## 4 РЕАЛИЗАЦИЯ

**4.1 Разработка классов информационной системы**

Так как мы используем шаблон MVC, идет отдельно разработка классов модели (сущности), отдельно вида (страниц для браузера) и контроллеров (обработки действий пользователя).

Так как данные в шаблоне будут изменяться по требованию администратора ресурса, нам нужно создать класс-сущность для хранения этих данных.

Сущность (entity) – это реальный или представляемый тип объекта, информация о котором должна сохраняться и быть доступна.

Сущности уже были спроектированы раньше для базы данных. На их основе пишутся классы сервиса.

Пример описания сущности приведен ниже.

Листинг 4.1 - Сущность дневной нагрузки

package com.hescha.trainingdaily.model;

import lombok.Data;

import org.jetbrains.annotations.NotNull;

import javax.persistence.ElementCollection;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

import javax.persistence.MapKeyColumn;

import javax.persistence.OneToMany;

import java.sql.Date;

import java.util.\*;

@Entity

@Data

public class DailyRoutine extends AbstractEntity implements Comparable<DailyRoutine> {

@ManyToOne

@JoinColumn(name = "user")

private User user;

private Date date;

@ElementCollection(targetClass=Integer.class)

@MapKeyColumn(name="Exercise")

private Map<Exercise, Integer> exercises = new HashMap<>();

@OneToMany(mappedBy = "dailyRoutine", fetch = FetchType.LAZY)

private Set<MealForDay> mealForDays = new TreeSet<>();

public float getAllExercisesKkal() {

double kkal = exercises.entrySet()

.stream()

.mapToDouble(exercise -> exercise.getKey().getKkalInMinute() \* exercise.getValue().intValue())

.sum();

return new Float(kkal).shortValue();

}

public float getAllProductsKkal() {

double kkal = mealForDays

.stream()

.mapToDouble(meal->meal.getAllKkalInTime())

.sum();

return new Float(kkal).shortValue();

}

@Override

public int compareTo(@NotNull DailyRoutine o) {

return date.compareTo(o.getDate());

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

if (!super.equals(o)) return false;

DailyRoutine that = (DailyRoutine) o;

return Objects.equals(user.getUsername().hashCode(), that.user.getUsername().hashCode())

&& Objects.equals(date, that.date)

&& Objects.equals(exercises, that.exercises)

&& Objects.equals(mealForDays, that.mealForDays);

}

@Override

public int hashCode() {

return Objects.hash(super.hashCode(), user.getUsername().hashCode(), date, exercises, mealForDays);

}

}

По описанию сущности автоматически будет сгенерирована таблица в базе данных, при ее отсутствии.

Сущность написана, переходим к созданию сервисов для доступа к бд. Сперва опишем интерфейс с основными командами.



Рисунок 4.1 – Интерфейс с основными командами

Затем переходим к написанию реализации этого интерфейса. На самом деле, реализация простая, мы вызываем соответствующие методы репозитория.

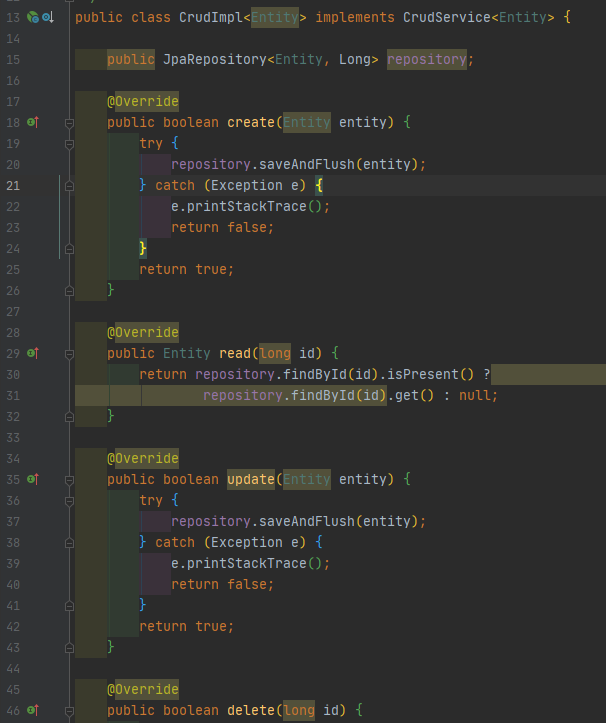


Рисунок 4.2 – Реализация интерфейса

Реализация есть, осталось связать нашу сущность шаблона с доступом к бд. Для этого наследуемся от только-что созданной реализации, и в конструктор передаем нашу сущность.

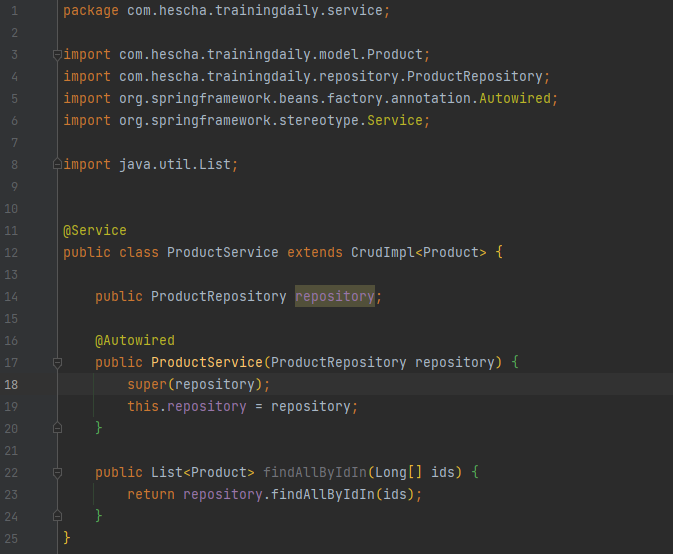


Рисунок 4.3 – Сервис доступа к таблице

Проделываем так для каждой сущности и готово, теперь, используя данный сервис, мы можем добавлять/редактировать/удалять данные в коде приложения.

Но это только в коде, пользователю это пока недоступно. Чтобы пользователь смог как-то взаимодействовать с приложением, создаем контроллеры страниц.

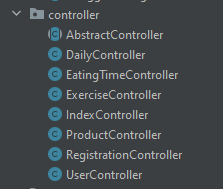


Рисунок 4.4 – Контроллеры страниц

По одному контроллеру для работы с соответствующими таблицами в базе данных.

Контроллер работает с данными таблиц базы данных, считывая или записывая в нее данные от пользователя. При запросе отображения, он возвращает страницу отображения. В этом контроллере реализованы методы создания, редактирования и удаления шаблонов по соответствующим адресам.

