



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)

Кафедра инструментального и прикладного программного обеспечения (ИиПО)

КУРСОВАЯ РАБОТА

по дисциплине: Разработка клиент-серверных приложений

по профилю: Разработка программных продуктов и проектирование информационных систем
направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: Разработка и развёртывание в облачном сервисе клиент-серверного фуллстек-приложения для контроля выдачи книг в библиотеке

Студент: Московка Артём Александрович

Группа: ИКБО-20-19

Работа представлена к защите 15.05.2022 (дата)

_____/Московка А.А./
(подпись и ф.и.о. студента)

Руководитель: доцент, Куликов Александр Анатольевич

Работа допущена к защите 15.05.2022 (дата)

_____/Куликов А.А./
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: _____

_____/ 21.06.2022, доцент, Куликов Александр Анатольевич /

_____/ 21.06.2022, _____ /

(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

2022 г.

Синицын
Синицын Анатолий Васильевич

Заведующему кафедрой
инструментального и прикладного
программного обеспечения (ИиППО)
Института информационных технологий (ИТ)
Болбакову Роману Геннадьевичу
От студента Московки

Артёма Александровича
ФИО
ИКБО-20-19
группа
3 курс
курс

Контактный номер: 7 926 706 59 15

Заявление

Прошу утвердить мне тему курсовой работы по дисциплине «Разработка клиент-серверных приложений» образовательной программы бакалавриата 09.03.04 (Программная инженерия)

Тема: Разработка и развёртывание в облачном сервисе клиент-серверного фуллстек-приложения для контроля выдачи книг в библиотеке

Приложение: лист задания на КР/КП в 2-ух экземплярах на двухстороннем листе (проект)

Подпись студента

Маслов А. А.
подпись ФИО

Дата

26.02.2022

Подпись руководителя

Романов
подпись Должность, ФИО

Дата

26.02.2022



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА - Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине: Разработка клиент-серверных приложений
по профилю: Разработка программных продуктов и проектирование информационных систем

Студент: Московка Артём Александрович
Группа: ИКБО-20-19
Срок представления к защите: 20.05.2022
Руководитель: доцент, Куликов Александр Анатольевич

Тема Разработка и развёртывание в облачном сервисе клиент-серверного фуллстек-приложения для контроля выдачи книг в библиотеке

Исходные данные: Flutter, Dart, BLoC, Cubit, DI, WebSocket, CI, Git, GitHub

Перечень вопросов, подлежащих разработке, и обязательного графического материала:

1. Провести анализ предметной области для выбранной темы с обоснованием выбора клиент-серверной архитектуры для разрабатываемого приложения; 2. Выбрать, программный стек для реализации проекта; 3. Дать описание архитектуры разрабатываемого клиент-серверного приложения 4. Провести реализацию фронтенд и бекенд части клиент-серверного приложения, обеспечив версионный контроль процесса разработки с помощью Git. 6. Интегрировать проект на GitHub с Heroku, с целью развёртывания разработанного клиент-серверного приложения в облаке. 7. Провести тестирование разработанного продукта 8. Разработать презентацию с графическими материалами.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Зав. кафедрой ИиППО: Болбаков Р. Г. /, «26» 05 2022 г.

Задание на КР выдал: Куликов А.А. /, «26» 02 2022 г.

Задание на КР получил: Московка А.А. /, «26» 02 2022 г.

Аннотация

В данной курсовой работе содержится пять основных разделов.

Основные разделы имеют следующие названия: введение, основная часть, заключение, информационные источники, приложения.

Весь текст курсовой работы занимает 36 страниц. Работа содержит 27 рисунков, 2 приложения, а также 24 информационных источника. Для защиты курсовой работы была сделана презентация, отражающая ключевые моменты аналитической и практической составляющей работы.

Данная курсовая работа направлена на исследование и разработку программного продукта, позволяющего вести учет выданных в библиотеке книг с возможностью узнать актуальных читателей.

Оглавление

Введение.....	7
Основная часть	8
1. Общие сведения	8
1.1. Обозначение и наименование программы.....	8
1.2. ПО, необходимое для функционирования программы.....	8
1.3. Языки программирования, на которых написана программа.....	8
2. Функциональное назначение.....	8
3. Описание логической структуры.....	9
3.1. Анализ предметной области.....	9
3.1.1. Описание предметной области	9
3.1.2. Анализ аналогов разрабатываемого приложения	9
3.2. Методология разработки сервиса	10
3.2.1. Выбор языка разработки.....	10
3.2.2. Выбор инструментов для разработки интерфейса приложения.....	10
3.2.3. Выбор среды для разработки приложения и БД	10
3.3. Реализация сервиса.....	13
3.3.1. Стадии и этапы разработки	13
3.3.2. Структурная схема сервиса	13
3.3.3. UML-диаграмма структуры БД	14
3.3.4. UML-диаграмма деятельности.....	14
3.3.5. Разработка серверной части	16
3.3.6. UML-диаграмма классов.....	19
3.3.7. Разработка интерфейса приложения	20
3.3.8. Выводы к разделу	25
4. Руководство по использованию сервиса	26
5. Вызов и загрузка	31
Заключение	33

Информационные источники.....	34
Приложения	36

Введение

Актуальность выбранной темы крайне высокая по той причине, что в настоящее время все еще сохраняется и поддерживается возможность брать бумажные книги и учебные пособия из библиотек без необходимости покупки книг конечными пользователями библиотек, а также с целью более бережного обращения с ограниченными на земле органическими ресурсами, такими, как древесина и материалы для печатных чернил.

Но у многих библиотек сохраняется советская система учета выданных книг с использованием бумажных карточек или читательских билетов, а такие информационные носители подвержены материальному износу и могут быть потеряны или испорчены.

Целью же данной работы является создание сервиса, позволяющего вести учет выданных книг в библиотеке с возможностью узнать нынешнего читателя. Для достижения данной цели были выдвинуты следующие задачи: изучение вопроса о содержимом читательских билетов, разработка информационной системы и внедрение этой системы в программную реализацию с возможностью последующего использования данного программного продукта.

Объектом исследования в рамках данной работы являются библиотеки с использующимися в них читательскими билетами, необходимыми для ведения учета выданных и полученных обратно бумажных изданий научной, технической и художественной литературы.

Основная часть

1. Общие сведения

1.1. Обозначение и наименование программы

Полное наименование системы: «Система контроля выдачи книг в библиотеке»

Краткое наименование системы: «СКВКБ»

1.2. ПО, необходимое для функционирования программы

Для функционирования ПО на стороне сервера необходимо наличие установленного DartSDK. Для работы фронт-энд составляющей ресурса требуется наличие FlutterSDK. Для поддержки взаимодействия с БД требуется установленный пакет работы с базами данных Hive [1-3].

Для взаимодействия с программным продуктом со стороны пользователей требуется наличие браузера, например Google Chrome или Mozilla Firefox, а также стабильного интернет-соединения, так как сервис развернут в облачном пространстве в сети Интернет [4-5].

1.3. Языки программирования, на которых написана программа

Основным языком для программирования является Dart, функционал веб-сервера реализован с помощью пакета Shelf. База данных написана на noSQL языке Hive. Фронт-энд реализован на фреймворке Flutter. Развертывание происходит в попавшем под санкции февраля 2022 года сервисе облачного развертывания Heroku [3, 6-9].

2. Функциональное назначение

Функциональные назначения перечислены ниже:

- 1) Авторизация для библиотекарей;
- 2) Добавление новых книг;

- 3) Удаление ранее добавленных книг;
- 4) Внесение изменений в доступность определенной книги с уточнением действующего читателя;
- 5) Добавление названия книге, автора и изображения обложки.

3. Описание логической структуры

3.1. Анализ предметной области

3.1.1. Описание предметной области

В качестве предметной области рассматривается раздел деятельности библиотекарей по контролю и учету имеющихся в наличии бумажных изданий книг, которые могут быть взяты посетителями библиотеки с имеющимся у них документом, подтверждающим право на пользование услугами библиотеки. У студентов и преподавателей, если рассматривать на примере библиотеки при университете, таким документом будет являться студенческий или преподавательский билет. У районных и городских библиотек таким документом могут являться специализированные читательские билеты, которые получают посетители библиотек при первом визите. Но в контексте разрабатываемого программного продукта наличие документа, подтверждающего право на пользование услугами библиотеки, не предусмотрено, поскольку этот уже другой раздел деятельности библиотекарей и такое требование не было представлено в выбранном варианте на выполнение курсовой работы.

