Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина «Распределённые системы обработки информации»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  ассистент кафедры ЭИ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.П. Лыщик |
|  | \_\_\_.\_\_\_\_.2023 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«Разработка системы автоматизации работы санатория»**

|  |  |
| --- | --- |
|  | Выполнил студент группы 074002  Лянной Вадим Александрович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_.\_\_\_\_.2023  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2023

Содержание

[ВВЕДЕНИЕ 5](#_Toc134467264)

[1. Анализ и моделирование предметной области программного средства 6](#_Toc134467265)

[1.1 Описание предметной области 6](#_Toc134467266)

[1.2 Разработка функциональной модели предметной области 8](#_Toc134467267)

[1.3 Анализ требований к разрабатываемому программному средству. 13](#_Toc134467268)

[1.4 Разработка информационной модели предметной области 14](#_Toc134467269)

[1.5 UML-модели представления программного средства и их описание 18](#_Toc134467270)

[2. Проектирование и конструирование программного средства 27](#_Toc134467271)

[2.1 Постановка задачи 27](#_Toc134467272)

[2.2 Обоснование выбора компонентов и технологий для реализации программного средства 28](#_Toc134467273)

[2.3 Архитектурные решения 28](#_Toc134467274)

[2.4 Описание алгоритмов, реализующих ключевую бизнес-логику разрабатываемого программного средства 30](#_Toc134467275)

[2.5 Проектирование пользовательского интерфейса 31](#_Toc134467276)

[2.6 Методы и средства, используемые для обеспечения безопасности данных 33](#_Toc134467277)

[3. Тестирование и проверка работоспособности программного средства 34](#_Toc134467278)

[4. Руководство по развертыванию и использованию программного средства 35](#_Toc134467279)

[4.1 Руководство по установке (развертыванию) программного средства 35](#_Toc134467280)

[4.2 Руководство пользователя 38](#_Toc134467281)

[ЗАКЛЮЧЕНИЕ 41](#_Toc134467282)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 42](#_Toc134467283)

[Приложение А (обязательное) - Отчет о проверке на заимствования в системе «Антиплагиат» 43](#_Toc134467284)

[Приложение Б (обязательное) - Листинг кода алгоритмов, реализующих бизнес-логику 43](#_Toc134467285)

[Приложение В (обязательное) - Листинг скрипта генерации базы данных 59](#_Toc134467286)

# **ВВЕДЕНИЕ**

В данной работе будет рассматриваться разработка системы автоматизации деятельности санаторно-оздоровительного учреждения. Данная тема является достаточно актуальной, т.к. санаторий охватывает множество аспектов экономической деятельности. Специфика работы санатория — совмещение отдыха с лечением. Его инфраструктура чрезвычайно развита, а сотрудники вынуждены иметь дело с огромными объемами типовой информации. Службы сервиса, медицинских услуг, питания в санатории испытывают в современных условиях острую необходимость в автоматизации. Поэтому является необходимым проводить комплексную автоматизацию.

Внедрение АСУ в данную область позволит значительно систематизировать обмен данными, регламентировать состав и формы представления данных, а также структуру информационных потоков в системе (информационных и командных связей между субъектами санатория, а также информационный обмен с внешними по отношению к санаторию организациями). Также значительно повысить точность и четкость ведения документации, гарантировать сохранность, предоставлять полную взаимоувязанную информацию по всем субъектам санатория. Все это должно привести к слаженной работе сотрудников данного учреждения и как следствие повысить эффективность функционирования предприятия в целом.

Данный курсовой проект состоит из 4 частей (глав). Первая глава описывает общее содержание предметной области, организационную структуру предприятия в целом, раскрывает особенности отдельных подразделений. Во второй главе представлена структура самой автоматизированной системы управления санаторным комплексом, архитектура подразделений и АРМ сотрудников, работающих на данном предприятии. Третья часть курсового проекта содержит конкретный пример разработанного программного обеспечения. Четвертая часть содержит инструкции по использованию программного средства.

# **Анализ и моделирование предметной области программного средства**

## **Описание предметной области**

В данном курсовом проекте в качестве предметной области рассматривается - санаторий. Санаторий представляет собой лечебно-профилактическое учреждение, в котором осуществляется деятельность оздоровительного характера. Данное учреждение характеризуется представлением разного рода услуг: медицинских, развлекательных, культурных и т.д. Возрастной диапазон пациентов включает все возрасты от детей 3-х лет до пенсионеров. В санатории отдыхающие могут принять лечебные процедуры и получить специальное диетическое питание, посещать спортивные залы, бассейн и другие развлекательные мероприятия.

Внутреннюю структуру рассматриваемого санаторного комплекса можно представить состоящей из следующих компонентов:

* Приемное отделение (регистратура) – осуществляет регистрацию прибывших клиентов и распределение их по комнатам;
* Жилой комплекс – совокупность жилых корпусов и отделов, которые обеспечивают проживание пациентов в санатории (ведение информации о свободных и занятых номерах, подбор номера и расселение пациентов);
* Лечебный комплекс – обеспечивает проведение лечебных процедур для пациентов (проведение обследования пациентов, назначение соответствующего поставленному диагнозу лечения, снабжение пациентов необходимыми им лекарственными препаратами);
* Комплекс питания – обеспечивает пациентов необходимым для каждого питанием (обследование пациента врачом-диетологом, выбор системы питания и диеты, назначение соответствующего рациона питания).
* Культурно-развлекательный комплекс – занимается культурно-развлекательными программами для пациентов санатория (кино, экскурсии и т.д.).

Главный врач

Отдел снабжения

Отдел кадров

Финансовый отдел

Отдел работы с клиентами

Отдел работы с клиентами

Культ. - развлекат.блок

Диет-блок

Лечебный блок

Приемное отделение

Врач-специалист

Лаборатория

Жилой комплекс

Лечащий врач

Медицинская сестра

Отдел кадров

Зав. отделом

Блок-учет работников

Финансовый блок

Отдел реализации путевок

Бухгалтерия

Зав. отделом

Главный бухгалтер

Рисунок 1.1 – схема структуры санатория

## **Разработка функциональной модели предметной области**

Построение модели информационной системы в системе моделирования начинается с описания функционирования системы в целом для того, чтобы конкретно определить входящую и исходящую информацию.

Моделирование предметной области, как правило, выполняется с помощью CASE-средств. К таким средствам относятся BPwin (AllFusion Process Modeler), Oracle Designer (Oracle), Rational Rose (Rational Software) и другие.

BPwin поддерживает три методологии моделирования: функциональное моделирование (IDEF0) — верхнеуровневое; описание бизнес-процессов (IDEF3) — поток работ и диаграммы потоков данных (DFD). Чаще всего применяется для создания функциональной модели предметной области на начальных этапах проектирования информационной системы, а также для анализа существующей или проектируемой ИС.

Методология IDEF0 предназначена для моделирования деятельности организации. Функциональная модель рассматривает систему как набор действий, в котором каждое действие преобразует некоторый объект или набор объектов. Технология IDEF0 использует принцип функциональной декомпозиции систем (разбиение системы на фрагменты). Принцип декомпозиции означает, что функциональную модель следует строить по правилу «сверху вниз», от общего вида модели к частным моделям.

Как можно понять из определения, методология описывает бизнес-процессы, их иерархичность и взаимодействие в рамках определённой системы.

Бизнес-процесс – это совокупность взаимосвязанных мероприятий или работ, направленных на создание определённого продукта или услуги для потребителей. В качестве графического описания деятельности применяются блок-схемы бизнес-процессов.

Бизнес-процесс начинается со спроса потребителя и заканчивается его удовлетворением. Процессно-ориентированные организации стараются устранять барьеры и задержки, возникающие на стыке двух различных подразделений организации при выполнении одного бизнес-процесса.

Бизнес-процесс может быть декомпозирован на несколько подпроцессов, процедур и функций, которые имеют собственные атрибуты, однако также направлены на достижение цели основного бизнес-процесса. Такой анализ бизнес-процессов обычно включает в себя составление карты бизнес-процесса и его подпроцессов, разнесенных между определенными уровнями активности.

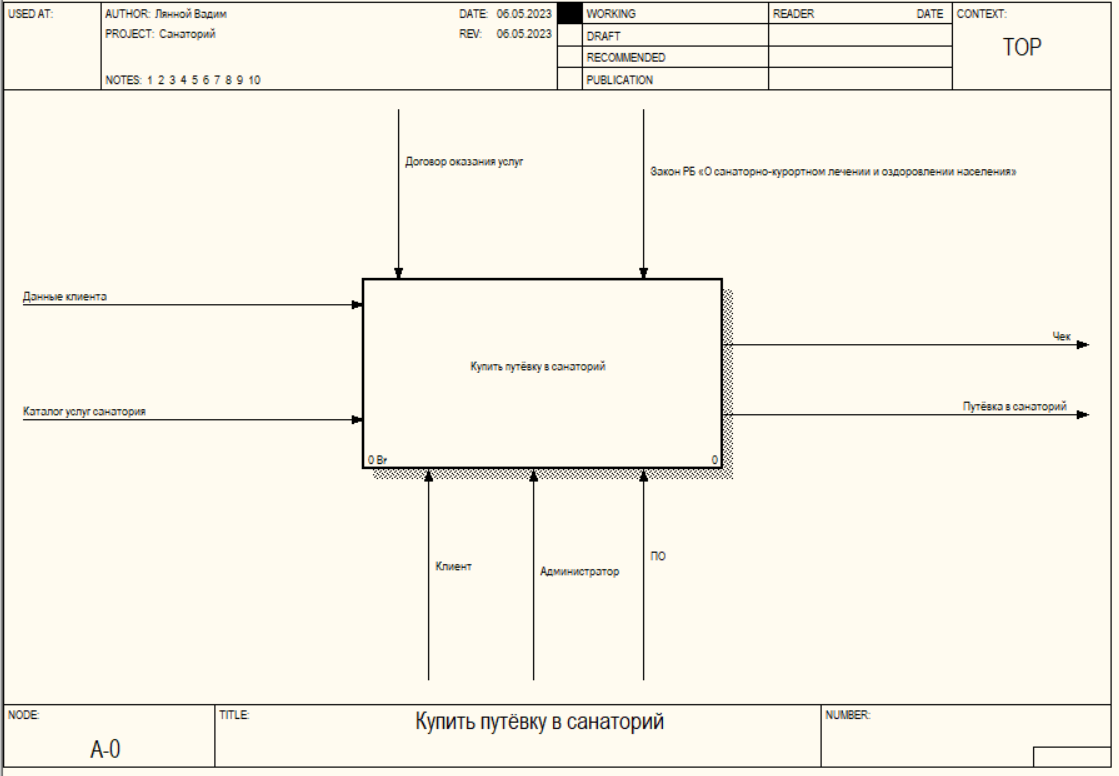
На контекстной диаграмме верхнего уровня представлена функциональная модель «Купить путёвку в санаторий»

Рисунок 1.2 – Контекстная диаграмма «Купить путёвку в санаторий»

Главным был выбран процесс «Купить путёвку в санаторий». Внутри данного процесса будет происходить обработка информации о клиенте и проверка возможности бронирования путёвки в выбранный период. В результате проведённой работы для нового клиента будет создана путёвка в санаторий, поэтому работать с клиентом будет администратор. Весь процесс взаимодействия клиента и санатория будет происходить в соответствии с договором оказания услуг, а также законом РБ «О санаторно-курортном лечении и оздоровлении населения».

Были определены входные и выходные данные, механизмы и управление.

На вход подаются параметры: данные клиента, каталог услуг санатория. Данные параметры будут преобразованы благодаря процессу в выходные параметры. Выходные параметры из данного блока: чек, путёвка в санаторий. Управление – договор оказания услуг и закон РБ «О санаторно-курортном лечении и оздоровлении населения». Механизмы – те или то, кто осуществляет сам процесс: администратор, клиент, программное обеспечение.

Контекстная диаграмма разбивается на процессы:

* Отправить заявку на бронирование;
* Выбрать услуги санатория;
* Оплатить услуги;
* Подтвердить путёвку.

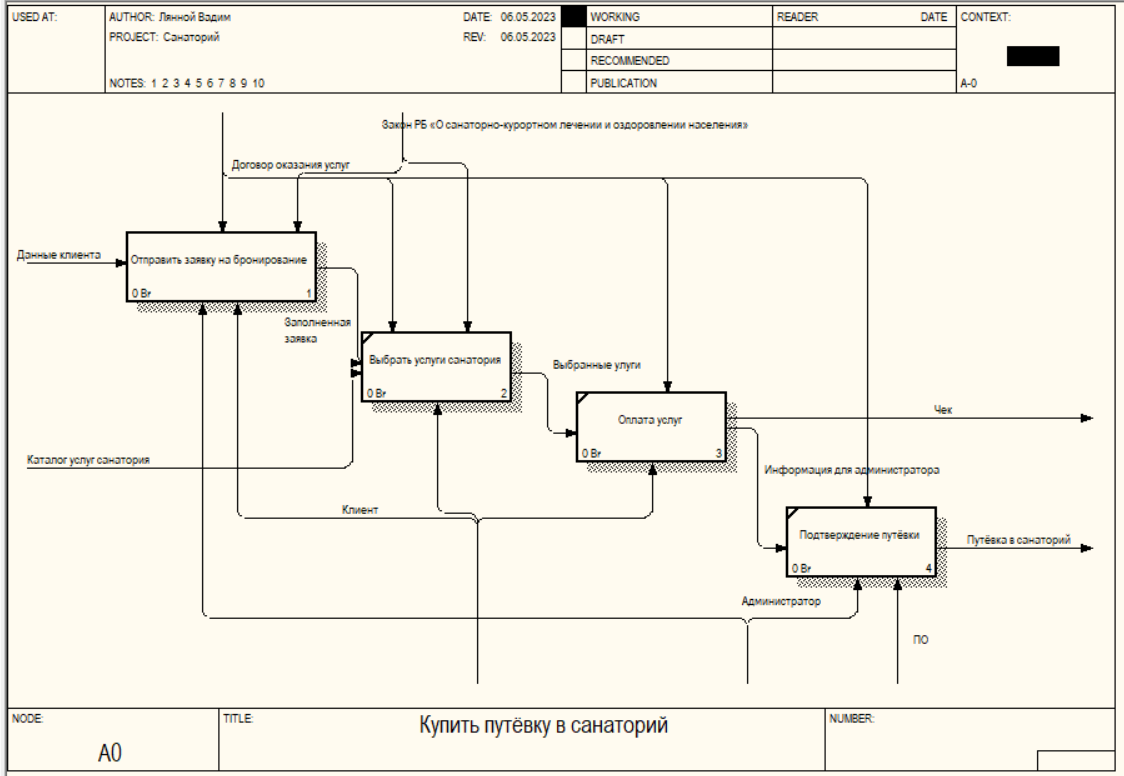


Рисунок 1.3 – Декомпозиция контекстной диаграммы «Отправить заявку на бронирование»

Процесс начинается с принятия запроса клиента. Здесь администратор узнает, что необходимо клиенту согласно перечню предлагаемых услуг, и принимает заказ. Далее необходимо обработать заказ, внеся его в информационную систему и расписание мастеров. Исходя из всех запросов, сделанных клиентами в адрес конкретных мастеров, мастера принимают список клиентов и выполняют требуемые ими услуги.

