Министерство образования и науки Республики Казахстан

Некоммерческое акционерное общество «Торайгыров университет»

**ДИПЛОМНЫЙ ПРОЕКТ**

**Создание информационно аналитической системы по потреблению основных продуктов питания разными слоями населения на основе открытых данных РК**

По образовательной программе 6B06102 – «Информационные системы»

Павлодар

2022Министерство образования и науки Республики Казахстан

Некоммерческое акционерное общество «Торайгыров университет»

Факультет Computer Science

Допущен(а) к защите

Руководитель ОП \_\_\_\_\_\_\_\_\_\_\_\_ Н. Н. Оспанова

(подпись)

\_\_ \_\_\_\_\_\_\_\_\_ 20\_\_года

**ДИПЛОМнЫЙ ПРОЕКТ**

**Создание информационно аналитической системы по потреблению основных продуктов питания разными слоями населения на основе открытых данных РК**

**по образовательно программе 6B06102 – Информационные системы**

**Выполнил(а)** \_\_\_\_\_\_\_\_\_\_\_\_ С. А. Зайцев

(подпись)

Группа ИС-402

**Научный руководитель**

магистр, ст. преподаватель \_\_\_\_\_\_\_\_\_\_\_\_ С. Н. Талипов

(подпись)

**Нормоконтроль**

Ст. преподаватель \_\_\_\_\_\_\_\_\_\_\_\_ А. З. Даутова

(подпись)

Павлодар

2022

|  |  |  |
| --- | --- | --- |
| Задание  на выполнение ДР/П | лого 2020 | Ф МИ СМК 4.01.6/02 |

Министерство образования и науки Республики Казахстан

Некоммерческое акционерное общество «Торайгыров университет»

Факультет «Computer Science»

Специальность Информационные системы

**Утверждаю**

Руководитель образовательных программ канд. Пед. наук, профессор

Н. Н. Оспанова

20 года

**ЗАДАНИЕ**

на выполнение дипломной (го) работы/проекта

Обучающемуся Шухарт Диане Владимировне

Тема ДР/П Создание информационно аналитической системы по потреблению основных продуктов питания разными слоями населения на основе открытых данных РК

утверждена приказом по университету от 20 года №

Срок сдачи ДР/П 20 года

Исходные данные к ДР/П Методические рекомендации по выполнению ДР/П

Перечень подлежащих разработке в ДР/П вопросов или краткое содержание

выпускной работы (дипломной работы/дипломного проекта):

1 Описание и сравнительный анализ средств для разработки

2 Обзор возможностей TELEGRAM

3 Чат бот. Основные понятия. Функции. Виды

4 Выбор среды разработки и языка программирования

5 Общий алгоритм реализации проекта

6 Целевая аудитория

7 Постановка цели

8 Актуальность проекта

|  |  |  |
| --- | --- | --- |
| Задание  на выполнение ДР/П | лого 2020 | Ф МИ СМК 4.01.6/02 |

9 Разработка чат бота

10 Архитектура чат бота

11 Регистрация чат бота в TELEGRAM

12 Api

13 Добавление библиотек

14 Написание программы

Перечень графического материала с точным указанием обязательных чертежей:38

Рекомендуемая основная литература: 5

**ГРАФИК**

подготовки дипломной работы/проекта

|  |  |  |
| --- | --- | --- |
| Наименование разделов, перечень  разрабатываемых вопросов | Сроки представления научному  руководителю и консультантам | Подпись |
| Описание и сравнительный анализ средств для разработки |  |  |
| Постановка цели |  |  |
| Разработка чат бота |  |  |

**Подписи**

консультантов на законченную дипломную работу (проект), с указанием относящихся к ним разделов ДР/П

|  |  |  |  |
| --- | --- | --- | --- |
| Наименования разделов | Консультанты, Ф.И.О.  (уч. степень, звание) | Дата  подписания | Подпись |
| Описание и сравнительный анализ средств для разработки |  |  |  |
| Постановка цели |  |  |  |
| Разработка чат бота |  |  |  |

Научный руководитель С. Н. Талипов

(подпись) (инициалы, фамилия)

Задание принял к исполнению

обучающийся С. А. Зайцев

(подпись) (инициалы, фамилия)

Дата 20\_\_года

**Содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1 | Теоретическая часть | 6 |
| 1.1 | Цель и актуальность проекта | 6 |
| 1.2 | Обзор основных задач и функций проекта | 7 |
| 1.3 | Выбор средств и инструментов для разработки проекта | 7 |
| 1.4 | Общий алгоритм реализации проекта | 8 |
| 2. | Практическая часть | 9 |
| 2.1 | Архитектура проекта | 9 |
| 2.2 | Разработка интерфейса | 9 |
| 2.3 | Разработка базы данных |  |
| 2.4 | Разработка API | 10 |
| 2.5 | Разработка программного кода | 10 |
|  | Заключение | 29 |
|  | Список использованных источников | 30 |
|  | Приложение А | 31 |

**Введение**

В настоящее время, многие люди предпочитают чтение книг в электронном формате. Однако, существует большая группа любителей бумажных книг, которые также хотят иметь доступ к большому количеству литературы и обмениваться книгами между собой. В связи с этим, создание платформы по буккросингу и обмену книгами представляется актуальным и полезным проектом.

Цель данной платформы - облегчить процесс обмена книгами между пользователями, предоставить им доступ к широкому выбору литературы и создать удобное пространство для общения и обмена мнениями о книгах.

В рамках данного проекта была создана платформа на основе современных технологий, таких как React JS, TypeScript, Redux и Json-server. Платформа включает в себя несколько страниц, таких как страница списка всех книг, страница о компании, страница авторизации клиента, страница личного аккаунта клиента и страница конкретной книги.

Благодаря использованию современных технологий, пользователи могут быстро находить нужные книги, ознакамливаться с их основными характеристиками и связываться с владельцами книг. Это создает удобную платформу для обмена книгами между любителями чтения и способствует распространению книжной культуры.

Таким образом, разработка платформы по буккросингу и обмену книгами на основе современных технологий представляет собой важный проект, который способен удовлетворить потребности любителей чтения и обеспечить им доступ к большому количеству литературы.;

1. **Теоретическая часть**

**1.1 Цель и актуальность проекта**

В настоящее время, существует большая группа людей, которые не хотят отказываться от бумажных книг и хотят иметь доступ к широкому выбору литературы. Однако, не всегда у них есть возможность купить нужную книгу, а также не все книги доступны в их регионе. В этом случае, платформа по буккросингу и обмену книгами представляет собой идеальное решение проблемы, которое облегчает процесс обмена книгами между пользователями и позволяет им получить доступ к большому количеству книг.

Несмотря на то, что электронные книги являются более компактными и удобными в путешествиях, они не могут заменить уникальный опыт чтения бумажных книг.

Бумажные книги имеют важную значимость перед электронными, поскольку они предоставляют своим читателям уникальный опыт чтения, который невозможен сравнить с электронными форматами. Бумажные книги ощущаются в руках, позволяют насладиться ароматом бумаги и чернил, их страницы можно ощупать, перелистывать, делать пометки и выделения.

Кроме того, бумажные книги не требуют подключения к интернету и не зависят от устройств, что обеспечивает надежность и доступность. Также, многие люди предпочитают читать бумажные книги в качестве отдыха от экранов мониторов и смартфонов.

Актуальность проекта заключается в том, что он способствует развитию книжной культуры и увеличению доступности литературы для людей, проживающих в разных регионах. Кроме того, платформа представляет собой удобное пространство для общения и обмена мнениями о книгах, что способствует развитию книжного сообщества.

**1.2 Обзор основных задач и функций проекта**

Приложение по буккросингу и обмену книгами предоставляет следующие функции:

- поиск книг по различным критериям. Это позволяет быстро и удобно найти нужные книги, используя поиск по названию, автору или жанру. Благодаря этой функции пользователи могут быстрее находить интересующие их книги, а также изучать новые книги, читать аннотации и отзывы других пользователей, тем самым создавая подборку желаемых книг.

- добавление книг в список доступных для обмена. Это позволяет пользователям добавлять книги, которые они хотели бы обменять или поделиться с другими участниками платформы. Таким образом, пользователи могут предложить к обмену книги, которые уже прочитали, или обнаружили лишние в своей коллекции, что поможет другим пользователям получить доступ к большему количеству книг.

- просмотр подробной информации о книге. Это позволяет пользователям получить всю необходимую информацию о книге, включая фотографию, описание, автора, жанр и контакты пользователя, который добавил эту книгу. Благодаря этой функции пользователи могут быстрее принимать решение о том, стоит ли им читать эту книгу, а также связываться с владельцами книг, которые они хотели бы обменять или поделиться.

- создание личного аккаунта и хранение данных о своих книгах. Это позволяет пользователям создать личный аккаунт, где они могут хранить информацию о своих книгах и управлять списком доступных книг для обмена. Благодаря этой функции пользователи могут создать свою персональную библиотеку книг, контролировать доступность книг для обмена и управлять своими обменами.

- оценка книг и обмен мнениями. Это позволяет пользователям оценивать книги и обмениваться мнениями и рекомендациями с другими участниками платформы. Благодаря этой функции пользователи могут выражать свое мнение о книгах, обмениваться рекомендациями с другими читателями и находить новых друзей с общими интересами.

- просмотр информации о компании. Это позволяет пользователям ознакомиться с информацией о компании, которая разработала платформу, и получить контактные данные для обратной связи. Благодаря этой функции пользователи могут узнать больше о платформе, ее целях и задачах, а также связаться с разработчиками, чтобы получить помощь или задать вопросы.

Все эти функции позволяют пользователям нашей платформы наслаждаться чтением книг и общением с другими читателями в удобной и комфортной обстановке. Они сделают процесс поиска, обмена и чтения книг более удобным и доступным, а также позволят нашим пользователям находить новые друзья и единомышленников.

**1.3 Выбор средств и инструментов для разработки проекта**

IntelliJ IDEA - это интегрированная среда разработки для создания высококачественных приложений на Java, Kotlin, Groovy, Scala и других языках программирования. Она обладает широким спектром функциональных возможностей, интуитивно понятным интерфейсом и инструментами для улучшения производительности и эффективности разработки. IntelliJ IDEA позволяет быстро создавать и отлаживать код, а также облегчает процесс совместной работы между разработчиками.

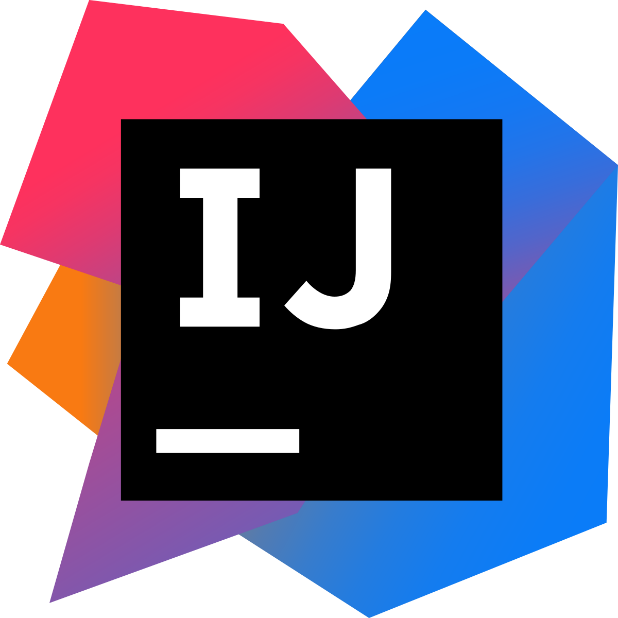


Рисунок 1 − Логотип IntelliJ Idea

Одним из принципов разработки веб-приложений является Single Page Application (SPA). Этот подход был придуман в начале 2000-х годов, когда появились первые богатые интернет-приложения (RIA) и было необходимо улучшить производительность и удобство работы с веб-сайтами.

SPA - это веб-приложение, которое загружает только одну страницу HTML и динамически обновляет ее содержимое, используя JavaScript для связи с сервером и манипулирования DOM. Этот подход позволяет создавать приложения с более быстрым откликом и более удобным пользовательским интерфейсом. Кроме того, SPA позволяет создавать более гибкие и масштабируемые приложения, поскольку они могут работать в фоновом режиме, загружать только те данные, которые нужны в данный момент, и более эффективно использовать ресурсы клиента.