3.1.2. Анализ аналогов разрабатываемого приложения

Популярных специализированных решений не было обнаружено. Для анализа аналогов разрабатываемого приложения и предметной области было совершено неоднократное посещение Научно-технической библиотеки МИРЭА. В НТБ библиотеке МИРЭА все еще используются бумажные читательские билеты, которые вставляются в приклеенный кармашек первого разворота обложки книги,

что может свидетельствовать об отсутствии конкурентов и аналогов на рынке как таковых. В сети Интернет можно найти сервисы управления контентом и сущностями, но все они не обладают всей полнотой требуемого функционала либо должны быть индивидуально настроены под требуемую задачу [10].

3.2. Методология разработки сервиса

3.2.1. Выбор языка разработки

Для серверной части был выбран язык Dart по причине того, что ранее на нем были написаны курсовые работы, а также из личных предпочтений. Функционал веб-сервера был реализован с помощью пакета Shelf. Стоит уточнить, что это не полноценный бэкенд фреймворк, а middleware сервер. Однако выбран он был по причине того, что предоставляет достаточно функционала благодаря множеству дополнительных модулей, о которых будет сказано далее. Полноценные же фреймворки на Dart слабо поддерживаются в сравнении с используемым в данном проекте. В качестве базы данных использована hive – очень быстрая и простая noSQL БД, позволяющая вести кроссплатформенную разработку программных продуктов [3, 6-7].

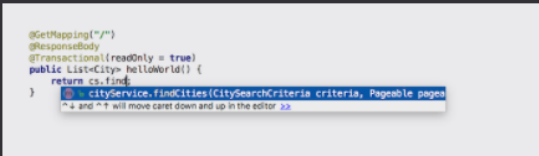
3.2.2. Выбор инструментов для разработки интерфейса приложения

С целью разработать фронтенд часть приложения был использован фреймворк Flutter, разработанный компанией Google в дополнение к Dart. Также Flutter дополняется такими пакетами, как Dio для запросов в интернет, get_it для инъекции зависимостей, bloc для стейт менеджмента, что немаловажно в нынешних реалиях разработки приложений [6, 8, 11-14].

3.2.3. Выбор среды для разработки приложения и БД

Из личных предпочтений был сделан выбор интерактивной среды разработки IntelliJ IDEA от JetBrains. Данная IDE имеет приятный внешний вид, много опций

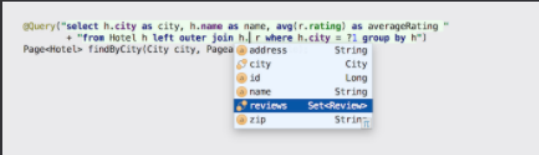
для кастомизации, поддерживает все используемые для написания курсовой работы технологии и, что не менее важно, данное ПО можно приобрести бесплатно в образовательных целях. На рисунках 3.1-3.3 приведены основные преимущества, встроенные модули, и инструменты для разработки ПО [15-16].



```
@GetMapping("/")
@ResponseBody
@Transactional(readOnly = true)
public List<City> helloWorld() {
    return cs.find(
    cityService.findCities(CitySearchCriteria criteria, Pageable pagea
    ~^ and ~^+ will move caret down and up in the editor ~^
```

Умное автодополнение кода

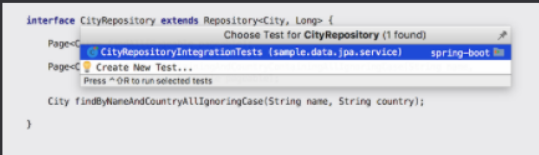
В то время как базовое автодополнение предлагает имена классов, методов, полей и ключевых слов в области видимости, умное автодополнение предлагает только те элементы кода, которые актуальны в текущем контексте.



```
@Query("select h.city as city, h.name as name, avg(r.rating) as averageRating "
+ "from Hotel h left outer join h.r r where h.city = ?1 group by h")
Page<Hotel> findByCity(City city, Pageable pageable) {
    @address String
    @city City
    @id Long
    @name String
    @reviews Set<Review>
    @zip String
```

Разработка на разных языках

Несмотря на то, что IntelliJ IDEA — в первую очередь IDE для Java, она понимает и предоставляет интеллектуальную помощь при написании кода на SQL, JPQL, HTML, JavaScript и многих других языках и позволяет редактировать код, написанный не на Java, внутри строковых литералов Java-кода.



```
interface CityRepository extends Repository<City, Long> {
    Choose Test for CityRepository (1 found)
    CityRepositoryIntegrationTests (sample.data.jpa.service) spring-boot
    Page<C> Create New Test...
    Press ~ or ~ to run selected tests
    City findByNameAndCountryAllIgnoringCase(String name, String country);
}
```

Продуктивная работа

IntelliJ IDEA анализирует однообразные задачи в процессе разработки и автоматизирует их, чтобы вы могли сосредоточиться на общей картине.

Рисунок 3.1 – Скриншот основных преимуществ среды IntelliJ IDEA

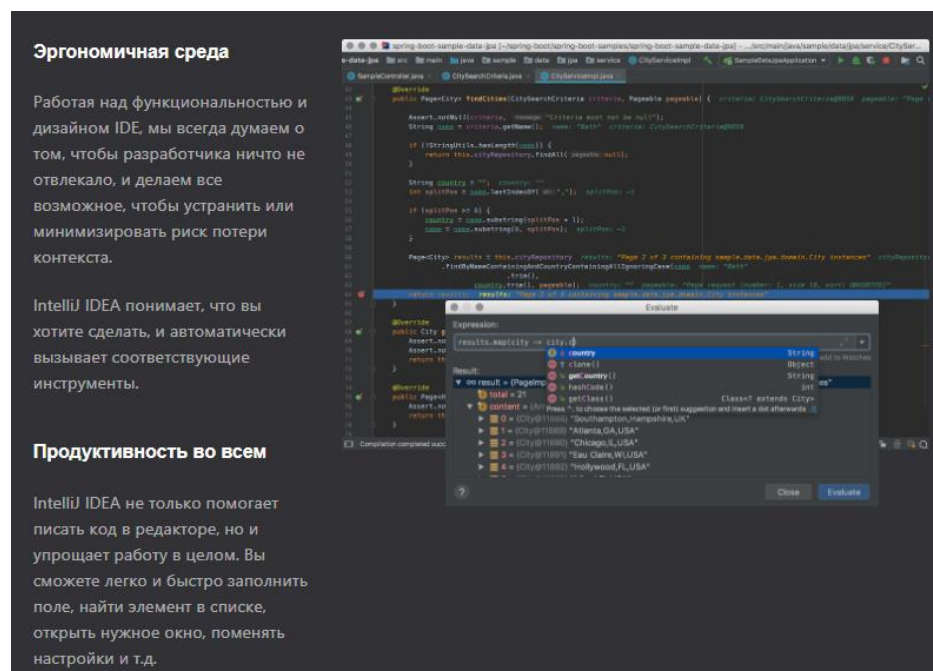


Рисунок 3.2 – Скриншот основных преимуществ среды IntelliJ IDEA






				
Встроенные инструменты	Языки JVM	Фреймворки для разработки корпоративных приложений	Мобильная разработка	Веб-разработка
Инструменты сборки	Java	Spring	Android	JavaScript
Интеграция с системами контроля версий	Kotlin	Java EE	React Native	TypeScript
HTTP-клиент	Scala	Jakarta EE	Cordova	HTML и CSS
Инструменты профилирования	Groovy	Micronaut	Ionic	Node.js
Декомпилятор		Quarkus		Angular
Покрывтие кода		Helidon		React
Работа с базами данных и SQL		Grails		Vue.js

Рисунок 3.3 – Скриншот встроенных инструментов и поддерживаемых фреймворков IntelliJ IDEA

3.3. Реализация сервиса

3.3.1. Стадии и этапы разработки

Ниже представлен последовательный список стадий и этапов разработки программы:

1. Анализ предметной области;
2. Изучение аналогов и конкурентов;
3. Выбор подходящих инструментов, фреймворков и пакетов для разработки;
4. Разработка модели базы данных и сущностей в ней;
5. Реализация моделей БД в коде;
6. Написание главной функции программы;
7. Реализация репозитория для работы с книгами;
8. Написание роутов для манипуляций с книгами;
9. Роутинг запросов;
10. Создание интерфейса приложения;
11. Развертывание приложения с помощью Heroku;
12. Тестирование всех возможных точек соприкосновения интерфейса с конечным пользователем.