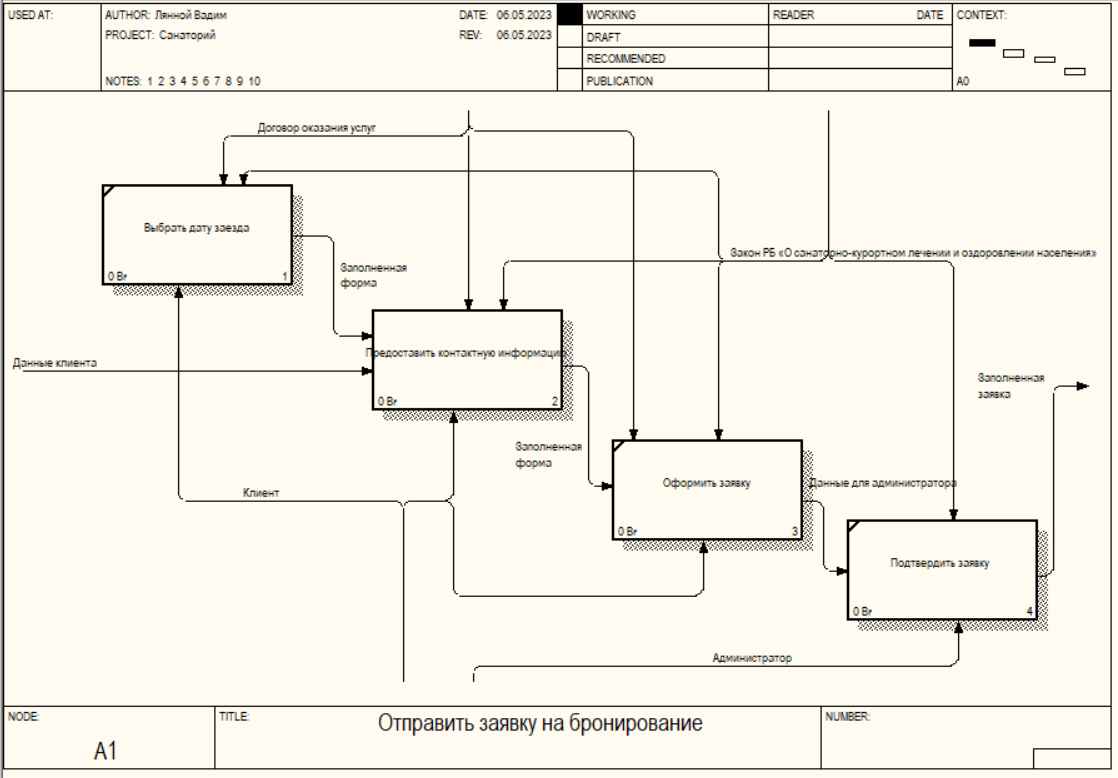


Рисунок 1.4 – Декомпозиция контекстной диаграммы «Отправить заявку на бронирование»

Так же проведём декомпозицию процесса «Отправить заявку на бронирование». Данный процесс разделим на четыре подпроцесса: «Выбрать дату заезда», «Предоставить контактную информацию», «Оформить заявку» и «Подтвердить заявку». Форма для заявки должна быть заполнена в соответствии с договором оказания услуг, а также законом РБ «О санаторно-курортном лечении и оздоровлении населения». Результатом данного процесса будет оформленная заявка.

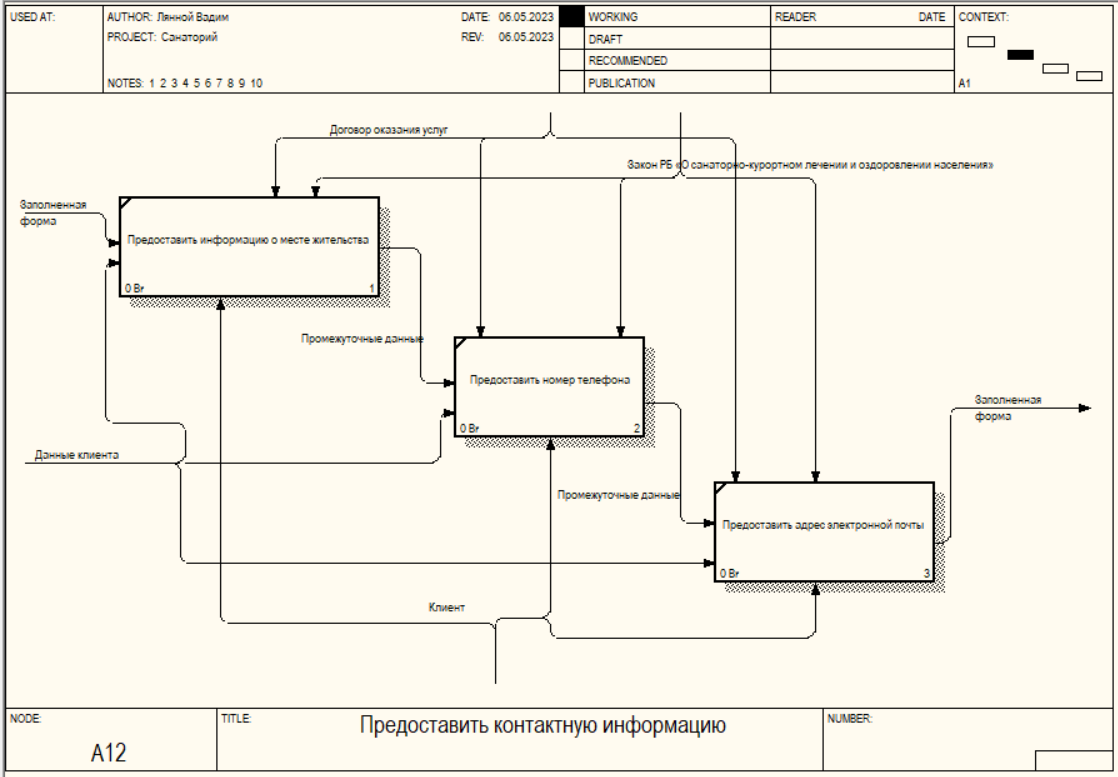


Рисунок 1.5 – Декомпозиция контекстной диаграммы «Предоставить контактную информацию»

При декомпозиции процесса «Предоставить контактную информацию» были определены следующие процессы:

* предоставить информацию о месте жительства;
* предоставить номер телефона;
* предоставить адрес электронной почты.

На данном уровне производится проверка предоставленной информации в соответствии с договором оказания услуг и доработка недочётов. На выходе будет получена проверенная контактная информация для оформления заявки на бронирование путёвки.

В результате проведения всех операций клиент оформит путёвку в санаторий на определённую дату. А администратор на этом завершит работу с клиентом.

Таким образом было проведено описание основного процесса предметной области: «Купить путёвку в санаторий».

## **Анализ требований к разрабатываемому программному средству.**

Цель процесса анализа требований к программному средству заключается в установлении и документировании требований к программному обеспечению.

Для реализации Back-end части будут использованы язык программирования Java, сервер базы данных PostgresSQL. Для реализации Front-end части будут использованы следующие технологии: HTML, CSS и JavaScript. Среда разработки – Intelij IDEA. Браузер Google Chrome – для проверки результатов работы программного средства при разработке.

Каждая из выбранных технологий отвечает за разные аспекты работы программы.

Основные требования к приложению:

Пользователь:

* заселение в санаторий;
* запись на прием к врачу;
* заказ трансфера в санаторий;
* заказ еды из столовой санатория;
* возможность записаться на реабилитацию после операций;
* регистрация;
* наличие личного кабинета с информацией о записях и т.д.;
* обратная связь с администрацией.

Администратор:

* вносить, изменять и удалять данные;
* отвечать на вопросы обычных пользователей.

## **Разработка информационной модели предметной области**

Построение информационной модели требует представления сущностей таблицами, а атрибутов сущностей – столбцами таблиц; для возможных ключей определяется ограничение unique, внешние ключи становятся декларациями ссылочной целостности. Конструкции в информационной модели должны быть реализуемы в выбранной СУБД. Процесс построения информационной модели состоит из следующих шагов: определение сущностей, определение зависимостей между сущностями, задание первичных и альтернативных ключей, определение атрибутов сущностей, приведение модели к требуемому уровню нормальной формы.

Для хранения данных, используемых в приложении, применяется база данных PostgresSQL. Схема данных приведена на рисунке 1.6.

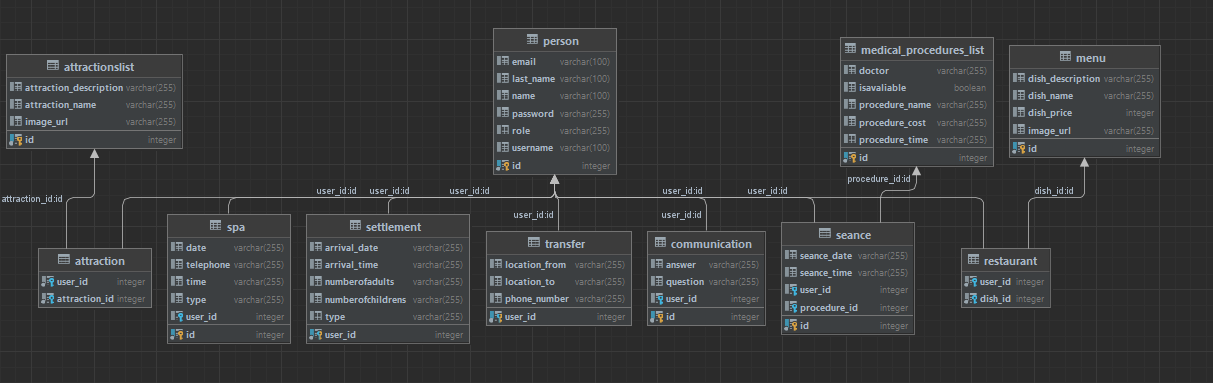


Рисунок. 1.6 – Схема данных

Для решения поставленной задачи созданы и использованы следующие сущности:

* Достопримечательность (attraction);
* Список достопримечательностей (attractionslist);
* Обратная связь (communication);
* Список медицинских процедур(medical\_procedures\_list);
* Меню ресторана (menu);
* Пользователь (person);
* Ресторан санатория (restaurant);
* Сеанс (seance);
* Заселение в отель (settlement);
* СПА (SPA);
* Трансфер в отель (transfer)

Далее описаны структура и тип хранящихся данных каждой сущности.

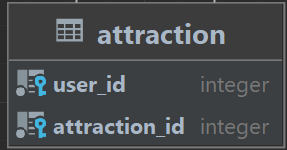


Рисунок. 1.7 – сущность «Достопримечательность»

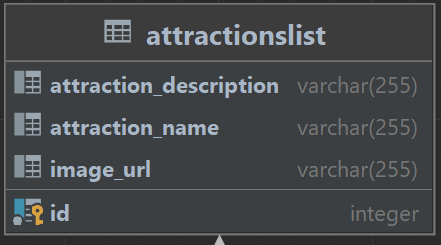
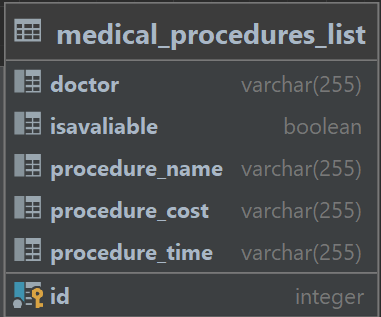
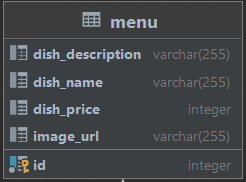


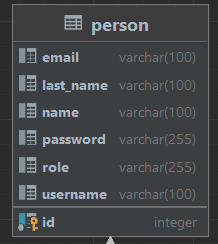
Рисунок. 1.8 – сущность «Список достопримечательностей»



Рисунок. 1.9 – сущность «Обратная связь»



Рисунок. 1.10 – сущность «Список медицинских процедур»

Рисунок. 1.11 – сущность «Меню ресторана»

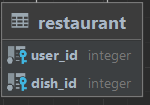
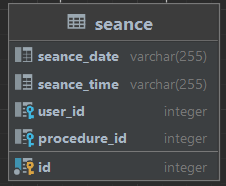
Рисунок. 1.12 – сущность «Пользователь»

Рисунок. 1.13 – сущность «Ресторан санатория»

Рисунок. 1.14 – сущность «Сеанс»

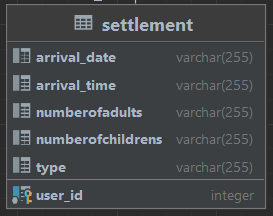


Рисунок. 1.15– сущность «Заселение в отель»

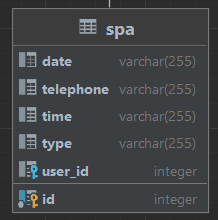


Рисунок. 1.16 – сущность «СПА»

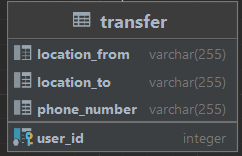


Рисунок. 1.17 – сущность «Трансфер в отель»

Созданная информационная модель представляет собой структуру информации, необходимой для поддержки сайта санатория.

## **UML-модели представления программного средства и их описание**

UML (Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML является языком широкого профиля, это — открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода.

В UML используются следующие виды диаграмм:

* диаграмма классов;
* диаграмма компонентов;
* диаграмма развёртывания;
* диаграмма прецедентов (Диаграмма вариантов использования);
* диаграмма состояний;
* диаграмма последовательности и другие.

Система будет описана следующими графическими представлениями, которые приведены далее:

* диаграмма вариантов использования;
* диаграмма последовательности;
* диаграмма состояний;
* диаграмма классов.

Диаграмма прецедентов или диаграмма вариантов использования (use case diagram) в UML — диаграмма, отражающая отношения между акторами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Основное назначение диаграммы — описание функциональности и поведения, позволяющее заказчику, конечному пользователю и разработчику совместно обсуждать проектируемую или существующую систему.