Листинг 4.2 - Контроллер пользователей

package com.hescha.trainingdaily.controller;

import com.hescha.trainingdaily.model.User;

import com.hescha.trainingdaily.service.UserService;

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

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

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

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

import java.util.List;

@Controller

@RequestMapping("/users")

public class UserController {

@Autowired

private UserService service;

@Autowired

private PasswordEncoder passwordEncoder;

@GetMapping

String get(Model model) {

List<User> list = service.findAll();

model.addAttribute("list", list);

return "users";

}

@GetMapping(path = {"/edit", "/edit/{id}"})

public String edit(Model model, @PathVariable(name = "id", required =

false) Long id) {

if (id != null) {

User entity = service.read(id);

model.addAttribute("entity", entity);

} else {

model.addAttribute("entity", new User());

}

return "users-one";

}

@PostMapping(path = "/create")

public String createOrUpdate(User entity) {

if (entity.getId() == null)

service.registerNew(entity);

else {

User read = service.read(entity.getId());

read.setFio(entity.getFio());

read.setEmail(entity.getEmail());

read.setDateBorn(entity.getDateBorn());

read.setPassword(passwordEncoder.encode(entity.getPassword()));

service.update(read);

}

return "redirect:/users";

}

@GetMapping(path = "/delete/{id}")

public String delete(@PathVariable("id") Long id) {

service.delete(id);

return "redirect:/users";

}

}

Класс наследует стандартный класс настройки безопасности. В нем мы сразу создаем данные для входа администратора и в методе конфигурации прописываем, какие пользователи куда могут попасть.

Поскольку использованные шаблоны нужно где-то хранить, создаем настройку подключения к базе данных. Все это делается в файле application.properties.

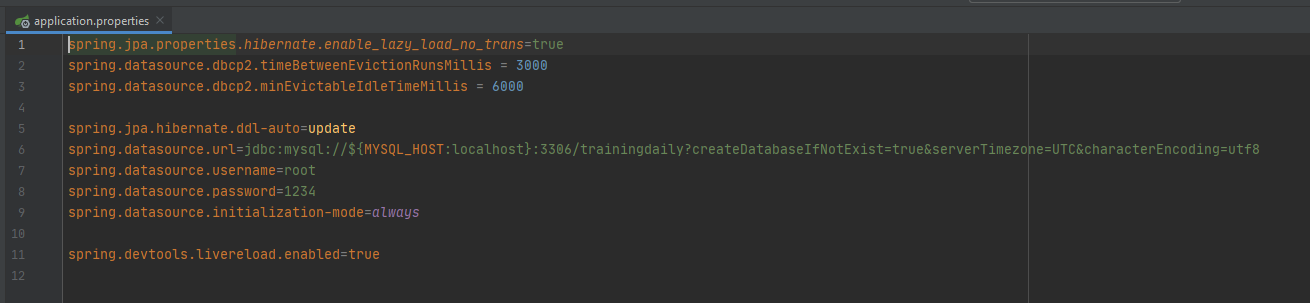


Рисунок 4.5 – Файл настройщик приложений

Проделав все эти операции, мы имеем следующую диаграмму классов



Рисунок 4.6 – Диаграмма классов

## 4.2 Разработка интерфейса программного продукта

Остался последний шаг – связывание страниц шаблонов с данными приложения. Для этого воспользуемся шаблонизатором Thymeleaf, который работает в связке с Spring.

Суть его проста – те данные, что мы передавали в модель отображения, таймлиф можем выводить простой командой th:text=”${}”. В скобках пишется имя переменной. Обращение идет как к переменным класса. В нужных местах вставляем этот код.

Так, для отображения списка тарифов будут использоваться строки листинга 4.3.

Листинг 4.3 - Шаблон для отображения доступных упражнений

<section class="coaches-area section-padding">

<div class="container">

<div class="row">

<div class="col-lg-12">

<div class="section-top text-center">

<h3>Доступные активности</h3>

</div>

</div>

</div>

<div class="row">

<div class="col-lg-12">

<div th:switch="${list}" class="container my-5">

<div class="col-md-12">

<h2 th:case="null"> Записей не найдено </h2>

<div th:case="\*" class="overScroll">

<button class="accordion">Показать все</button>

<table class="table table-striped table-responsive-md panel">

<thead>

<tr>

<th>Активность</th>

<th>Краткое описание</th>

<th>Затраты ккал в час</th>

</tr>

</thead>

<tbody>

<tr th:each="entity : ${list}"

th:if="${entity.approved == true}">

<td th:text="${entity.name}"></td>

<td th:text="${entity.description}"></td>

<td th:text="${entity.kkalInHour}"></td>

<td class="btnInTable widthLastCol" th:if="${#request.isUserInRole('ROLE\_ADMIN')}">

<a th:href="@{/exercises/edit/{id}(id=${entity.id})}" class="btn btn-primary">

Редактировать

</a>

</td>

<td class="btnInTable widthLastCol" th:if="${#request.isUserInRole('ROLE\_ADMIN')}">

<a th:href="@{/exercises/delete/{id}(id=${entity.id})}" class="btn btn-danger">

Удалить

</a>

</td>

</tr>

</tbody>

</table>

</div>

<p class="d-flex justify-content-end">

<a href="/exercises/edit" class="btn btn-primary mx-2">

Добавить запись</a>

</p>

</div>

</div>

</div>

</div>

</div>

</section>

После создания всех шаблонов можно просмотреть полученный интерфейс.

При переходе на главную страницу, пользователю будет отображена информация о сервисе, а также даны будут возможности присоединиться к нему.

