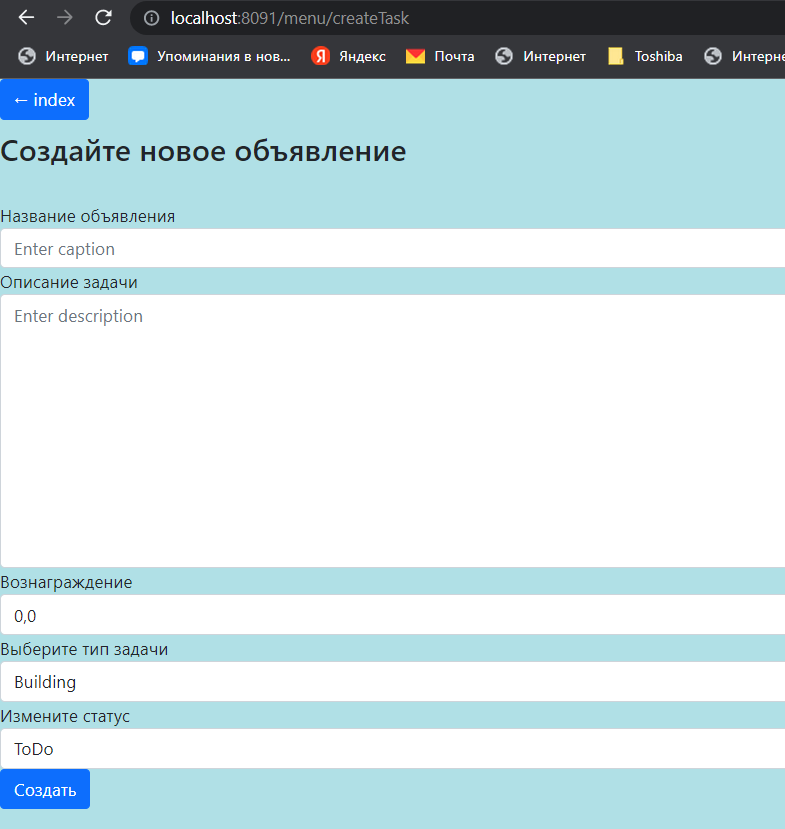


Рисунок 9.11. Панель создания нового объявления

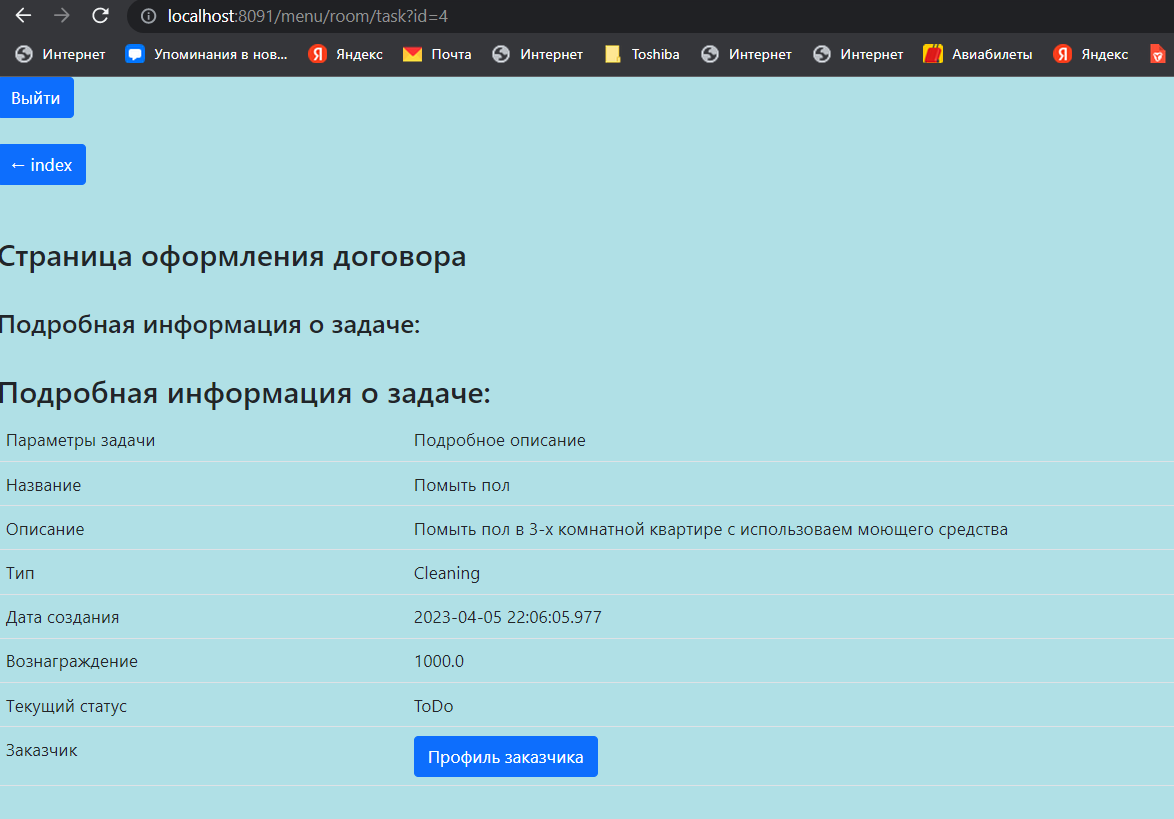


Рисунок 9.12. Страница оформления договора по объявлению

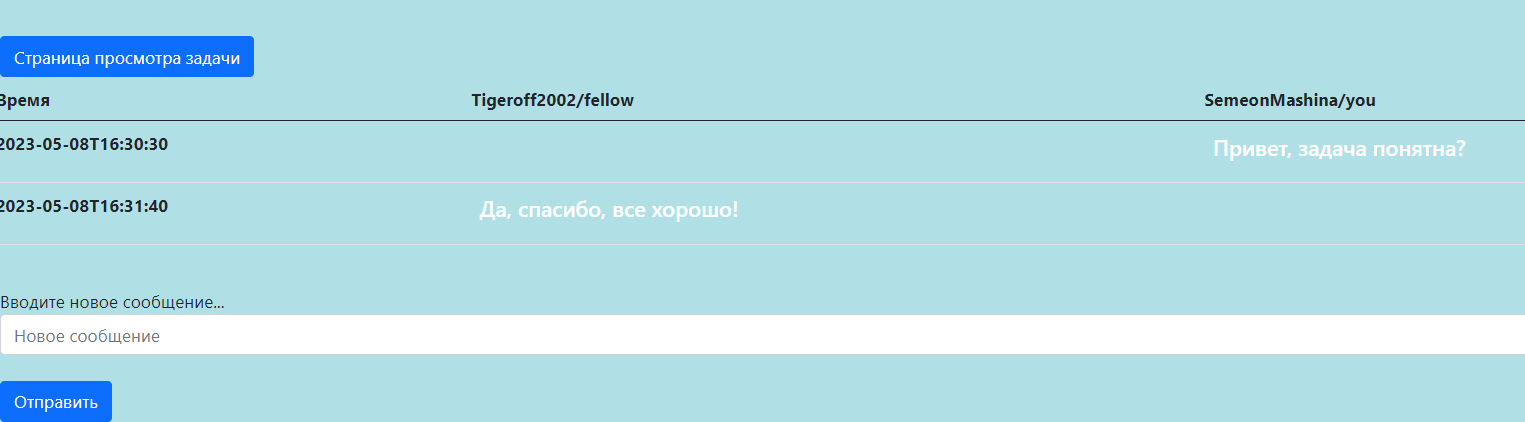


Рисунок 9.13. Панель чата на сайте

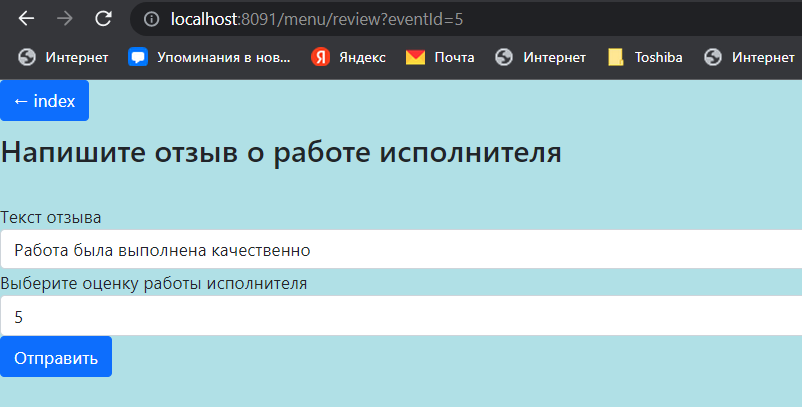


Рисунок 9.14. Форма написания отзыва о исполнителе

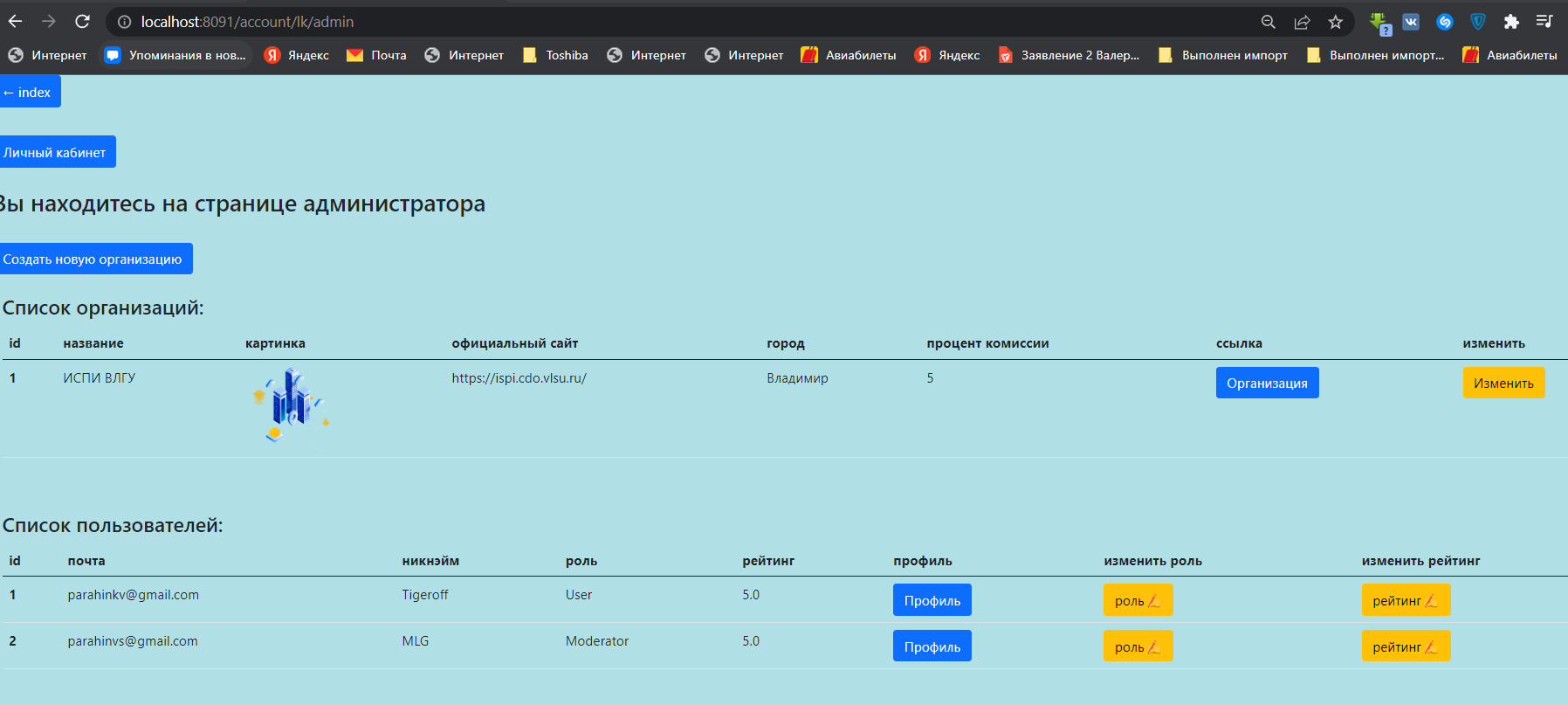


Рисунок 9.15. Панель администратора сайта

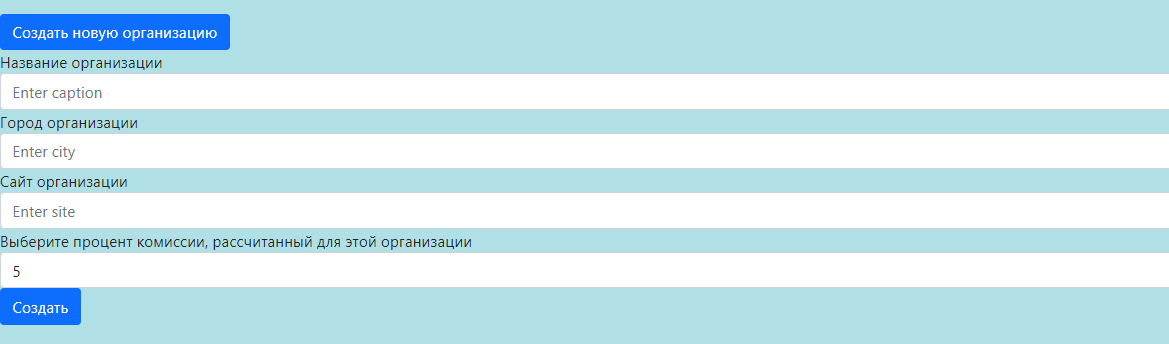


Рисунок 9.16. Панель создания новой организации

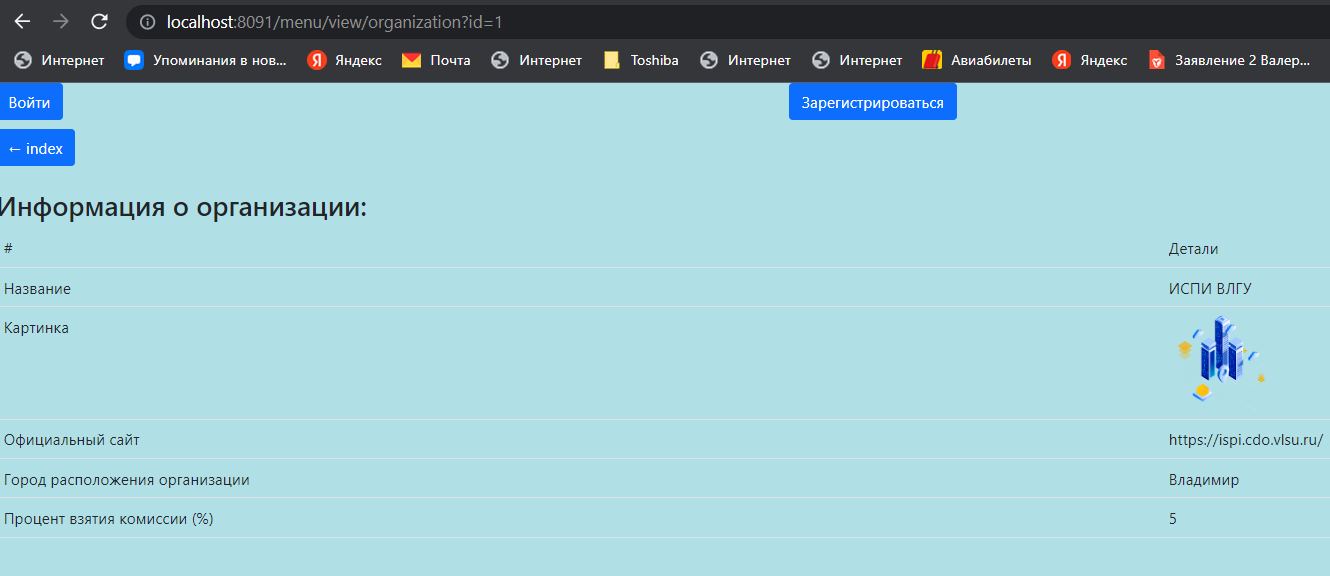


Рисунок 9.17. Просмотр информации об организации

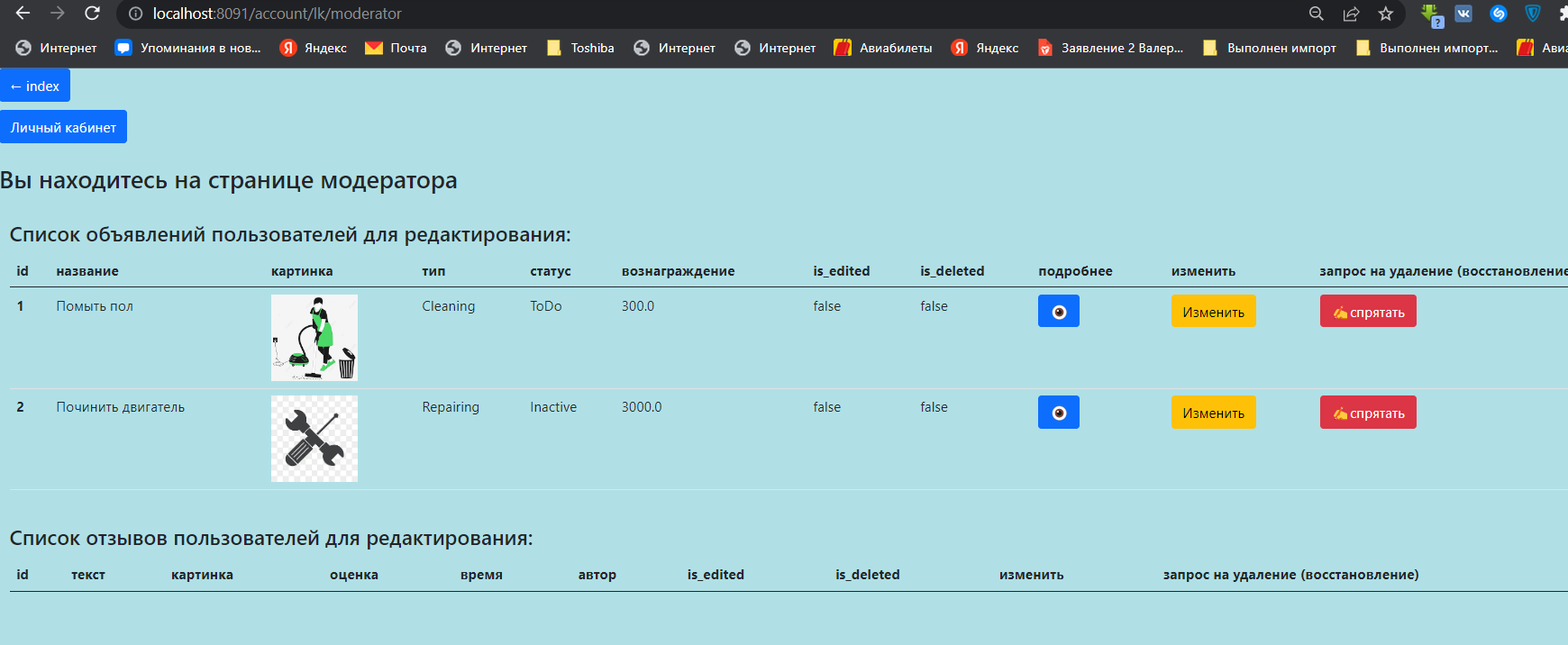


Рисунок 9.18. Панель модератора сайта

ТЕСТИРОВАНИЕ СИСТЕМЫ

Модульное тестирование

Для написания unit-тестов Spring Boot веб-приложения я использовал библиотеки junit.jupiter.api и mockito.