3.3.2. Структурная схема сервиса

Структурная схема сервиса представлена на рисунке 3.4:

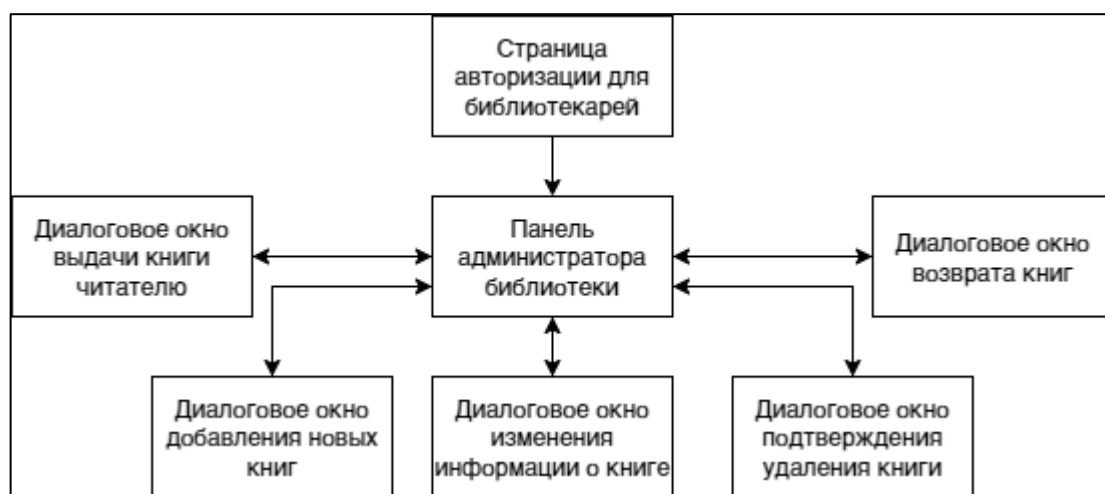


Рисунок 3.4 – Скриншот структурной схемы сервиса

3.3.3. UML-диаграмма структуры БД

На рисунке 3.5 представлена UML-диаграмма структуры БД. Поскольку используется poSQL БД, то типы связей большую роль не играют. Имеется три таблицы: author содержит информацию об авторе книги, admin содержит информацию о библиотекаре, который использует логин и пароль для авторизации. Наконец, самая важная таблица book содержит в себе информацию о названии книги, данные для изображения, статус доступности книги для выдачи, а также имя действующего читателя.

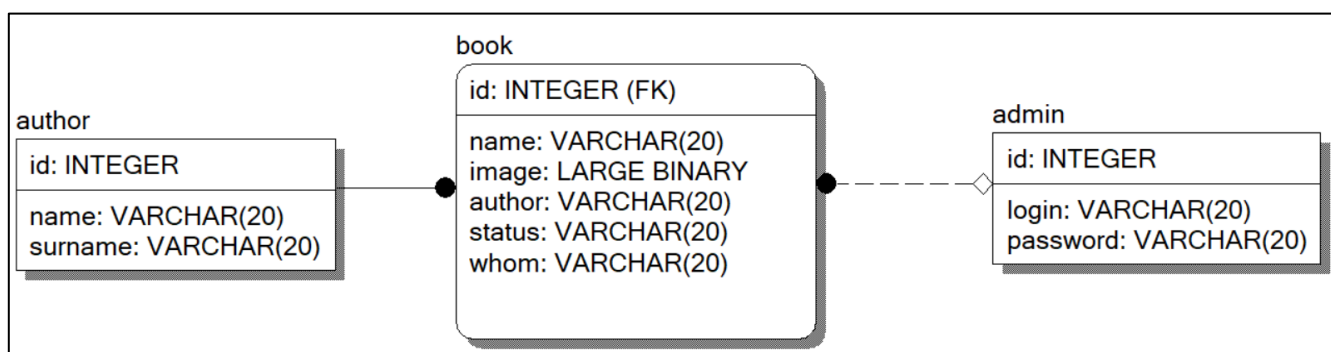


Рисунок 3.5 – Скриншот UML-диаграммы структуры БД в физическом виде

3.3.4. UML-диаграмма деятельности

На рисунке 3.6 представлена UML-диаграмма деятельности, отображающая

деятельность библиотекаря при взаимодействии с читателем. Первым делом библиотекарь должен авторизоваться в панель администратора, чтобы получить доступ к функциям приложения и информации по имеющимся в наличии книгам. При посещении библиотеки читателем у него уточняется, с какой целью: если требуется сдать книгу, то библиотекарь вносит информацию по сдаче книги в библиотеку и деятельность заканчивается; если же требуется получить книгу, то происходит проверка наличия книги в библиотеке: если требуемая книга в библиотеке не имеется, то деятельность заканчивается; если же требуемая книга есть в наличии, то читатель предоставляет личные данные о ФИО и номере группы, если это студент. После чего библиотекарь вносит информацию по выдаче книги, затем деятельность заканчивается.

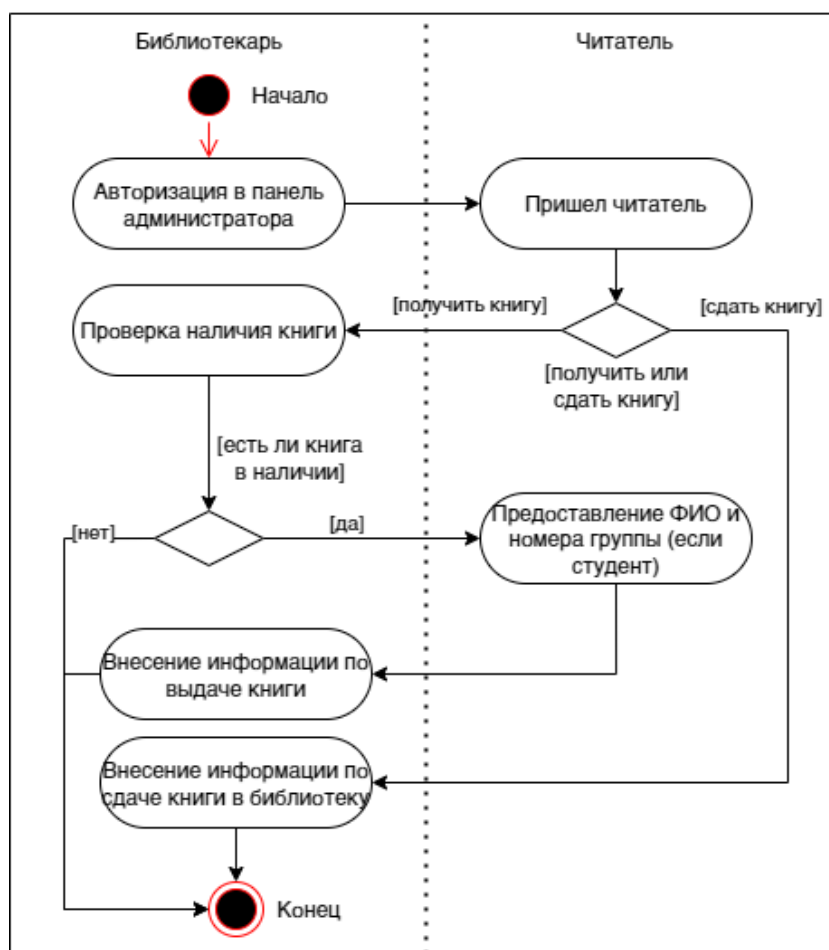


Рисунок 3.6 – Скриншот UML-диаграммы деятельности, отображающей процесс взаимодействия библиотекаря с читателем