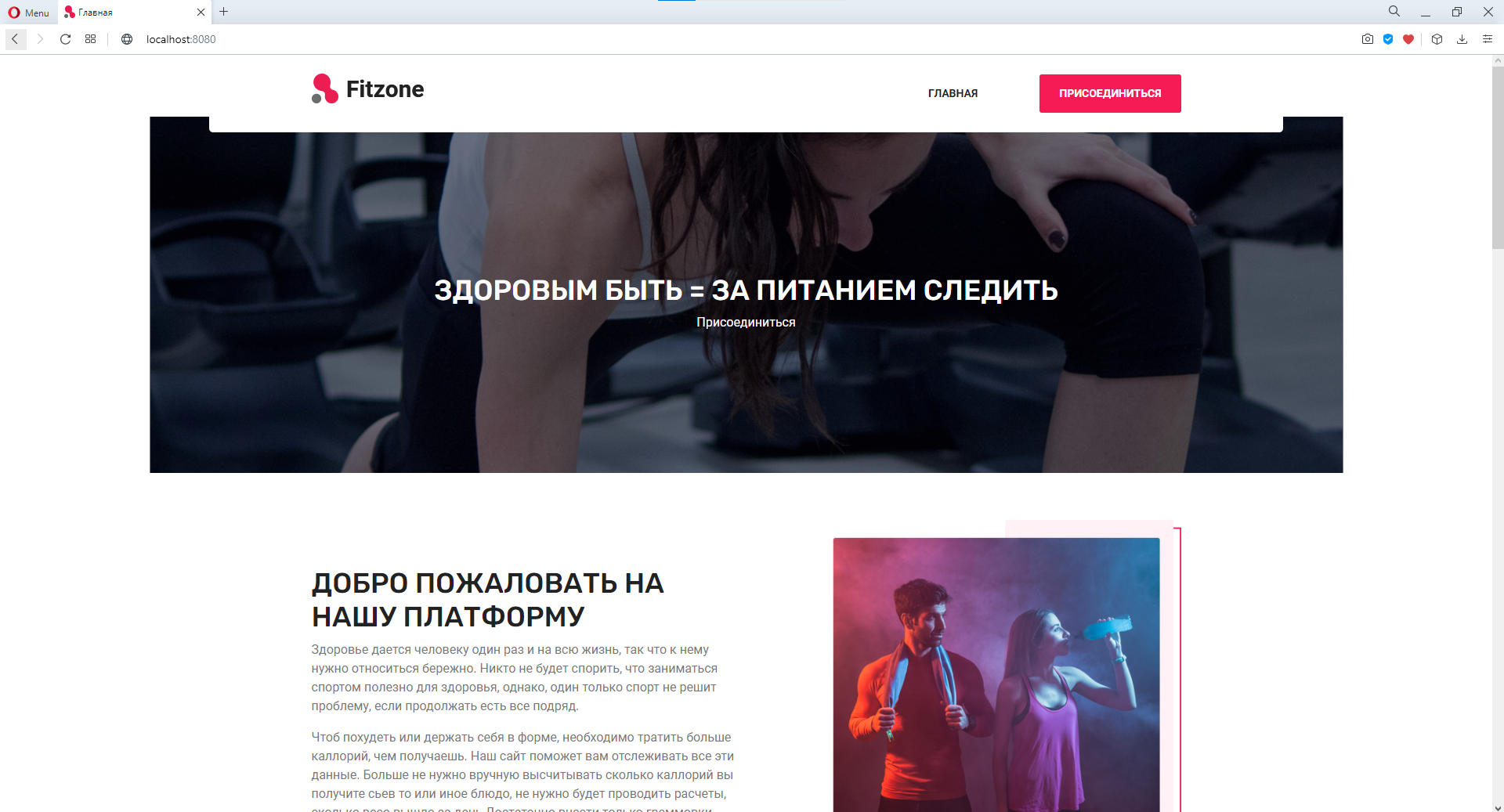


Рисунок 4.7 – Главная страница

Если на главной странице опуститься ниже, то можно пользователь увидит перед собой секцию с калькулятором. Не важно, это вошедший в систему пользователь или просто гость, он при любом раскладе может воспользоваться данным калькулятором.

Первым делом нужно заполнить все поля, после чего выбрать свой пол и уровень активности и нажать кнопку узнать результат. После этого в поля ниже будет выведен результат о дневной норме калорий и индекс массы тела пользователя с кратким комментарием.

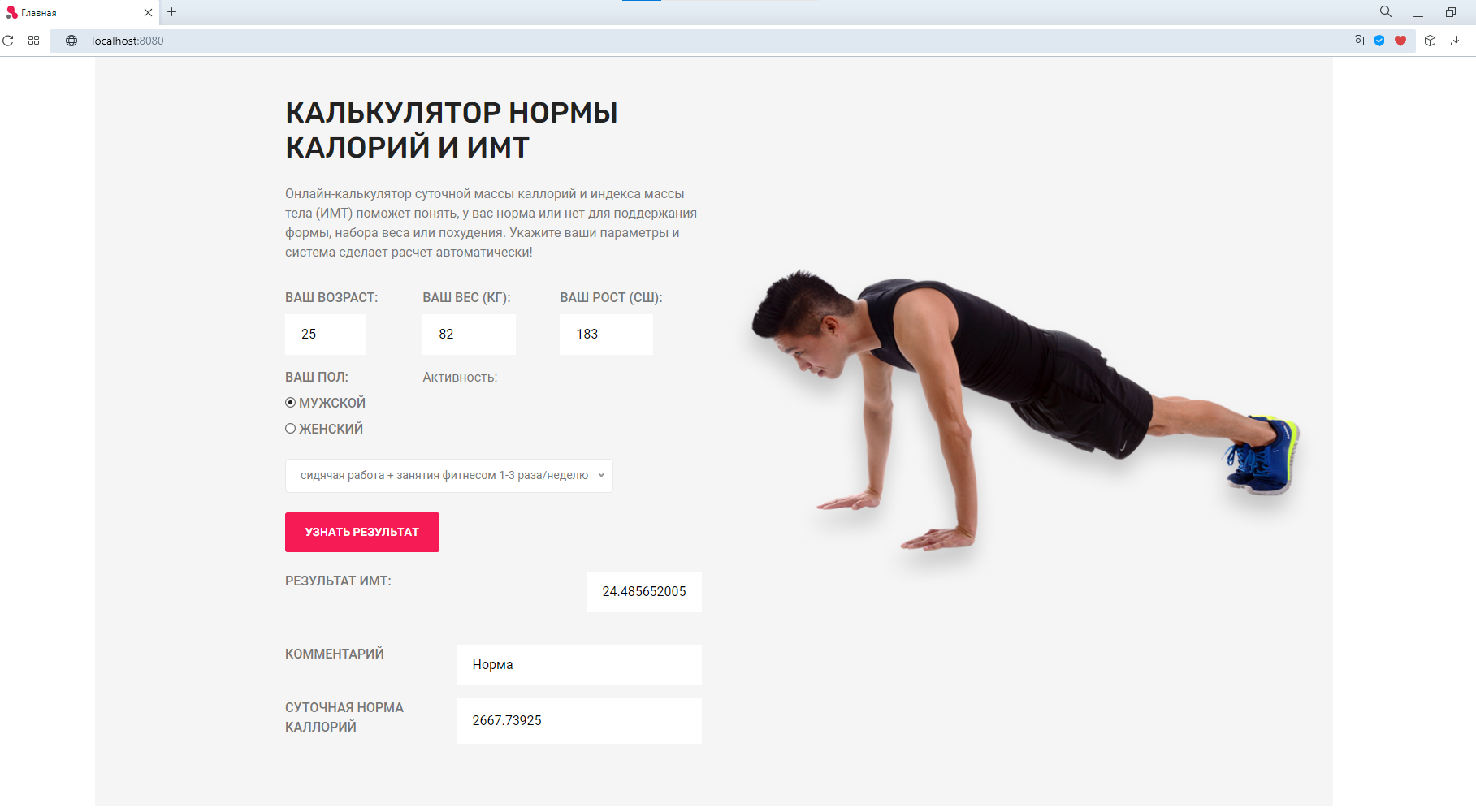


Рисунок 4.8 – Калькулятор

Для дальнейшего пользования системой необходимо войти в нее, или если нет аккаунта – зарегистрироваться.