Преимущества использования SPA включают:

- более быструю загрузку страниц и удобство работы с приложением;

- меньшую нагрузку на сервер и возможность кеширования данных на клиенте;

- более гибкую архитектуру приложения и возможность использования различных фреймворков и библиотек для разработки.

Использование инструментов, таких как IntelliJ IDEA, в сочетании с принципом SPA, позволяет создавать высококачественные, быстрые и масштабируемые веб-приложения, которые могут удовлетворить потребности самых требовательных пользователей.

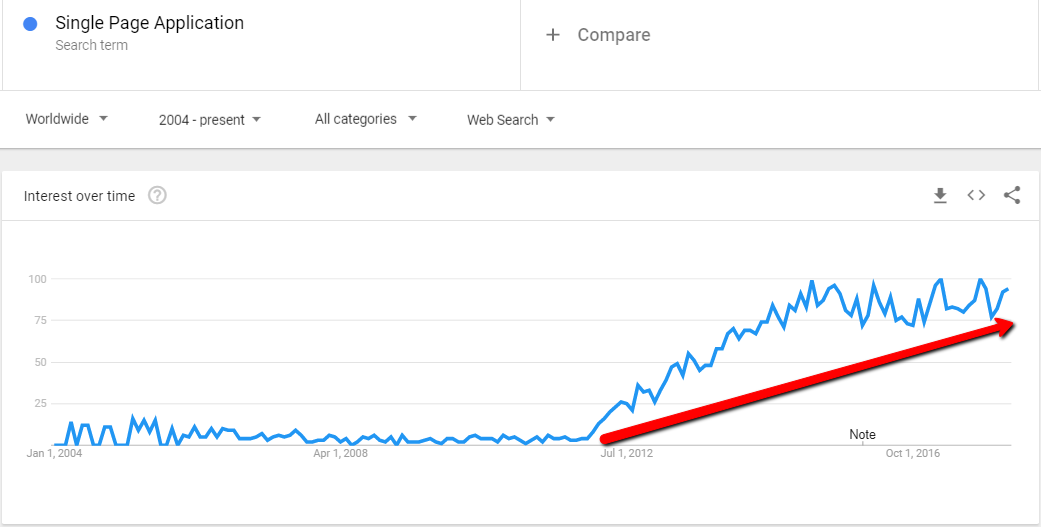


Рисунок 2 − График развития популярности SPA

Чтобы создать SPA использовались такие технологии как React JS в сочетании с TypeScript и Redux.

React JS - это библиотека JavaScript для создания пользовательских интерфейсов (UI), которая была разработана компанией Meta. Он был впервые выпущен в 2013 году и быстро стал одним из самых популярных инструментов для создания фронтенд-приложений. Он используется для создания динамических и масштабируемых интерфейсов для веб-приложений.



Рисунок 3 − Логотип компании Meta

React JS основан на принципе компонентов, что означает, что интерфейс пользователя создается из набора отдельных компонентов. Эти компоненты могут быть переиспользованы в различных частях приложения, что упрощает разработку и ускоряет процесс создания приложений. Кроме того, React использует виртуальный DOM (VDOM), который позволяет ускорить процесс отображения изменений на странице и повысить производительность приложения.

React JS имеет множество преимуществ, таких как:

- удобный и интуитивно понятный синтаксис;

- высокая производительность благодаря использованию виртуального DOM;

- возможность создавать многокомпонентные приложения, которые легко масштабировать;

- широкий выбор инструментов и библиотек для разработки.

React также обладает большим сообществом разработчиков, которые создают множество инструментов и библиотек для улучшения процесса разработки. Для создания отдельной страницы для каждой книги была использована библиотека React Router. React Router - это набор инструментов для реализации навигации в веб-приложении, который позволяет создавать SPA с множеством страниц и маршрутов.

С помощью React Router был реализован механизм маршрутизации, который позволяет перенаправлять пользователя на нужную страницу приложения в зависимости от запроса. Каждая книга имеет свой уникальный идентификатор, поэтому был создан маршрут для отображения информации о конкретной книге, основанный на этом идентификаторе.

При обращении к маршруту для отображения информации о книге, приложение загружает соответствующую страницу с информацией о книге, включая фотографию, описание, автора, жанр и контакты пользователя, который добавил эту книгу.

Таким образом, использование React Router позволило создать удобный и легко навигируемый интерфейс для отображения информации о книгах и обеспечить быстрый доступ к нужной книге с помощью уникальных маршрутов для каждой книги.

TypeScript - это язык программирования, который является надмножеством JavaScript, добавляющим статическую типизацию и другие новые функции к языку. Он был разработан Microsoft и впервые выпущен в 2012 году. TypeScript позволяет создавать более безопасный и поддерживаемый код, улучшая производительность и уменьшая количество ошибок.

TypeScript добавляет статическую типизацию к JavaScript, что означает, что переменные, функции и объекты имеют явно определенные типы. Это улучшает производительность приложения и облегчает его поддержку и разработку. TypeScript также имеет множество других возможностей, таких как поддержка классов, наследования, интерфейсов и многие другие.

React JS в сочетании с TypeScript позволяет разработчикам создавать более безопасный и поддерживаемый код, улучшая производительность и уменьшая количество ошибок. Это особенно полезно для больших проектов, где множество разработчиков работают вместе. React JS и TypeScript также интегрируются между собой очень хорошо, что обеспечивает более эффективную и безопасную разработку.

В настоящее время React JS и TypeScript считаются стандартом технологии для создания веб-приложений. Большинство крупных компаний используют React JS и TypeScript для разработки своих приложений. React JS и TypeScript имеют большое сообщество разработчиков и множество инструментов и библиотек для разработки, что делает их одними из наиболее популярных технологий для создания современных веб-приложений.

Redux - это библиотека управления состоянием для JavaScript-приложений, которая работает в связке с React JS. Redux позволяет управлять сложным состоянием приложения в единой глобальной области состояния, которая доступна для всех компонентов React. Redux помогает управлять состоянием приложения, упрощает его разработку и облегчает отладку.

Redux использует паттерн Flux, который представляет собой архитектурный шаблон для управления состоянием веб-приложения. Redux работает с единственным глобальным объектом, который содержит всю информацию о состоянии приложения. Каждый раз, когда происходит изменение состояния, создается новый объект состояния, который сохраняется в глобальном хранилище.

Redux позволяет управлять состоянием приложения с помощью actions, reducers и store. Actions - это объекты, которые описывают изменение состояния приложения. Reducers - это функции, которые принимают текущее состояние и действие и возвращают новое состояние. Store - это единственное место, где хранится состояние приложения.

React JS и Redux часто используются вместе, что обеспечивает более удобную и эффективную разработку веб-приложений. С React JS можно легко интегрировать Redux, что позволяет управлять сложным состоянием приложения с помощью простых и понятных инструментов.

Сегодня React JS и Redux считаются одними из самых популярных технологий для разработки веб-приложений. Они имеют широкое сообщество разработчиков и множество инструментов и библиотек для разработки, что делает их одними из наиболее популярных технологий для создания современных веб-приложений.

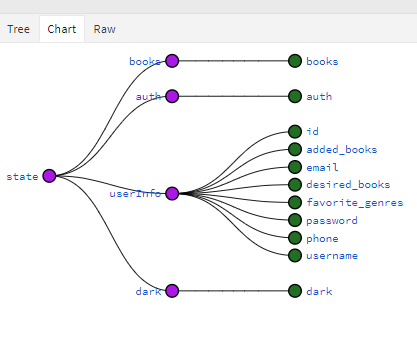


Рисунок 4 − Пример расположения данных в Redux

Для веб-приложения по буккросингу и обмену книгами был разработан современный минималистичный дизайн. Это такой подход к дизайну, который убирает все лишнее и фокусируется на существенных элементах. Этот подход создает простые, но красивые интерфейсы, которые легко воспринимаются пользователем. Такой подход позволил убрать все ненужное и сосредоточиться на функциях, которые действительно необходимы пользователям.

UI UX (user interface и user experience) - это еще один важный аспект в разработке приложений. Цель UI UX - создать интерфейс, который будет максимально простым, интуитивно понятным и удобным для пользователя. Поэтому был разработан интерфейс с интуитивно понятным пользовательским опытом, который позволяет быстро находить нужные книги и добавлять свои книги для обмена.

Также для веб-приложения по буккросингу и обмену книгами был разработана современна возможность перейти на темную тему. Темная тема, это альтернативный режим интерфейса приложения, где цветовая схема использует темные оттенки вместо светлых цветов. Такой режим имеет ряд преимуществ, которые сделали его популярным в последнее время.

Во-первых, темная тема позволяет снизить нагрузку на глаза пользователей в условиях недостаточной освещенности, таких как ночное время или темные помещения. Она также может снизить утомляемость глаз во время длительного использования приложения, что особенно важно для людей, которые проводят много времени за экраном.

Во-вторых, темная тема позволяет снизить энергопотребление смартфонов и ноутбуков с OLED-дисплеями. Такие дисплеи используют меньше энергии, когда на экране отображаются темные цвета.

В приложении по буккросингу и обмену книгами была реализована темная тема, чтобы предоставить пользователям возможность выбора цветовой схемы, которая наиболее удобна для них. Кроме того, темная тема делает приложение более современным и стильным, что может привлечь новых пользователей.

Таким образом, темная тема является полезной функцией, которая улучшает пользовательский опыт и делает приложение более удобным и доступным для пользователей.

* 1. **Общий алгоритм реализации проекта**

Проект по буккроссингу и обмену книгами был успешно реализован при помощи следующего алгоритма.

Сначала были определены требования к проекту, проведен анализ рынка и потребностей пользователей, определены функции и основные характеристики приложения.

Затем был разработан дизайн интерфейса приложения, созданы макеты и определены функциональные возможности пользовательского интерфейса.

Для разработки проекта были выбраны и установлены необходимые инструменты и технологии, включая фреймворк React, TypeScript, Redux и Json-server.

После этого была выполнена разработка серверной и клиентской частей приложения, включая разработку функций и логику приложения.

После завершения разработки приложение было тщательно протестировано на разных устройствах и браузерах, были обнаружены и исправлены ошибки.

**2 Практическая часть**

**2.1 Архитектура проекта**

Архитектура проекта была задумана таким образом, чтобы обеспечить четкую и логичную структуру приложения. В корневой папке был создан файл modules.ts, в котором содержались все текстовые константы, используемые в action и reducer, а также интерфейсы для определения формата данных в приложении. Это позволило уменьшить количество ошибок при написании кода.

Далее, в папке redux были созданы файлы, где хранился глобальный стор, а также все actions и reducers. Использование Redux обеспечивало удобное хранение состояния приложения и позволяло легко расширять его функциональность в будущем.

В папке pages были храниться все страницы приложения, в которых использовались компоненты из папки components. Компоненты были разбиты по логическим блокам, таким как header, footer, book, form, и т.д., что облегчало их использование и повторное использование в других частях приложения.

Таким образом, архитектура проекта была спроектирована так, чтобы обеспечить чистый и понятный код, удобное хранение состояния приложения и легкое масштабирование функциональности.

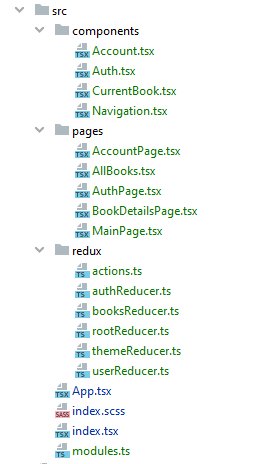


Рисунок 5 − Архитектура папок и файлов в проекте

В проекте был применен подход чистых функций в Redux, где все reducers были реализованы как чистые функции без побочных эффектов. Этот принцип показан на рисунке 6. Это позволило обеспечить стабильное и предсказуемое поведение приложения, а также упростить тестирование и отладку кода. Использование чистых функций помогло избежать проблем, связанных с разделяемым состоянием, такими как гонки данных и блокировки, и обеспечило более предсказуемое и удобное использование кода.