Для отражения модели прецедентов на диаграмме используются:

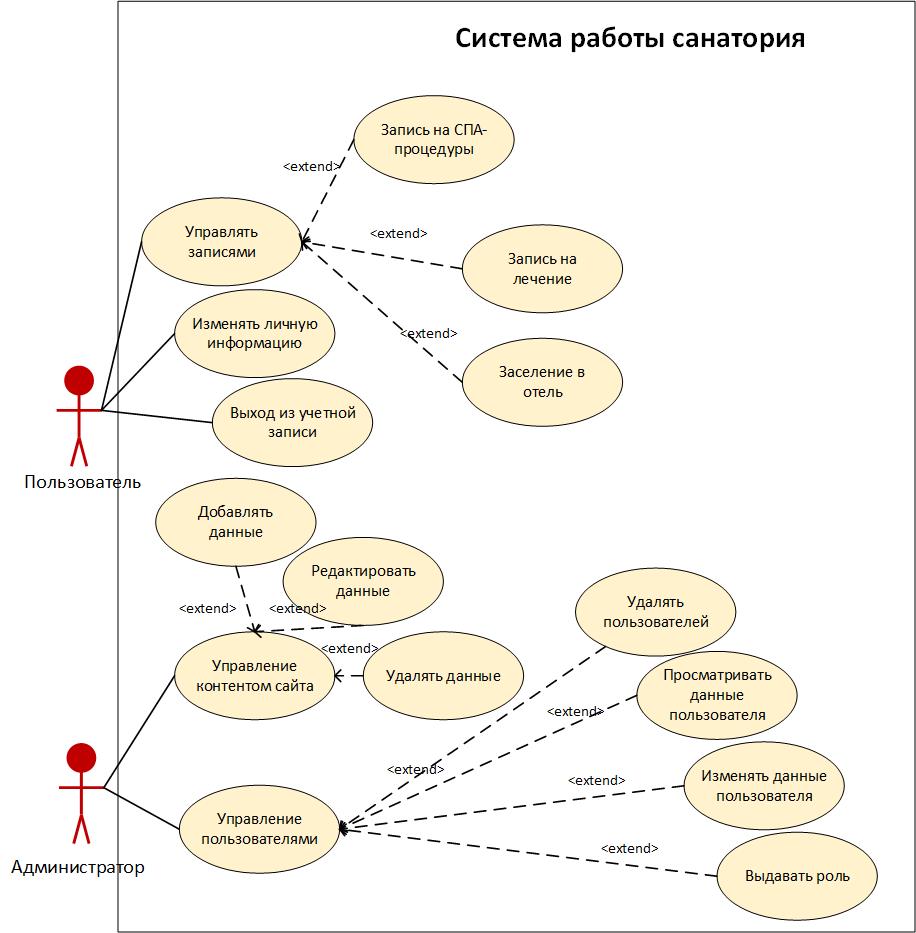
* рамки системы (system boundary) — прямоугольник с названием в верхней части и эллипсами (прецедентами) внутри. Часто может быть опущен без потери полезной информации;
* актор (actor) — стилизованный человечек, обозначающий набор ролей пользователя (понимается в широком смысле: человек, внешняя сущность, класс, другая система), взаимодействующий с некоторой сущностью (системой, подсистемой, классом). Акторы не могут быть связаны друг с другом (за исключением отношений обобщения/наследования);
* прецедент — эллипс с надписью, обозначающий выполняемые системой действия (могут включать возможные варианты), приводящие к наблюдаемым акторами результатам. Надпись может быть именем или описанием (с точки зрения актора) того, «что» делает система (а не «как»). Имя прецедента связано с непрерывным (атомарным) сценарием — конкретной последовательностью действий, иллюстрирующей поведение. В ходе сценария акторы обмениваются с системой сообщениями. Сценарий может быть приведён на диаграмме прецедентов в виде UML-комментария. С одним прецедентом может быть связано несколько различных сценариев.

Рисунок 1.18 – Диаграмма вариантов использования

Диаграмма последовательности — UML-диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие акторов (действующих лиц) информационной системы в рамках прецедента.

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни» (англ. lifeline), отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни» — фокусы контроля, англ. focus of control), и стрелки, показывающие обмен сигналами или сообщениями между объектами.

Как было сказано выше, взаимодействие между акторами отображается при помощи специальных стрелок, передающих управление от отправителя (от кого идёт стрелка) к получателю (тот, к кому направлена стрелка). Стрелки демонстрируют ход сценария и те события, которые происходят во время анализируемого прецедента. Всего существует 5 видов стрелок:

Синхронное сообщение — актор-отправитель передаёт ход управления актору-получателю, которому необходимо провести в прецеденте некоторое действие. Пока проводимое актором-получателем действие не будет завершено (соответственно, не будет получено ответное сообщение), актор-отправитель теряет возможность производить какие-либо действия. Графически изображается как сплошная линия со стрелкой в виде закрашенного треугольника, после которой идёт прямоугольник, отражающий деятельность объекта, в конце которого находится ответное сообщение.

Ответное сообщение — данное сообщение является ответом на синхронное сообщение. Обычно, содержит какое-либо возвращаемое изначальному актору-отправителю значение, также возвращающее ему управление (возможность действовать). Графически изображается пунктирной линией с открытой стрелкой.

Асинхронное сообщение — актор-отправитель передаёт ход управления актору-получателю, которому необходимо провести в прецеденте некоторое действие. Основное отличие от синхронного сообщения состоит в том, что актор-отправитель не теряет возможности совершать другие действия. Графически изображается сплошной линией с открытой стрелкой.

Потерянное сообщение — сообщение без адресата (есть отправитель, нет получателя).

Найденное сообщение — сообщение без отправителя.

Последние два вида стрелок (взаимодействий) используются крайне редко. В основном они используются для демонстрации взаимодействия имеющихся объектов в данном прецеденте с внешними системами.

На рисунке 1.19 представлена диаграмма последовательности, отображающая процесс работы санатория.

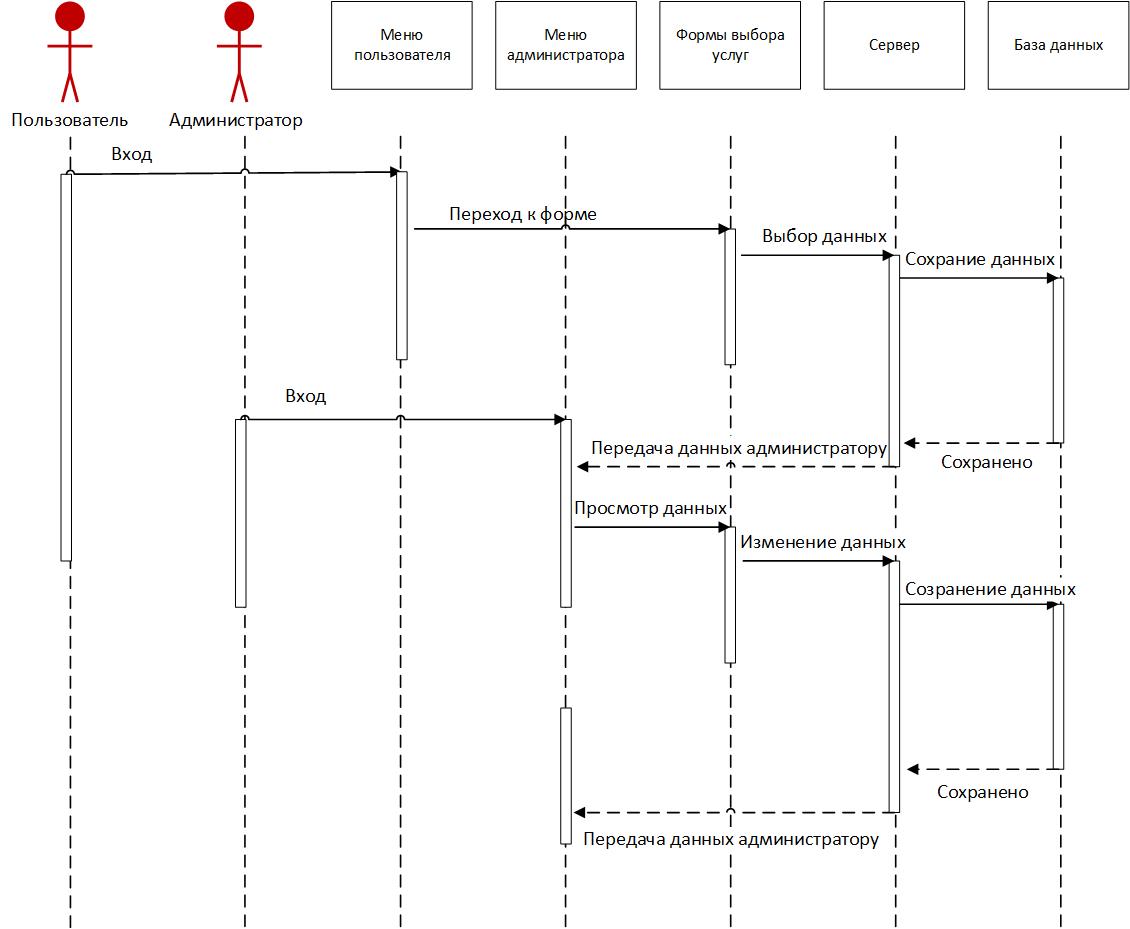


Рисунок 1.19 — Диаграмма последовательности

Далее диаграмма состояний. Диаграмма состояний — это, по существу, диаграмма состояний из теории автоматов со стандартизированными условными обозначениями, которая может определять множество систем от компьютерных программ до бизнес-процессов. Используются следующие условные обозначения:

* круг, обозначающий начальное состояние;
* окружность с маленьким кругом внутри, обозначающая конечное состояние (если есть);
* скруглённый прямоугольник, обозначающий состояние. Верхушка прямоугольника содержит название состояния. В середине может быть горизонтальная линия, под которой записываются активности, происходящие в данном состоянии;
* стрелка, обозначающая переход. Название события (если есть), вызывающего переход, отмечается рядом со стрелкой. Охраняющее выражение может быть добавлено перед «/» и заключено в квадратные скобки (название\_события[охраняющее\_выражение]), что значит, что это выражение должно быть истинным, чтобы переход имел место. Если при переходе производится какое-то действие, то оно добавляется после «/» (название\_события[охраняющее\_выражение]/действие);
* толстая горизонтальная линия с либо множеством входящих линий и одной выходящей, либо одной входящей линией и множеством выходящих. Это обозначает объединение и разветвление соответственно.

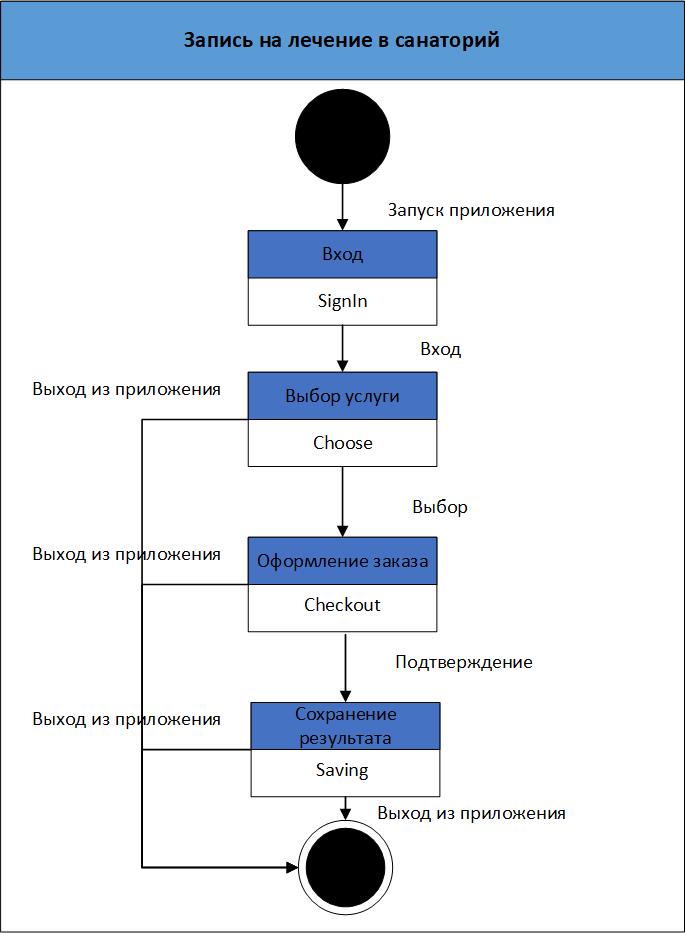


Рисунок 1.19 — Диаграмма состояний

Данная диаграмма представляет главное состояние – запись на лечение в санаторий. Сам процесс начинается и завершается соответствующими блоками – закрашенным кругом и полым с меньшем закрашенным кругом внутри соответственно.

В рамках процесса представлена последовательность состояний:

– вход в систему;

– выбор услуги;

– оформление заказа.

При этом, в любой момент может быть осуществлён переход к завершающему состоянию, в виду того, что пользователь в любой момент может закрыть программу.

Диаграмма классов (class diagram) — структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей (отношений) между ними. Широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого или обратного проектирования.

Класс является ключевым элементом в объектно-ориентированном моделировании. На диаграмме классы представлены в рамках, содержащих три компонента:

* В верхней части написано имя класса. Имя класса выравнивается по центру и пишется полужирным шрифтом. Имена классов начинаются с заглавной буквы. Если класс абстрактный — то его имя пишется полужирным курсивом.
* Посередине располагаются поля (атрибуты) класса. Они выровнены по левому краю и начинаются с маленькой буквы.
* Нижняя часть содержит методы класса. Они также выровнены по левому краю и пишутся с маленькой буквы.

Язык UML предоставляет механизмы для представления членов класса, например атрибутов и методов, а также дополнительной информации о них.

Рисунок 1.20 – диаграмма классов пакета models

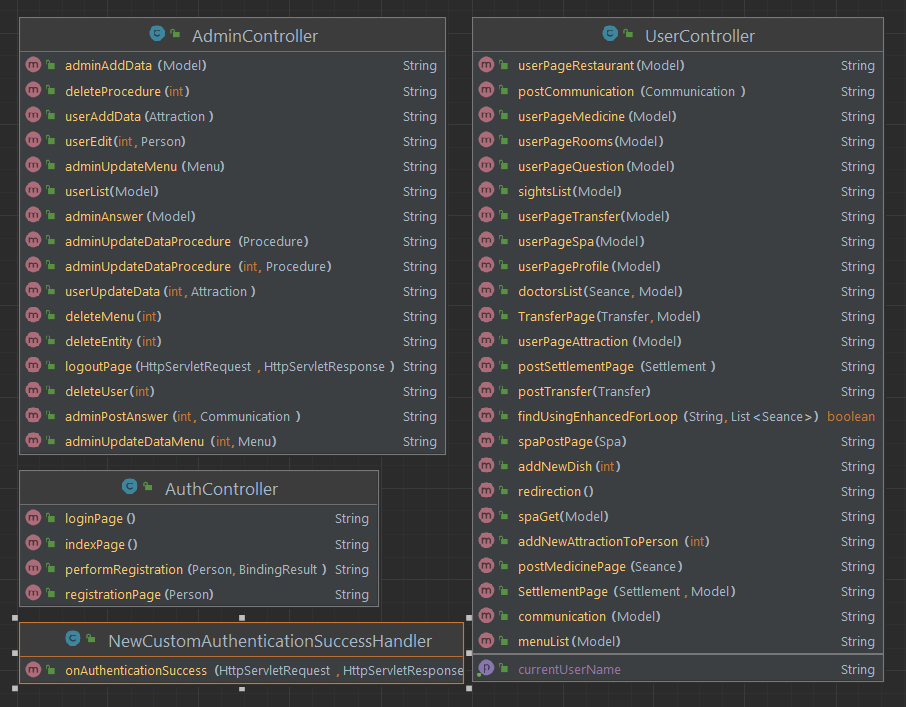


Рисунок 1.21 – диаграмма классов пакета controllers

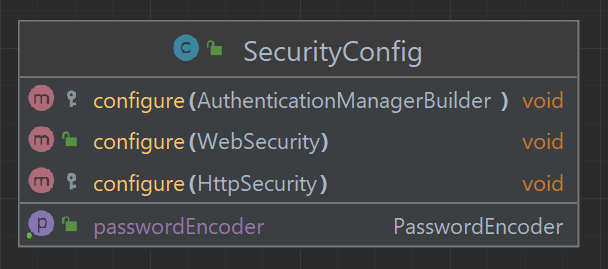


Рисунок 1.22 – диаграмма классов пакета config

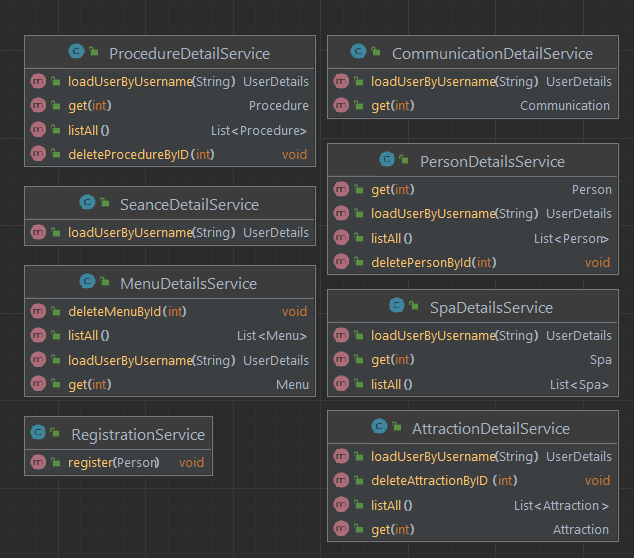


Рисунок 1.23 – диаграмма классов пакета service

# **Проектирование и конструирование программного средства**

## **Постановка задачи**

Задачей курсовой работы является разработка сайта санатория, который будет позволять пользователям выбирать услуги и бронировать их, а администратору работать с информацией на сайте.