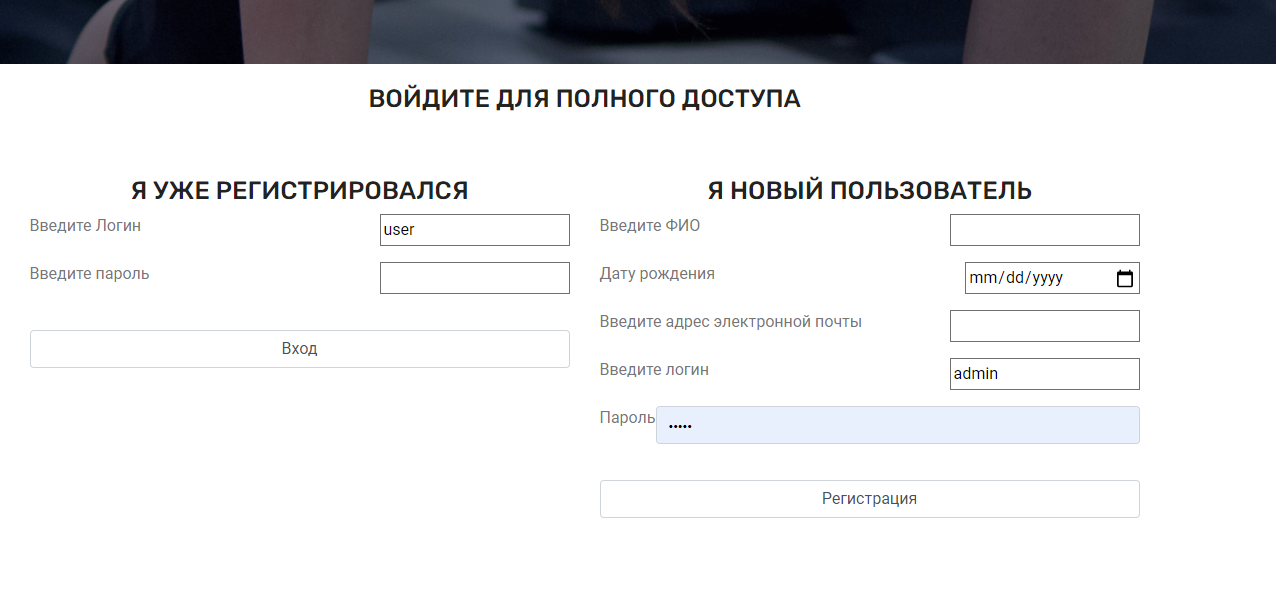


Рисунок 4.9 – Страница входа и регистрации

Если попытаться войти в систему под данными пользователя, которого не существует – система выдаст сообщение об ошибке.

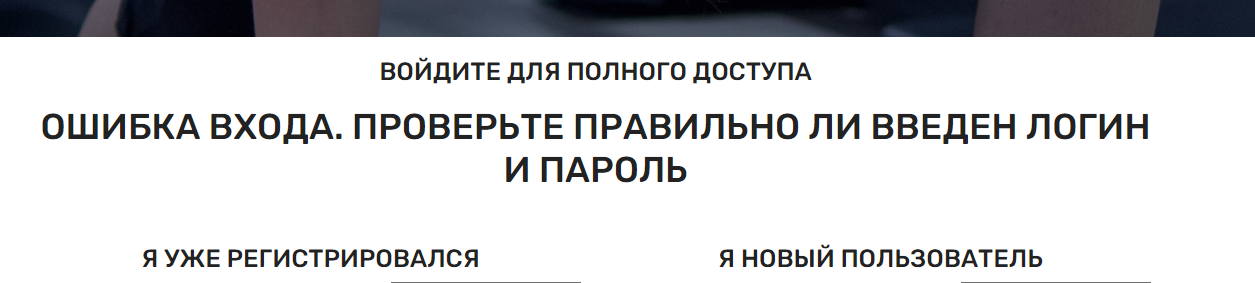


Рисунок 4.10 – Уведомление об ошибке

После заполнения всех полей и нажатия кнопки регистрации появится сообщение об успешной регистрации, после чего можно войти на сайт.

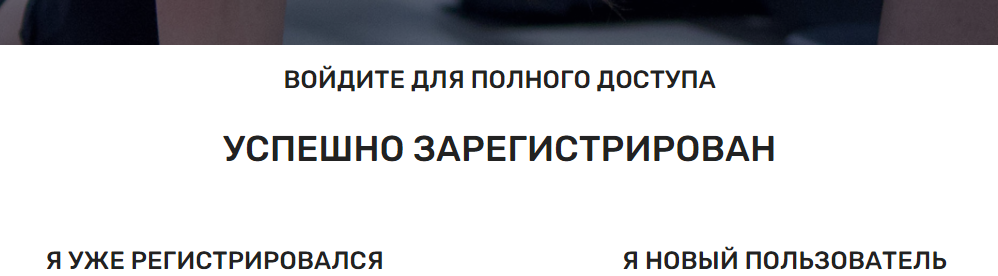


Рисунок 4.11 – Уведомление о регистрации

Самая главная страница для пользователя при работе с системой – это страница расписания. На ней пользователи в зависимости от времени приема пищи будут вносить то, что они съели или какой активностью занимались.

По умолчанию идет переход на сегодняшний день расписания. Если на выбранный день еще не было расписания, то система предложит создать его.

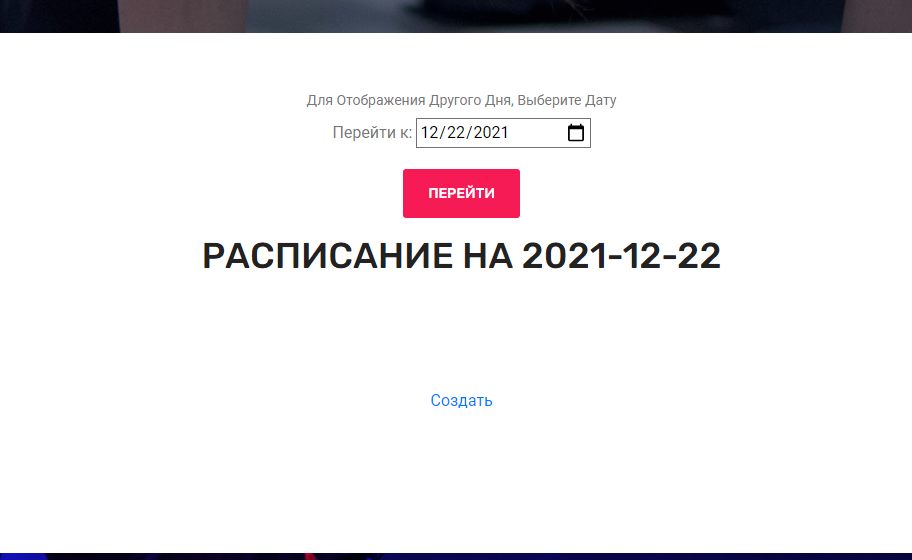


Рисунок 4.12 – Страница расписания

После того, как расписание будет создано, на странице отобразится список приемов пищи (настроенный администратором) и активности. В каждый пункт этого списка пользователь будет вносить свои данные по текущему дню.