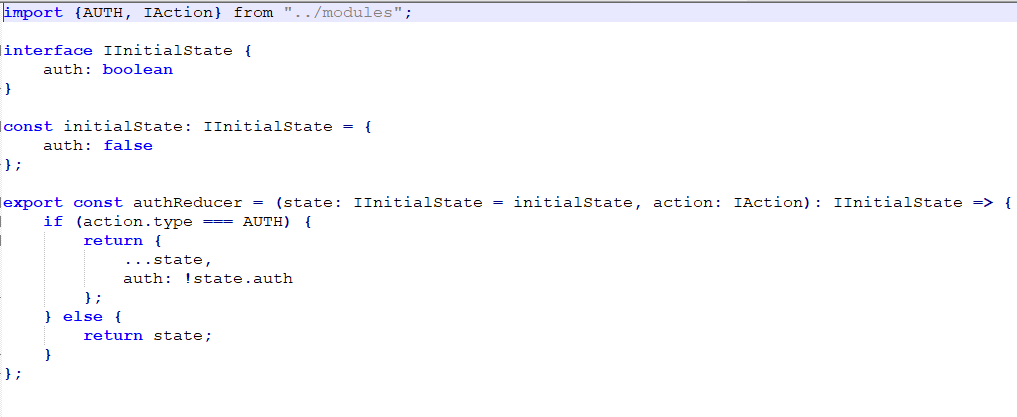


Рисунок 6 − Принцип чистых функций

Также во время разработки приложения использовался принцип тупых компонентов, которые отвечают только за отображение данных, не содержат внутри себя логики и не имеют своего собственного состояния. Такой подход к проектированию компонентов позволяет упростить код, сделать его более понятным и легким для поддержки. Этот принцип показан на рисунке 7.



Рисунок 7 − Принцип тупых компонент

При использовании тупых компонентов логика обработки данных или управления состоянием приложения остается на более высоких уровнях иерархии компонентов, что позволяет избежать появления ошибок в коде. Кроме того, такие компоненты не производят побочных эффектов, что снижает вероятность появления проблем при работе с ними.

Использование тупых компонентов в React является хорошей практикой и позволяет создавать более удобное и надежное приложение.

**2.2 Разработка интерфейса**

Разработка интерфейса програмы происходила по определенному процессу. Сначала составился подробный план разработки, где были определены функциональные и нефункциональные требования к программе, ее цели и задачи, а также ее целевая аудитория. Затем проектируется пользовательский интерфейс (UI) и пользовательский опыт (UX) с помощью специализированных программных средств, таких как Figma. На рисунках 8 и 9 изображены примеры интерфейса.

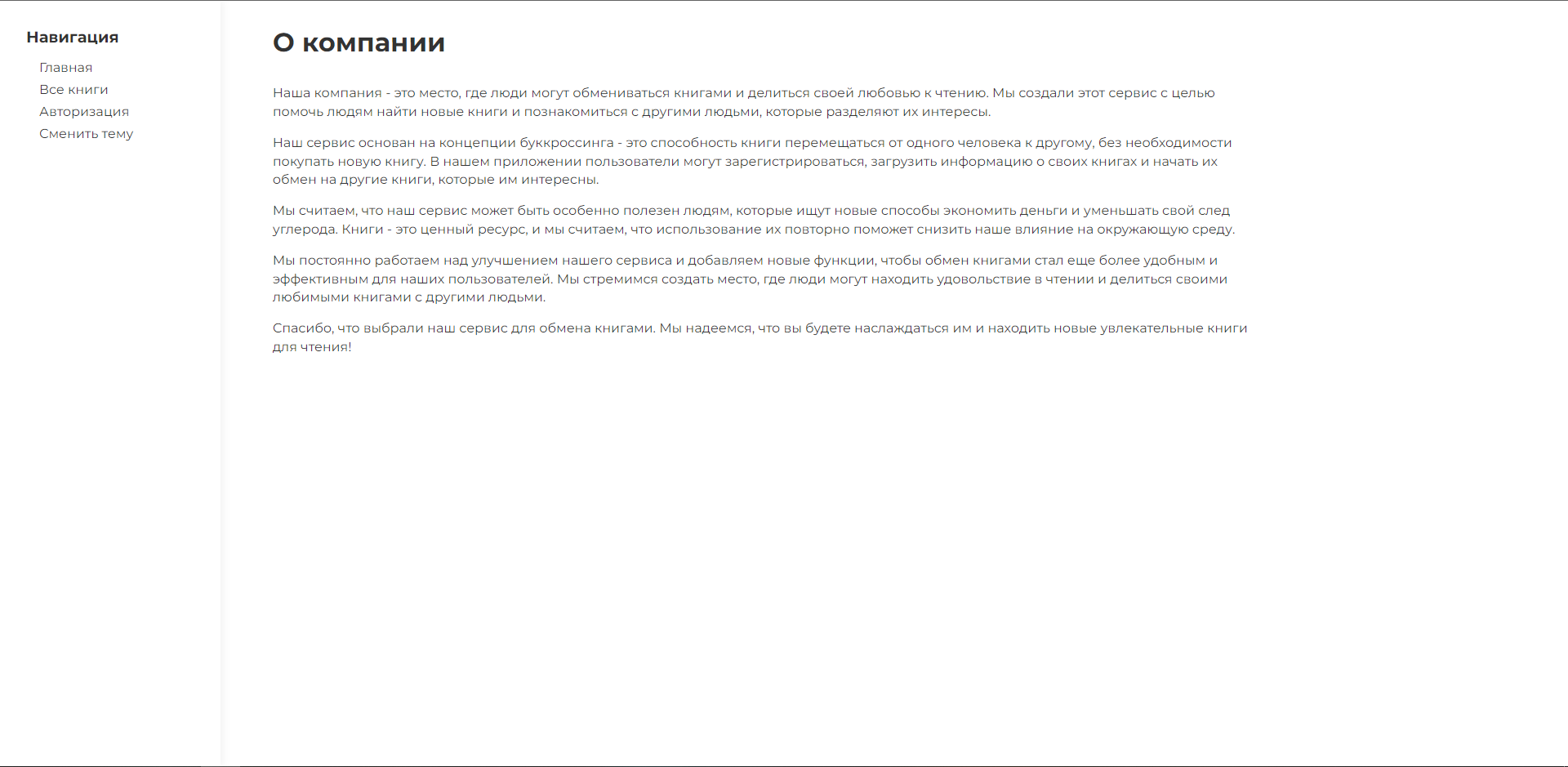


Рисунок 8 − Дизайн страницы о компании

Для обеспечения согласованности и качества интерфейса, обычно были созданы библиотеки компонентов, которые могут быть использованы в различных проектах. В этих компонентах определяются стили, расположение элементов и поведение интерфейса.

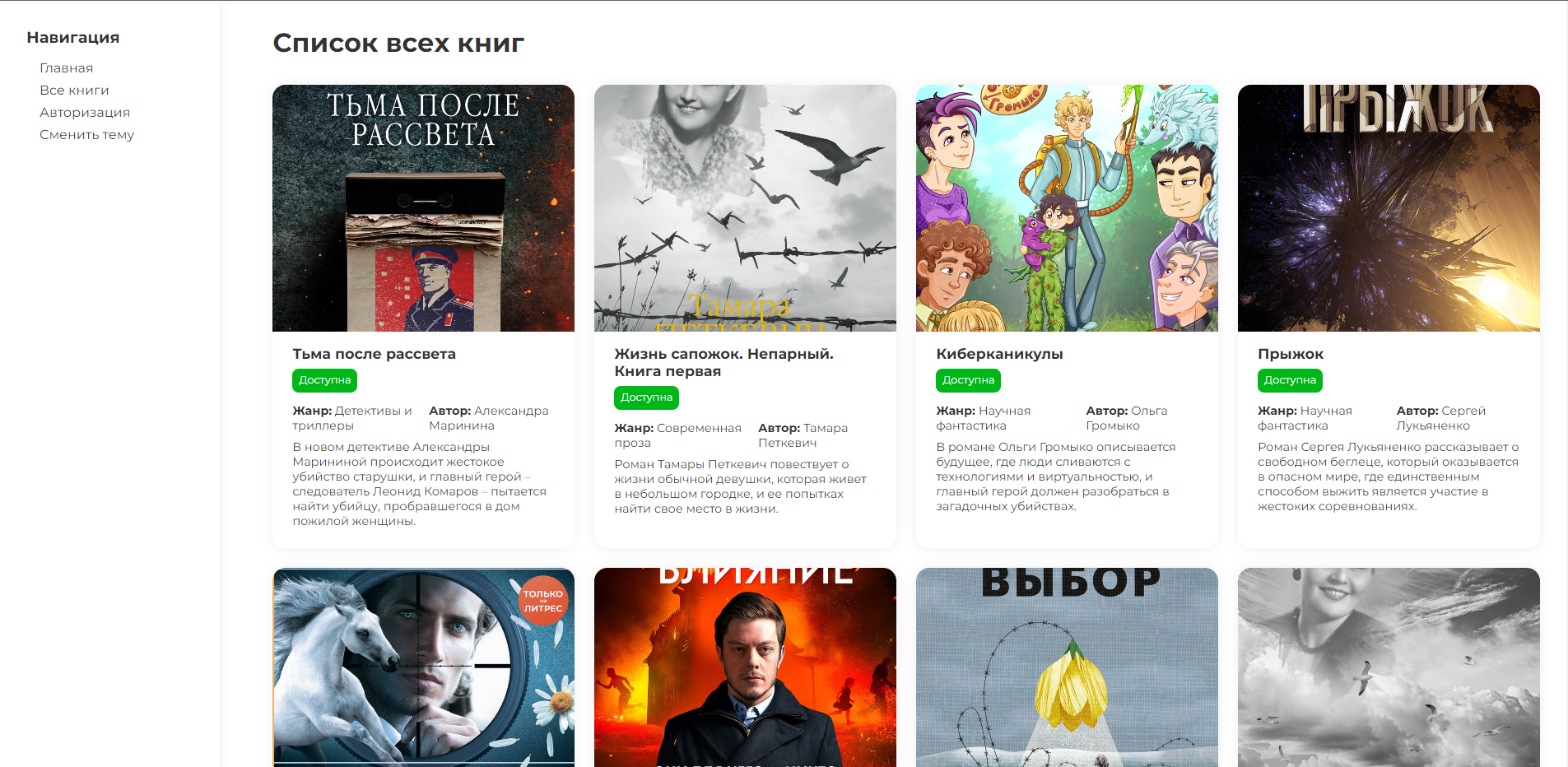


Рисунок 9 − Дизайн страницы для вывода всех книг

В процессе разработки приложения по буккросингу и обмену книгами были созданы различные компоненты, включая компонент для отображения книги, компонент формы с элементами ввода данных, а также компонент плашки, который информирует пользователя, доступна ли книга для обмена или нет.

Создание библиотеки компонентов помогает обеспечить согласованность и качество интерфейса в различных проектах. В такой библиотеке определяются стили, расположение элементов и поведение интерфейса. Это также позволяет значительно сократить время разработки и упростить процесс тестирования и обновления интерфейса.

**2.3 Разработка базы данных**

Когда дело касается разработки базы данных для проекта, необходимо определить, какая технология будет использоваться для хранения и получения данных. В данном проекте было решено использовать json-server в формате json для создания базы данных.

Json-server - это библиотека для Node.js, которая позволяет быстро и легко создать RESTful API на основе JSON файлов. Она позволяет создавать и запускать локальные серверы с фиктивными данными, что очень удобно при разработке и тестировании приложений.

Для получения данных из json-server в проекте был использован axios. Он является мощной библиотекой для работы с HTTP-запросами и поддерживает типизацию от TypeScript. Это позволяет точно определить, какие данные вы ожидаете получить от сервера, что улучшает надежность и читаемость кода.

Для обеспечения типизации данных, которые приходят с сервера, были созданы специальные интерфейсы в файле modules.ts показанные на рисунке 10 и 11. Они определяют структуру данных, что позволяет использовать эти данные в проекте без ошибок. Это также упрощает понимание того, какие данные будут получены и какие свойства они имеют.