Программное средство должно обладать следующими функциями доступными зарегистрированному пользователю:

* просмотр информации сайта;
* заказ услуг на сайте;
* общение с администрацией сайта.

В свою очередь администратор должен иметь возможность:

* добавлять, редактировать и удалять информацию об услугах на сайте;
* редактировать, удалять данные пользователей;
* отвечать на вопросы пользователей.

## **Обоснование выбора компонентов и технологий для реализации программного средства**

Для разработки автоматизированной системы работы санатория использованы средства, предоставляемые фреймворком Spring, так как на момент написания курсового проекта данная технология является актуальной, и решает весь спектр задач при построении enterprise-приложений, предоставляя такие инструменты как Spring Boot, Spring Data, Spring Security.

Spring обеспечивает решения многих задач, с которыми сталкиваются Java-разработчики и организации, которые хотят создать информационную систему, основанную на платформе Java. Из-за широкой функциональности трудно определить наиболее значимые структурные элементы, из которых он состоит. Spring не всецело связан с платформой Java Enterprise, несмотря на свою масштабную интеграцию с ней, что является важной причиной его популярности.

Spring, вероятно, наиболее известен как источник расширений (features), нужных для эффективной разработки сложных бизнес-приложений вне тяжеловесных программных моделей, которые исторически были доминирующими в промышленности. Ещё одно его достоинство в том, что он ввел ранее неиспользуемые функциональные возможности в сегодняшние господствующие методы разработки, даже вне платформы Java.

Этот фреймворк предлагает последовательную модель и делает её применимой к большинству типов приложений, которые уже созданы на основе платформы Java. Считается, что Spring реализует модель разработки, основанную на лучших стандартах индустрии, и делает её доступной во многих областях Java.

Для тестирования автоматизированной системы работы санатория использован фреймворк Spring (spring-boot-starter-test, spring-security-test).

При разработке автоматизированной системы использована СУБД PostgreSQL, так как она является оптимальным решением для построения сложных систем с большим числом запросов.

## **Архитектурные решения**

Архитектура программного обеспечения – описание структуры программной системы, включающее программные компоненты, их свойства и отношения между ними. Правильно выбранная архитектура помогает создать успешный программный продукт, который в дальнейшем может легко адаптироваться к изменяющимся требованиям заказчика. Благодаря правильно спроектированной архитектуре программный продукт удовлетворяет не только функциональным требованиям, но и атрибутом качества.

Концептуальная архитектура (conceptual architecture) – идейное представление об архитектуре программного приложения.

Архитектурный стиль (reference architecture, шаблонная архитектура) – это абстрактная архитектура, применимая к некоторому множеству программных продуктов. Распространенными архитектурными стилями являются следующие архитектурные стили:

* клиент-серверный;
* многослойный;
* многоуровневый;
* сервис-ориентированный;
* шина сообщений.

Шаблонные архитектурные решения имеют следующие преимущества:

* предоставляют общую терминологию;
* являются решением, проверенным на уже существующем программном обеспечении.

В проекте будет использоваться паттерн MVC и Spring boot.

Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.

Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.

Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Spring Boot — это полезный проект, целью которого является упрощение создания приложений на основе Spring. Он позволяет наиболее простым способом создать web-приложение, требуя от разработчиков минимум усилий по его настройке и написанию кода.

Spring Boot обладает большим функционалом, но его наиболее значимыми особенностями являются: управление зависимостями, автоматическая конфигурация и встроенные контейнеры сервлетов.

## **Описание алгоритмов, реализующих ключевую бизнес-логику разрабатываемого программного средства**

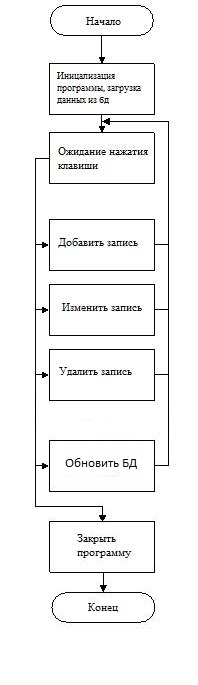
Бизнес-логика — в разработке информационных систем — совокупность правил, принципов, зависимостей поведения объектов предметной области (области человеческой деятельности, которую система поддерживает). Иначе можно сказать, что бизнес-логика — это реализация правил и ограничений автоматизируемых операций. Является синонимом термина «логика предметной области».

Проще говоря, бизнес-логика — это реализация предметной области в информационной системе. К ней относятся, например, формулы расчёта ежемесячных выплат по ссудам (в финансовой индустрии), автоматизированная отправка сообщений электронной почты руководителю проекта по окончании выполнения частей задания всеми подчиненными (в системах управления проектами), отказ от отеля при отмене рейса авиакомпанией (в туристическом бизнесе) и т. д.

В фазе бизнес-моделирования и разработки требований бизнес-логика может описываться в виде:

* текста;
* концептуальных аналитических моделей предметной области;
* бизнес-правил;
* разнообразных алгоритмов;
* диаграмм деятельности;
* графов и диаграмм перехода состояний;
* моделей бизнес-процессов.

В фазе анализа и проектирования системы бизнес-логика воплощается в различных диаграммах языка UML или ему подобных. В фазе программирования бизнес-логика воплощается в коде классов и их методов, в случае использования объектно-ориентированных языков программирования, или процедур и функций, в случае применения процедурных языков.

Рисунок 2.24 - Алгоритм описания бизнес-логики процессов редактирования содержимого приложения

## **Проектирование пользовательского интерфейса**

Интерфейс пользователя, он же пользовательский интерфейс (UI — англ. user interface) — интерфейс, обеспечивающий передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы.

Под совокупностью средств и методов интерфейса пользователя подразумеваются:

* средства вывода информации из устройства к пользователю — весь доступный диапазон воздействий на организм человека (зрительных, слуховых, тактильных, обонятельных и т. д.) — экраны (дисплеи, проекторы) и лампочки, динамики, зуммеры и сирены, вибромоторы;
* средства ввода информации/команд пользователем в устройство — множество всевозможных устройств для контроля состояния человека — кнопки, переключатели, потенциометры, датчики положения и движения, сервоприводы, жесты лицом и руками, даже съём мозговой активности пользователя.

По наличию тех или иных средств ввода, интерфейсы разделяются на типы — жестовый, голосовой, брэйн, и т. д., возможны смешанные варианты. Средства эти должны быть необходимыми и достаточными, быть удобными и практичными, расположенными и скомпонованными разумно и понятно, соответствовать физиологии человека, не должны приводить к негативным последствиям для организма пользователя (всё это входит в понятие эргономики).

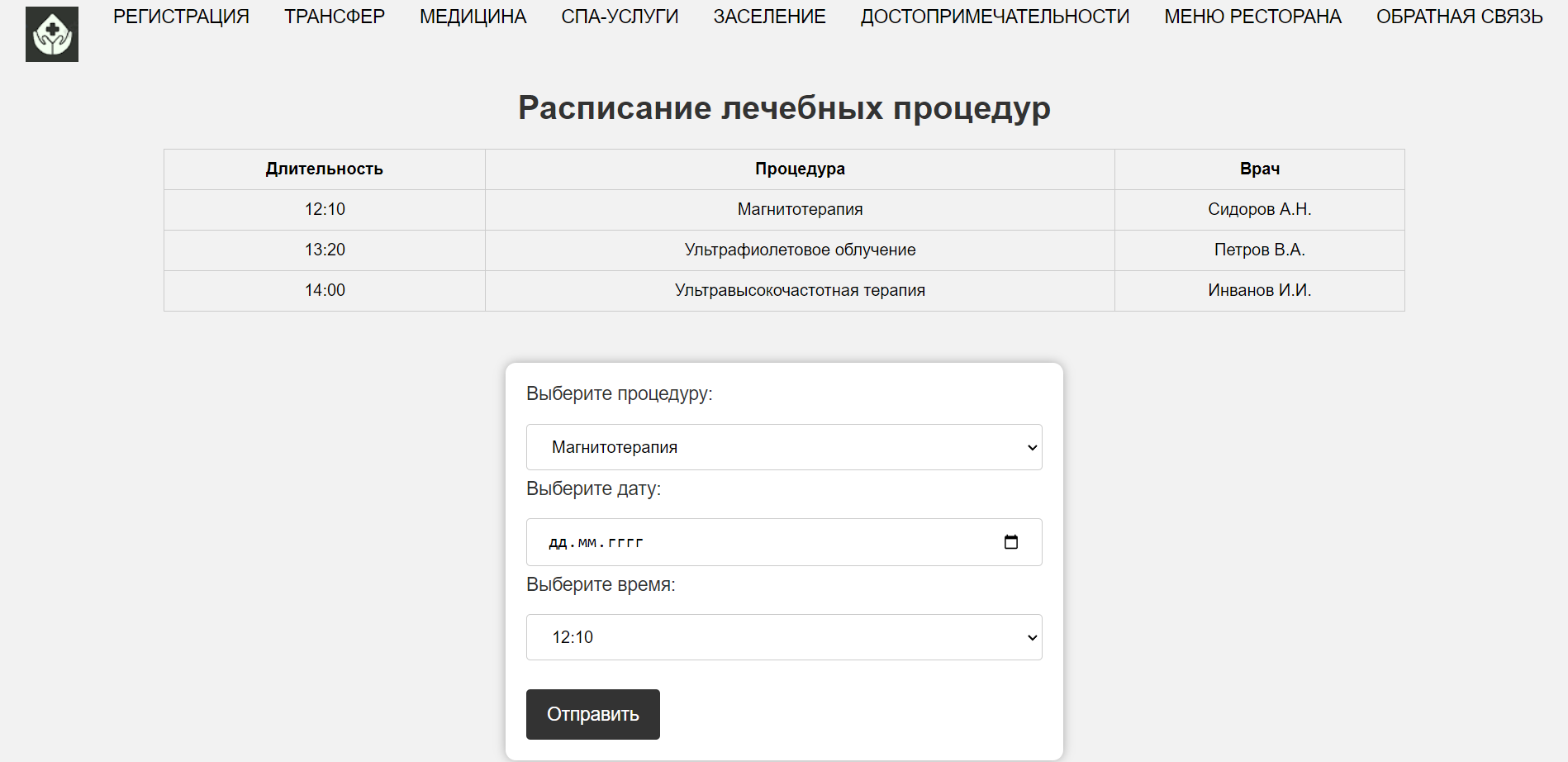
Методы — набор правил, заложенных разработчиком устройства, согласно которым совокупность действий пользователя должна привести к необходимой реакции устройства и выполнения требуемой задачи — так называемый логический интерфейс.

Рисунок 2. – Запись на лечебные процедуры

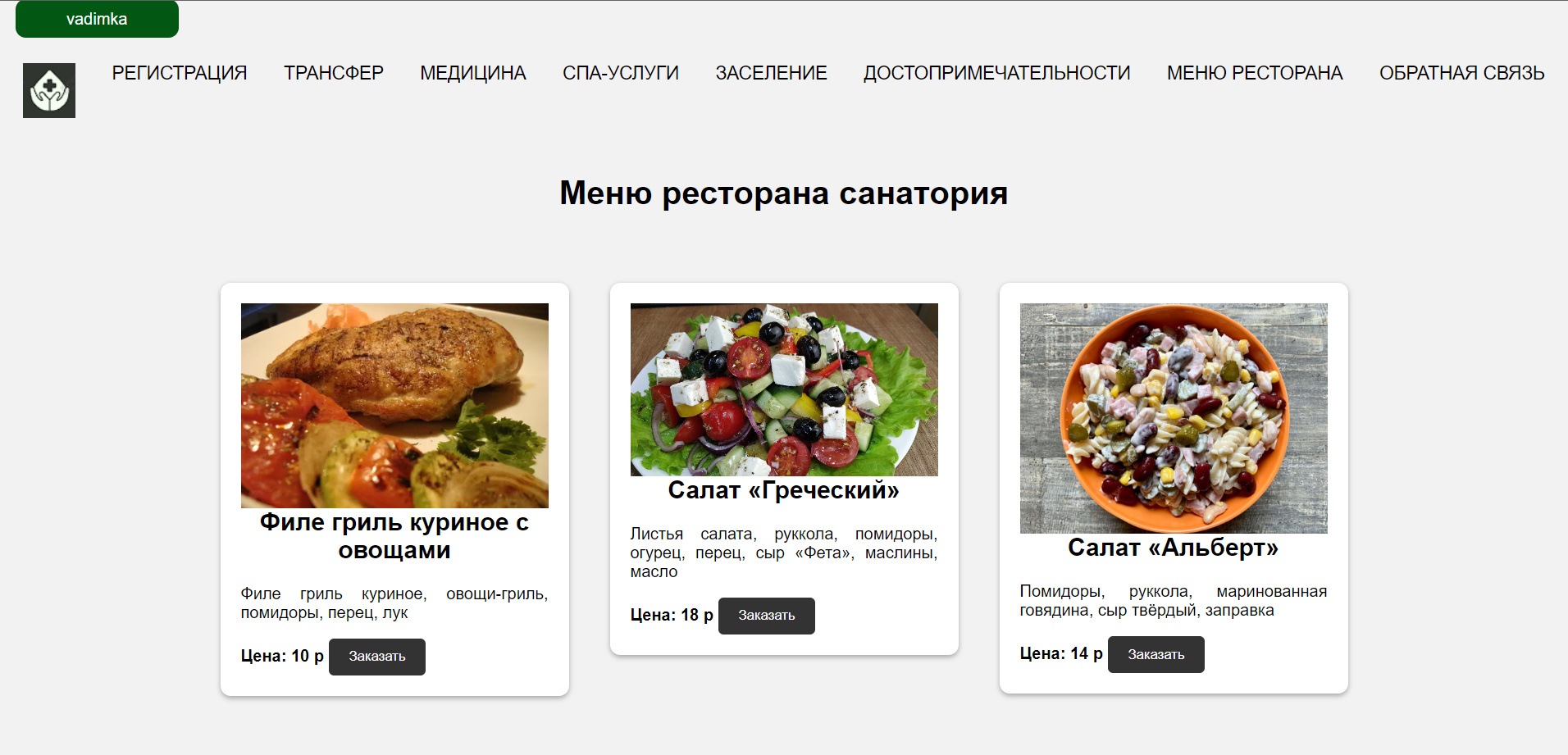


Рисунок 2. – Меню ресторана санатория

Рисунок 2. - Достопримечательности

## **Методы и средства, используемые для обеспечения безопасности данных**

В данной работе безопасность была обеспечена посредством фреймворка Spring Security.