Мною было написано 2 ознакомительных интеграционных теста (проверяющих корректность запуска веб-приложения и работы БД), а также тесты на модели и сервисы.

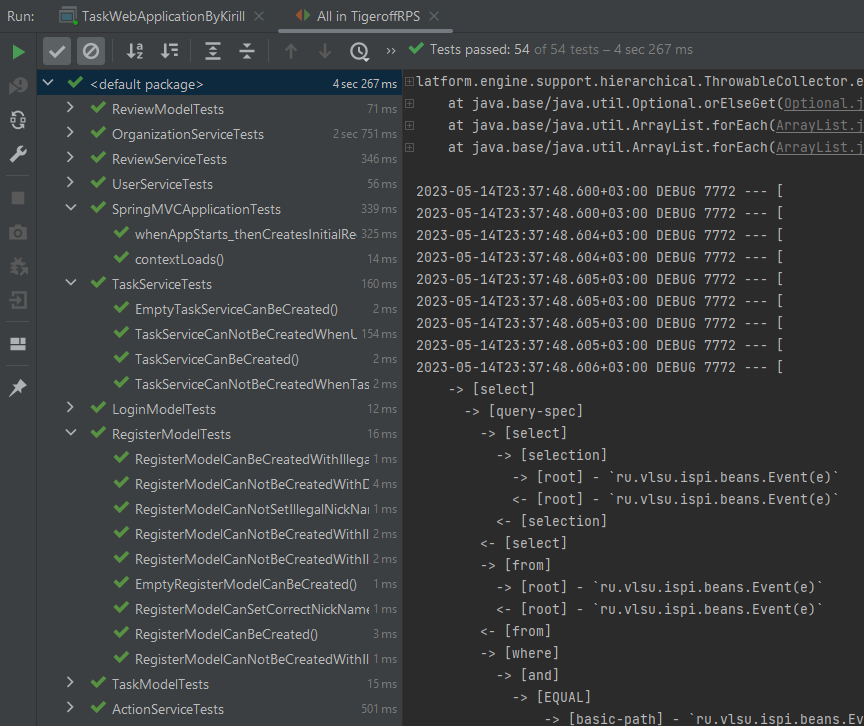


Рисунок 10. Unit-тестирование веб-приложения Spring

Нагрузочное тестирование

Для тестирование производительности Spring Boot веб-приложения я использовал программное обеспечение под названием Jmeter 5.

В ходе тестирования воплотим в жизнь такой сценарий:

Большое количество пользователей (например, n = 100 человек) одновременно входят (авторизуются) в свой аккаунт, затем заходят в личный кабинет, оттуда создают новое одинаковое объявление, а затем его сразу удаляют (в рамках одной транзакции).

В рамках этого сценария мы получим выполнение несколькоих типов запросов:

1. Авторизация – POST-запрос с выборкой (просмотром личной информации пользователя) из БД
2. Вход в личный кабинет – GET-запрос с выборкой данных (объявлений пользователя) из БД
3. Создание нового объявления – POST-запрос с добавлением новой записи в БД

Последние 2 действия придется сделать в рамках одной транзакции, чтобы избежать возможных проблем из-за доступа множества клиентов к общему ресурсу (таблице БД) во многопоточном окружении Spring веб-приложения.

В ходе проведения нагрузочного тестирования попробуем повышать число одновременных запросов к приложению, поступающих от разных пользователей (в интервале – от 100 до 10.000 пользователей). Если не произойдет закрытия приложения и получения результата запроса наподобие «Error 500», то важнейшее функциональное требование к выдерживанию нагрузки будет выполнено.