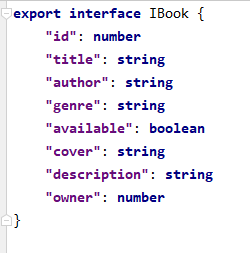


Рисунок 10 − Интерфейс IBook

Таким образом, использование json-server и axios позволило создать базу данных для проекта и получать данные из нее с помощью HTTP-запросов. Использование интерфейсов позволило обеспечить типизацию и повысить надежность проекта.

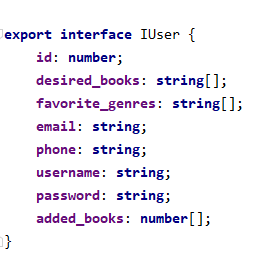


Рисунок 11 − Интерфейс IUser

**2.4 Разработка API**

Как уже было сказано, Json-server предоставляет RESTful API, который позволяет получать данные из локального файла в формате JSON. По умолчанию сервер запускается на порту 3000, но в данном проекте был использован порт 3004. Он принимает запросы GET, POST, PUT, PATCH и DELETE

Однако, если необходимо получить данные из других файлов в формате JSON, можно использовать ключевое слово "routes" в файле db.json и задать путь к этим файлам. Это хорошее качество при масштабировании приложения.

В данном проекте имеются две ссылки: http://localhost:3004/users и http://localhost:3004/books. По умолчанию, происходит GET-запрос, но можно также использовать POST, PUT, PATCH и DELETE, для добавления, обновления и удаления данных.

GET-запрос позволяет получать данные из сервера по заданному URL. Например, GET-запрос к http://localhost:3004/users вернет список пользователей, а GET-запрос к http://localhost:3004/books вернет список книг. Пример того что вернет этот запрос можно увидеть на рисунке 12 и 13.

POST-запрос используется для создания новых данных на сервере. Например, POST-запрос к http://localhost:3004/users создаст нового пользователя, а POST-запрос к http://localhost:3004/books создаст новую книгу.

PUT-запрос используется для обновления существующих данных на сервере. Например, PUT-запрос к http://localhost:3004/users/:id обновит данные пользователя с определенным идентификатором, а PUT-запрос к http://localhost:3004/books/:id обновит данные книги с определенным идентификатором.

PATCH-запрос используется для частичного обновления существующих данных на сервере. Например, PATCH-запрос к http://localhost:3004/users/:id обновит только те данные пользователя, которые были указаны в запросе, а PATCH-запрос к http://localhost:3004/books/:id обновит только те данные книги, которые были указаны в запросе.

DELETE-запрос используется для удаления существующих данных на сервере. Например, DELETE-запрос к http://localhost:3004/users/:id удалит пользователя с определенным идентификатором, а DELETE-запрос к http://localhost:3004/books/:id удалит книгу с определенным идентификатором.

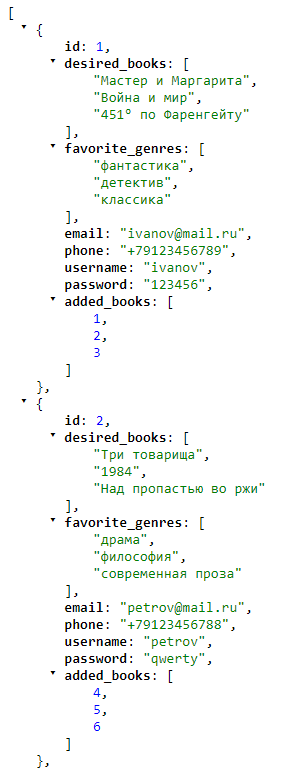


Рисунок 12 − GET запрос к users



Рисунок 13 − GET запрос к books

**2.5 Разработка программного кода**

Перед началом написания кода необходимо создать Telegram бота. Для этого мы обращаемся к BotFather, который уже есть в мессенжере Telegram, изображен на рисунке 2.



Рисунок 2 − BotFather

Данный бот это самый простой и единственный способ регистрации, настройки и управления другими telegram – ботами на данной площадке. Работа с ним не требует особых навыков, с его помощью можно зарегистрировать неограниченное количество новых ботов

Данным ботом можно управлять с помощью команд, которые представлены на рисунке 3

Для создания нового бота нужно ввести команду /start, изображенную на рисунке 3.

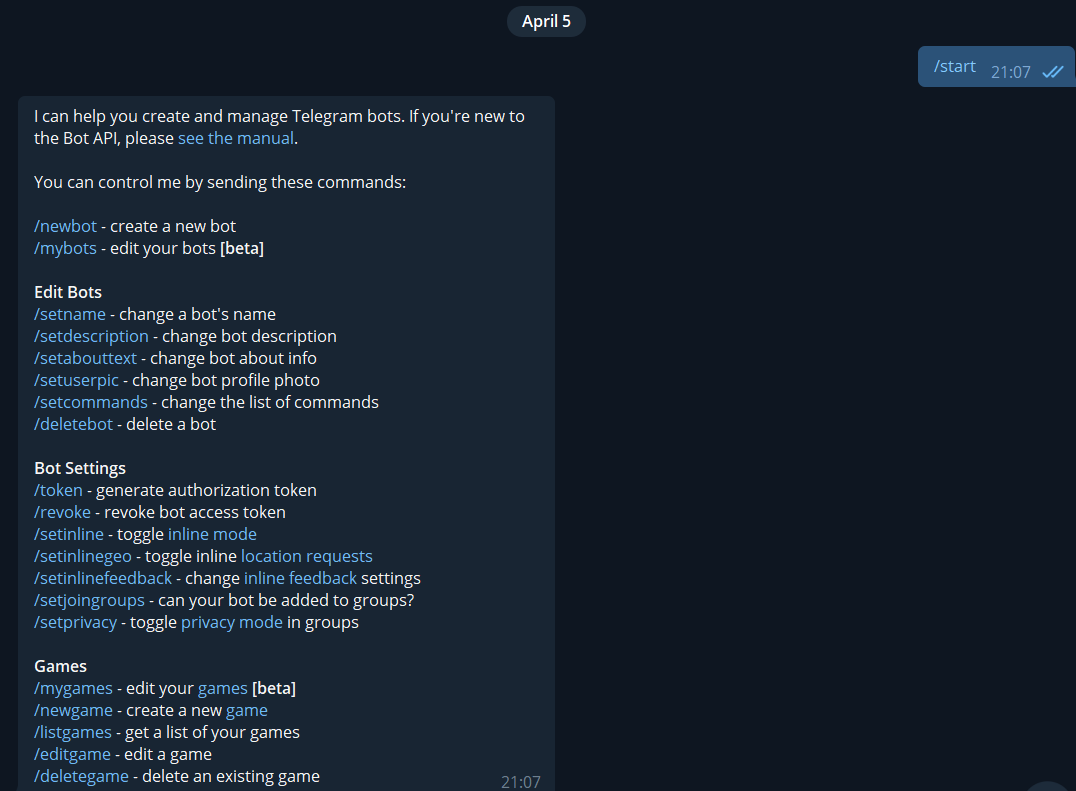


Рисунок 3 − Команда /start

Следующим шагом идет создание имени бота и его ник, это продемонстрировано на рисунке 4, который необходим для поиска среди других пользователей

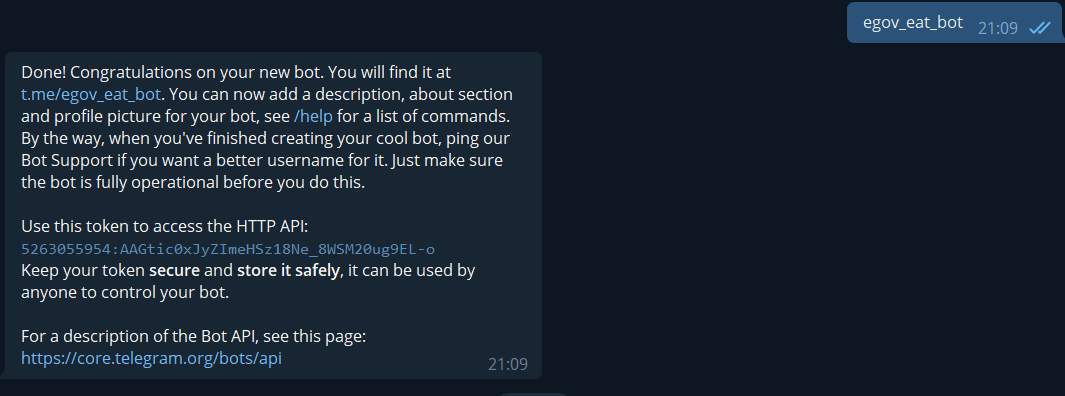


Рисунок 4 − Создание имени бота

Далее мы прописываем описание, при необходимости и команды, команды используемые в данном проекте представлены на рисунке 5

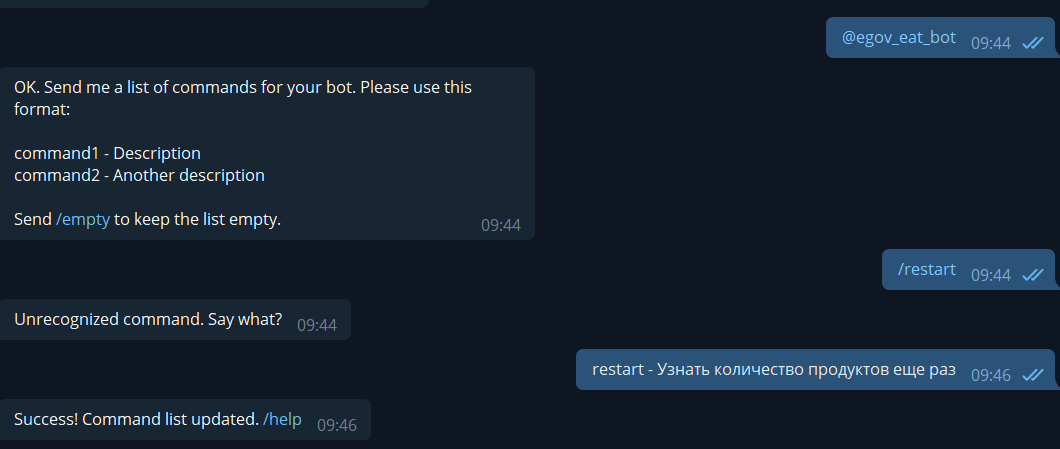


Рисунок 5 − Создание команды

Также бот дает уникальный ключ, изображенный на рисунке 6, который в дальнейшем будет осуществлять связь между пользователем и сервером

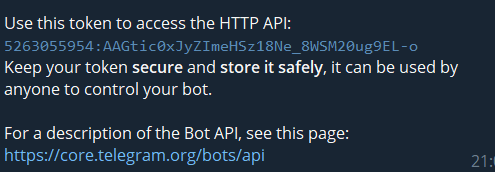


Рисунок 6 − Уникальный ключ

**3.3 API**

Application programming interface, или API это интерфейс, при помощи которого, сервер взаимодействуют с другим сервером, в данном случае это сервер egov с открытыми данными.

Открытые данные предоставленные на данном сайте, являются частью инициативы правительства Республики казахстан и являются отдельным элементом «Электронного правительства» Республики Казахстан.

При помощи «открытых данных», любой желающий может без проблем находить интересующую его информацию, которые разбиты по разным темам и разделам, и предоставляются государственными органами.

Были использованы данные по потреблению основных продуктов питания разными слоями населения на основе открытых данных Республики К

Это специальный набор способов и правил, с помощью которых программы обмениваются данными между собой, и в этом им помогают функции, классы, структуры, к которым обращаются программы.

В случае с данным проектом сервис Telegram обменивается данными с сервисом EGOV представленный на Рисунке 7



Рисунок 7 − Сервис EGOV

Telegram применяет личный протокол шифрования MTProto. MTProto API или же Telegram API − через данное API приложение Telegram связывается с сервером, за счет того, что Telegram API полностью открыт, любой разработчик может без проблем написать свой клиент мессенжер.

Для написания ботов в Telegram был создан специальный TelegramBot Api, чтобы воспользоваться им не нужно знать о том, как он работает, ведь протокол шифрования, указанный выше. Будет сам обрабатывать все шифрование и связь с Telegram API.