3.3.5. Разработка серверной части

Для реализации модели книги (Рисунок 3.7) были использованы аннотации из пакетов `json_serializable` и `copy_with_extension_gen`, а также `hive` для автогенерации рутинного кода конструкторов `copyWith`, `fromJson` и для адаптера БД [3, 17-18].

```
import 'package:copy_with_extension/copy_with_extension.dart';
import 'package:hive/hive.dart';
import 'package:json_annotation/json_annotation.dart';

part 'book.g.dart';

@CopyWith()
@JsonSerializable()
@HiveType(typeId: 0)
You, 17 hours ago | 1 author (You)
class Book {
  const Book({
    required this.id,
    required this.name,
    required this.image,
    required this.author,
    required this.status,
    this.whom = '',
    this.dateOfReturn = '',
  });

  factory Book.fromJson(Map<String, dynamic> json) => _$BookFromJson(json);

  @HiveField(3)
  final String author;

  @HiveField(5)
  final String dateOfReturn;

  @HiveField(6)
  final String whom;

  @HiveField(0)
  final int id;

  @HiveField(2)
  final String image;

  @HiveField(1)
  final String name;

  @HiveField(4)
  final BookStatus status;

  Map<String, dynamic> toJson() => _$BookToJson(this);
}
```

Рисунок 3.7 – Скриншот кода реализации модели книги

На рисунке ниже представлена главная функция программы (Рисунок 3.8):

```
Future<void> main(List<String> args) async {  
    Environment(  
        args: args,  
    ).spinTheWheelOfSamsara();  
}
```

Рисунок 3.8 – Скриншот кода главной функции программы

Пакет Shelf позволяет включить сервер и слушать заданный адрес и порт, а также предоставляет возможность установки необходимых прослоек (Рисунок 3.9):

```
Future<void> beAGuestInAMortalReality() async {  
    final handler = const shelf.Pipeline()  
        .addMiddleware(corsHeaders(headers: corsHeaders1))  
        .addMiddleware(shelf.logRequests())  
        .addHandler(router);  
  
    var server = await shelf_io.serve(  
        handler,  
        isProd ? '0.0.0.0' : 'localhost',  
        port,  
    );  
}
```

Рисунок 3.9 – Скриншот кода сервера с использованием пакета Shelf

Пакет shelf_cors_headers предоставляет механизм для установки хедеров для ответов на preflight cors запросы [19-20].

Пакет shelf_router позволяет реализовать роутинг запросов. На рисунке 3.10 приведен пример роута на /books/ с post методом для создания книги в БД [21].

```

@Route.post('/:urlName/')
Future<Response> createBook(Request request) async {
  try {
    final data = await request.readAsString();
    final book = Book.fromJson(json.decode(data));
    repo.create(book);
    return Response.ok(
      json.encode({
        'res': 'success',
        'data': data,
      }),
      headers: overrideHeaders,
    );
  } catch (e) {
    print(e.toString());
    return Response.internalServerError(
      body: json.encode({
        'res': 'fail',
        'reason': 'Book was not created.',
      }),
      headers: overrideHeaders,
    );
  }
}

```

Рисунок 3.10 – Скриншот кода реализации роута post метода для создания новой книги в базе данных

Также стоит обратить внимание на все реализованные роуты для манипуляций с книгами в БД (Рисунок 3.11):

```
// GENERATED CODE - DO NOT MODIFY BY HAND

part of 'handler.dart';

// *****
// ShelfRouterGenerator
// *****

Router _$BookServiceRouter(BookService service) {
  final router = Router();
  router.add('GET', r'/books/', service.listBooks);
  router.add('GET', r'/books/<bookId>', service.fetchBook);
  router.add('POST', r'/books/', service.createBook);
  router.add('PUT', r'/books/<bookId>', service.updateBook);
  router.add('DELETE', r'/books/<bookId>', service.deleteBook);
  return router;
}
```

Рисунок 3.11 – Скриншот автоматически сгенерированного кода всех реализованных роутов для взаимодействия с книгами в БД

3.3.6. UML-диаграмма классов

Далее описана структура расположения классов в приложении. Важно обратить внимание, что структура относится ближе к чистому ООП, то есть объектно-ориентированному коду, а не классово-ориентированному. По этой причине в данной модели не совсем та структура, где существует группа классов, которые работают друг с другом, посылая данные. В действительном случае реализована матрешечная вложенность, отображенная через композицию (Рисунок 3.12).

1. Класс Environment принимает внешние аргументы (массив строк) и запускает систему
2. Класс Server принимает номер порта, переменную режима среды isProd, подготовленные Environment и экземпляр роутера (класс BookService) и запускает механизм прослушивания
3. Класс BookService принимает BooksRepo (репозиторий книг, работающий с базой) и AdminsRepo (репозиторий админов) и определяет какие

ответы на какие запросы выдаёт сервер

4. Класс BooksRepo принимает базу (коробку hive из класса Environment) и реализует работу с данными в бд. Аналогичным образом происходит работа с классом AdminsRepo.

5. Класс Book описывает данные, которые хранятся в базе и с которыми идёт работа. Аналогичным образом происходит работа с классом Admin.

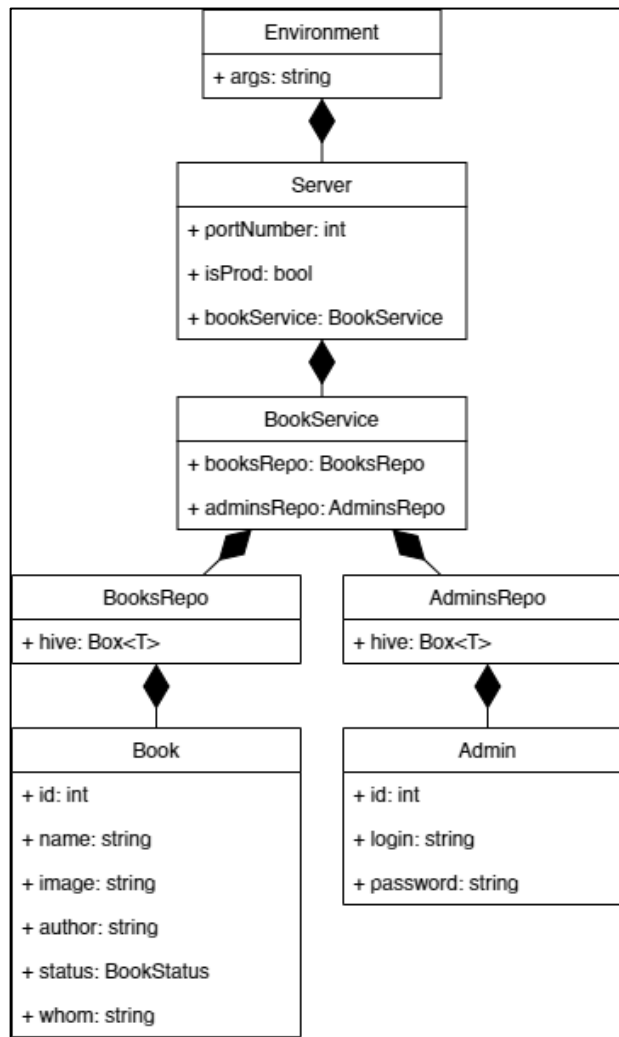


Рисунок 3.12 – Скриншот UML-диаграммы классов построенной системы

3.3.7. Разработка интерфейса приложения

В процессе разработки был реализован репозиторий с книгами. Он позволяет добавить книгу, удалить, выдать или вернуть в библиотеку, изменить информацию

о книге. Кроме того, используется поток, подключившись к которому можно в реактивном (асинхронном) стиле получать данные.

Реализация репозитория разделена с его интерфейсом. Некоторые программисты называют такую структуру паттерном мост (bridge) [22].