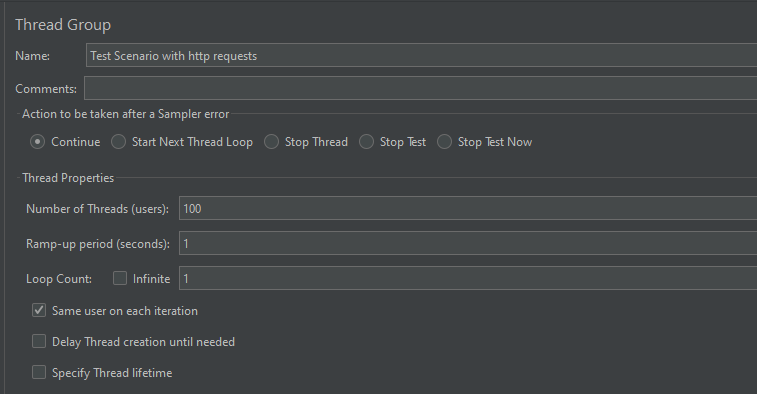


Рисунок 11.1. Создание общего тестового сценария для 100 пользователей

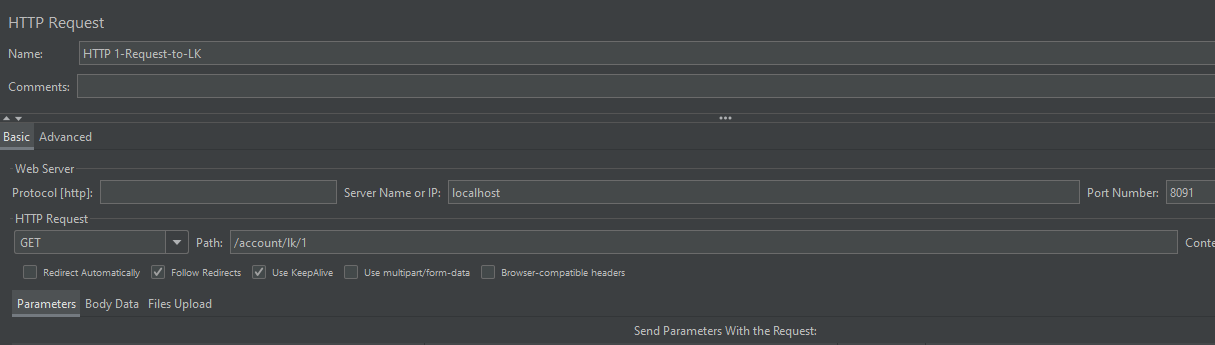


Рисунок 12.1. Первый GET-запрос (вход в личный кабинет)

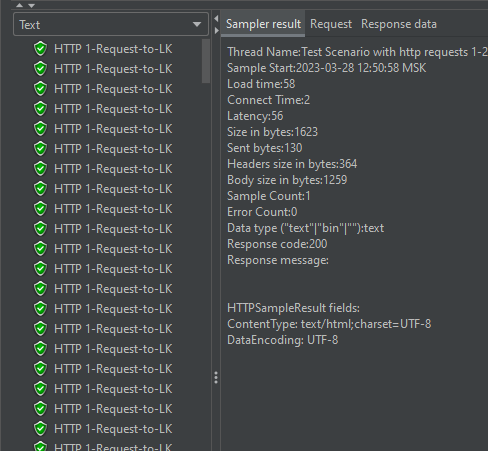


Рисунок 12.2. Результат GET-запроса для 100 пользователей

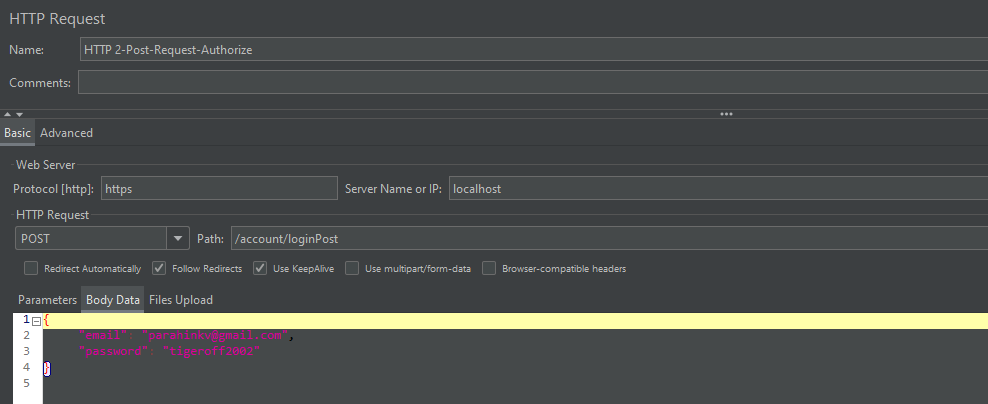


Рисунок 13.1. POST-запрос на авторизацию пользователя

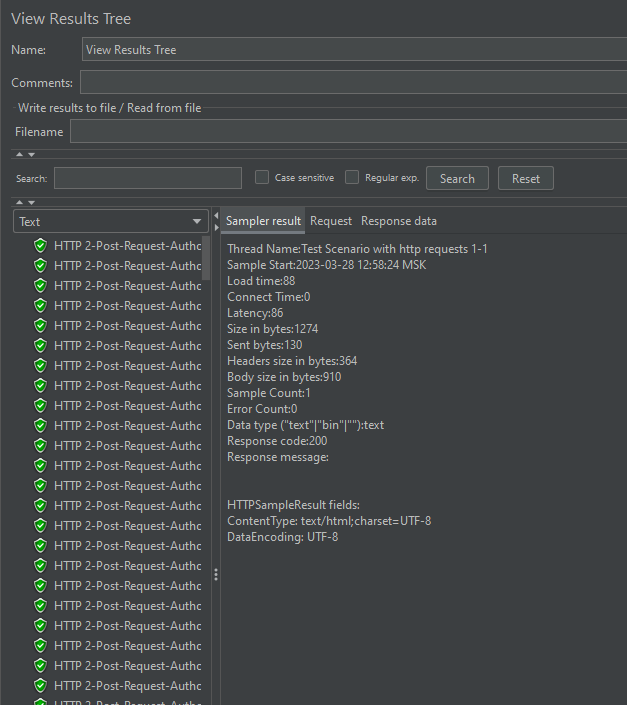


Рисунок 13.2. Результат POST-запроса для 100 пользователей

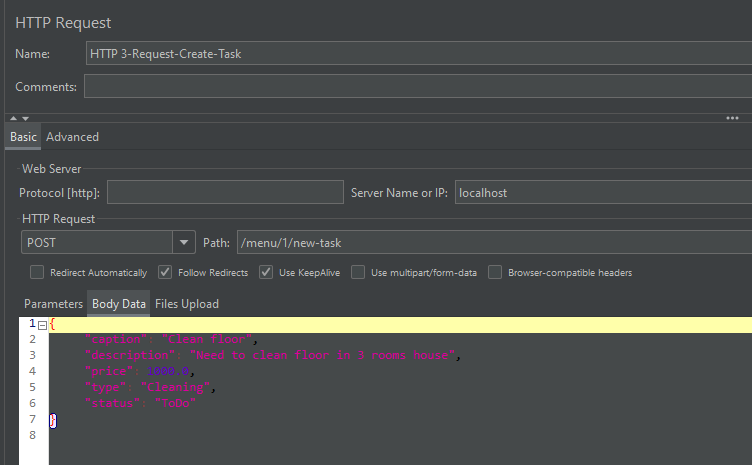


Рисунок 14.1. POST-запрос на создание нового объявления на сайте

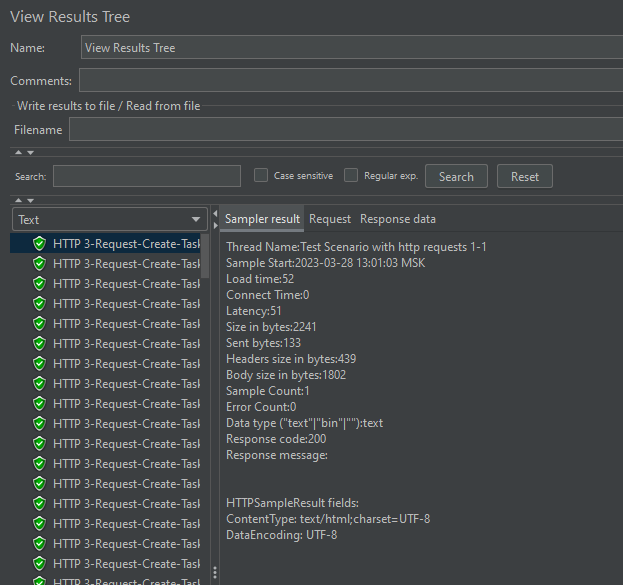


Рисунок 14.2. Результат POST-запроса для 100 пользователей

РЕПОЗИТОРИЙ ПРОЕКТА

Вся разработка велась с использованием системы контроли версий git и сервиса для хранения удаленных репозиториев Github, на котором размещен код моей программной системы по ссылке: <https://github.com/javavlsu/course-work-team-100-parakhin> (рис.15 – представлен скришот курсового проекта, расположенный на сайте GitHub).