**3.4 Добавление библиотек**

Несмотря на то, что множество задач при написании кода можно решить при помощи стандартных средств InteliJIdea, в данном дипломном проекте использовались сторонние библиотеки, так как это было не просто полезно, а необходимо.

Библиотеки используются для упрощения написания кода, работы программы они избегают нагромождения функций в коде и использование сложных конструкций.

В данном случае можно было бы не использовать сторонние библиотеки, но в таком случае были бы сложности с написанием кода, а библиотеки упростили синтаксис программы, и тем самым всю работу.

В данном проекте использовалась библиотека Telegram Bot Java Library. В базовый класс, который реализовывает все взаимодействие бота с пользователем и с Telegram.

Работает данная библиотека с помощью технологии LongPolling, так как он не требует каких либо сертификатов шифрования и одним из его плюсов является запуск программного кода на любом устройстве.

Для корректной работы были использованы методы, представленные на рисунке 8, которые наследуются от класса TelegramLongPollingBot. Такие как:

- public void onUpdateReceived(Update update)

- public String getBotUsername()

- public String getBotToken()

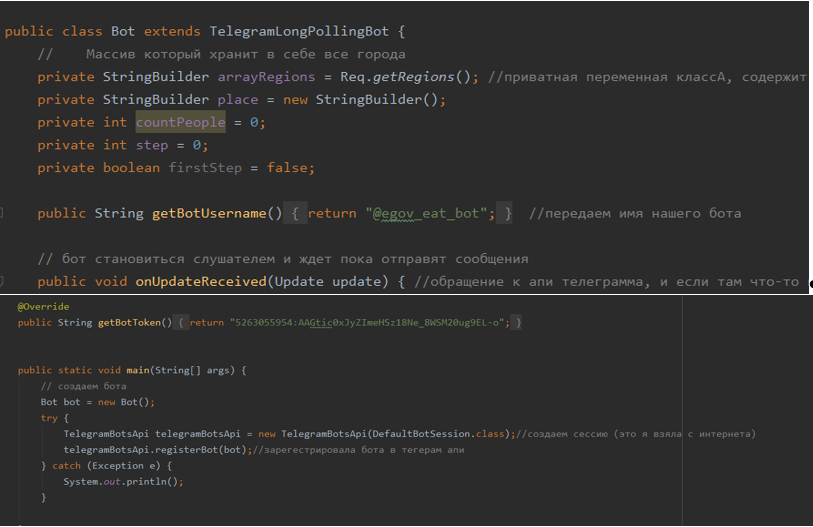


Рисунок 8 − методы класса

**3.5 Написание программы**

В начале написания кода был создан класс Bot, изображённый на рисунке 9, который наследуется от класса используемой библиотеки TelegramLongPollingBot



Рисунок 9 − Класс Bot

После данного этапа было установлено соединение с Telegram ботом, при помощи методов , которые среда попросила реализовать от материнского класса:

* public String getBotUsername()
* public String getBotToken()

Которые показаны на рисунках 10 и 11

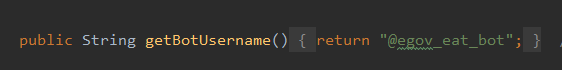


Рисунок 10 − Методы класса

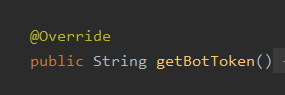


Рисунок 11 − Методы класса

После подключения к Telegram боту в данном проекте был добавлен так называемый «слушатель», через метод onUpdateReceived, изображенный на рисунке 12, который ждет отправки сообщения от пользователя, таких команд как /star или /restar

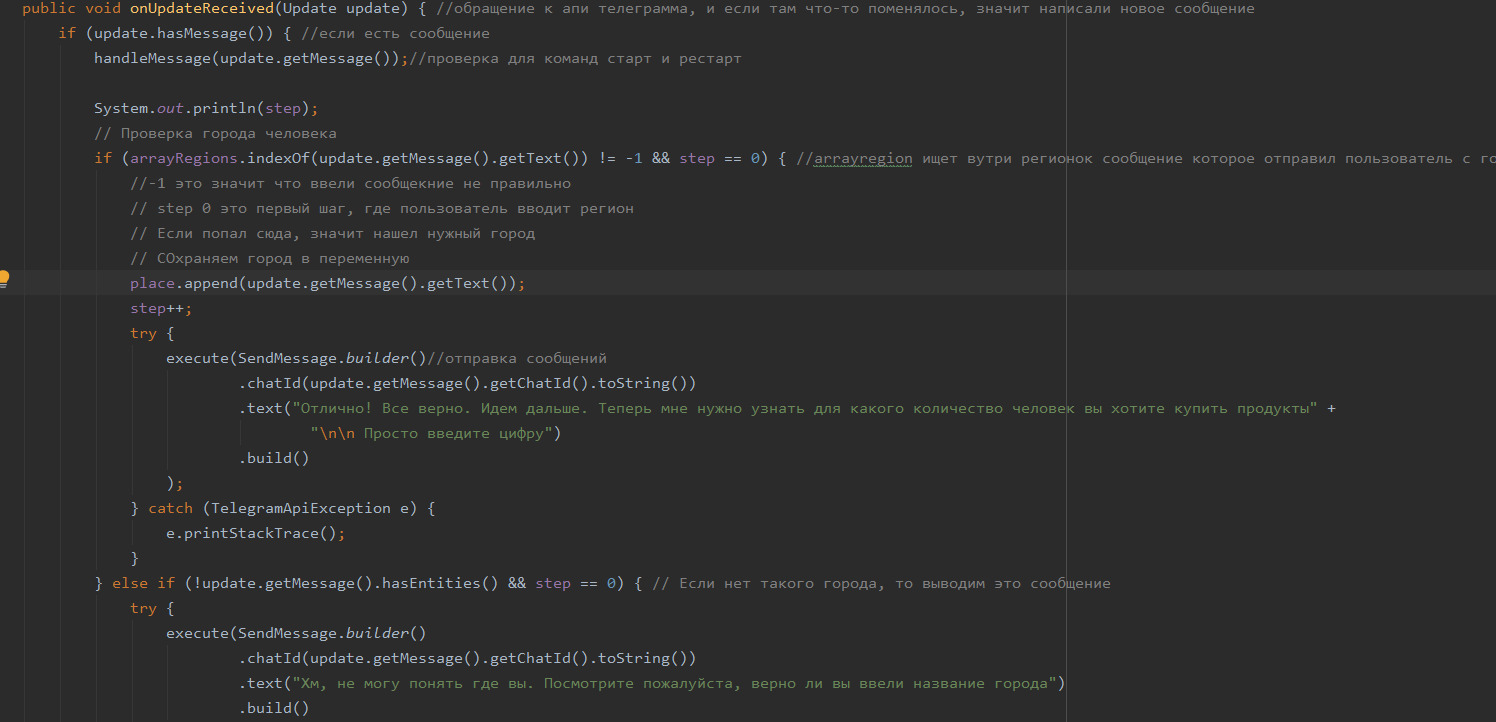


Рисунок 12 − Метод onUpdateReceived

После этого была реализация проверки на наличие команд от пользователя при помощи метода обработки сообщений handleMessage.

Данный метод показан на рисунке 13, также в данном методе присутствует условие с проверкой данного сообщения на текст или команду

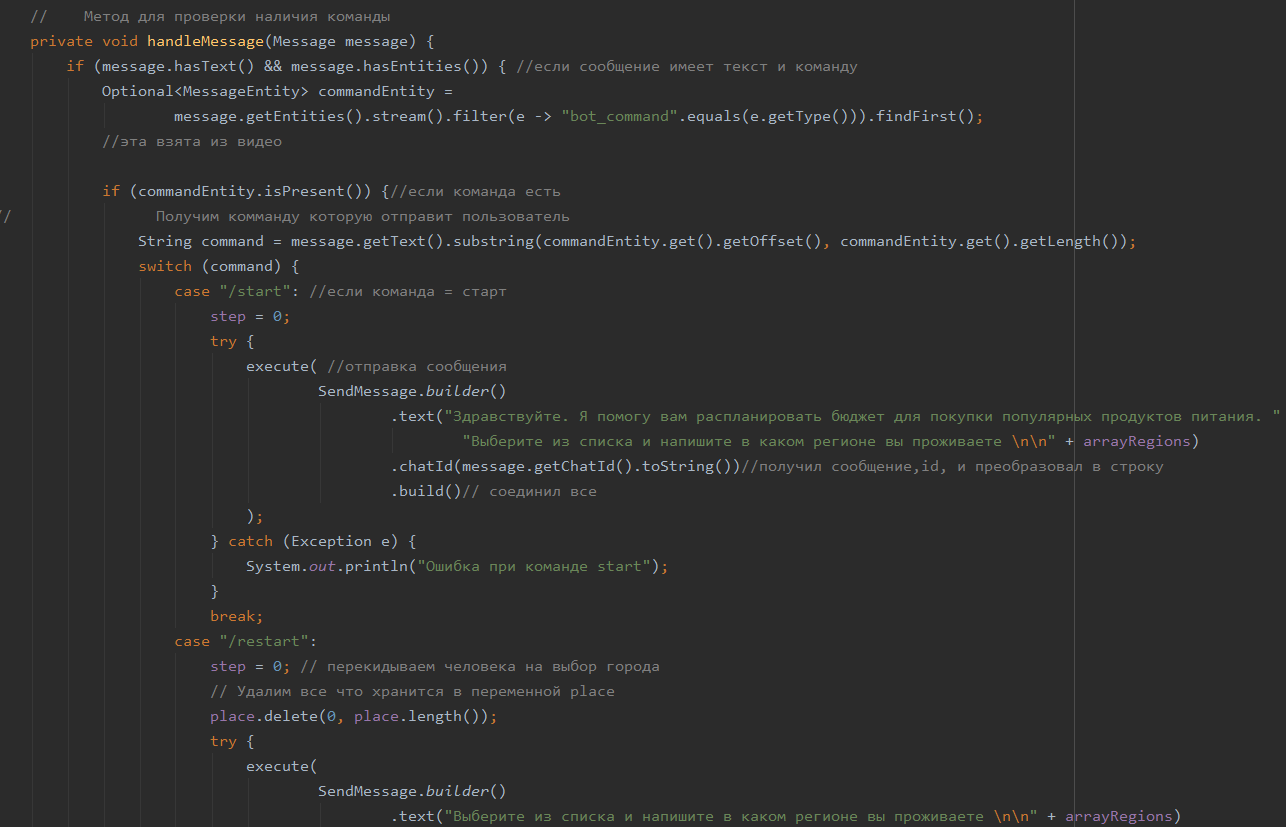


Рисунок 13 − Метод handleMessage

Кроме базовой команды /start, которая является стандартной в каждом боте Telegram, в данном проекте были добавлена команда /restart, через главный бот Telegram – BotFather, о котором было написано выше

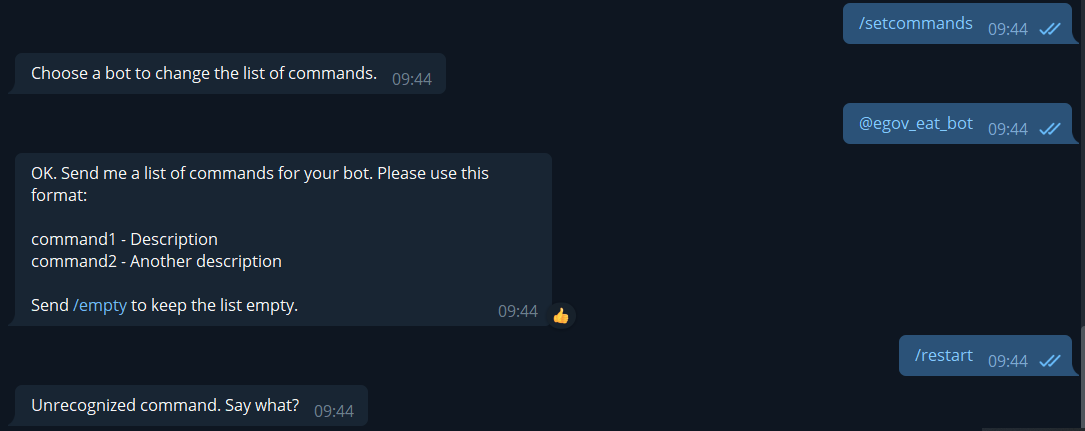


Рисунок 13 − Команда Restart

Команда /restart нужна для того чтобы пользователь мог повторно отправить запрос

Кода пользователь вводит сообщение идет проверка на содержание команды /start или /restart через оператор switch/

Switch это специальный оператор, который последовательно сравнивает введенные команды в чат Telegram, которые находятся после каждого ключевого слова case в коде. Реализация этого оператора изображена на рисунке 14



Рисунок 14 − Реализация оператора switch

Если была введена команда /start пользователю выводится приветственное сообщение и массив городов, которые получены из egov API и инструкция по использованию бота, которая была реализована с помощью метода SendMessage.builder().