Абстрактный класс находится в папке domain/repositories, а реализация находится в data/repositories. На следующих двух рисунках приведены скриншоты примеров кода реализации абстрактного класса BooksRepository и его реализации (Рисунок 3.13-3.14):

```
abstract class BooksRepository {
    BooksRepository();

    final StreamController<List<Book>> content =
        StreamController<List<Book>>.broadcast();

    Future<void> giveBook(Book book);

    Future<void> createBook(Book book);

    Future<void> returnBook(Book book);

    Future<void> deleteBook(Book book);

    Future<List<Book>> getBooks();
}
```

Рисунок 3.13 – Скриншот кода абстрактного класса BooksRepository

```

class BooksRepositoryImpl extends BooksRepository {
  BooksRepositoryImpl({
    required this.dio,
  });

  final Dio dio;
  final String url = 'https://bookstore-backend-paper.herokuapp.com/books/';

  @override
  Future<void> createBook(Book book) async {
    try {
      log('Create book ${book.name}');
      final response = await dio.post(
        url,
        data: json.encode(book.toJson()),
      );
      log(response.data.toString());
    } catch (e) {
      log('BooksRepository createBook Error: $e');
    }
  }
}

```

Рисунок 3.14 – Скриншот кода реализации класса BookRepository под названием BooksRepositoryImpl

Далее следует фундаментальная база логики фронтенд составляющей – блок книг. Реализована подписка на обновление данных по факту их получения и реакции системы на события (действия с книгами) [14] (Рисунок 3.15).

```

class BooksBloc extends Bloc<BooksEvent, BooksState> {
  final BooksRepository repository;
  BooksBloc({
    required this.repository,
  }) : super(BooksLoading()) {
    on<UpdateBooks>((event, emit) {
      emit(BooksLoaded(books: event.books));
    });
    on<BookAction>(_onBookAction);

    repository.content.stream.listen((event) {
      add(UpdateBooks(books: event));
    });
    repository.getBooks();
  }

  Future<void> _onBookAction(
    BookAction event,
    Emitter<BooksState> emitter,
  ) async {
    switch (event.runtimeType) {
      case GiveBook:
        await repository.giveBook(event.book);
        break;
      case ReturnBook:
        await repository.returnBook(event.book);
        break;
      case CreateBook:
        await repository.createBook(event.book);
        break;
      case DeleteBook:
        await repository.deleteBook(event.book);
        break;
    }
    emitter(BooksLoading());

    final books = await repository.getBooks();
    emitter(BooksLoaded(books: books));
  }
}

```

Рисунок 3.15 – Скриншот структуры bloc с книгами с реализованным слушателем событий

Также стоит упомянуть про реализацию инъекции зависимостей (DI). Инстанс `dio` для запросов, репозиторий `BooksRepository` и блок `BooksBloc` записаны в контейнер, представленный на рисунке 3.16 [23].

```
final getIt = GetIt.instance;

void setup() {
  getIt.registerSingleton<Dio>(Dio());

  getIt.registerSingleton<BooksRepository>(
    BooksRepositoryImpl(
      dio: getIt<Dio>(),
    ),
  );

  getIt.registerFactory<BooksBloc>(
    () => BooksBloc(
      repository: getIt<BooksRepository>(),
    ),
  );
}
```

Рисунок 3.16 – Скриншот кода примера DI в проекте

В заключение к разделу разработки интерфейса приложения хочется добавить изображение, иллюстрирующее исполнение диалога, подтверждающего удаление книги в качестве примера реализации виджета (Рисунок 3.17):


```

class ConfirmDeleteDialog extends StatelessWidget {
  final Book book;
  const ConfirmDeleteDialog({
    Key? key,
    required this.book,
  }) : super(key: key);

  Widget cancelButton(BuildContext context) {
    return TextButton(
      child: const Text("Отмена"),
      onPressed: () {
        Navigator.pop(context);
      },
    ); // TextButton
  }

  Widget deleteButton(BuildContext context) {
    return TextButton(
      child: const Text(
        "Удалить",
        style: TextStyle(color: Colors.red),
      ), // Text
      onPressed: () {
        BlocProvider.of<BooksBloc>(context).add(DeleteBook(book: book));
        Navigator.pop(context);
      },
    ); // TextButton
  }

  Widget title(BuildContext context) {
    return Text(
      'Вы действительно хотите удалить ${book.name}?',
      style: Theme.of(context)
        .textTheme
        .subtitle1!
        .copyWith(fontWeight: FontWeight.bold),
    );
  }
}

```

Рисунок 3.17 – Скриншот реализации диалога при удалении книги

3.3.8. Выводы к разделу

С помощью сравнительно низкоуровневого пакета Shelf для Dart удалось развернуть приложение, способное слушать входящие запросы, корректно обрабатывать их, взаимодействуя с базой данных, предоставляющих все необходимые запрашиваемые у нее данные. Этап разработки приложения можно

считать успешно выполненным.

4. Руководство по использованию сервиса

При первом запуске библиотекаря встречает окно авторизации. Для удобства и быстроты тестирования логин и пароль установлены admin и admin соответственно (Рисунок 4.1).

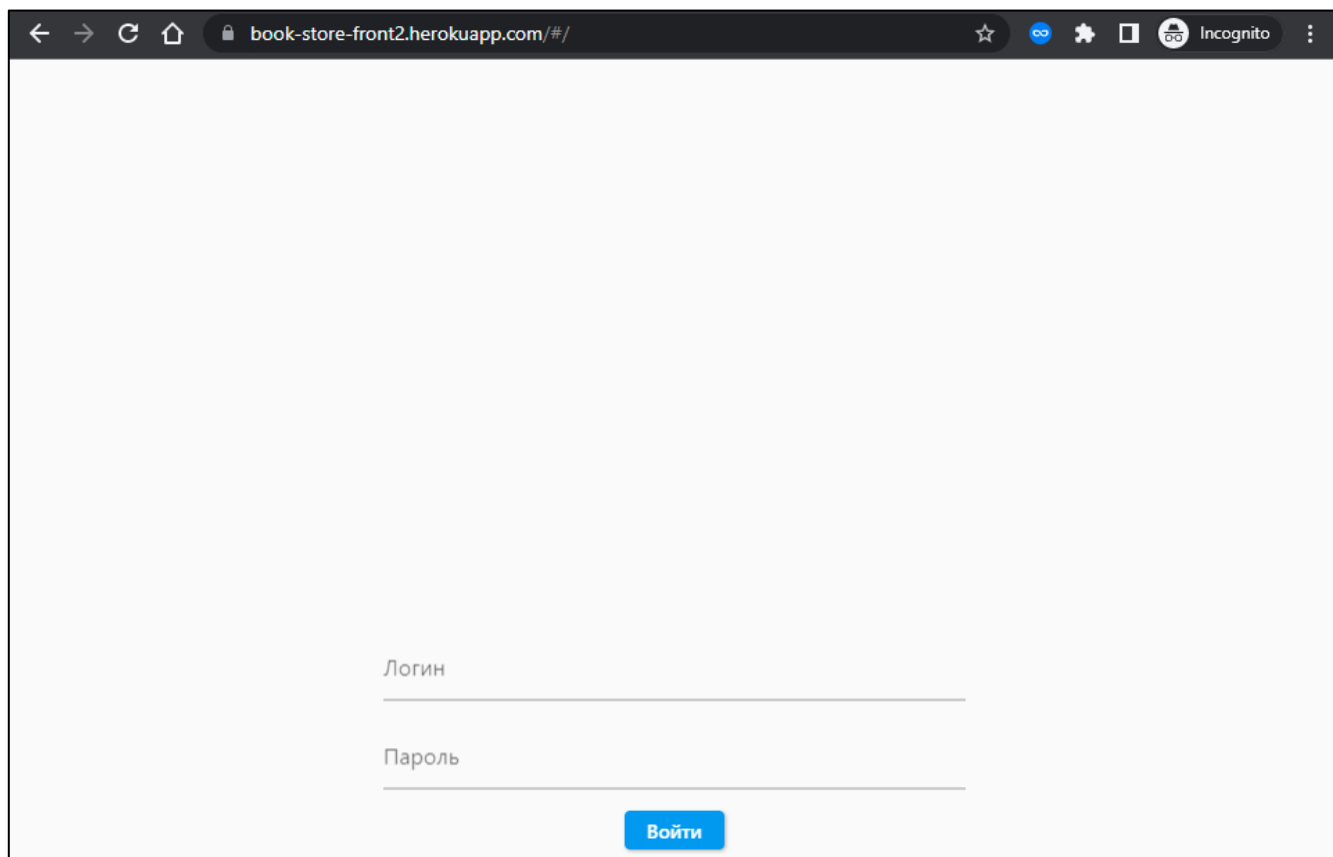


Рисунок 4.1 – Скриншот окна авторизации при входе в приложение

После успешной авторизации открывается панель администратора библиотеки. В правом верхнем углу присутствует кнопка «Добавить книгу» для добавления новых книг в требуемом количестве, с указанием названия, автора и ссылки на изображение обложки в сети Интернет (Рисунок 4.2-4.4):