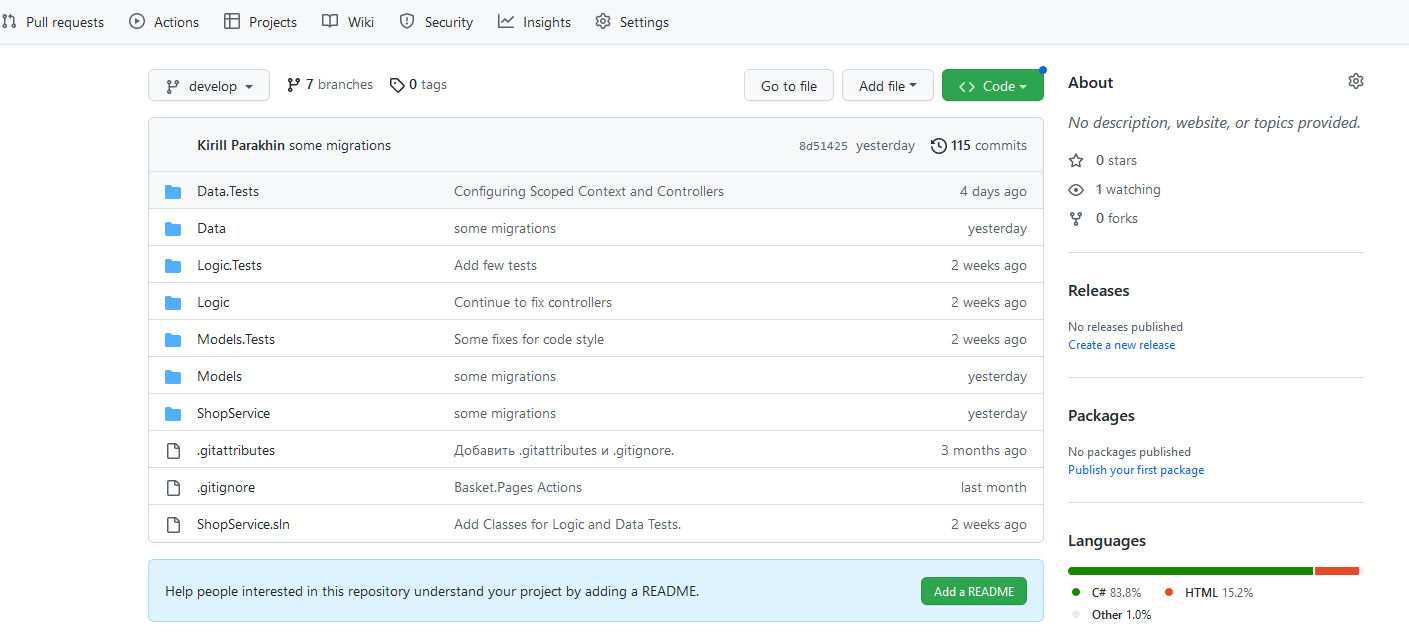


Рисунок 15. Репозиторий проекта на GitHub

# Заключение

В ходе выполнения курсового проекта была разработана программная система для автоматизации работы агенства, занимающегося предоставлением и выполненим бытовых услуг пользователям (исполнителям и заказчикам) в сети Интернет. По итогу был получен рабочий протип данной программно-информационной системы, который соответствует заявленным функциональным и нефункциональным требованиям), а также обладает приятным электронным сайтом и рабочим понятным функционалом.

# Список использованных источников

1. Будилов, В. А. Интернет-программирование на Java / В. А. Будилов – СПб.: БХВ-Петербург, 2003.

2. Дейтел, Х. М. Технологии программирования на Java 2 / Х. М. Дей-тел, П. Дж. Дейтел, С. И. Сантри – М.: ООО "Бином-Пресс", 2003.

3. Вершинин, М. М. Java 2 Enterprise Edition. Технологии проектиро-вания и разработки / М. М. Вершинин, Е. Б. Иванова– СПб.: БХВ-Петербург, 2003

1. METANIT.COM – Сайт о программировании. [Электронный ресурс]: URL: <https://metanit.com/sharp/> (Дата обращения: 27.05.2023).
2. Habr - Сайт о программировании. [Электронный ресурс] URL: <https://habr.com/ru/hub/csharp/page3/> (Дата обращения: 28.05.2023).