Spring Security это Java/Java EE фреймворк, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для промышленных приложений, созданных с помощью Spring Framework.

Ключевые объекты контекста Spring Security:

SecurityContextHolder, в нем содержится информация о текущем контексте безопасности приложения, который включает в себя подробную информацию о пользователе (Principal), работающем в настоящее время с приложением. По умолчанию SecurityContextHolder используетThreadLocal для хранения такой информации, что означает, что контекст безопасности всегда доступен для методов, исполняющихся в том же самом потоке. Для того что бы изменить стратегию хранения этой информации можно воспользоваться статическим методом класса SecurityContextHolder.setStrategyName(String strategy). Более подробно SecurityContextHolder.

SecurityContext, содержит объект Authentication и в случае необходимости информацию системы безопасности, связанную с запросом от пользователя.

Authentication представляет пользователя (Principal) с точки зрения Spring Security.

GrantedAuthority отражает разрешения, выданные пользователю в масштабе всего приложения, такие разрешения (как правило называются «роли»), например ROLE\_ANONYMOUS, ROLE\_USER, ROLE\_ADMIN.

UserDetails предоставляет необходимую информацию для построения объекта Authentication из DAO объектов приложения или других источников данных системы безопасности. Объект UserDetailsсодержит имя пользователя, пароль, флаги: isAccountNonExpired, isAccountNonLocked, isCredentialsNonExpired, isEnabled и Collection — прав (ролей) пользователя.

UserDetailsService, используется чтобы создать UserDetails объект путем реализации единственного метода этого интерфейса.

# **Тестирование и проверка работоспособности программного средства**



Рисунок 3.1 – Результаты тестирования класса SpringSecurityTests

# **Руководство по развертыванию и использованию программного средства**

## **Руководство по установке (развертыванию) программного средства**

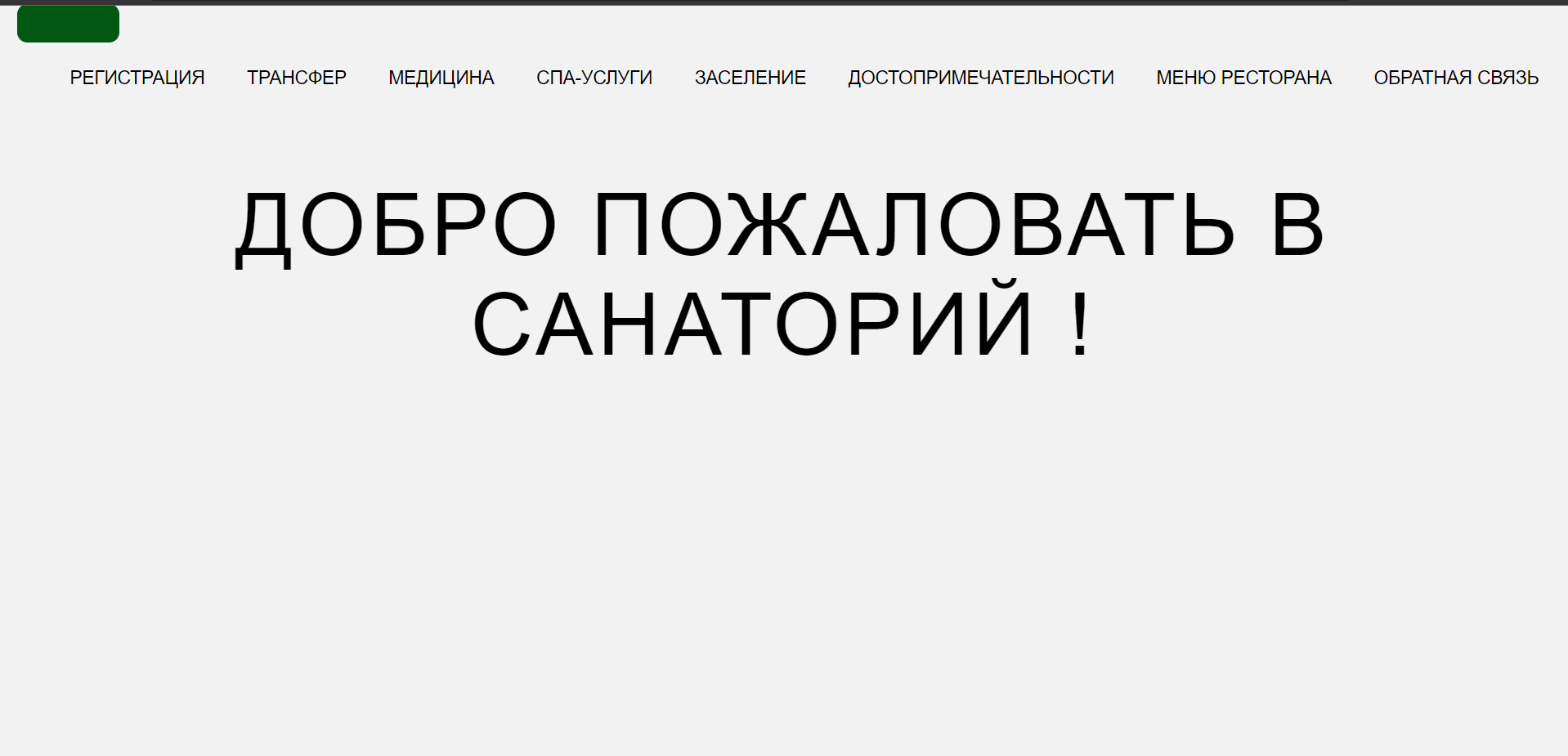
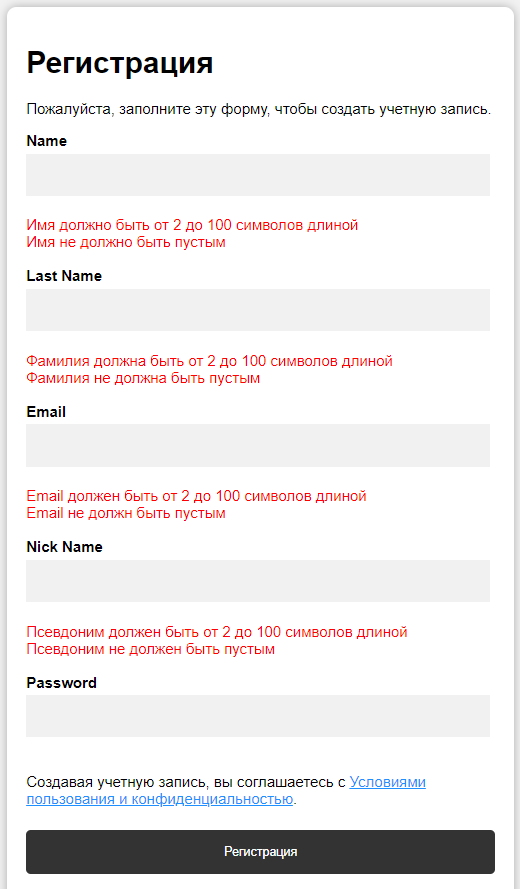
При запуске приложения появляется главная страница сайта. С неё невозможно перейти на все остальные страницы из меню если пользователь не зарегистрировался.

Рисунок 4.1 – Главная страница приложения

Форма регистрации имеет свои правила, при несоблюдении которых пользователь не сможет зарегистрироваться.

Рисунок 4.2 – Форма регистрации

При попытке зарегистрироваться с данными уже существующего пользователя выводится сообщение об ошибке, представленное на рисунке 4.3.

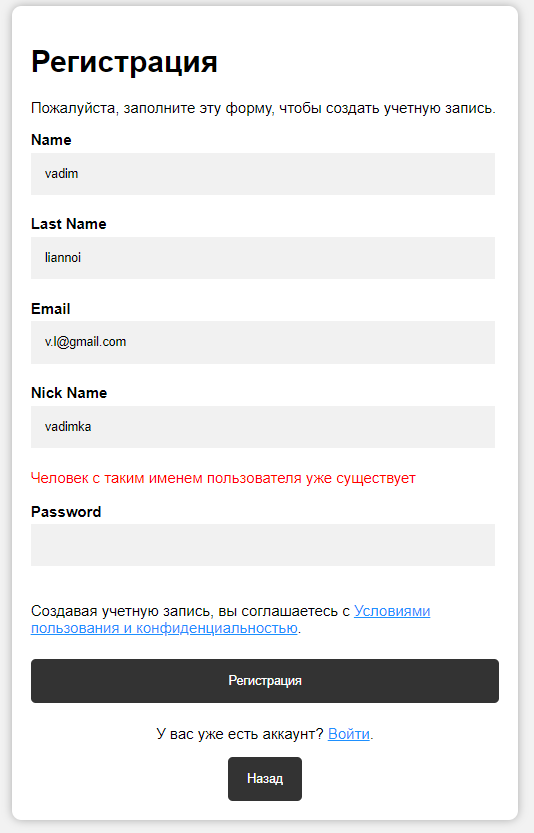


Рисунок 4.3 – Ошибка «Пользователь с таким именем пользователя уже существует»

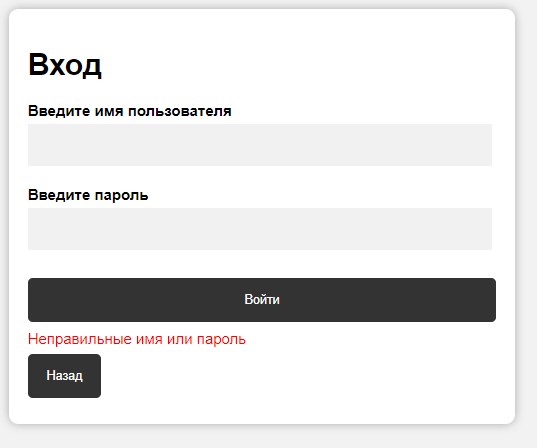
При прохождении авторизации необходимо ввести верный логин и пароль зарегистрированного пользователя, в противном случае выводится сообщение об ошибке регистрации.

Рисунок 4.4 – Ошибка авторизации

## **Руководство пользователя**

После прохождения регистрации, пользователю становится доступными меню приложения. Ему открывается возможность просмотра меню ресторана санатория, выбор медицинских и оздоровительных услуг, посещение экскурсий, возможность заселиться в номер, а также обратная связь с администрацией санатория.

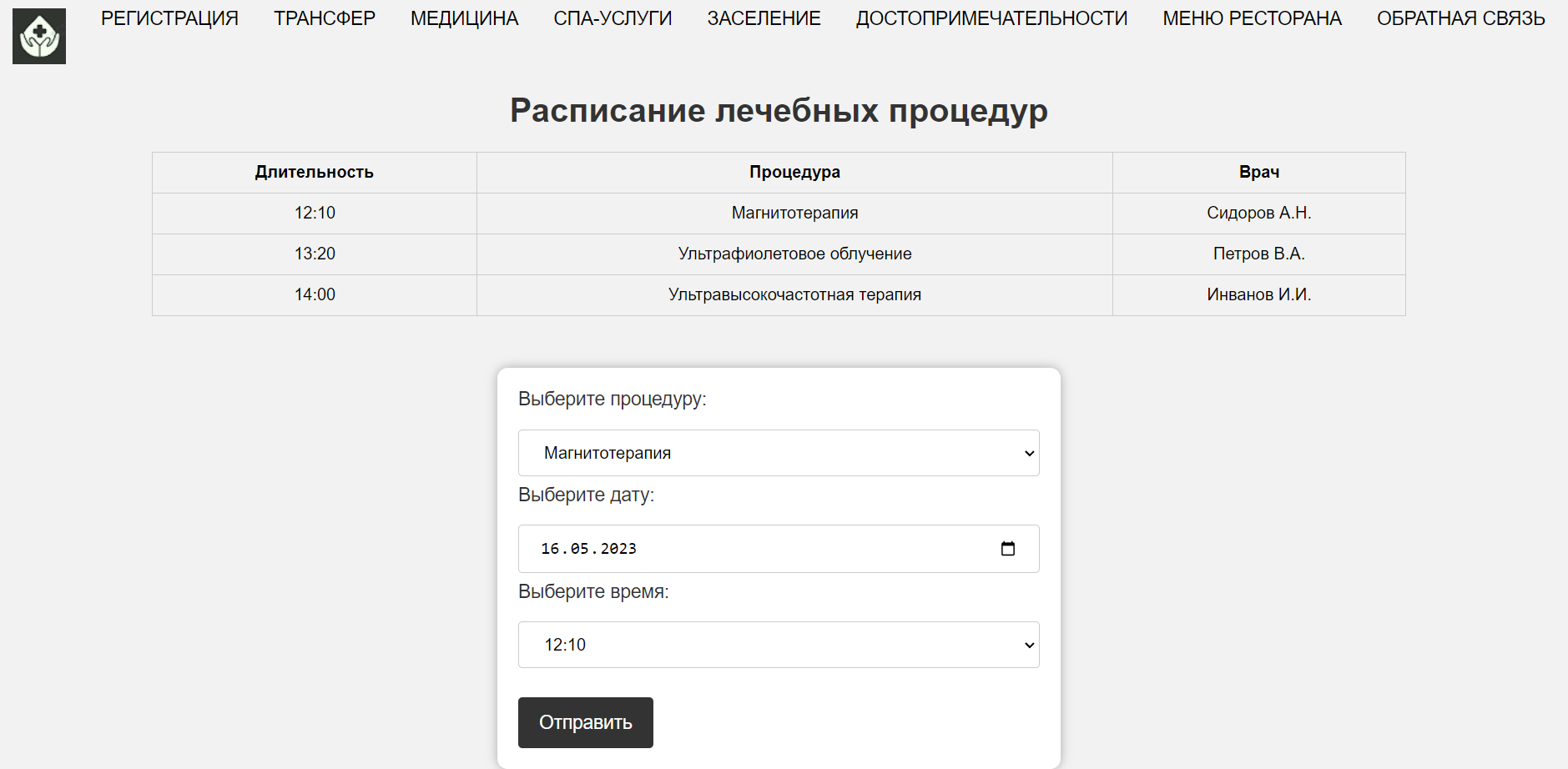


Рисунок 4.5 – Страница «Лечение»