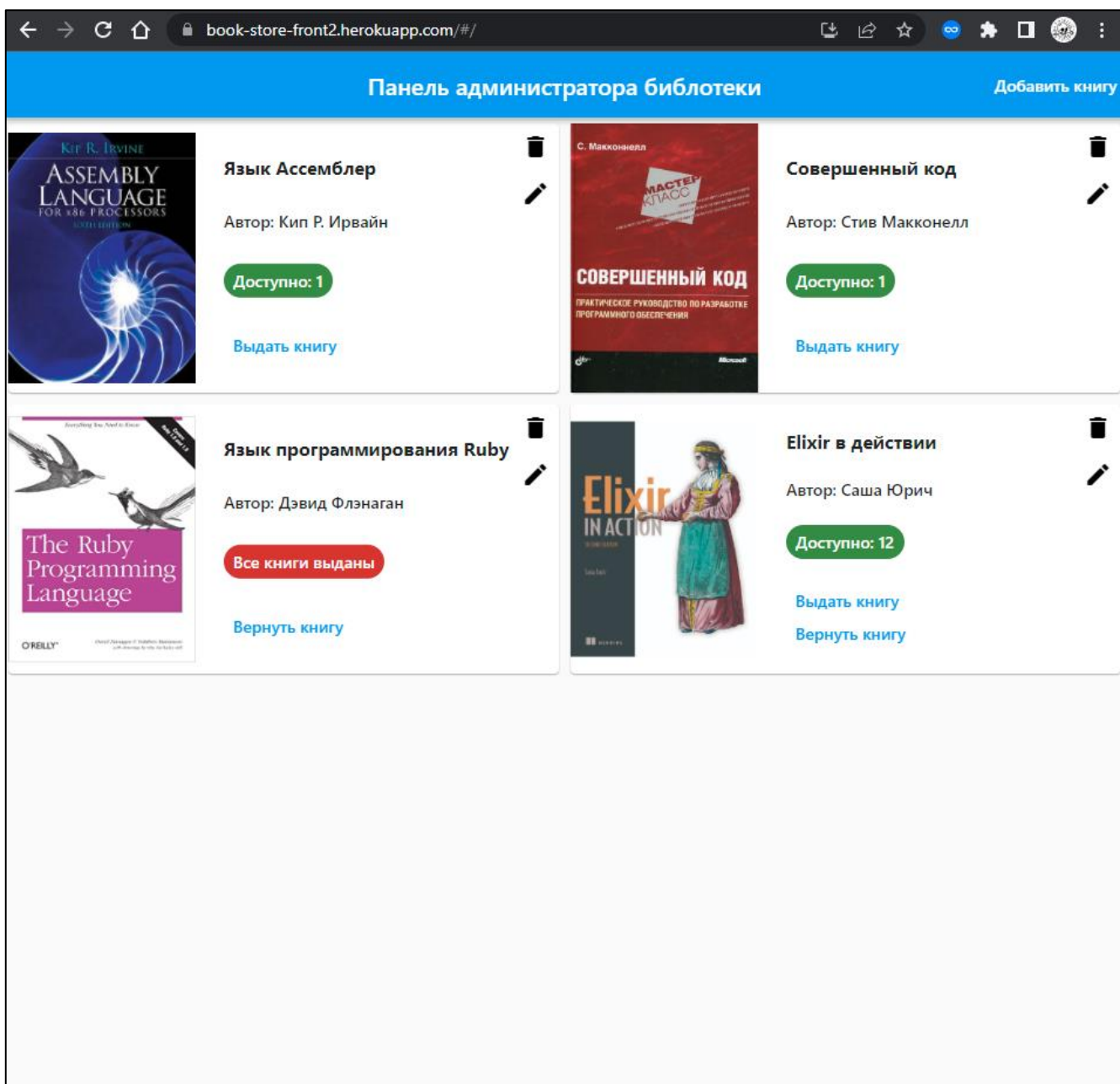


Рисунок 4.2 – Скриншот панели администратора библиотеки

Форма добавления книги

Введение в машинное обучение с помощью Python

Андреас Мюллер

<https://cdn1.ozone.ru/multimedia/wc1200/1018802929.jpg>

5|

[Добавить книгу](#)

Рисунок 4.3 – Скриншот формы добавления новой книги

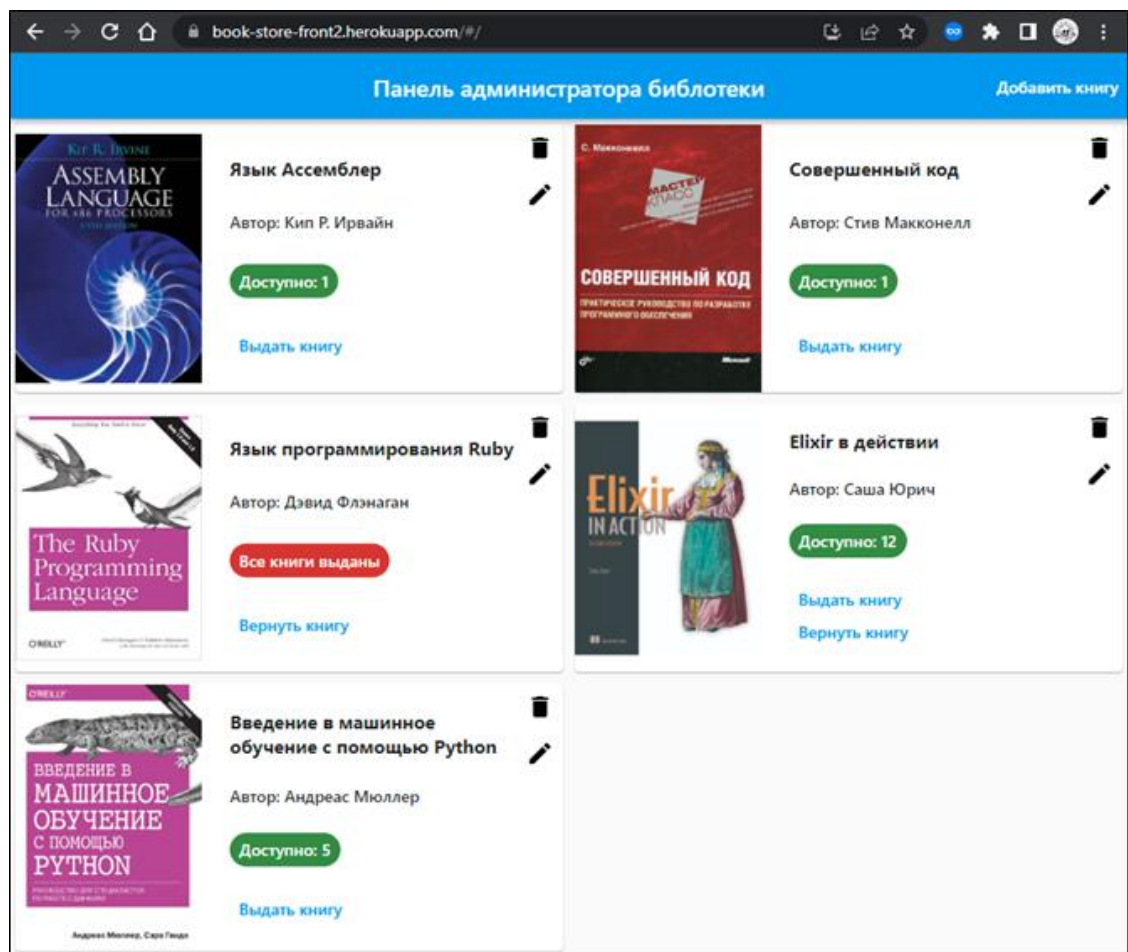
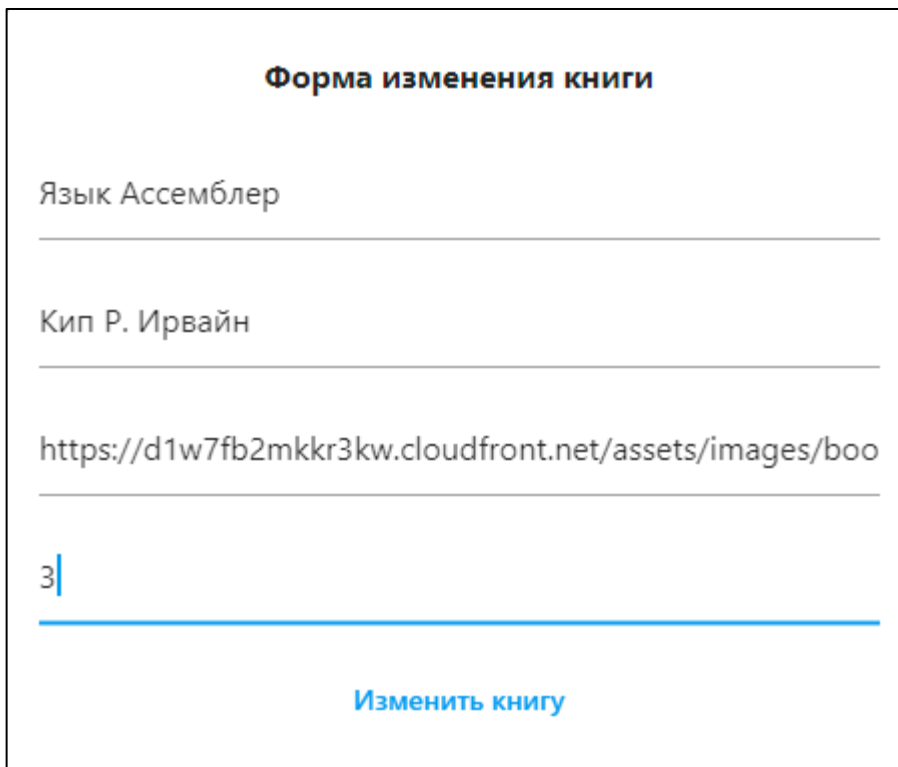