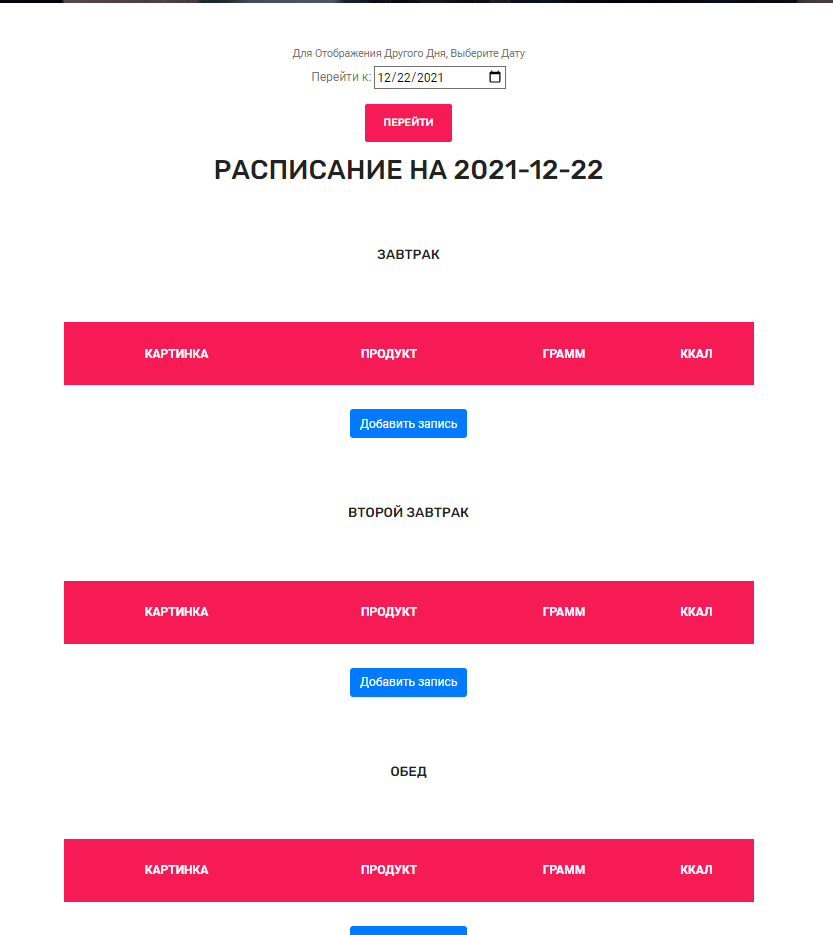


Рисунок 4.13 – Страница расписания после создания

Сразу при создании пункты пусты. Чтоб заполнить выбранный период приема пищи, достаточно около него нажать кнопку добавить запись, после чего будет осуществен переход на страницу добавления продукта в выбранный период.

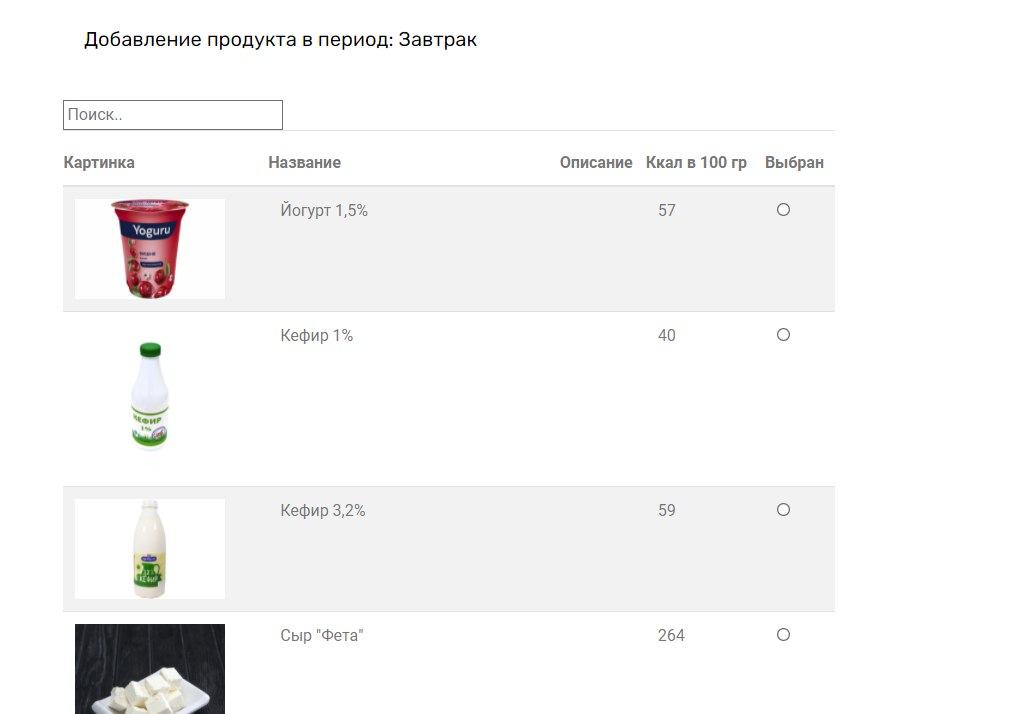


Рисунок 4.14 – Выбор продукта

Так как продуктов в списке много, реализован поиск по названию. Достаточно ввести часть названия и результаты будут отображены ниже. После выбора продукта указывается сколько грамм его было употреблено и все данные отправляются на сервер.