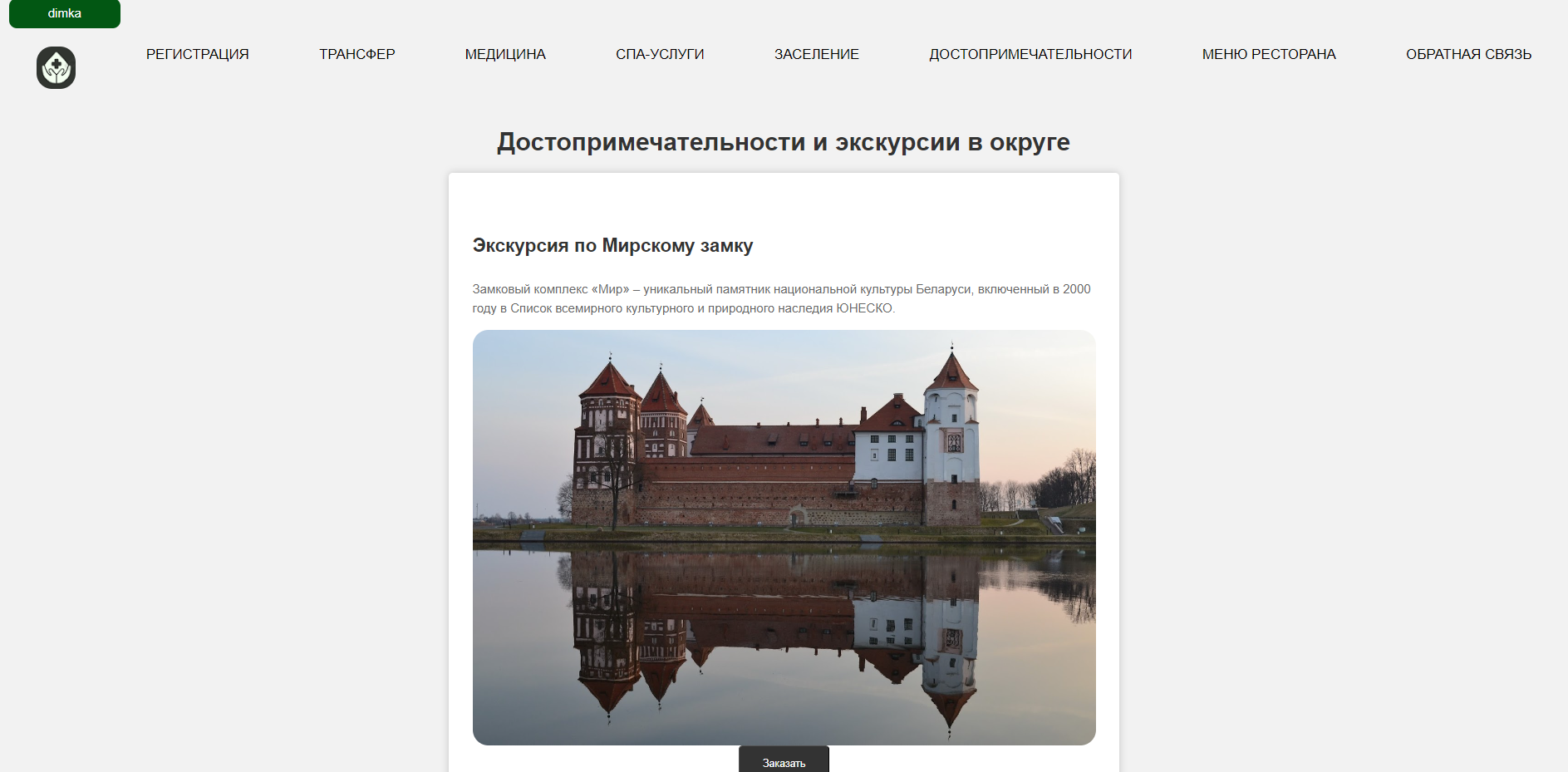


Рисунок 4.6 – Страница «Достопримечательности»

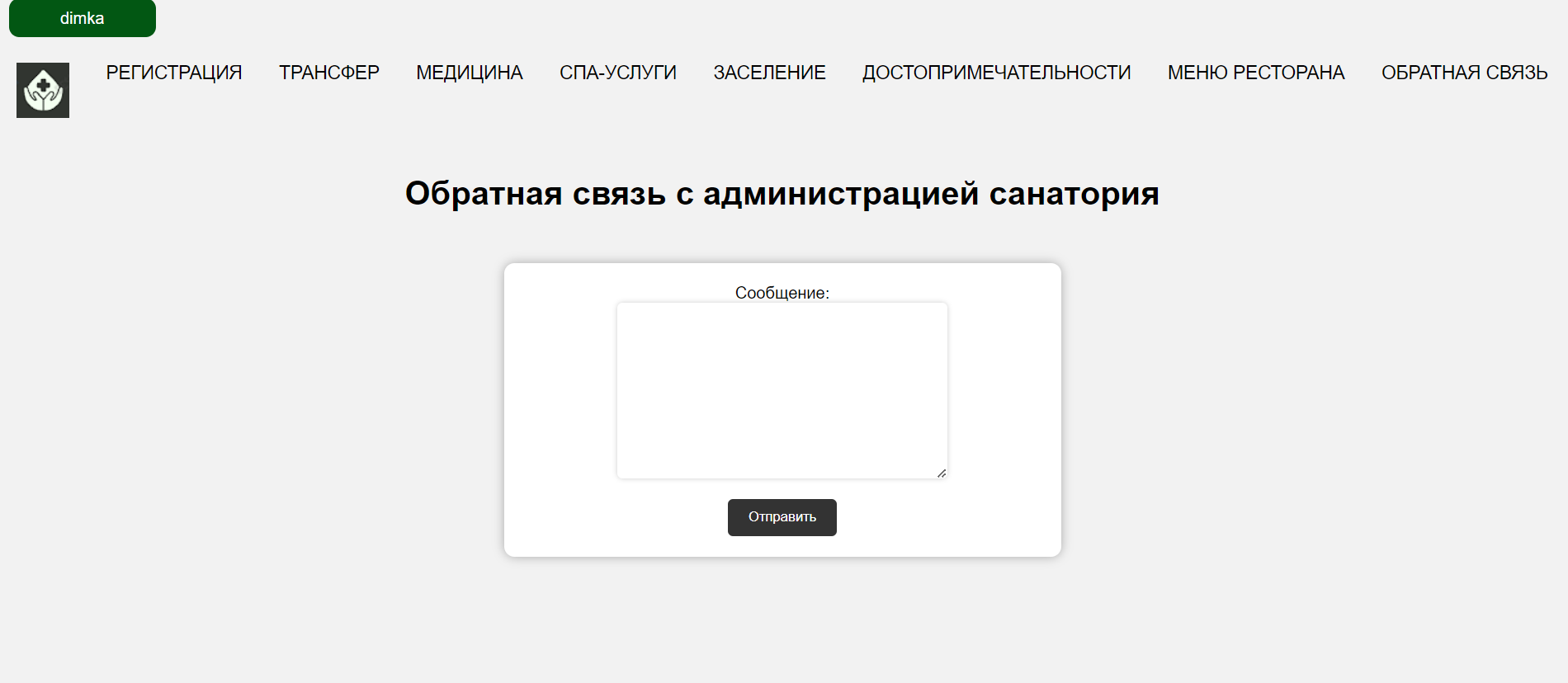


Рисунок 4.7 – Страница «Обратная связь»

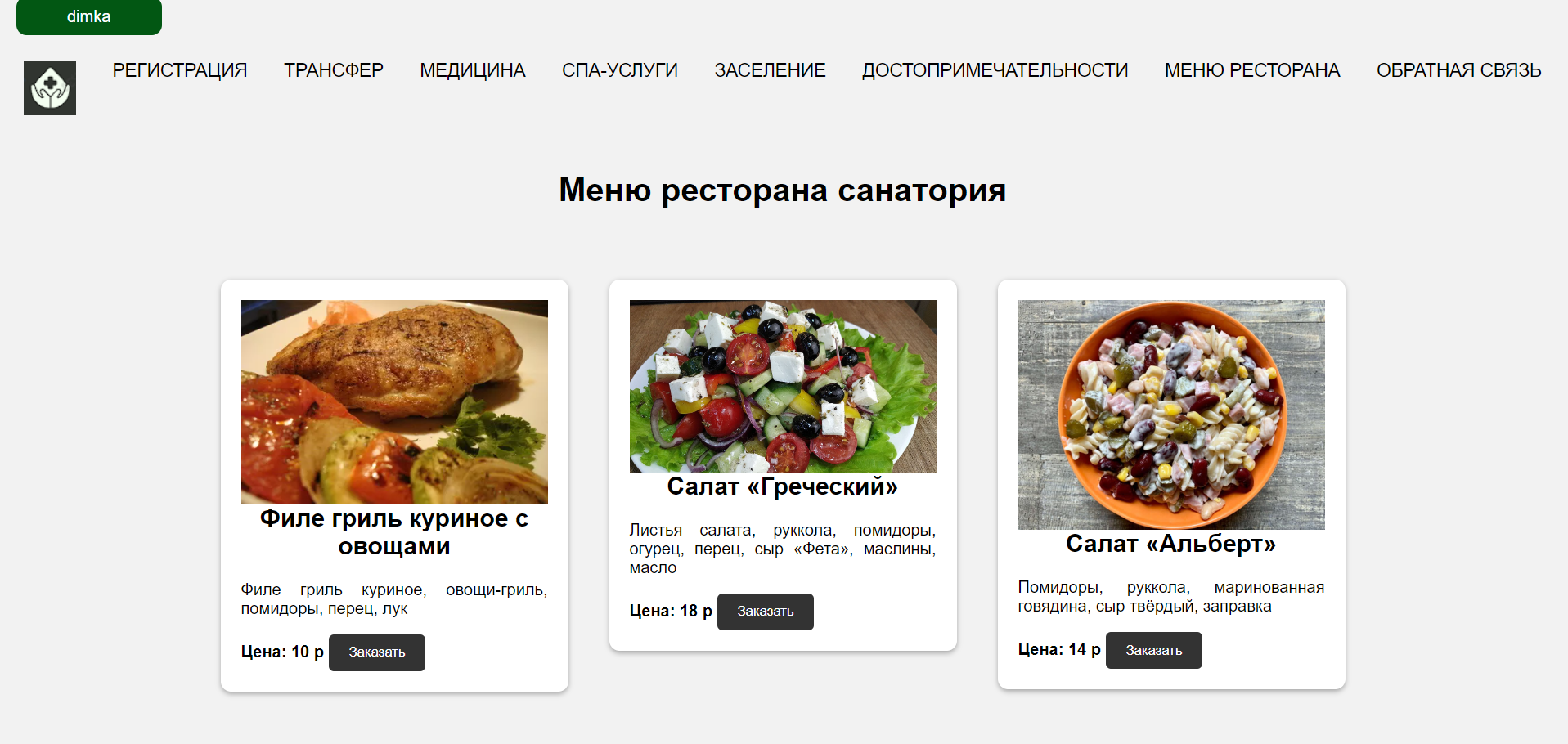
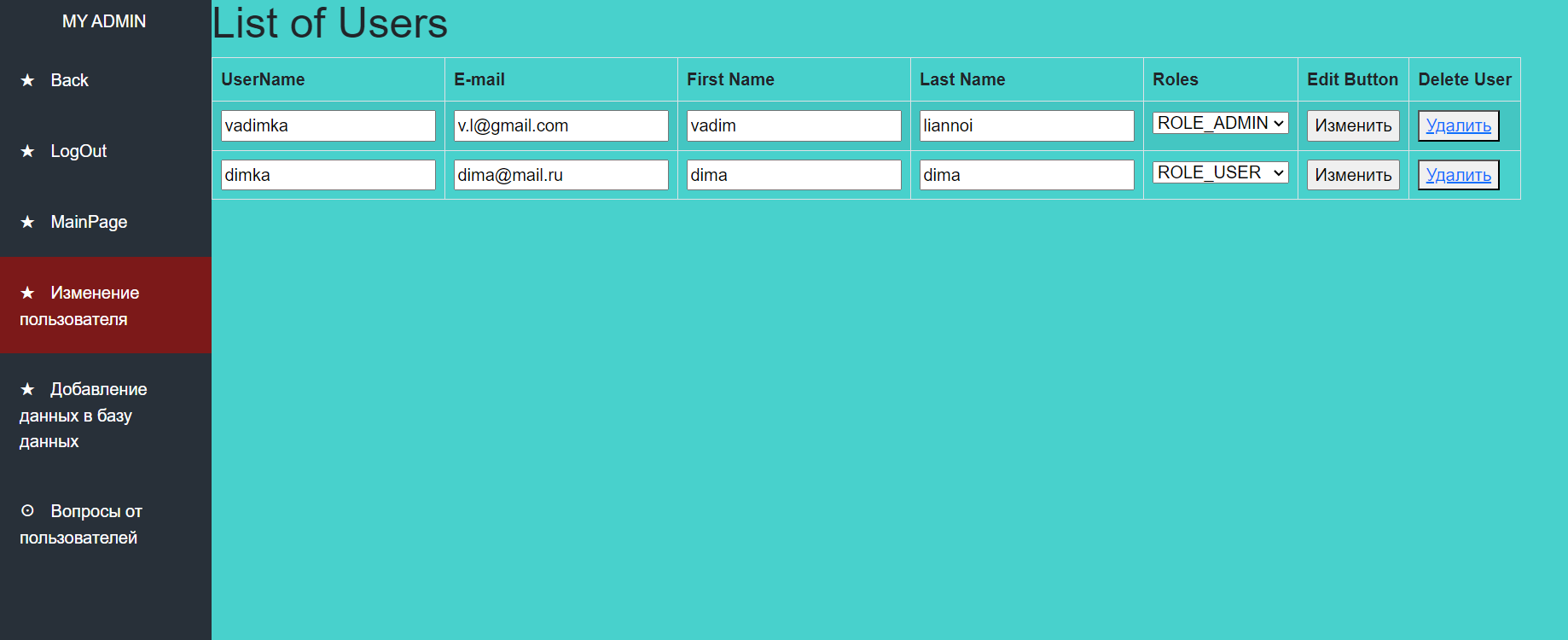


Рисунок 4.8 – Страница «Меню ресторана»

При авторизации под ролью администратора доступны все функции, которые есть у пользователя, а также переход на страницу управления пользователями, где присутствует добавление, удаление позиций меню ресторана, лечебных процедур, достопримечательностей.