В данном методе мы указываем текст сообщения и id чата с пользователем с помощью строк: text("Здравствуйте. Я помогу вам распланировать бюджет для покупки популярных продуктов питания. " +  
 "Выберите из списка и напишите в каком регионе вы проживаете \n\n" + arrayRegions) и chatId(message.getChatId().toString())

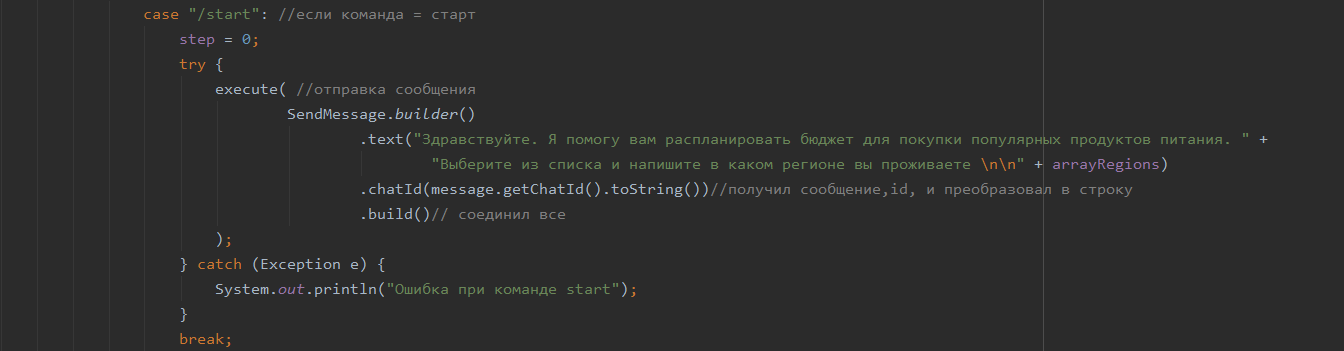


Рисунок 15 − Метод SendMessage.builder

Для того, чтобы подключиться к egov API был реализован класс Req, изображённый на рисунке 16



Рисунок 16 − Класс Req

В данном классе расположен главный метод, который получает данные.

Для этого создаётся защищенное соединение и открывается соединение, изображённое на рисунке 17

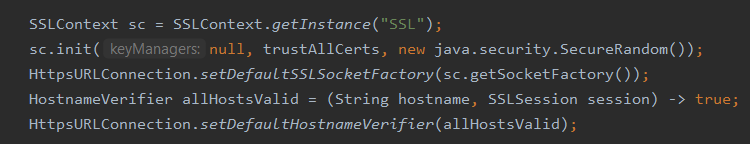


Рисунок 17 − Защищенное соединение

Далее формируется строка, с записью ответа от сервера, изображенная на рисунке 18 в которую записывается ответ, полученный от сервера egov

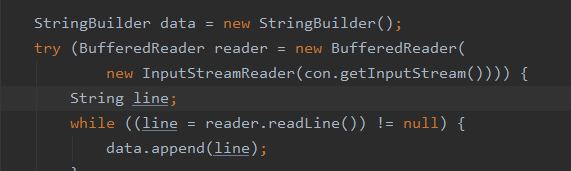


Рисунок 18 − Запись ответа

Для того чтобы вывести пользователю какие города доступны, реализован второй класс getRegions

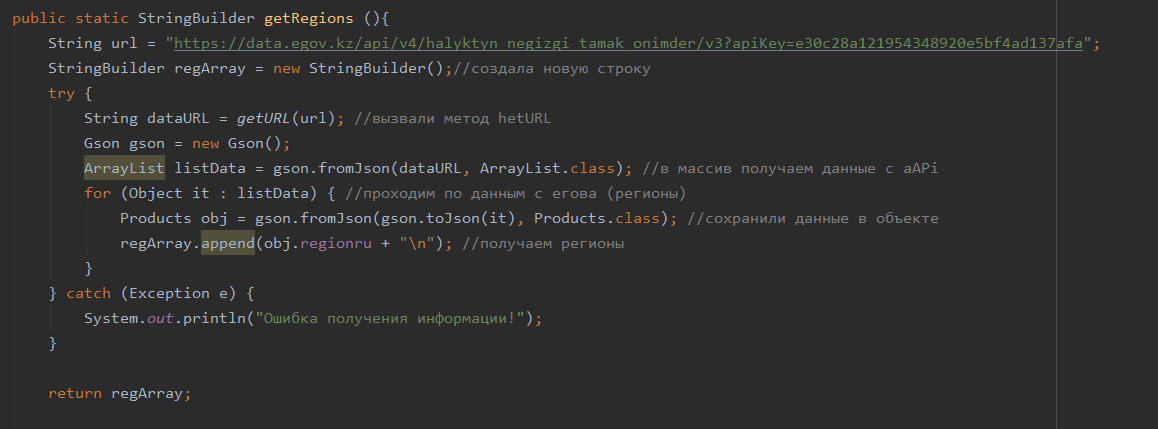


Рисунок 19 − Класс getRegions

Для более удобной разработки полученные данные были преобразованы в формат gson с помощью библиотеки gson, изображенной на рисунке 20



Рисунок 20 − Библиотека Gson

Gson – это библиотека на языке Java c открытым исходным кодом для перевода какой либо необходимой структуры данных в последовательность битов, и обратной данной операции.

Данная библиотека требует, чтобы для данных был создан специальный класс, который должен содержать такие же переменные, как и на сервере egov, как представлены данные на сервере egov, представлено на рисунке 21.

Для того чтобы данные на данном сайте приняли вид как на рисунке 21 необходимо было использовать специальное расширение браузера под название GsonViewer



Рисунок 21 − Egov

Поэтому был реализован класс products, изображенный на рисунке 22, в котором необходимо было прописать данные без ошибочно и точно, как в формате gson

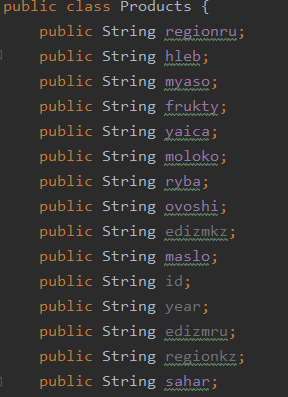


Рисунок 22 − Класс Products

Здесь все переменные имеют такое же наименование, как и в сервере egov.

Для сохранения всех данных был создан массив arrayList, в котором хранился ответ от сервера egov, массив представлен на рисунке 23

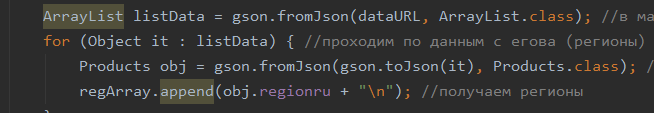


Рисунок 23 − Массив

Далее необходимо извлечь данные для каждого региона с помощью Products obj = gson.fromJson(gson.toJson(it), Products.class), который находится в методе getRegions, представленный на рисунке 24

Также в этом методе прописывается ссылка на Egov Api, в соответствии с полученным ключом Api при регистрации

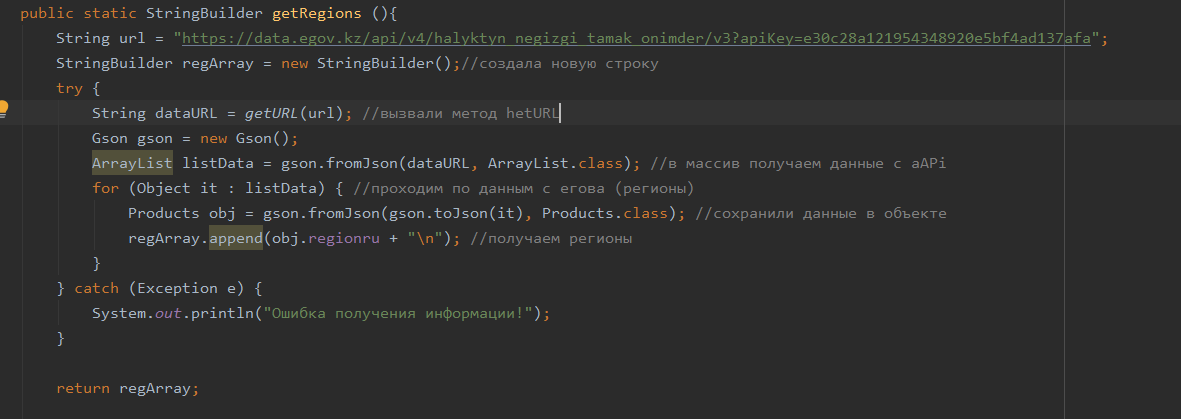


Рисунок 24 − Метод getRegions

После реализации все данные с сервера хранились в объектах obj.

Так как все это делалось для получения списков регионов в строку req.array определенную ранее, из каждого объекта obj было получено информация о регионах через obj.regionru



Рисунок 25 − Получение регионов

После работы цикла, изображенного на рисунке выше, было получено готовую строку со всеми городами, и данную строку мы отправляли пользователю в приветственном сообщении.

После этого необходимо было дождаться ответа пользователя чтобы он точно указал название города, для проверки было создано условие, изображенное на рисунке 26



Рисунок 26 − Условие

Если пользователь ввел не корректный ответ ему отправлялось сообщение, представленное на рисунке 27

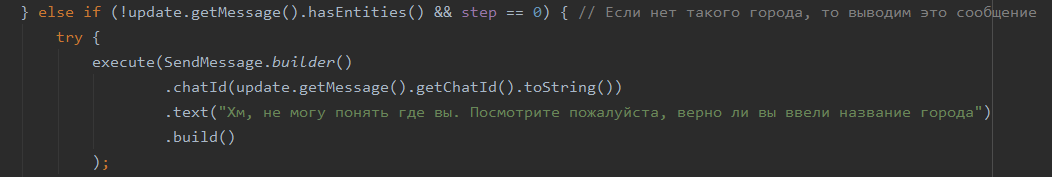


Рисунок 27 − Получение сообщения

В версии Telegram сообщение представлено в видео, показанном на рисунке 28, дальше бот ждет корректное ответное сообщение от пользователя, при введении некорректного ответа бот будет отправлять данное сообщение столько раз, сколько получит неверный ответ

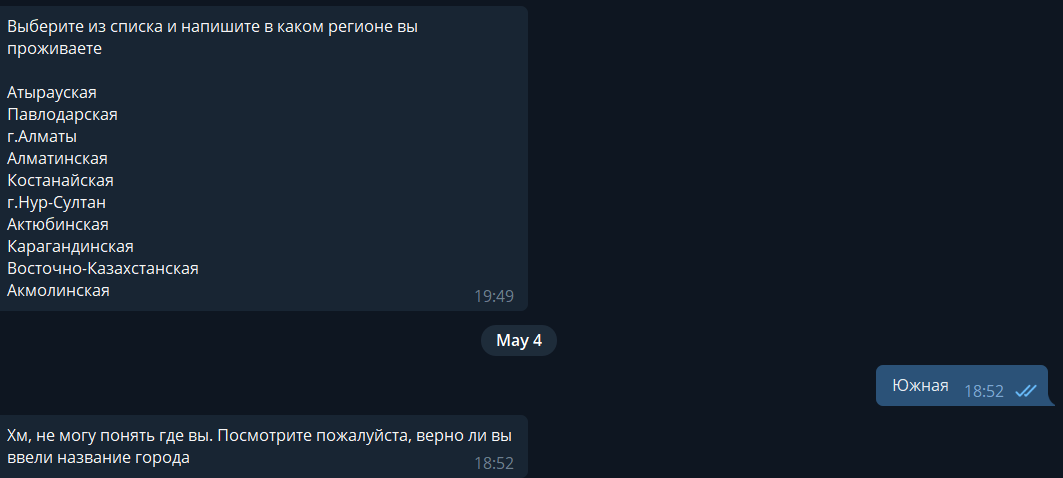


Рисунок 28 − Некорректное введение данных

Если данные введены корректно, переходим к следующему шагу. Следующий шаг — это вывод сообщения: «Отлично! Все верно. Теперь мне нужно узнать для какого количества человек вы хотите купить продукты», в котором бот узнает точное количество человек

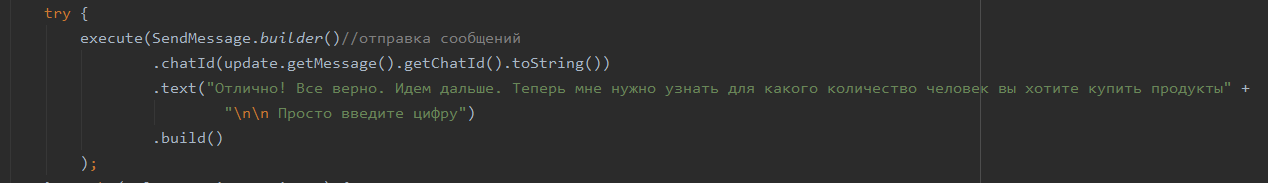


Рисунок 29 − Вывод сообщения

На этом этапе бот ждет чтобы пользователь ввел для какого количества человек необходимо купить продуктов.