Рисунок 4.4 – Скриншот демонстрации добавления новой книги

При помощи иконки карандаша возле книги можно изменять данные у добавленных ранее книг (Рисунок 4.5):



Форма изменения книги

Язык Ассемблер

Кип Р. Ирвайн

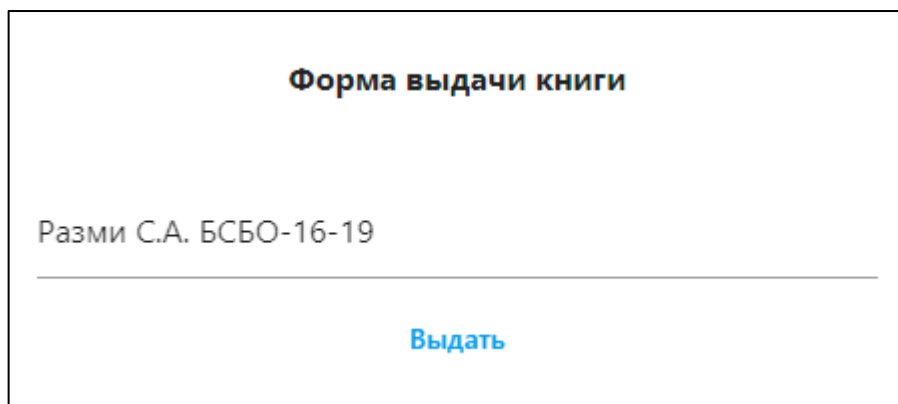
<https://d1w7fb2mkk3kw.cloudfront.net/assets/images/boo>

3

Изменить книгу

Рисунок 4.5 – Скриншот формы изменения информации в добавленной книги

При нажатии кнопки «Выдать книгу» откроется форма выдачи книги, в которой указывается ФИО читателя (Рисунок 4.6):



Форма выдачи книги

Разми С.А. БСБО-16-19

Выдать

Рисунок 4.6 – Скриншот формы выдачи книги

При нажатии кнопки «Вернуть книгу» откроется список читателей, в котором

можно выбрать, какой читатель возвращает книгу в библиотеку (Рисунок 4.7):



Рисунок 4.7 – Скриншот диалогового окна возврата книги в библиотеку

При нажатии на иконку мусорной корзины появится диалоговое окно для подтверждения удаления книги (Рисунок 4.8):

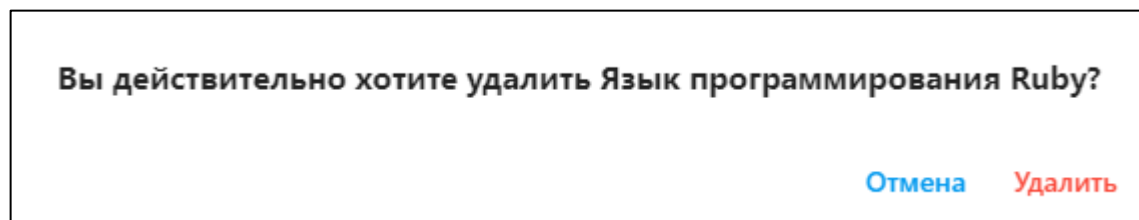


Рисунок 4.8 – Скриншот диалогового окна подтверждения удаления книги

После подтверждения удаления книги, ее больше не будет в списке добавленных книг в панели администратора библиотеки (Рисунок 4.9):

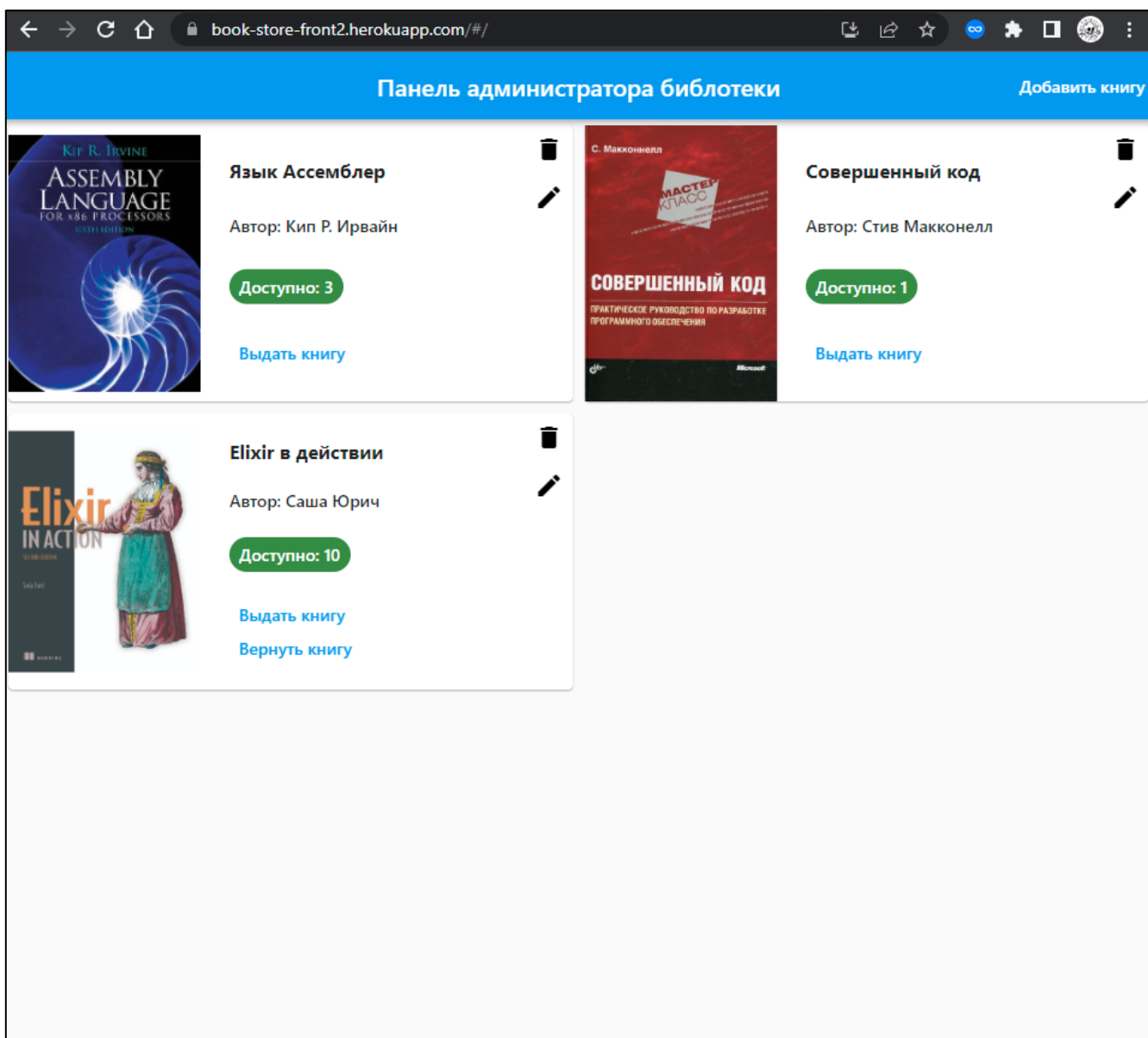
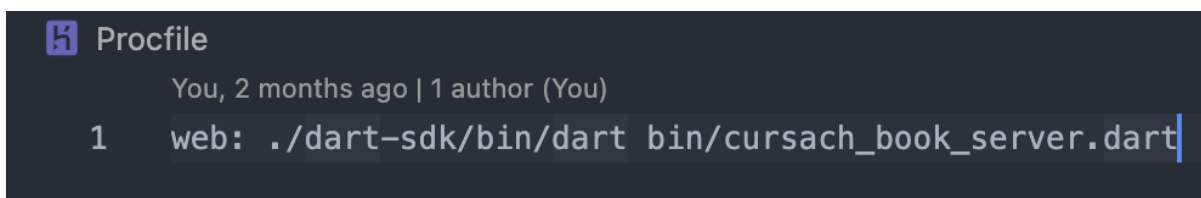