Рисунок 4.9 – Часть функционала администратора «список пользователей»

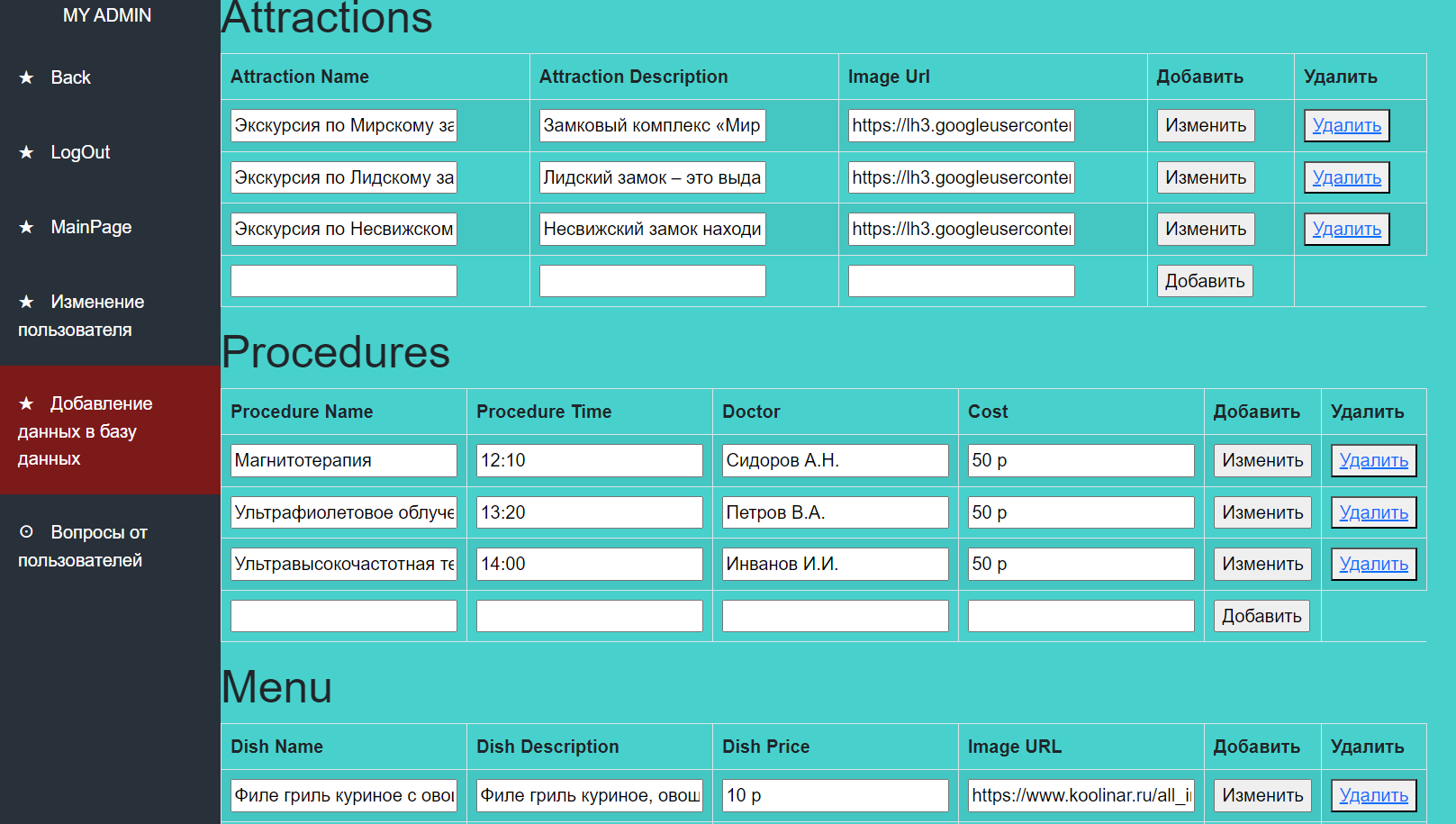


Рисунок 4.10 – Часть функционала администратора «Редактирование информации на сайте»

В главе был описан функционал для авторизованного пользователя как администратора, так и рядового посетителя.

# **ЗАКЛЮЧЕНИЕ**

Целью данной работы являлось создание приложения для автоматизации работы санатория. Стояла необходимость в создании механизмов, которые дадут возможность снизить трудозатраты, сократить временные расходы и повысить комфорт условий труда медицинского персонала учреждений. Для выполнения данной задачи требовалось исследовать и определить все пользовательские требования. Отталкиваясь от них, формировались функциональные требования, на основе которых проходила разработка программного продукта.

После окончания анализа требований мы приступили к проектированию и моделированию основных бизнес-процессов учреждения, в нотации IDEF0. Поскольку создание приложения происходило в среде intelij IDEA, требовалась классификация и нормализация данных, и моделирование их архитектуры. Создание пользовательских интерфейсов позволило представить, как будет выглядеть приложение в конце. Целью создания интерфейса являлось обеспечение возможности обновления представленных данных по деятельности санатория. Создание блок-схемы действия алгоритма программы оказалось удобным средством описания, демонстрирующее наглядный принцип работы ключевого бизнес-процесса с помощью геометрических фигур с линиями-связями.

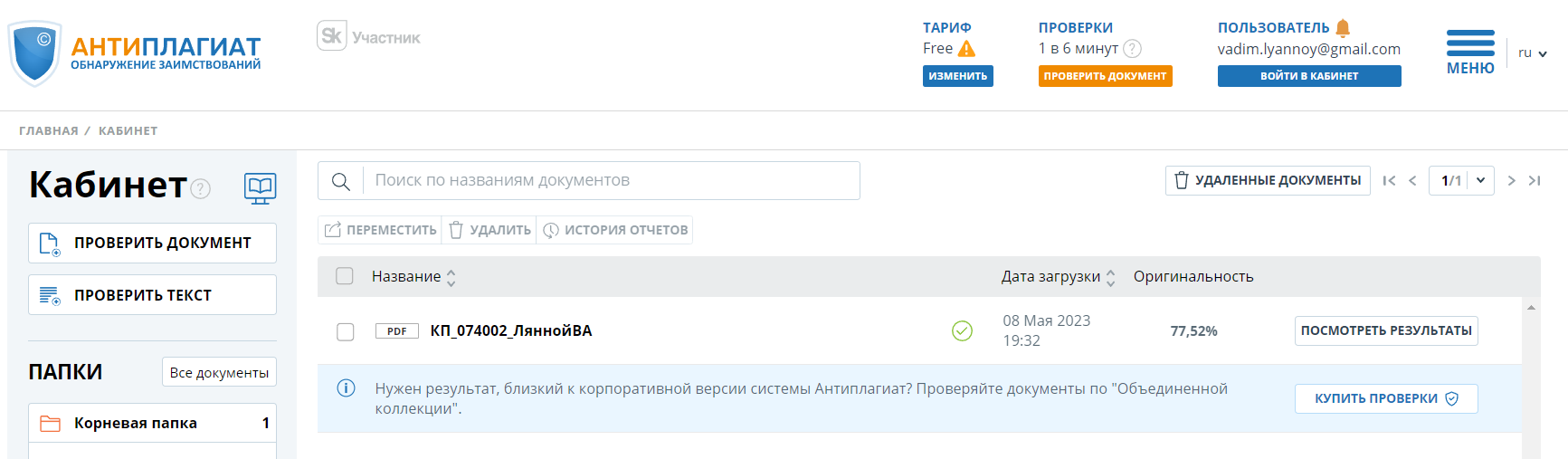
Заключительным этапом созданного приложения являлось тестирование. В ходе прохождения тестов при различном количестве записей в БД, было выявлено, что приложение отличается высокой производительностью.

Подведя итоги, можно сказать, разработанное программное средство отвечает всем требованиям функциональности, так как выполняет все возложенные на нее функции и требования надежности.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Свободная энциклопедия Википедия, статья «Анализ требований» [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Анализ\_требований
2. Бизнес-процессы – разработка, описание и управление [Электронный ресурс]. – Режим доступа: https://first-expert.ru/upravlenie-biznes-processami.
3. Спецификация требований программного обеспечения [Электронный ресурс]. – Режим доступа: http://unetway.com/tutorial/specifikacia-trebovanij-programmnogo-obespecenia.
4. Руководство по HTML [Электронный ресурс]. – Режим доступа: https://metanit.com/web/html5.
5. Руководство по CSS [Электронный ресурс]. – Режим доступа: https://metanit.com/web/CSS
6. Общая характеристика деятельности санатория [Электронный ресурс]. – Режим доступа: <https://studwood.net/1187182/pravo/obschaya_harakteristika_sanatoriya>
7. Введение в Spring Boot: создание простого REST API на Java [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/435144/>
8. PostgreSQL: Документация [Электронный ресурс]. – Режим доступа: <https://postgrespro.ru/docs/postgresql>
9. Стандарт IDEF0: примеры и правила построения [Электронный ресурс]. – Режим доступа: [https://technolakpiter.ru/standart-idef0-primery-idef0- diagramma-primery-i-pravila-postroeniya/](https://technolakpiter.ru/standart-idef0-primery-idef0-%20diagramma-primery-i-pravila-postroeniya/)

# **Приложение А (обязательное) - Отчет о проверке на заимствования в системе «Антиплагиат»**



# **Приложение Б (обязательное) - Листинг кода алгоритмов, реализующих бизнес-логику**

Класс Securityconfig:

package ru.vadim.spingapp.FirstSecurityApp.config;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.builders.WebSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import ru.vadim.spingapp.FirstSecurityApp.controllers.NewCustomAuthenticationSuccessHandler;

import ru.vadim.spingapp.FirstSecurityApp.services.PersonDetailsService;

@EnableWebSecurity

@EnableGlobalMethodSecurity(prePostEnabled = true)

public class SecurityConfig extends WebSecurityConfigurerAdapter {

private final PersonDetailsService personDetailsService;

@Autowired

public SecurityConfig(PersonDetailsService personDetailsService) {

this.personDetailsService = personDetailsService;

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests()

.antMatchers("/admin/admin","/admin/AdminRest").hasRole("ADMIN")

.antMatchers("/auth/login", "/auth/registration",

"/error","/auth/index","auth/homePage","/logout").permitAll()

.antMatchers("/auth/\*\*","/auth/sights/\*\*","/auth/medicine/\*\*","/user/\*\*").authenticated()

.anyRequest().hasAnyRole("USER", "ADMIN")

.and()

.formLogin().loginPage("/auth/index")

.loginProcessingUrl("/process\_login")

.defaultSuccessUrl("/auth/index", true)

.successHandler(new NewCustomAuthenticationSuccessHandler())

.failureUrl("/auth/login?error")

.and()

.logout()

.logoutUrl("/logout")

.logoutSuccessUrl("/auth/login");

}

// Настраиваем аутентификацию

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(personDetailsService)

.passwordEncoder(getPasswordEncoder());

}

@Override

public void configure(WebSecurity web) throws Exception {

web.ignoring().antMatchers("/css/\*\*","/images/\*\*");

}

@Bean

public PasswordEncoder getPasswordEncoder() {

return new BCryptPasswordEncoder();

}

}

Класс AdminController:

package ru.vadim.spingapp.FirstSecurityApp.controllers;

import org.apache.tomcat.jni.Proc;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.\*;

import ru.vadim.spingapp.FirstSecurityApp.models.\*;

import ru.vadim.spingapp.FirstSecurityApp.repositories.\*;

import ru.vadim.spingapp.FirstSecurityApp.services.\*;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.util.ArrayList;

import java.util.List;

@Controller

public class AdminController {

List<Person> listPersons;

List<Menu> listMenu;

@Autowired

PeopleRepository peopleRepository;

@Autowired

PersonDetailsService personDetailsService;

@Autowired

CommunicationDetailService communicationDetailService;

@Autowired

CommunicationPepository communicationPepository;

@Autowired

AttractionDetailService attractionDetailService;

@Autowired

AttractionRepository attractionRepository;

@Autowired

MenuDetailsService menuDetailsService;

@Autowired

MenuRepository menuRepository;

@Autowired

ProcedureDetailService procedureDetailService;

@Autowired

ProcedureRepository procedureRepository;

@RequestMapping(value="/logout", method = RequestMethod.GET)

public String logoutPage (HttpServletRequest request, HttpServletResponse response) {

Authentication auth = SecurityContextHolder.getContext().getAuthentication();

if (auth != null){

new SecurityContextLogoutHandler().logout(request, response, auth);

}

return "redirect:/login?logout";

}

@GetMapping("/admin/admin")

public String userList(Model model)

{

listPersons = personDetailsService.listAll();

model.addAttribute("users",listPersons);

return "/admin/admin";

}

@PostMapping("/admin/admin/{id}")

public String userEdit(@PathVariable("id") int id,@ModelAttribute("person") Person newPerson)

{

Person person = personDetailsService.get(id);

person.setName(newPerson.getName());

person.setLast\_name(newPerson.getLast\_name());

person.setRole(newPerson.getRole());

person.setEmail(newPerson.getEmail());

person.setUsername(newPerson.getUsername());

peopleRepository.save(person);

return "redirect:/admin/admin";

}

/\* @GetMapping("/admin/menuOut/{id}")

public String menuUserList(@PathVariable("id") int id, Model model){

Person person = personDetailsService.get(id);

listMenu = person.getMenu();

model.addAttribute("listMenu",listMenu);

return "/admin/adminRest";

}\*/

@GetMapping("/admin/deleteUser/{id}")

public String deleteUser(@PathVariable int id) {

personDetailsService.deletePersonById(id);

return "redirect:/admin/admin";

}

@GetMapping("/admin/adminAnswer")

public String adminAnswer(Model model)

{

List<Communication> communicationList = communicationPepository.findAll();

model.addAttribute("communicationList",communicationList);

return "/admin/adminAnswer";

}

@PostMapping("/admin/adminAnswer/{id}")

public String adminPostAnswer(@PathVariable int id,@ModelAttribute("communication") Communication communication)

{

Communication communication1 = communicationDetailService.get(id);

communication1.setAnswer(communication.getAnswer());

communicationPepository.save(communication1);

return "redirect:/admin/adminAnswer";

}

@GetMapping("/admin/adminAddData")

public String adminAddData(Model model)

{

List<Attraction> attractionList = attractionDetailService.listAll();

Attraction newAttraction = new Attraction();

List<Procedure> procedureList = procedureDetailService.listAll();

Procedure newProcedure = new Procedure();

List<Menu> menuList = menuDetailsService.listAll();

Menu newMenu = new Menu();

model.addAttribute("menuList",menuList);

model.addAttribute("men",newMenu);

model.addAttribute("procedureList",procedureList);

model.addAttribute("proc",newProcedure);

model.addAttribute("attract",newAttraction);

model.addAttribute("attractionList",attractionList);

return "/admin/adminAddData";

}

@PostMapping("/admin/adminUpdateData/{id}")

public String userUpdateData(@PathVariable("id") int id,@ModelAttribute("attraction") Attraction newAttraction)

{

Attraction attraction = attractionDetailService.get(id);

attraction.setAttraction\_name(newAttraction.getAttraction\_name());

attraction.setAttraction\_description(newAttraction.getAttraction\_description());

attraction.setImage\_url(newAttraction.getImage\_url());

attractionRepository.save(attraction);

return "redirect:/admin/adminAddData";

}

@PostMapping("/admin/adminUpdateDataProcedure/{id}")

public String adminUpdateDataProcedure(@PathVariable("id") int id,@ModelAttribute("procedure") Procedure newProcedure)