ПРИЛОЖЕНИЕ А. Диаграмма прецедентов

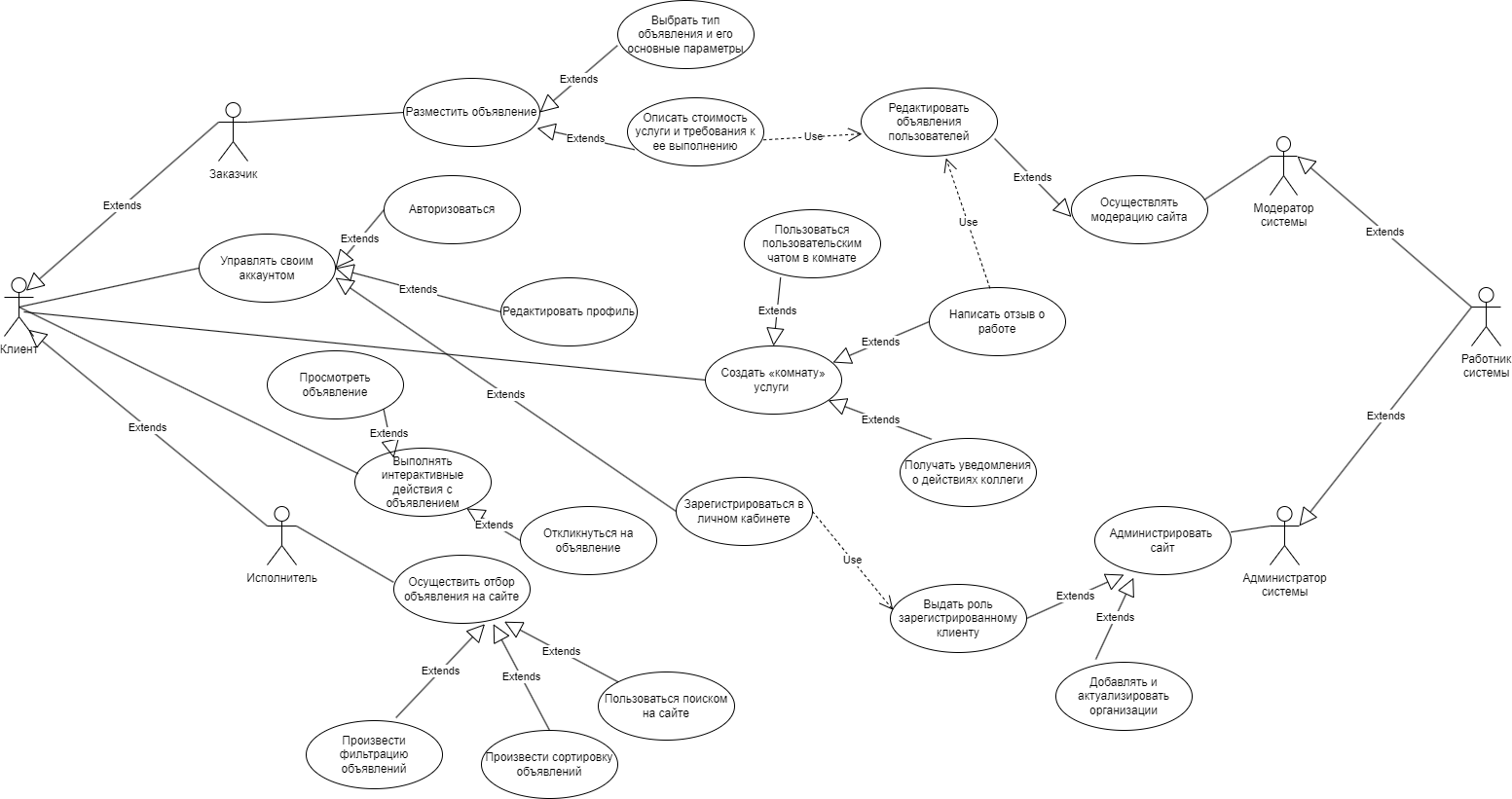


Рисунок 1. Диаграмма прецедентов

ПРИЛОЖЕНИЕ Б. Диаграмма классов

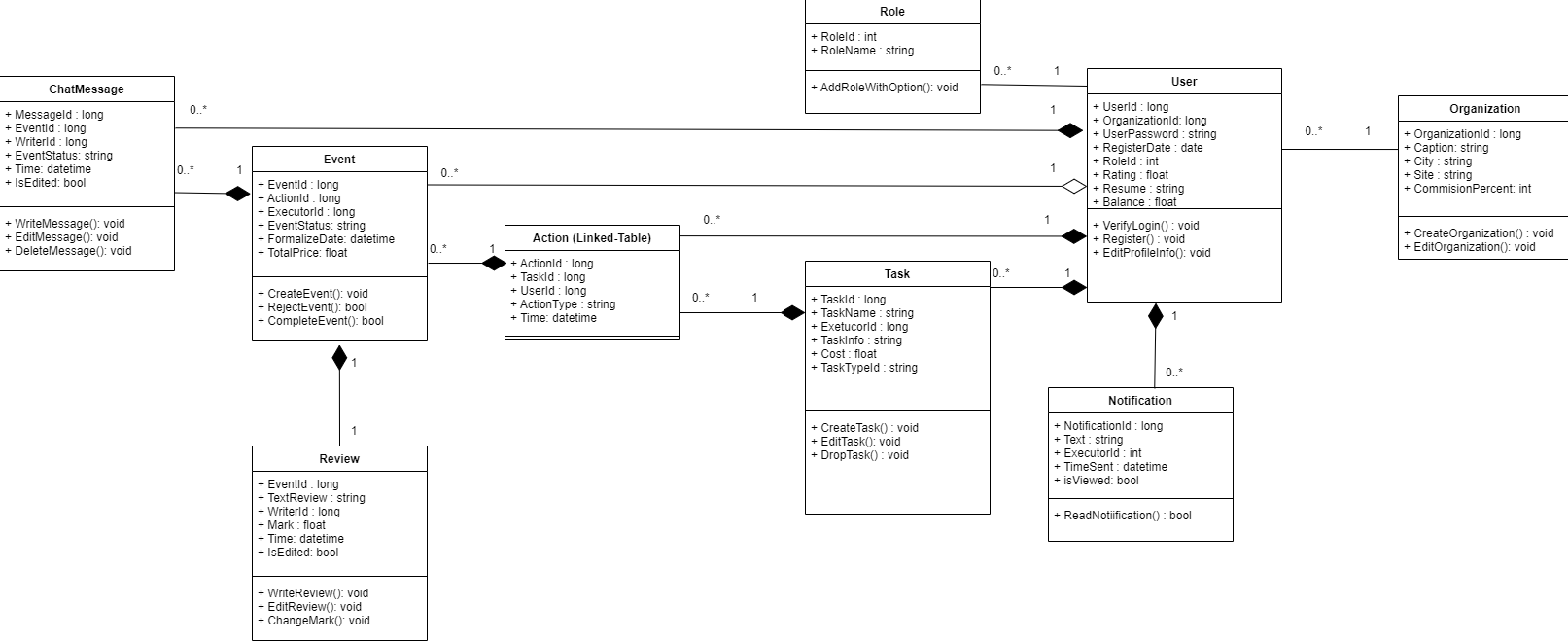


Рисунок 2. Диаграмма классов

ПРИЛОЖЕНИЕ В. Код программной системы

@Data  
@Entity  
@Table(name="Users")  
@AllArgsConstructor  
public class User implements UserDetails {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private String email;  
 private String nickname;  
 private String password;  
  
 @Enumerated(value = EnumType.*STRING*)  
 private RoleType role;  
  
 @Enumerated(value = EnumType.*STRING*)  
 private Gender gender;  
  
 private String contactnumber;  
  
 @ManyToOne  
 private Organization organization;  
   
 private float rating;

public User() {  
  
 }  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 return null;  
 }  
  
 @Override  
 public String getUsername() {  
 return null;  
 }  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return false;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return false;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return false;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return false;  
 }  
}

@Service  
public class UserService implements UserDetailsService {  
 @Autowired  
 private UserRepository userRepository;  
  
 private PasswordEncoder passwordEncoder;  
  
 public UserService(){  
  
 }  
  
 public UserService(UserRepository userRepository, PasswordEncoder passwordEncoder){  
  
 if (userRepository == null){  
 throw new IllegalArgumentException("Null user repository was provided");  
 }  
 this.userRepository = userRepository;  
  
 if (passwordEncoder == null){  
 throw new IllegalArgumentException("Null passwordEncoder was provided");  
 }  
 this.passwordEncoder = passwordEncoder;  
 }  
  
 public User RegisterUser(RegisterModel model) throws SQLException {  
 if (model == null){  
 throw new IllegalArgumentException("Null register model was provided");  
 }  
  
 User user = userRepository.findUserByEmail(model.getEmail());  
  
 if (user == null) {  
  
 if (!Objects.*equals*(model.getPassword(), model.getConfirmpassword())){  
 User testUser = new User();  
 testUser.setId(-1L);  
 return testUser;  
 }  
  
 User newUser = new User();  
 newUser.setNickname(model.getNickname());  
 newUser.setPassword(model.getPassword());  
 newUser.setContactnumber(model.getContactnumber());  
 newUser.setEmail(model.getEmail());  
 newUser.setRole(model.getRole());  
 newUser.setGender(model.getGender());  
  
 userRepository.save(newUser);  
  
 int newId = userRepository.calculateMaxUserId(newUser.getEmail());  
  
 newUser.setId(Integer.*toUnsignedLong*(newId));  
  
 return newUser;  
 }  
  
 return null;  
 }  
  
 public User LoginUser(LoginModel model) throws SQLException{  
 if (model == null){  
 throw new IllegalArgumentException("Null login model was provided");  
 }  
  
 User user = userRepository.findUserByEmail(model.getEmail());  
  
 if (user != null){  
 if (Objects.*equals*(user.getPassword(), model.getPassword())){  
 return user;  
 }  
 else {  
 return null;  
 }  
 }  
 return user;  
 }  
  
 public User FindUserById(Long id) throws SQLException{  
 if (id == null){  
 throw new IllegalArgumentException("Null id was provided");  
 }  
 return userRepository.findUserById(id);  
 }  
  
 public List<User> getAllUsers(){  
 return userRepository.findAll();  
 }  
  
 public ExtraUser nameRoleUser(Long userId){  
 User user = userRepository.findUserById(userId);  
  
 if (user != null){  
 ExtraUser extraUser = new ExtraUser();  
 extraUser.setId(user.getId());  
 extraUser.setRole(user.getRole());  
 extraUser.setGender(user.getGender());  
 extraUser.setEmail(user.getEmail());  
 extraUser.setNickname(user.getNickname());  
 extraUser.setPassword(user.getPassword());  
 extraUser.setContactnumber(user.getContactnumber());  
 extraUser.isModerator = false;  
 extraUser.isAdmin = false;  
 extraUser.isUser = false;  
 if (extraUser.getRole() == RoleType.*Admin*){  
 extraUser.isAdmin = true;  
 }  
 else if (extraUser.getRole() == RoleType.*Moderator*){  
 extraUser.isModerator = true;  
 }  
 else {  
 extraUser.isUser = true;  
 }  
  