Так же здесь реализована защита c помощью условия if, представленная на рисунке 30 , которое защищает бота от некорректного введения данных, например цифр меньше 0, или слов.

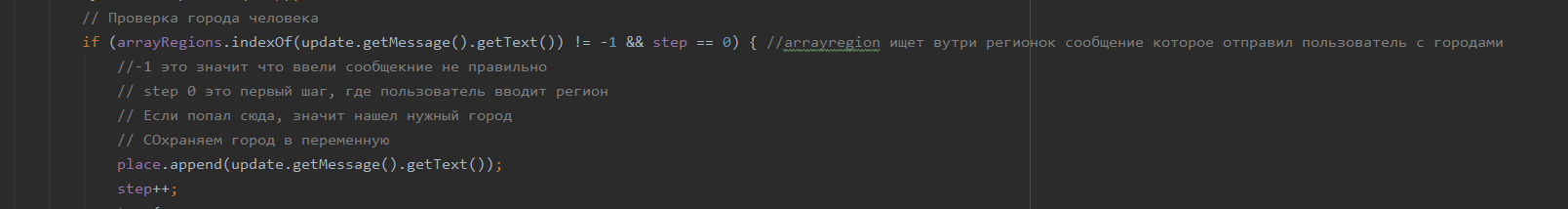


Рисунок 30 − Защита от некорректного введения

Если был введен текст, то ничего не происходит, если введено число равное или меньше 0, выводится сообщение, представленное на рисунке 31

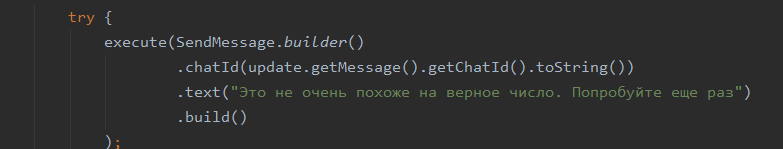


Рисунок 31 − Сообщение при некорректном введении данных

Если же данные корректны, идет следующий запрос на сервер. Для этого в классе req реализован метод getInfo.

Данный метод получает ответ от пользователя, регион и количество человек, он работает по такому же принципу что и getRegions, описанный выше, за исключением условия, представленного на рисунке 32

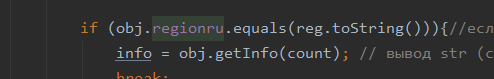


Рисунок 32 − Условие

Далее из всех регионов мы выбираем тот регион, который ввел пользователь и запрашиваем информацию про этот регион.

Полную информацию про этот регион подготавливает класс products с помощью метода getInfo, представленного на рисунке 33

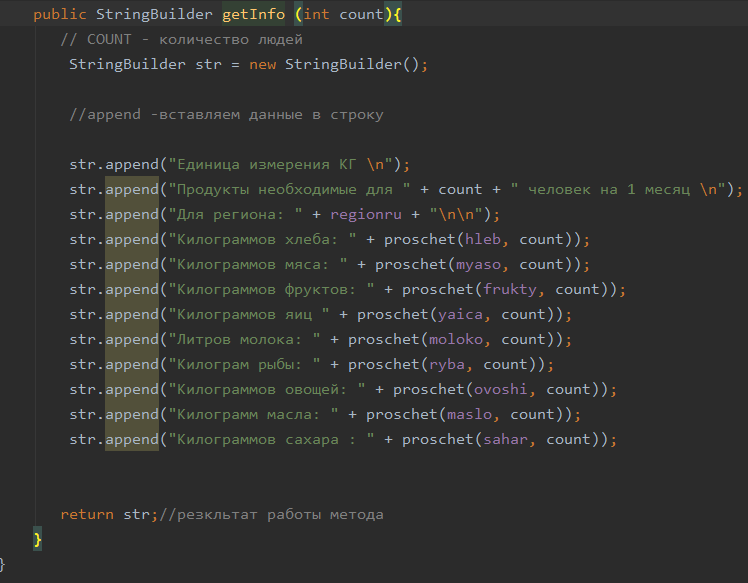


Рисунок 33 − Метод getInfo

Так как внутри объекта хранятся данные от сервера, то нам эти данные необходимо вывести для пользователя в понятном для него формате, что было сделано с помощью StringBuilder и пояснению к каждому из полей. Пояснение к каждому из полей берется с сайта открытых данных egov, в данном случае это продукты, необходимо ввести данные также как они представлены на сайте.

После данного этапа, необходимо вернуться в класс bot

Здесь отправляется пользователю сообщение об обработке данных, представленный на рисунке 34



Рисунок 34 − Сообщение об обработке данных

Когда был получен ответ от сервера пользователю сразу отправляется сообщение, которое было получено от метода getInfo, представленный на рисунке 35

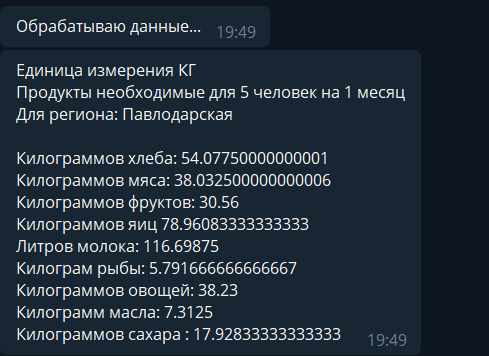


Рисунок 35 − Вывод данных

В случае если пользователь хочет воспользоваться ботом повторно, ему предлагается ввести команду /restart, бот также выводит это сообщение, как пользователь видит сообщение представлено на рисунке 36

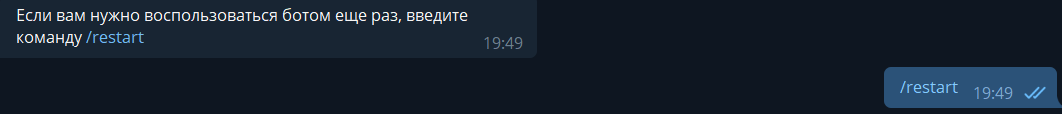


Рисунок 36 − Команда restart

Команда /restart реализуется с помощью метода execute, представленный на рисунке 37

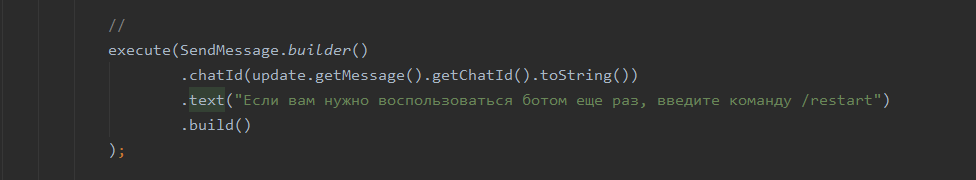


Рисунок 37 − Метод execute

Когда пользователь нажимает команду /restart, ему заново отправляется список со всеми городами и работа бота повторяется заново, код данной команды представлен на рисунке 38

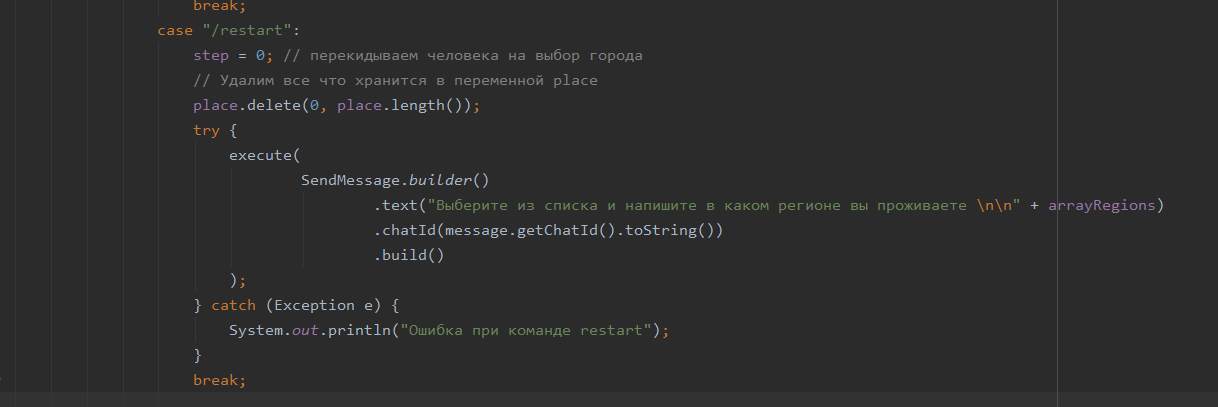


Рисунок 38 − Обновление работы бота

**Заключение**

В настоящее время популярность мессенджером растет. Также стоит отметить рост популярности Telegram, как приводилось выше мессенджер насчитывает более 100 миллионов пользователей, и соответственно Telegram ботов, которые работают на данной платформе.

Благодаря социальным сетям мы можем не только общаться, делиться фотографиями, но и автоматизировать процессы и передавать рутинную работу службам автоматизации, в нашем случае это Telegram боты.

В рамках данной работы были выполнены поставлены задачи. В начале работы были изучены материалы и ресурсы, которые помогли в построении и реализации Telegram бота, также были выявлены возможности применения, функции и технологии ботов в Telegram, и в конечном итоге спроектирован и разработан чат бот.

Тема данного дипломного проекта является наиболее острой и актуальной, ведь это не просто получение данных из открытых источников, которое предоставляет государство, а возможность сэкономить средства как для людей с разным уровнем достатка, так и для благотворительных фондов, которые являются основной целевой аудиторией данного TELEGRAM бота.

В настоящее время идет тенденция на благотворительность, сохранение природы и экологии, и несмотря на то, что данный проект разрабатывался в сфере IT, он может помочь решить такие проблемы как:

- проблема экономической доступности продуктов питания

- расчёт количества продуктов питания, необходимых для покупки в благотворительных фондах

Открытые данные мы можем использовать не только для аналитической деятельности, но и для практического использования. Также сейчас стоит остро проблема автоматизации процессов в различных социально экономических областях. В результате автоматизации процессов освобождается человеческий ресурс, который в свою очередь может быть реализован в других областях, тем самым это повышает эффективность работы.

Практическая значимость данного проекта, заключается в том, что данный TELEGRAM бот можно использовать как для личных целей, так и для социально значимых проектов

**Список использованных источников**

1 Н. А. Вязовик. Программирование на Java Курс лекций, Интернет университет информационных технологий, 2003. − 592 с.

2 О. В. Герман. − Санкт-Петербург.: Программирование на Java и C#, 2012. − 512 c.

3 Берд, Барри Java для чайников / Барри Берд. − М. : Диалектика / Вильямс, 2013. − 521c.