{

Procedure procedure = procedureDetailService.get(id);

procedure.setLocation\_from(newProcedure.getLocation\_from());

procedure.setProcedure\_time(newProcedure.getProcedure\_time());

procedure.setProcedure\_cost(newProcedure.getProcedure\_cost());

procedure.setDoctor(newProcedure.getDoctor());

procedureRepository.save(procedure);

return "redirect:/admin/adminAddData";

}

@GetMapping("/admin/deleteEntity/{id}")

public String deleteEntity(@PathVariable int id) {

attractionDetailService.deleteAttractionByID(id);

return "redirect:/admin/adminAddData";

}

@GetMapping("/admin/deleteProcedure/{id}")

public String deleteProcedure(@PathVariable int id) {

procedureDetailService.deleteProcedureByID(id);

return "redirect:/admin/adminAddData";

}

@PostMapping("/admin/adminAddData")

public String userAddData(@ModelAttribute("attract") Attraction newAttraction)

{

attractionRepository.save(newAttraction);

return "redirect:/admin/admin";

}

@PostMapping("/admin/adminUpdateDataProcedure")

public String adminUpdateDataProcedure(@ModelAttribute("proc") Procedure newProcedure)

{

procedureRepository.save(newProcedure);

return "redirect:/admin/admin";

}

@PostMapping("/admin/adminUpdateDataMenu/{id}")

public String adminUpdateDataMenu(@PathVariable("id") int id,@ModelAttribute("menu") Menu newMenu)

{

Menu menu = menuDetailsService.get(id);

menu.setDish\_name(newMenu.getDish\_name());

menu.setDish\_description(newMenu.getDish\_description());

menu.setDish\_price(newMenu.getDish\_price());

menu.setImage\_url(newMenu.getImage\_url());

menuRepository.save(menu);

return "redirect:/admin/adminAddData";

}

@PostMapping("/admin/adminUpdateDataMenu")

public String adminUpdateMenu(@ModelAttribute("men") Menu newMenu)

{

menuRepository.save(newMenu);

return "redirect:/admin/adminAddData";

}

@GetMapping("/admin/deleteMenu/{id}")

public String deleteMenu(@PathVariable int id) {

menuDetailsService.deleteMenuById(id);

return "redirect:/admin/adminAddData";

}

}

Класс AuthController:

package ru.vadim.spingapp.FirstSecurityApp.controllers;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import ru.vadim.spingapp.FirstSecurityApp.models.Person;

import ru.vadim.spingapp.FirstSecurityApp.repositories.PeopleRepository;

import ru.vadim.spingapp.FirstSecurityApp.services.RegistrationService;

import ru.vadim.spingapp.FirstSecurityApp.util.PersonValidator;

import javax.validation.Valid;

@Controller

@RequestMapping("/auth")

public class AuthController {

PeopleRepository peopleRepository;

private final RegistrationService registrationService;

private final PersonValidator personValidator;

@Autowired

public AuthController(RegistrationService registrationService, PersonValidator personValidator) {

this.registrationService = registrationService;

this.personValidator = personValidator;

}

@GetMapping("/login")

public String loginPage(){

return "auth/login";

}

@GetMapping("/index")

public String indexPage() {

return "auth/index";

}

@GetMapping("/registration")

public String registrationPage(@ModelAttribute("person") Person person) {

return "auth/registration";

}

@PostMapping("/registration")

public String performRegistration(@ModelAttribute("person") @Valid Person person,

BindingResult bindingResult) {

personValidator.validate(person, bindingResult);

if (bindingResult.hasErrors())

return "/auth/registration";

registrationService.register(person);

return "redirect:/auth/login";

}

}

Класс UserController:

package ru.vadim.spingapp.FirstSecurityApp.controllers;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.\*;

import ru.vadim.spingapp.FirstSecurityApp.services.AttractionDetailService;

import ru.vadim.spingapp.FirstSecurityApp.services.MenuDetailsService;

import ru.vadim.spingapp.FirstSecurityApp.services.PersonDetailsService;

import ru.vadim.spingapp.FirstSecurityApp.services.ProcedureDetailService;

import ru.vadim.spingapp.FirstSecurityApp.models.\*;

import ru.vadim.spingapp.FirstSecurityApp.repositories.\*;

import javax.persistence.EntityExistsException;

import javax.validation.Valid;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

@Controller

public class UserController {

@Autowired

PersonDetailsService personDetailsService;

List<Seance> newSeance = new ArrayList<>();

List<Attraction> newAttractions = new ArrayList<>();

List<Menu> newMenu = new ArrayList<>();

List<Person> newPerson = new ArrayList<>();

@Autowired

MenuDetailsService menuDetailsService;

@Autowired

ProcedureDetailService procedureDetailService;

@Autowired

AttractionDetailService attractionDetailService;

@Autowired

AttractionRepository attractionRepository;

@Autowired

MenuRepository menuRepository;

@Autowired

PeopleRepository peopleRepository;

@Autowired

ProcedureRepository procedureRepository;

@Autowired

SeanceRepository seanceRepository;

@Autowired

CommunicationPepository communicationPepository;

@GetMapping("auth/homePage")

public String redirection(){

Person person = peopleRepository.findByUsername(getCurrentUserName());

if(person.getRole().equals("ROLE\_ADMIN")){

return "redirect:/admin/admin";

}

else

return "redirect:/user/userPage";

}

@GetMapping( "/auth/transfer" )

public String TransferPage(@ModelAttribute("transfer") Transfer transfer, Model model) {

Person person = peopleRepository.findByUsername(getCurrentUserName());

Transfer tr = person.getTransfer();

model.addAttribute("Transfer",tr);

return "/auth/transfer";

}

@PostMapping("/auth/{id}")

public String addNewDish(@PathVariable("id") int id) {

Menu menu = menuDetailsService.get(id);

Person person = peopleRepository.findByUsername(getCurrentUserName());

newMenu.add(menu);

newPerson.add(person);

person.setMenu(newMenu);

menu.setPersons(newPerson);

menuRepository.save(menu);

peopleRepository.save(person);

return "redirect:/auth/food";

}

@PostMapping("/auth/transfer")

public String postTransfer(@ModelAttribute("transfer") @Valid Transfer transfer) throws EntityExistsException{

Person person = peopleRepository.findByUsername(getCurrentUserName());

if(person.getTransfer() == null) {

transfer.setPerson(person);

person.setTransfer(transfer);

peopleRepository.save(person);

}

return "/auth/transfer";

}

public String getCurrentUserName(){

Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

return authentication.getName();

}

@GetMapping("/auth/food")

public String menuList(Model model)

{

List<Menu> menuList = menuDetailsService.listAll();

model.addAttribute("menuList",menuList);

return "/auth/food";

}

@GetMapping("/auth/sights")

public String sightsList(Model model)

{

List<Attraction> attractionList = attractionDetailService.listAll();

model.addAttribute("attractionList",attractionList);

return "/auth/sights";

}

@PostMapping("/auth/sights/{id}")

public String addNewAttractionToPerson(@PathVariable("id") int id) {

Attraction attraction= attractionDetailService.get(id);

Person person = peopleRepository.findByUsername(getCurrentUserName());

newAttractions.add(attraction);

newPerson.add(person);

person.setAttractions(newAttractions);

attraction.setPersons(newPerson);

attractionRepository.save(attraction);

peopleRepository.save(person);

return "redirect:/auth/sights";

}

@GetMapping( "/auth/rooms" )

public String SettlementPage(@ModelAttribute("settlement") Settlement settlement, Model model) {

Person person = peopleRepository.findByUsername(getCurrentUserName());

Settlement st = person.getSettlement();

model.addAttribute("Settlement",st);

return "/auth/rooms";

}

@PostMapping("/auth/settlement")

public String postSettlementPage(@ModelAttribute("settlement") @Valid Settlement settlement) throws EntityExistsException{

Person person = peopleRepository.findByUsername(getCurrentUserName());

if(person.getSettlement() == null) {

settlement.setPerson(person);

person.setSettlement(settlement);

peopleRepository.save(person);

}

return "/auth/rooms";

}

@PostMapping("/auth/medicine")

public String postMedicinePage(@ModelAttribute("seance") @Valid Seance seance) {

Person person = peopleRepository.findByUsername(getCurrentUserName());

seance.setPerson(person);

Set<Seance> sc = new HashSet<>();

sc.add(seance);

newSeance.add(seance);

if(seance.getProcedure().isIsavaliable()) {

// seance.getProcedure().setIsavaliable(false);

person.setSeances(sc);

seance.getProcedure().setSeances(newSeance);

procedureRepository.save(seance.getProcedure());

peopleRepository.save(person);

seanceRepository.save(seance);

}

return "redirect:/auth/medicine";

}

@GetMapping("/auth/medicine")

public String doctorsList(@ModelAttribute("seance") Seance seance,Model model)

{

List<Procedure> procedureList = procedureDetailService.listAll();

model.addAttribute("procedureList",procedureList);

return "/auth/medicine";

}

public boolean findUsingEnhancedForLoop(String name, List<Seance> seance) {

for (Seance seance1: seance) {

if (seance1.getSeance\_time().equals(name)) {

return false;

}

}

return true;

}

@GetMapping("/auth/questions")

public String communication(Model model)

{

Person person = peopleRepository.findByUsername(getCurrentUserName());

Communication communication = person.getCommunication();

model.addAttribute("communication",communication);

return "/auth/questions";

}

@PostMapping("/auth/questions")

public String postCommunication(@ModelAttribute("communication") @Valid Communication communication) throws EntityExistsException{

Person person = peopleRepository.findByUsername(getCurrentUserName());

if(person.getCommunication() == null){

communication.setPerson(person);

person.setCommunication(communication);

communicationPepository.save(communication);

}

return "/auth/questions";

}

@GetMapping("/auth/spa")

public String spaGet(Model model)

{

Person person = peopleRepository.findByUsername(getCurrentUserName());

Spa spa = person.getSpa();

model.addAttribute("spa",spa);

return "/auth/spa";

}

@PostMapping("/auth/spa")

public String spaPostPage(@ModelAttribute("spa") @Valid Spa spa) throws EntityExistsException{

Person person = peopleRepository.findByUsername(getCurrentUserName());

if(person.getSpa() == null){

spa.setPerson(person);

person.setSpa(spa);

peopleRepository.save(person);

}

return "redirect:/auth/spa";

}

@GetMapping("/user/userPage")

public String userPageProfile(Model model)

{

Person person = peopleRepository.findByUsername(getCurrentUserName());

model.addAttribute("person",person);

return "/user/userPage";

}

@GetMapping("/user/userPageTransfer")

public String userPageTransfer(Model model)

{

Person person = peopleRepository.findByUsername(getCurrentUserName());

Transfer transfer = person.getTransfer();

model.addAttribute("transfer",transfer);

return "/user/userPageTransfer";

}

@GetMapping("/user/userPageMedicine")

public String userPageMedicine(Model model)

{

Person person = peopleRepository.findByUsername(getCurrentUserName());

Set<Seance> seances = person.getSeances();

model.addAttribute("seances",seances);

return "/user/userPageMedicine";

}

@GetMapping("/user/userPageSpa")

public String userPageSpa(Model model)

{

Person person = peopleRepository.findByUsername(getCurrentUserName());

Spa spa = person.getSpa();

model.addAttribute("spa",spa);

return "/user/userPageSpa";

}

@GetMapping("/user/userPageRooms")

public String userPageRooms(Model model)

{

Person person = peopleRepository.findByUsername(getCurrentUserName());

Settlement settlement = person.getSettlement();

model.addAttribute("settlement",settlement);

return "/user/userPageRooms";

}

@GetMapping("/user/userPageAttraction")

public String userPageAttraction(Model model)

{

Person person = peopleRepository.findByUsername(getCurrentUserName());

List<Attraction> attraction = person.getAttractions();

model.addAttribute("attractionList",attraction);

return "/user/userPageAttraction";

}

@GetMapping("/user/userPageRestourant")

public String userPageRestaurant(Model model)

{

Person person = peopleRepository.findByUsername(getCurrentUserName());

List<Menu> menu = person.getMenu();

model.addAttribute("menuList",menu);

return "/user/userPageRestourant";

}

@GetMapping("/user/userPageQuestion")

public String userPageQuestion(Model model)

{

Person person = peopleRepository.findByUsername(getCurrentUserName());

Communication communication = person.getCommunication();

model.addAttribute("communication",communication);

return "/user/userPageQuestion";

}

}

# **Приложение В (обязательное) - Листинг скрипта генерации базы данных**

create table attraction (user\_id int4 not null, attraction\_id int4 not null)

create table attractionslist (id serial not null, attraction\_description varchar(255), attraction\_name varchar(255), image\_url varchar(255), primary key (id))

create table communication (id serial not null, answer varchar(255), question varchar(255), user\_id int4, primary key (id))

create table medical\_procedures\_list (id serial not null, doctor varchar(255), isavaliable boolean, procedure\_name varchar(255), procedure\_cost varchar(255), procedure\_time varchar(255), primary key (id))

create table menu (id serial not null, dish\_description varchar(255), dish\_name varchar(255), dish\_price int4, image\_url varchar(255), primary key (id))

create table person (id serial not null, email varchar(100), last\_name varchar(100), name varchar(100), password varchar(255), role varchar(255), username varchar(100), primary key (id))

create table restaurant (user\_id int4 not null, dish\_id int4 not null)

create table seance (id serial not null, seance\_date varchar(255), seance\_time varchar(255), user\_id int4, procedure\_id int4, primary key (id))

create table settlement (user\_id int4 not null, arrival\_date varchar(255), arrival\_time varchar(255), numberofadults varchar(255), numberofchildrens varchar(255), type varchar(255), primary key (user\_id))

create table spa (id serial not null, date varchar(255), telephone varchar(255), time varchar(255), type varchar(255), user\_id int4, primary key (id))

create table transfer (user\_id int4 not null, location\_from varchar(255), location\_to varchar(255), phone\_number varchar(255), primary key (user\_id))

alter table attraction add constraint FK935bi7n59kx4v5uj13hntq8f4 foreign key (attraction\_id) references attractionslist

alter table attraction add constraint FKqn7x4r41k3nf2e14wuw4jyv0j foreign key (user\_id) references person

alter table communication add constraint FK77y8wj38dhho0xsatcsw2rqip foreign key (user\_id) references person

alter table restaurant add constraint FK4ts6q9i59prn17m2k7969b0rl foreign key (dish\_id) references menu

alter table restaurant add constraint FKtljtoyc2kv34xvb2s8cl8g2ug foreign key (user\_id) references person

alter table seance add constraint FK4px9cuiko1yci1okg2oh8pjjo foreign key (user\_id) references person

alter table seance add constraint FKh1lbeup18718x1c3nu48eg8cs foreign key (procedure\_id) references medical\_procedures\_list

alter table settlement add constraint FKnp51xxouvds1kd7ujjvk0876w foreign key (user\_id) references person

alter table spa add constraint FKrxce4h7rjs0qj4aqa5a9jhhsa foreign key (user\_id) references person

alter table transfer add constraint FK748wmnjw1tfk36wuuot64fahg foreign key (user\_id) references person