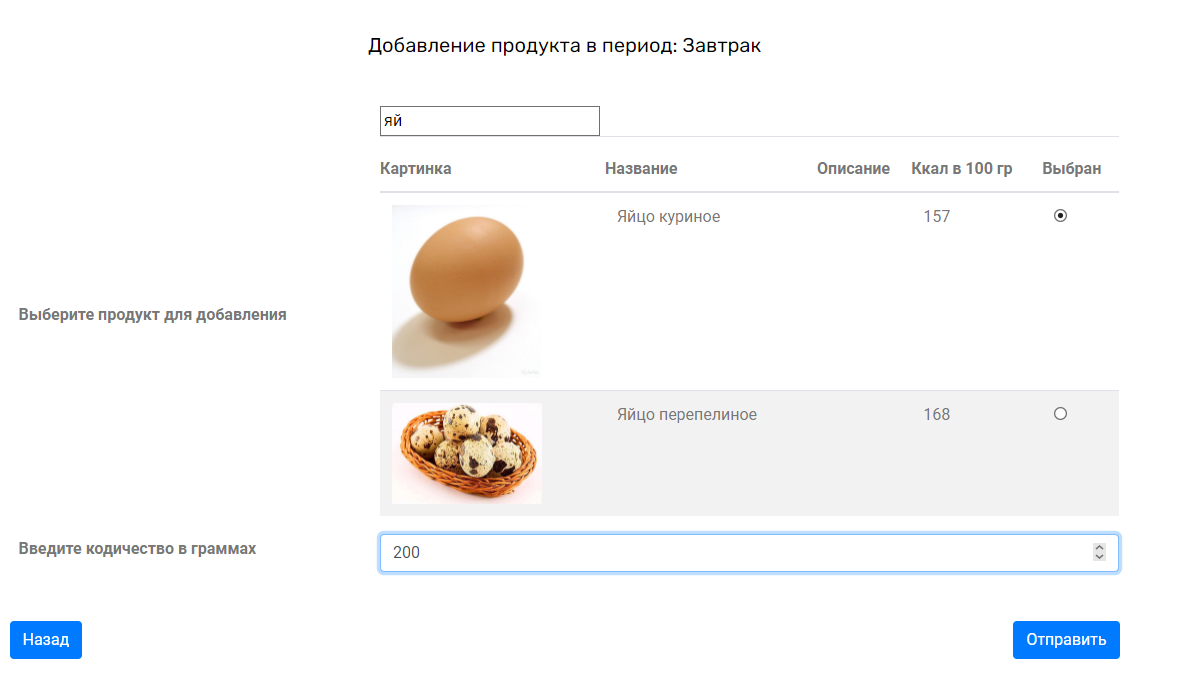


Рисунок 4.15 – Результат поиска при выборе продукта

Для внесения активности используется тот же метод на похожей странице.

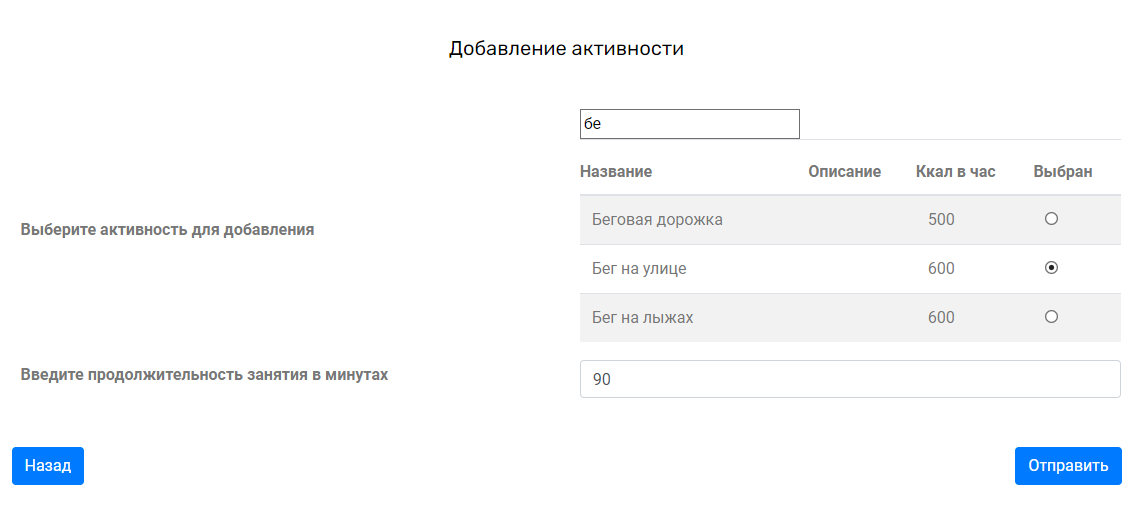


Рисунок 4.16 – Выбор активности

После заполнения выбранного дня данными, под каждым приемом пиши указано общее количество полученных калорий за этот прием, как и с активностями.

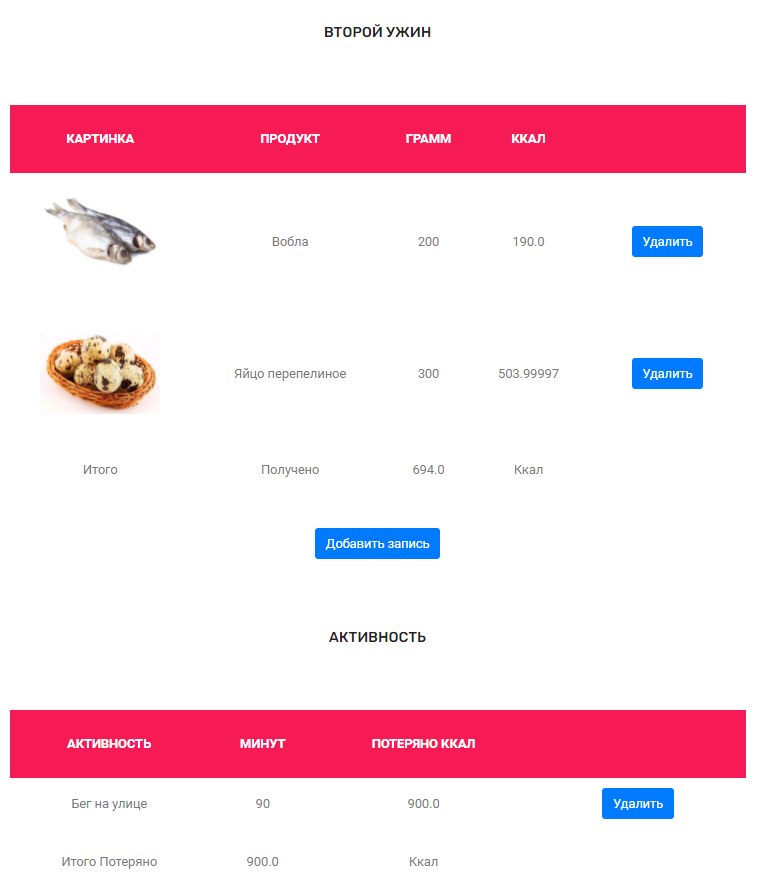


Рисунок 4.17 – Заполненная страница расписания

Итоговый результат дня отображен внизу страницы. На нем видна сумма полученных и затраченных калорий, а также разница между ними.

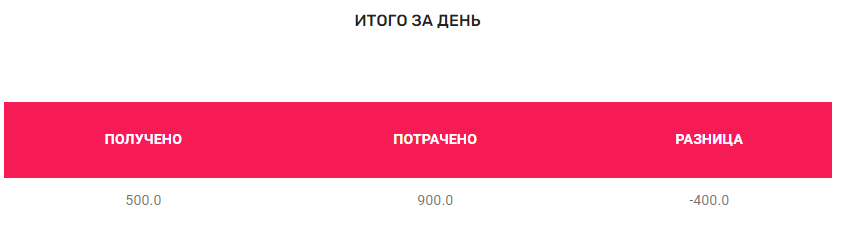


Рисунок 4.18 – Общий результат дня

Если продуктов или активностей в списке окажется недостаточно, пользователь может перейти в пункт продукты и предложить новый продукт.