 return extraUser;  
 }  
  
 return null;  
 }  
  
 public List<ExtraTask> filterByRowParameters(List<ExtraTask> currentTaskList, String rowToFind, WholeFilterSet filterSet, String sorter){  
  
 if (stringIsNullOrEmptyOrBlank(rowToFind) || filterSet == null || stringIsNullOrEmptyOrBlank(sorter)){  
 throw new IllegalArgumentException("String headers should not be null, empty or blank!");  
 }  
  
 rowToFind = rowToFind.toLowerCase();  
 sorter = sorter.toLowerCase();  
  
 var sortBy = SortBy.*valueOf*(sorter);  
 var parameters = new ArrayList<FilterParameter>();  
  
 if (filterSet.isAuthUser()){  
  
 var filterAuthCase1 = filterSet.getLiked();  
  
 if (!Objects.*equals*(filterAuthCase1, "default")){  
 var filterAuthParameter1 = new FilterParameter();  
 filterAuthParameter1.Filter = FilterBy.*liked*;  
 filterAuthParameter1.parameter = filterAuthCase1;  
 parameters.add(filterAuthParameter1);  
 }  
  
 var filterAuthCase2 = filterSet.getViewed();  
  
 if (!Objects.*equals*(filterAuthCase2, "default")){  
 var filterAuthParameter2 = new FilterParameter();  
 filterAuthParameter2.Filter = FilterBy.*viewed*;  
 filterAuthParameter2.parameter = filterAuthCase2;  
 parameters.add(filterAuthParameter2);  
 }  
 }  
  
 var filterCase1 = filterSet.getType();  
  
 if (!Objects.*equals*(filterCase1, "default")){  
 var filterParameter1 = new FilterParameter();  
 filterParameter1.Filter = FilterBy.*type*;  
 filterParameter1.parameter = filterSet.getType();  
 parameters.add(filterParameter1);  
 }  
  
 var filterCase2 = filterSet.getStatus();  
  
 if (!Objects.*equals*(filterCase2, "default")){  
 var filterParameter2 = new FilterParameter();  
 filterParameter2.Filter = FilterBy.*status*;  
 filterParameter2.parameter = filterSet.getStatus();  
 parameters.add(filterParameter2);  
 }  
  
 return filterByAllParameters(currentTaskList, rowToFind, parameters, sortBy);  
 }  
  
 private boolean stringIsNullOrEmptyOrBlank(String row){  
 return row == null || row.isEmpty() || row.trim().isEmpty();  
 }  
  
 private List<ExtraTask> filterByAllParameters(List<ExtraTask> currentTaskList, String rowToFind, List<FilterParameter> parameters, SortBy sorter){  
  
 var taskListAfterSearch = findByString(currentTaskList, rowToFind);  
  
 if (taskListAfterSearch.size() == 0){  
 return taskListAfterSearch;  
 }  
  
 var taskListAfterFiltering = filterBySetParameters(taskListAfterSearch, parameters);  
  
 if (taskListAfterFiltering.size() == 0){  
 return taskListAfterFiltering;  
 }  
  
 return sortByParameter(taskListAfterFiltering, sorter);  
 }  
  
 private List<ExtraTask> findByString(List<ExtraTask> currentTaskList, String row){  
 if (!row.equals("empty")){  
  
 var rowToFind = row.toLowerCase();  
  
 if (rowToFind.contains("+")){  
  
 List<String> subWords = new ArrayList<String>(Arrays.*asList*(row.split("\\+")));  
  
 var obtainedTaskList = new ArrayList<ExtraTask>();  
  
 for(var task:currentTaskList){  
 var caption = task.getCaption().toLowerCase();  
 var description = task.getDescription().toLowerCase();  
  
 if (containsAtLeastElementOfArray(caption, subWords)  
 || containsAtLeastElementOfArray(description, subWords)){  
 obtainedTaskList.add(task);  
 }  
 }  
  
 return obtainedTaskList;  
 }  
  
 else {  
 var newTaskList = new ArrayList<ExtraTask>();  
  
 for(var task:currentTaskList){  
 var caption = task.getCaption().toLowerCase();  
 var description = task.getDescription().toLowerCase();  
  
 if (caption.contains(rowToFind) || description.contains(rowToFind)){  
 newTaskList.add(task);  
 }  
 }  
  
 return newTaskList;  
 }  
 }  
  
 return currentTaskList;  
 }  
  
 private boolean containsAtLeastElementOfArray(String string, List<String> subWords){  
 for(var item: subWords){  
 if (string.contains(item)){  
 return true;  
 }  
 }  
 return false;  
 }  
  
 private List<ExtraTask> sortByParameter(List<ExtraTask> currentTaskList, SortBy direction){  
  
 if (direction == SortBy.*cheap\_first*){  
 currentTaskList.sort((o1, o2) -> Math.*round*(o1.getPrice() - o2.getPrice()));  
 }  
 else if (direction == SortBy.*dear\_first*){  
 currentTaskList.sort((o1, o2) -> Math.*round*(o2.getPrice() - o1.getPrice()));  
 }  
  
 return currentTaskList;  
 }  
  
 private List<ExtraTask> filterBySetParameters(List<ExtraTask> currentTaskList, List<FilterParameter> parameters){  
 for (var parameter:parameters) {  
 var filter = parameter.Filter;  
 var param = parameter.parameter;  
 currentTaskList = filterByParameter(currentTaskList, filter, param);  
 }  
 return currentTaskList;  
 }  
  
 private List<ExtraTask> filterByParameter(List<ExtraTask> currentTaskList, FilterBy filter, String parameter){  
 if (filter == FilterBy.*type*){  
 var param = TaskType.*valueOf*(parameter);  
 currentTaskList = currentTaskList.stream().filter(x -> x.getType() == param).collect(Collectors.*toList*());  
 }  
 else if (filter == FilterBy.*status*){  
 var param = TaskStatus.*valueOf*(parameter);  
 currentTaskList = currentTaskList.stream().filter(x -> x.getStatus() == param).collect(Collectors.*toList*());  
 }  
 else if (filter == FilterBy.*mine*){  
 currentTaskList = currentTaskList.stream().filter(x -> x.IsMine).collect(Collectors.*toList*());  
 }  
 else if (filter == FilterBy.*liked*){  
 currentTaskList = currentTaskList.stream().filter(x -> Objects.*equals*(x.Liked, parameter)).collect(Collectors.*toList*());  
 }  
 else if (filter == FilterBy.*viewed*){  
 if (Objects.*equals*(parameter, "viewed")){  
 currentTaskList = currentTaskList.stream().filter(x -> x.IsViewed).collect(Collectors.*toList*());  
 }  
 else {  
 currentTaskList = currentTaskList.stream().filter(x -> !x.IsViewed).collect(Collectors.*toList*());  
 }  
 }  
 return currentTaskList;  
 }  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 return userRepository.findUserByName(username);  
 }  
}

@Controller  
public class HomeController implements ErrorController {  
 @Autowired  
 private TaskService taskHandler;  
  
 @Autowired  
 private UserService userHandler;  
  
 @Autowired  
 private ActionService actionHandler;  
  
 @GetMapping("/")  
 public String DefaultIndex(HttpSession session){  
  
 if (session.getAttribute("rowToFind") == null){  
 session.setAttribute("rowToFind", "empty");  
 }  
 if (session.getAttribute("type") == null){  
 session.setAttribute("type", "default");  
 }  
 if (session.getAttribute("status") == null){  
 session.setAttribute("status", "default");  
 }  
 if (session.getAttribute("sorter") == null){  
 session.setAttribute("sorter", "default\_sort");  
 }  
 if (session.getAttribute("elementsPerPage") == null){  
 session.setAttribute("elementsPerPage", 5);  
 }  
  
 if (session.getAttribute("currentPageNumber") == null){  
 session.setAttribute("currentPageNumber", 1);  
 }  
  
 var rowToFind = (String) session.getAttribute("rowToFind");  
 var type = (String) session.getAttribute("type");  
 var status = (String) session.getAttribute("status");  
 var sorter = (String) session.getAttribute("sorter");  
  
 var elementsPerPage = (int) session.getAttribute("elementsPerPage");  
  
 var currentPageNumber = 1;  
 session.setAttribute("currentPageNumber", currentPageNumber);  
  
 return "redirect:/index?rowToFind=" + rowToFind + "&type=" + type + "&status=" + status + "&sorter="  
 + sorter + "&elementsPerPage=" + Integer.*toString*(elementsPerPage) + "&currentPageNumber=" + Integer.*toString*(currentPageNumber);  
 }  
  