4 Эккель, Брюс Философия Java / Брюс Эккель. − М. : Питер, 2016. − 809c.

5 Создание Telegram бота на Java: от идеи до деплоя [https://javarush.ru/groups/posts/504-sozdanie-telegram-bota-na-java-ot-idei-do-deploja]

**Приложение A**

import org.telegram.telegrambots.bots.DefaultBotOptions;  
import org.telegram.telegrambots.bots.TelegramLongPollingBot;  
import org.telegram.telegrambots.meta.TelegramBotsApi;  
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;  
import org.telegram.telegrambots.meta.api.objects.Message;  
import org.telegram.telegrambots.meta.api.objects.MessageEntity;  
import org.telegram.telegrambots.meta.api.objects.Update;  
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;  
import org.telegram.telegrambots.updatesreceivers.DefaultBotSession;  
  
import java.util.ArrayList;  
import java.util.Optional;  
  
  
public class Bot extends TelegramLongPollingBot {  
 // Массив который хранит в себе все города  
 private StringBuilder arrayRegions = Req.getRegions(); //приватная переменная классА, содержит в себе все города  
 private StringBuilder place = new StringBuilder();  
 private int countPeople = 0;  
 private int step = 0;  
 private boolean firstStep = false;  
  
 public String getBotUsername() {  
 return "@egov\_eat\_bot";  
 } //передаем имя нашего бота  
  
 // бот становиться слушателем и ждет пока отправят сообщения  
 public void onUpdateReceived(Update update) { //обращение к апи телеграмма, и если там что-то поменялось, значит написали новое сообщение  
 if (update.hasMessage()) { //если есть сообщение  
 handleMessage(update.getMessage());//проверка для команд старт и рестарт  
  
 System.out.println(step);  
 // Проверка города человека  
 if (arrayRegions.indexOf(update.getMessage().getText()) != -1 && step == 0) { //arrayregion ищет вутри регионок сообщение которое отправил пользователь с городами  
 //-1 это значит что ввели сообщекние не правильно  
 // step 0 это первый шаг, где пользователь вводит регион  
 // Если попал сюда, значит нашел нужный город  
 // СОхраняем город в переменную  
 place.append(update.getMessage().getText());  
 step++;  
 try {  
 execute(SendMessage.builder()//отправка сообщений  
 .chatId(update.getMessage().getChatId().toString())  
 .text("Отлично! Все верно. Идем дальше. Теперь мне нужно узнать для какого количество человек вы хотите купить продукты" +  
 "\n\n Просто введите цифру")  
 .build()  
 );  
 } catch (TelegramApiException e) {  
 e.printStackTrace();  
 }  
 } else if (!update.getMessage().hasEntities() && step == 0) { // Если нет такого города, то выводим это сообщение  
 try {  
 execute(SendMessage.builder()  
 .chatId(update.getMessage().getChatId().toString())  
 .text("Хм, не могу понять где вы. Посмотрите пожалуйста, верно ли вы ввели название города")  
 .build()  
 );  
 } catch (TelegramApiException e) {  
 e.printStackTrace();  
 }  
 }  
  
 // Проверка количества человек  
 if (Integer.parseInt(update.getMessage().getText()) > 0 && step == 1) {//проверка на число  
 countPeople = Integer.parseInt(update.getMessage().getText());//сохраняем кол-во человек в переменную  
 step++;  
 try {  
 execute(SendMessage.builder()  
 .chatId(update.getMessage().getChatId().toString())  
 .text("Обрабатываю данные...")  
 .build()  
 );  
  
// ЗАПРОС К ЕГОВ И ОТРАВКА ДАННЫХ ПОЛЬЗОВАТЕЛЮ  
 execute(SendMessage.builder()  
 .chatId(update.getMessage().getChatId().toString())  
 .text(Req.getInfo(place, countPeople).toString()) //передали текст с req (str) и бот сам вывел  
 .build()  
 );  
  
 //  
 execute(SendMessage.builder()  
 .chatId(update.getMessage().getChatId().toString())  
 .text("Если вам нужно воспользоваться ботом еще раз, введите команду /restart")  
 .build()  
 );  
  
  
 } catch (TelegramApiException e) {  
 e.printStackTrace();  
 }  
 } else if (step == 1) { // Если это не число, то выводим это сообщение  
 try {  
 execute(SendMessage.builder()  
 .chatId(update.getMessage().getChatId().toString())  
 .text("Это не очень похоже на верное число. Попробуйте еще раз")  
 .build()  
 );  
 } catch (TelegramApiException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 // Метод для проверки наличия команды  
 private void handleMessage(Message message) {  
 if (message.hasText() && message.hasEntities()) { //если сообщение имеет текст и команду  
 Optional<MessageEntity> commandEntity =  
 message.getEntities().stream().filter(e -> "bot\_command".equals(e.getType())).findFirst();  
 //эта взята из видео  
  
 if (commandEntity.isPresent()) {//если команда есть  
// Получим комманду которую отправит пользователь  
 String command = message.getText().substring(commandEntity.get().getOffset(), commandEntity.get().getLength());  
 switch (command) {  
 case "/start": //если команда = старт  
 step = 0;  
 try {  
 execute( //отправка сообщения  
 SendMessage.builder()  
 .text("Здравствуйте. Я помогу вам распланировать бюджет для покупки популярных продуктов питания. " +  
 "Выберите из списка и напишите в каком регионе вы проживаете \n\n" + arrayRegions)  
 .chatId(message.getChatId().toString())//получил сообщение,id, и преобразовал в строку  
 .build()// соединил все  
 );  
 } catch (Exception e) {  
 System.out.println("Ошибка при команде start");  
 }  
 break;  
 case "/restart":  
 step = 0; // перекидываем человека на выбор города  
 // Удалим все что хранится в переменной place  
 place.delete(0, place.length());  
 try {  
 execute(  
 SendMessage.builder()  
 .text("Выберите из списка и напишите в каком регионе вы проживаете \n\n" + arrayRegions)  
 .chatId(message.getChatId().toString())  
 .build()  
 );  
 } catch (Exception e) {  
 System.out.println("Ошибка при команде restart");  
 }  
 break;  
  
 }  
 }  
 }  
 }  
  
 @Override  
 public String getBotToken() {  
 return "5263055954:AAGtic0xJyZImeHSz18Ne\_8WSM20ug9EL-o";  
 }  
  
  
 public static void main(String[] args) {  
 // создаем бота  
 Bot bot = new Bot();  
 try {  
 TelegramBotsApi telegramBotsApi = new TelegramBotsApi(DefaultBotSession.class);//создаем сессию (это я взяла с интернета)  
 telegramBotsApi.registerBot(bot);//зарегестрировала бота в тегерам апи  
 } catch (Exception e) {  
 System.out.println();  
 }  
  
 }  
  
  
}

public class Products {  
 public String regionru;  
 public String hleb;  
 public String myaso;  
 public String frukty;  
 public String yaica;  
 public String moloko;  
 public String ryba;  
 public String ovoshi;  
 public String edizmkz;  
 public String maslo;  
 public String id;  
 public String year;  
 public String edizmru;  
 public String regionkz;  
 public String sahar;  
  
 @Override  
 public String toString() {  
 return "Регион = " + regionru;  
 }  
  
 private String proschet (String a, int b){  
 return String.valueOf(Double.parseDouble(a) \* b / 12) + "\n";  
 }  
  
 public StringBuilder getInfo (int count){  
 // COUNT - количество людей  
 StringBuilder str = new StringBuilder();  
  
 //append -вставляем данные в строку  
  
 str.append("Единица измерения КГ \n");  
 str.append("Продукты необходимые для " + count + " человек на 1 месяц \n");  
 str.append("Для региона: " + regionru + "\n\n");  
 str.append("Килограммов хлеба: " + proschet(hleb, count));  
 str.append("Килограммов мяса: " + proschet(myaso, count));  
 str.append("Килограммов фруктов: " + proschet(frukty, count));  
 str.append("Килограммов яиц " + proschet(yaica, count));  
 str.append("Литров молока: " + proschet(moloko, count));  
 str.append("Килограм рыбы: " + proschet(ryba, count));  
 str.append("Килограммов овощей: " + proschet(ovoshi, count));  
 str.append("Килограмм масла: " + proschet(maslo, count));  
 str.append("Килограммов сахара : " + proschet(sahar, count));  
  
  
 return str;//резкльтат работы метода  
 }  
}

import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.net.MalformedURLException;  
import java.net.URL;  
import java.net.URLConnection;  
import java.security.KeyManagementException;  
import java.security.NoSuchAlgorithmException;  
import java.security.cert.X509Certificate;  
import java.util.ArrayList;  
import javax.net.ssl.HostnameVerifier;  
import javax.net.ssl.HttpsURLConnection;  
import javax.net.ssl.SSLContext;  
import javax.net.ssl.SSLSession;  
import javax.net.ssl.TrustManager;  
import javax.net.ssl.X509TrustManager;  
import com.google.gson.Gson;  
  
public class Req {  
//главный метод класса, он получает данные  
 public static String getURL(String urlStr) throws MalformedURLException, IOException, NoSuchAlgorithmException, KeyManagementException {  
  
 URL url = new URL(urlStr);  
  
 TrustManager[] trustAllCerts = new TrustManager[]{new X509TrustManager() {  
 @Override  
 public java.security.cert.X509Certificate[] getAcceptedIssuers() { return null; }  
 @Override  
 public void checkClientTrusted(X509Certificate[] certs, String authType) { }  
 @Override  
 public void checkServerTrusted(X509Certificate[] certs, String authType) { }  
 }};  
  
 SSLContext sc = SSLContext.getInstance("SSL");  
 sc.init(null, trustAllCerts, new java.security.SecureRandom());  
 HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());  
 HostnameVerifier allHostsValid = (String hostname, SSLSession session) -> true;  
 HttpsURLConnection.setDefaultHostnameVerifier(allHostsValid);  
  
 URLConnection con = url.openConnection();  
  
 StringBuilder data = new StringBuilder();  
 try (BufferedReader reader = new BufferedReader(  
 new InputStreamReader(con.getInputStream()))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 data.append(line);  
 }  
 } catch (IOException e) {  
 }  
 return data.toString();  
 }  
  
  
 public static StringBuilder getRegions (){  
 String url = "https://data.egov.kz/api/v4/halyktyn\_negizgi\_tamak\_onimder/v3?apiKey=e30c28a121954348920e5bf4ad137afa";  
 StringBuilder regArray = new StringBuilder();//создала новую строку  
 try {  
 String dataURL = getURL(url); //вызвали метод hetURL  
 Gson gson = new Gson();  
 ArrayList listData = gson.fromJson(dataURL, ArrayList.class); //в массив получаем данные с аAPi  
 for (Object it : listData) { //проходим по данным с егова (регионы)  
 Products obj = gson.fromJson(gson.toJson(it), Products.class); //сохранили данные в объекте  
 regArray.append(obj.regionru + "\n"); //получаем регионы  
 }  
 } catch (Exception e) {  
 System.out.println("Ошибка получения информации!");  
 }  
  
 return regArray;  
  
 }  
  
// Метод для получения инфы от сервера с учетеом полученных данных от пользователя  
 public static StringBuilder getInfo (StringBuilder reg, int count){  
 String url = "https://data.egov.kz/api/v4/halyktyn\_negizgi\_tamak\_onimder/v3?apiKey=e30c28a121954348920e5bf4ad137afa";  
 StringBuilder info = new StringBuilder();  
 try {  
 String dataURL = getURL(url); //вызвал метод  
 Gson gson = new Gson();  
 ArrayList listData = gson.fromJson(dataURL, ArrayList.class); //в массив получаем данные с апи  
 for (Object it : listData) {//проходим по данным егова  
 Products obj = gson.fromJson(gson.toJson(it), Products.class);// сохранили данные в объекте,которые в классе product.java  
  
 if (obj.regionru.equals(reg.toString())){//если регион в массиве и регион в сообщении одинаковые, то переходим к рассчетам  
 info = obj.getInfo(count); // вывод str (сообзщение с кг мяса)  
 break;  
 }  
 }  
 } catch (Exception e) {  
 System.out.println("Ошибка получения информации!");  
 }  
  
 return info;  
  
 }  
}