Рисунок 4.9 – Скриншот подтверждения удаления книги

5. Вызов и загрузка

Для запуска серверной части используется Procfile, созданный и наполненный согласно требованиям Heroku. Запуск происходит за счёт одной команды, представленной на рисунке 5.1:a



```
Procfile
You, 2 months ago | 1 author (You)
1 web: ./dart-sdk/bin/dart bin/cursach_book_server.dart
```

Рисунок 5.1 – Скриншот кода основного файла запуска программы

Чтобы все верно работало, необходимо предварительно установить на сервер DartSDK, упомянутый в подразделе 1.2. «ПО, необходимое для функционирования программы» раздела 1. «Общие сведения» Основной части. Этим занимается билдпак `heroku-buildpack-dart` [24]:

1. Скачивает DartSDK по указанной в переменных среды ссылке. Переменные среды устанавливаются в настройках приложения Heroku;
2. Скачивает зависимости с помощью `pub`;
3. Запускает команду из Procfile.

Заключение

Цель данной курсовой работы была успешно достигнута. В результате выполнения работы был изучен объект исследования – библиотеки с читательскими билетами, спроектирована информационная система по выбранному варианту, проведен анализ и описание предметной области и аналогов разрабатываемого приложения, выбраны язык разработки и инструменты для разработки интерфейса и серверной части приложения, выбрана среда для разработки БД, описаны стадии и этапы разработки, представлена структурная схема сервиса, построены диаграммы в нотации UML, написано руководство по использованию сервиса для библиотекарей и для пользователей системой учета выданных книг, продемонстрирован процесс развертывания и входные точки в программу, а также получены ценные теоретические и практические навыки анализа и разработки клиентских-серверных приложений, которые преподавались студентам в рамках данной дисциплины на лекционных занятиях.

Информационные источники

- 1) DartSDK [Электронный ресурс]. Режим доступа: <https://dart.dev/get-dart>, свободный. (дата обращения: 27.04.22).
- 2) Flutter sdk [Электронный ресурс]. Режим доступа: <https://docs.flutter.dev/get-started/install>, свободный. (дата обращения: 27.04.22).
- 3) Hive [Электронный ресурс]. Режим доступа: <https://pub.dev/packages/hive>, свободный. (дата обращения: 27.04.22).
- 4) Google Chrome [Электронный ресурс]. Режим доступа: https://www.google.com/intl/ru_ru/chrome/, свободный. (дата обращения: 27.04.22).
- 5) Mozilla Firefox [Электронный ресурс]. Режим доступа: <https://www.mozilla.org/ru/firefox/new/>, свободный. (дата обращения: 27.04.22).
- 6) Dart [Электронный ресурс]. Режим доступа: <https://dart.dev/>, свободный. (дата обращения: 27.04.22).
- 7) Shelf (Dart) [Электронный ресурс]. Режим доступа: <https://pub.dev/packages/shelf>, свободный. (дата обращения: 27.04.22).
- 8) Flutter [Электронный ресурс]. Режим доступа: <https://flutter.dev/>, свободный. (дата обращения: 27.04.22).
- 9) Heroku [Электронный ресурс]. Режим доступа: <https://id.heroku.com/>, свободный. (дата обращения: 27.04.22).
- 10) Научно-Техническая Библиотека МИРЭА [Электронный ресурс]. Режим доступа: <https://library.mirea.ru/>, свободный. (дата обращения: 27.04.22).
- 11) Google [Электронный ресурс]. Режим доступа: <https://about.google/>, свободный. (дата обращения: 27.04.22).
- 12) Dio (Dart) [Электронный ресурс]. Режим доступа: <https://pub.dev/packages/dio>, свободный. (дата обращения: 27.04.22).
- 13) Get_it (Dart) [Электронный ресурс]. Режим доступа: https://pub.dev/packages/get_it, свободный. (дата обращения: 27.04.22).
- 14) Bloc [Электронный ресурс]. Режим доступа: <https://bloclibrary.dev/#/>,

свободный. (дата обращения: 27.04.22).

15) IntelliJ IDEA [Электронный ресурс]. Режим доступа: <https://www.jetbrains.com/ru-ru/idea/>, свободный. (дата обращения: 27.04.22).

16) JetBrains [Электронный ресурс]. Режим доступа: <https://www.jetbrains.com/ru-ru/>, свободный. (дата обращения: 27.04.22).

17) Json_serializable (Dart) [Электронный ресурс]. Режим доступа: https://pub.dev/packages/json_serializable, свободный. (дата обращения: 27.04.22).

18) Copy_with_extension_gen (Dart) [Электронный ресурс]. Режим доступа: https://pub.dev/packages/copy_with_extension_gen, свободный. (дата обращения: 27.04.22).

19) Shelf_cors_headers (Dart) [Электронный ресурс]. Режим доступа: https://pub.dev/packages/shelf_cors_headers, свободный. (дата обращения: 27.04.22).

20) Preflight cors (Dart) [Электронный ресурс]. Режим доступа: <https://developer.mozilla.org/ru/docs/Web/HTTP/CORS>, свободный. (дата обращения: 27.04.22).

21) Shelf_router (Dart) [Электронный ресурс]. Режим доступа: https://pub.dev/packages/shelf_router, свободный. (дата обращения: 27.04.22).

22) Bridge (pattern) [Электронный ресурс]. Режим доступа: <https://medium.com/clean-code-channel/design-patterns-a-quick-guide-to-bridge-pattern-8688d8c4f5aa>, свободный. (дата обращения: 27.04.22).

23) DI (Dependency Injection) [Электронный ресурс]. Режим доступа: <https://habr.com/ru/post/350068/>, свободный. (дата обращения: 27.04.22).

24) Heroku-buildpack-dart [Электронный ресурс]. Режим доступа: <https://elements.heroku.com/buildpacks/igrigorik/heroku-buildpack-dart>, свободный. (дата обращения: 27.04.22).

Приложения

- 1) Ссылка для просмотра фронтенд составляющей развернутого приложения: <https://book-store-front2.herokuapp.com/>;
- 2) Ссылка для просмотра бэкенд составляющей развернутого приложения: <https://bookstore-backend-paper.herokuapp.com/>.