 @GetMapping("index1")  
 public String RedirectAmongThePages(@RequestParam("currentPageNumber") int currentPageNumber,  
 @RequestParam("elementsPerPage") int elementsPerPage,  
 HttpSession session){  
 if (currentPageNumber <= 0 || elementsPerPage <= 0){  
 return "redirect:/";  
 }  
 if (session.getAttribute("rowToFind") == null){  
 session.setAttribute("rowToFind", "empty");  
 }  
 if (session.getAttribute("type") == null){  
 session.setAttribute("type", "default");  
 }  
 if (session.getAttribute("status") == null){  
 session.setAttribute("status", "default");  
 }  
  
 if (session.getAttribute("sorter") == null){  
 session.setAttribute("sorter", "default\_sort");  
 }  
  
 session.setAttribute("elementsPerPage", elementsPerPage);  
  
 session.setAttribute("currentPageNumber", currentPageNumber);  
  
 var rowToFind = (String) session.getAttribute("rowToFind");  
 var type = (String) session.getAttribute("type");  
 var status = (String) session.getAttribute("status");  
 var sorter = (String) session.getAttribute("sorter");  
  
 return "redirect:/index?rowToFind=" + rowToFind + "&type=" + type + "&status=" + status + "&sorter="  
 + sorter + "&elementsPerPage=" + Integer.*toString*(elementsPerPage) + "&currentPageNumber=" + Integer.*toString*(currentPageNumber);  
 }  
  
 @GetMapping("index2")  
 public String RedirectAmongFilters(@RequestParam(name = "rowToFind") String rowToFind,  
 @RequestParam(name = "type") String type,  
 @RequestParam(name = "status") String status,  
 @RequestParam(name = "sorter") String sorter,  
 HttpSession session){  
  
 if (stringIsNullOrEmptyOrBlank(rowToFind) || stringIsNullOrEmptyOrBlank(type)  
 || stringIsNullOrEmptyOrBlank(sorter) || stringIsNullOrEmptyOrBlank(sorter)){  
 return "redirect:/";  
 }  
  
 session.setAttribute("rowToFind", rowToFind);  
  
 session.setAttribute("type", type);  
  
 session.setAttribute("status", status);  
  
 session.setAttribute("sorter", sorter);  
  
 var elementsPerPage = (int) session.getAttribute("elementsPerPage");  
  
 var currentPageNumber = 1;  
 session.setAttribute("currentPageNumber", currentPageNumber);  
  
 return "redirect:/index?rowToFind=" + rowToFind + "&type=" + type + "&status=" + status + "&sorter="  
 + sorter + "&elementsPerPage=" + Integer.*toString*(elementsPerPage) + "&currentPageNumber=" + Integer.*toString*(currentPageNumber);  
  
 }  
  
 @GetMapping("/index")  
 public String Index(@RequestParam(name = "rowToFind") String rowToFind,  
 @RequestParam(name = "type") String type,  
 @RequestParam(name = "status") String status,  
 @RequestParam(name = "sorter") String sorter,  
 @RequestParam(name = "elementsPerPage") int elementsPerPage,  
 @RequestParam(name = "currentPageNumber") int currentPageNumber,  
 Model model, HttpSession session) throws SQLException {  
  
 var userId = (Long) session.getAttribute("userId");  
  
 if (userId != null){  
 var user = userHandler.FindUserById(userId);  
  
 if (user != null){  
 return "redirect:/account/default\_index";  
 }  
 }  
  
 var taskList = actionHandler.nameAllLikedAndUnlikedTasks();  
  
 var filterSet = new WholeFilterSet(rowToFind, type, status, sorter);  
  
 var obtainedTaskList = userHandler.filterByRowParameters(taskList, rowToFind, filterSet, sorter);  
  
 if (obtainedTaskList.size() == 0 && taskList.size() != 0){  
  
 session.setAttribute("rowToFind", "empty");  
 session.setAttribute("type", "default");  
 session.setAttribute("status", "default");  
 session.setAttribute("sorter", "default\_sort");  
  
 return "redirect:/";  
 }  
  
 var pagination = new Pagination();  
  
 pagination.setElementsPerPage(elementsPerPage);  
  
 pagination.setCurrentPageNumber(currentPageNumber);  
  
 var listOfListsTasks = taskHandler.distributeTasksByPages(obtainedTaskList, pagination);  
  
 pagination.setLastPageNumber(listOfListsTasks.size() - 1);  
  
 session.setAttribute("lastPageNumber", pagination.getLastPageNumber());  
  
 var listTasksForCurrentPage = taskHandler.getElementsForCurrentPage(listOfListsTasks, currentPageNumber);  
  
 if (listTasksForCurrentPage == null && taskList.size() != 0) {  
  
 return "redirect:/";  
 }  
  
 model.addAttribute("taskList", listTasksForCurrentPage);  
  
 model.addAttribute("filterSet", new WholeFilterSet(rowToFind, type, status, sorter));  
  
 model.addAttribute("pagination", pagination);  
  
 return "mycatalog";  
 }

private boolean stringIsNullOrEmptyOrBlank(String row){  
 return row == null || row.isEmpty() || row.trim().isEmpty();  
 }  
  
 @RequestMapping("/error")  
 public String getErrorPath(HttpSession session) throws SQLException{  
  
 var userId = (Long) session.getAttribute("userId");  
  
 if (userId != null){  
 var user = userHandler.FindUserById(userId);  
 if (user != null){  
 return "redirect:/account/default\_index";  
 }  
 }  
  
 return "redirect:/";  
 }  
}

@Repository  
public interface UserRepository extends JpaRepository<User, Long> {  
  
 @Query("SELECT u FROM User u WHERE u.email = :email")  
 User findUserByEmail(@Param("email") String email);  
  
 @Query("SELECT u FROM User u WHERE u.nickname = :name")  
 User findUserByName(@Param("name") String name);  
  
 @Query("SELECT u FROM User u WHERE u.id = :id")  
 User findUserById(@Param("id") Long id);  
  
 @Query("SELECT id FROM User u WHERE u.email = :email")  
 int calculateMaxUserId(@Param("email") String email);  
  
 @Query("SELECT MAX(id) FROM User u")  
 Long calculateLastCreatedUserId();  
}

@Configuration  
@EnableWebSecurity  
@ConditionalOnProperty(name = "spring.security.enabled", havingValue = "false")  
public class WebSecurityConfig {  
 @Bean  
 public UserDetailsService userDetailsService(UserRepository userRepository) {  
 return username -> {  
 User user = userRepository.findUserByName(username);  
 if (user != null) return user;  
 throw new UsernameNotFoundException("User " + username + " not found!");  
 };  
 }  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {  
 return httpSecurity  
 .authorizeHttpRequests()  
 .requestMatchers("/account/lk/admin/").hasRole("Admin")  
 .requestMatchers("/account/lk/moderator/").hasRole("Moderator")  
 .requestMatchers("/", "/\*\*", "/js/\*\*", "/css/\*\*").permitAll()  
 .requestMatchers("/account/index/", "/account/lk/").authenticated()  
 .and()  
 .formLogin()  
 .loginPage("/account/login")  
 .defaultSuccessUrl("/account/index")  
 .and()  
 .build();  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder(12);  
 }  
  
 @Bean(name = "mvcHandlerMappingIntrospector")  
 public HandlerMappingIntrospector mvcHandlerMappingIntrospector() {  
 return new HandlerMappingIntrospector();  
 }