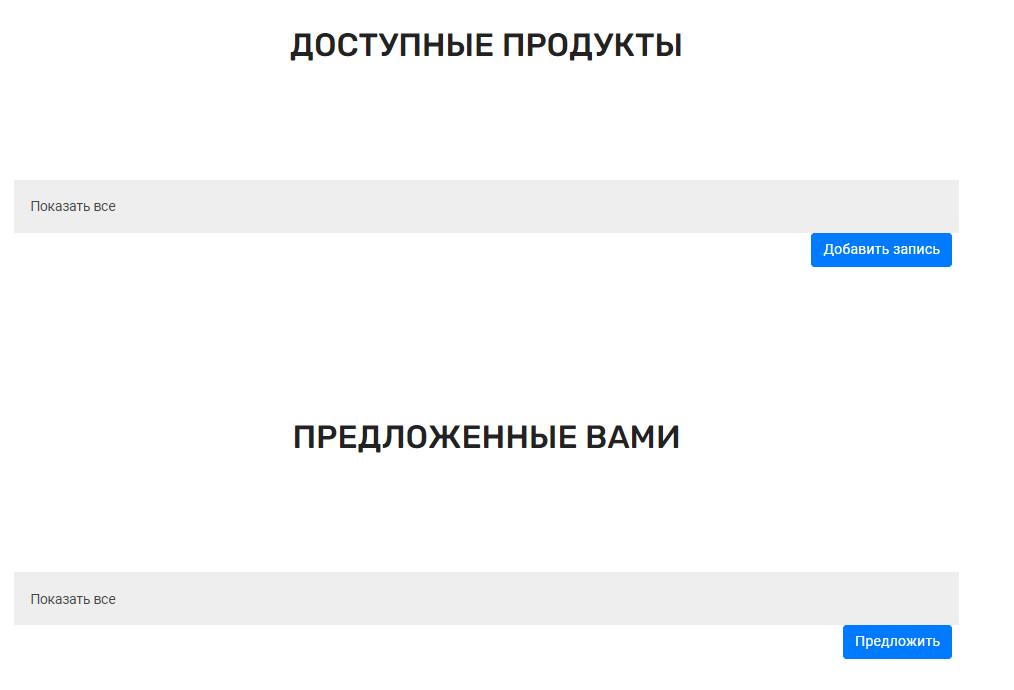
,--

Рисунок 4.19 – Страница продуктов

Для предложения продукта нужно нажать кнопку предложить, заполнить все поля и нажать кнопку отправить.

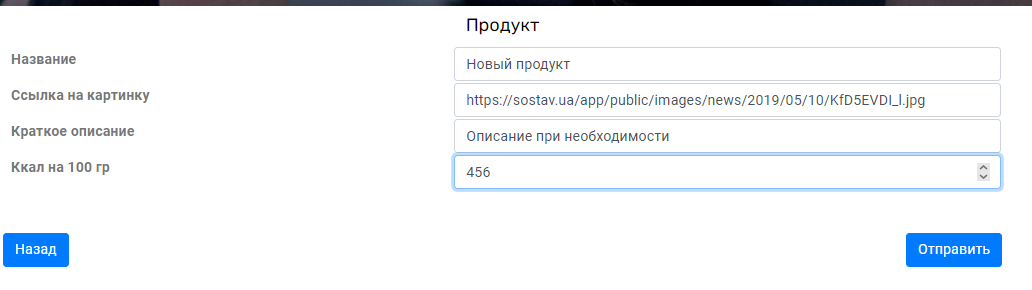


Рисунок 4.20 – Предложение нового продукта

После этого предложенный пользователем продукт появится в соответствующем списке и будет ждать одобрения администратора.

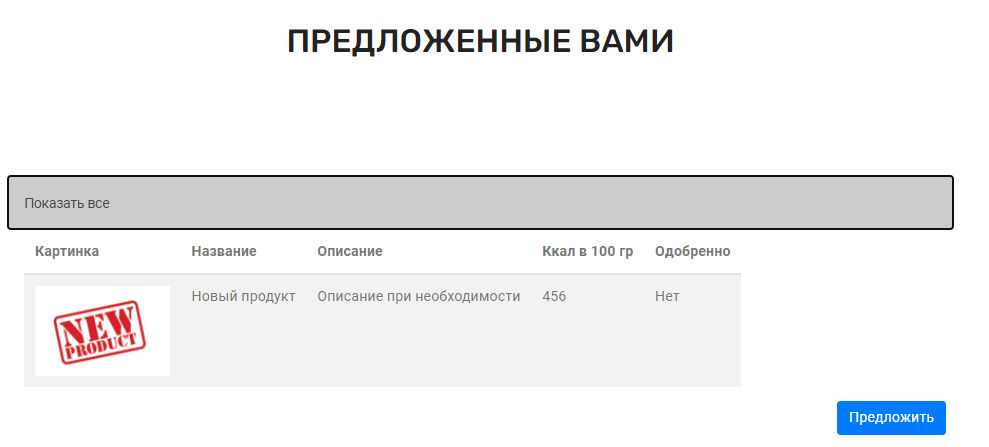


Рисунок 4.21 – Список предложенных продуктов

Администратор уже может соглашаться на предложенные варианты, редактировать их или удалять, как и остальные продукты или активности.

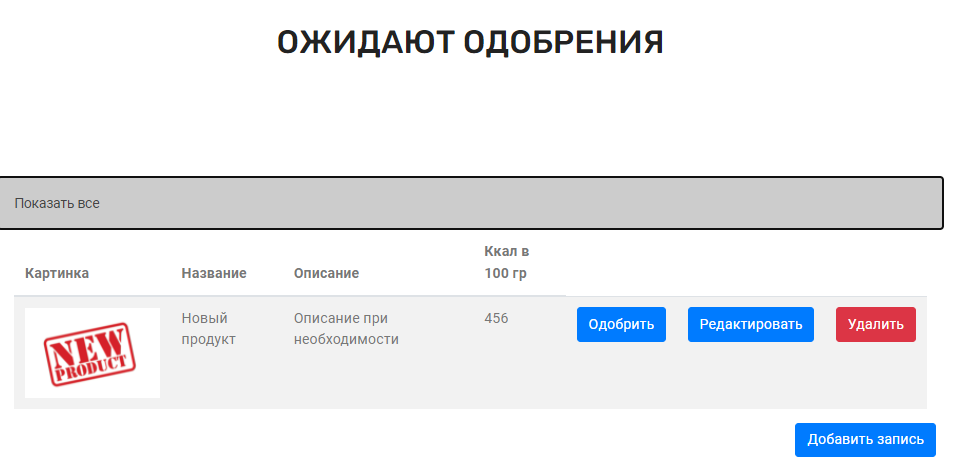


Рисунок 4.22 – Страница продуктов для администратора

Также администратору в соответствующем меню доступна функциональность по управлению пользователями.

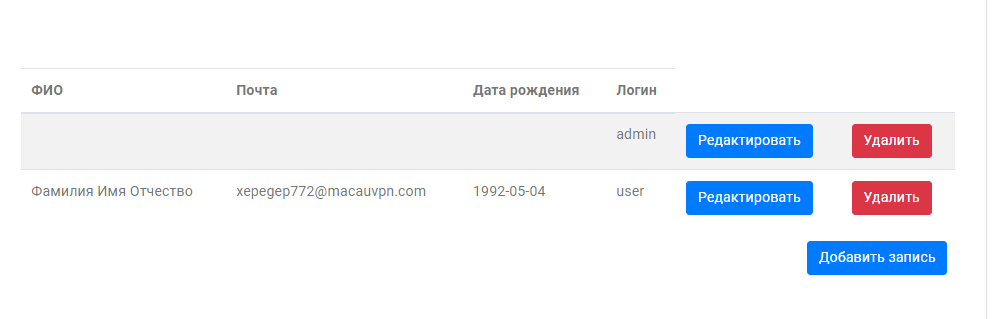


Рисунок 4.23 – Управление пользователями

При необходимости администратор настраивает какие приемы пищи будут отображаться всем пользователям.

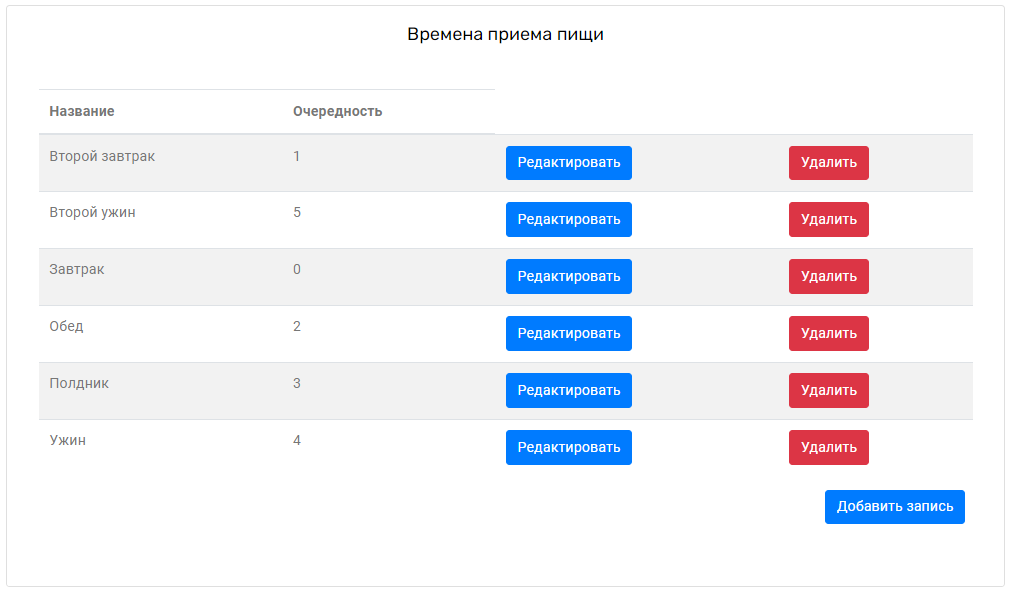


Рисунок 4.24 – Управление приемами пищи

## 4.3 Разработка алгоритмов реализации вариантов использования

При решении общей задачи возникали более мелкие, для решения которых необходимо было разрабатывать алгоритмы. Несколько примеров можно увидеть ниже:

1. Алгоритм сохранения в базе данных сущности, если ее еще там нет или обновления если есть:

public String createOrUpdate(Product e, Principal principal) {

if (e.getId() == null) {

float kkalIn1 = ((float) e.getKkalIn100gr())/((float)100);

e.setKkalIn1gr(kkalIn1);

e.setAddedBy(userService.findByUsername(principal.getName()));

service.create(e);

} else {

Product inBase = service.read(e.getId());

inBase.setImage(e.getImage());

inBase.setName(e.getName());

inBase.setDescription(e.getDescription());

inBase.setKkalIn100gr(e.getKkalIn100gr());

float kkalIn1 = ((float) e.getKkalIn100gr())/((float)100);

inBase.setKkalIn1gr(kkalIn1);

service.update(inBase);

}

return "redirect:/" + pagePath;

}

1. Алгоритм регистрации нового пользователя:

public boolean registerNew(User entity) {

if (repository.findByUsernameIgnoreCase(entity.getUsername()) != null)

return false;

entity.setPassword(passwordEncoder.encode(entity.getPassword()));

try {

if (roleService.findByName(DEFAULT\_ROLE) == null)

roleService.create(new Role(DEFAULT\_ROLE));

// просто создание

create(entity);

// добавление роли и сохранение

Role role = roleService.findByName(DEFAULT\_ROLE);

entity = repository.findByUsernameIgnoreCase(entity.getUsername());

entity.setRole(role);

update(entity);

// сохранение в списке ролей пользователя

role.getUsers().add(entity);

roleService.update(role);

return true;

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

## Модульное тестирование алгоритмов реализации вариантов использования

Модульное тестирование, или юнит-тестирование - это процесс в программировании, который направлен на проверку отдельных небольших частей приложения, также называемых атомарными, которые можно исследовать изолированно от других подобных частей. При выполнении данного тестирования могут проверяться как отдельные функции или методы классов, так и сами классы, взаимодействие классов, небольшие библиотеки и отдельные части приложения [15].

Довольно часто данный вид тестирования реализуется с использованием специальных технологий и инструментальных средств автоматизации тестирования, значительно упрощающих и ускоряющих разработку соответствующих тест-кейсов. Таким образом, оценивая каждый элемент отдельно и подтверждая правильность его работы, установить проблему значительно проще чем, если бы элемент был частью системы.

Примеры тестирования алгоритмов реализации вариантов использования:

1. Тестирование вывода преподавателей (таблица 4.1).

Таблица 4.1 - Тестирование вывода продуктов

|  |  |  |  |
| --- | --- | --- | --- |
| Краткое описание | Предварительные условия | Шаги для воспроизведения | Ожидаемый результат |
| Тестирование вывода списка продуктов | Запустить программу  Добавить по крайней мере два продукта в базу данных | Используя меню навигации перейти на страницу продуктов | При переходе на страницу отображен список продуктов из базы данных |

1. Тестирование добавления группы (таблица 4.2).

Таблица 4.2 - Тестирование добавления продукта в завтрак

|  |  |  |  |
| --- | --- | --- | --- |
| Краткое описание | Предварительные условия | Шаги для воспроизведения | Ожидаемый результат |
| Тестирование добавления продукта в завтрак | Запустить программу  Добавить прием пищи завтрак. Добавить любой продукт.  Создать расписание на текущий день. | Перейти на страницу расписания. Нажать кнопку добавить у завтрака. Выбрать продукт. Нажать кнопку отправить. | После сохранения, добавленный продукт с граммовкой отображается в расписании завтрака |

1. Тестирование изменения активности (таблица 4.3).

Таблица 4.3 - Тестирование изменения группы

|  |  |  |  |
| --- | --- | --- | --- |
| Краткое описание | Предварительные условия | Шаги для воспроизведения | Ожидаемый результат |
| Тестирование изменение активности | Запустить программу. Добавить активность. | Перейти на страницу активностей. Выбрать активность для редактирования. Изменить данные и отправить. | В таблице активности отображаются новые данные |