МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

Институт (Филиал) № 8 «Компьютерные науки и прикладная математика» **Кафедра** 806

Группа М8О-407Б-19 **Направление подготовки** 01.03.02 «Прикладная математика и информатика»

Профиль Информатика

Квалификация: бакалавр

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему: Сервис поиска, классификации и обработки тематических изображений в интернете

Автор ВКРБ:	Волков Матвей Андреевич	(_____)
Руководитель:	Филимонов Николай Сергеевич	(_____)
Консультант:	Чернышов Лев Николаевич	(_____)
Консультант:	—	(_____)
Рецензент:	—	(_____)

К защите допустить

Заведующий кафедрой № 806 Крылов Сергей Сергеевич (_____)

____ мая 2023 года

Москва 2023

РЕФЕРАТ

Выпускная квалификационная работа бакалавра состоит из 43 страниц, 25 рисунков, 11 использованных источников, 4 приложений.

МАНГА, БАЗА ДАННЫХ, МИКРОСЕРВИС, ПАРСИНГ, САЙТ, API

Работа состоит из введения, двух глав и заключения. Темой работы является веб приложение, основанное на технологии парсинга данных для сбора тематических картинок (манги) и компилирование их в сборники в одном pdf файле.

Целью работы является написание API, которое возможно использовать в других клиентских сервисах, таких как мессенджер, сайт. API должен предоставлять интерфейс для отправки собранных изображений на электронную книгу через отправку тех по электронной почте.

Проведение работы состоит из исследования целевых сайтов, список которых задается заранее. Написание парсинг алгоритма. Конструирование микросервиса, применяя методику чистой архитектуры для дальнейшего разввития проекта.

Областью применения является любой клиентский сервис, способный общаться с сделанным API.

Итог внедрения считается успешно работающее API любым из возможных клиентских сервисов, таких как сайт, мессенджер.

Дальнейшее развитие проекта предполагает оформление подписки на сайты для автоматического информирования пользователя о новых главах комиксов или манги. Оптимизирование работы сервиса для успешного обслуживания больше десятка пользователей.

Появление подобного микросервиса должно принести в рынок Open source новое кроссплатформенное решение по оффлайн прочтению манги, в том числе, и на электронных устройствах с искусственными чернилами (электронные книги).

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящей выпускной квалификационной работе бакалавра применяют следующие термины с соответствующими определениями:

API — описание способов взаимодействия одной компьютерной программы с другими

DOM-дерево — структурное представление HTML файла

Dos-атака — хакерская атака на вычислительную систему с целью довести её до отказа. Если атака выполняется одновременно с большого числа компьютеров, говорят о DDoS-атаке

GO — язык программирования, созданный компанией гугл преимущественно для создания бекэнд сервисов

HTML — стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере

HTTP — протокол передачи гипертекста, изначально в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных

HTTP запрос — сигнал серверу через http метод, всегда требующий какого-то ответа от сервара

JQuery — набор функций JavaScript, фокусирующийся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими

SQL-инъекция — один из распространённых способов взлома сайтов и программ, работающих с базами данных, основанный на внедрении в запрос произвольного SQL-кода

База данных — набор информации, которая хранится упорядоченно в электронном виде

Библиотека — сборник подпрограмм или объектов, используемых для разработки программного обеспечения

Клиент — приложение, которое путем отправки запросов на сервер способно визуализировать полученную информацию. Обычно приложение клиент общается с пользователем. При этом не присутствует никакой логики, связанной с основным приложением. Например, браузер является клиентом

Конститентность — согласованность данных друг с другом, целостность

данных, а также внутренняя непротиворечивость

Манга — японские комиксы, иногда называемые комикку

Метод HTTP — последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами

Миграция базы данных — переход от одной структуры базы данных к другой без потери косистентности

Модуль — называется пакет в проекте языка GO, который может быть переиспользован в других приложениях или проектах, как внешняя библиотека. Был добавлен в версии 1.11

Пайплайн — конвеер данных, поступающий из одной логической программы другой

Пакет — понятие в языке GO, обозначающее коллекцию исходного кода в одной дериктории скомпилируемых вместе. Пакет также является подключаемым модулем

Парсинг — это автоматизированный сбор и структурирование информации с сайтов при помощи программы или сервиса

Реверс-инженеринг — исследование некоторого готового устройства или программы, а также документации на него с целью понять принцип его работы. Также известно, как «обратная разработка»

Сервер — сервером в веб приложении называют программу, которая занимается обработкой запросов с клиентской части приложения. В основном на сервере происходит вся логика работы веб приложения. Обычно на серверной части запускается база данных

Транзация — это набор операций по работе с базой данных, объединенных в одну атомарную пачку

Хендлер — функция, которую выполняет веб-приложение, когда был сделан соответствующий http запрос

Чистая архитектура — понятие при конструировании микросервиса, созданное для разделения различных концепций, путем написания кода на нескольких уровнях с четким правилом, которое позволяет создать тестируемый и поддерживаемый проект

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей выпускной квалификационной работе бакалавра применяют следующие сокращения и обозначения:

API — Application Programming Interface
CSS — Cascading Style Sheets
DDOS — Distributed Denial of Service
DOM — Document Object Model
HTTP — HyperText Markup Language
ID — Identification
JS — JavaScript
JSON — Javascript object notation
PDF — Portable Document Format
RegExp — Regular Expression
SQL — Structured Query Language
БД — база данных
ВКР — выкускная квалификационная работа
ЯП — язык программирования

ВВЕДЕНИЕ

Сегодня манга является одним из самых популярных видов комиксов в мире, и ее популярность только растет. Однако, не всегда удобно читать мангу на экране компьютера или телефона. Именно поэтому возникает необходимость в сервисе, который позволяет скачивать мангу и отправлять ее на электронные книги. В данном случае мы говорим о микросервисе, который предоставляет API к скачиванию манги и отправки ее на устройства чтения. Задача данного сервиса заключается в том, чтобы обеспечить удобный и быстрый доступ к любимой манге, а также сделать ее чтение более комфортным для пользователей.

Плюсы API заключаются в том, что это, в первую очередь, кроссплатформенное решение. Исследовав рынок подобных сервисов, не было обнаружено чего-то удобного для операционных систем IOS. Из-за открытости системы android удалось найти похожее решение, однако удобство использования оставляло желать лучшего: есть привязанность к устройству, нет никакой синхронизации и прочих удобств современных приложений.

Чтение манги, используя электронную книгу — совершенно новое направление, которое открывает возможности для любителей манги. Так люди, которые обожают печатные издания, смогут насладиться технологией искусственных чернил, при этом не таская с собой в дорогу огромную библиотеку комиксов. Скачивать мангу на электронную книгу — очень трудная задача. С ней не справится пользователь, который не умеет писать даже базовые скрипты на компьютере. Сервис же будет предоставлять удобный API, который возможно подключить к любому клиетскому сервису.

Таким образом, скаченную мангу возможно хранить в мессенджере, электронной книге, наслаждаясь всеми прелестями технологии электронных чернил, а также, возможно сделать свое приложение со своим удобным оформлением.

1 ТЕХНОЛОГИИ И АРХИТЕКТУРА

1.1 Обоснование библиотек

В микросервисе были использованы внешние модули для упрощения частоиспользуемых действий в коде. Каждая из них представляет собой очень полезную функциональность, которая будет перечислена ниже с обоснованием использования.

1.1.1 Pdfcpu

Pdfcpu [1] — пакет, написанный на ЯП GO, яляющийся мощным сборником pdf файлов. Представляет интферфейс для создания сложноструктурированных pdf файлов.

В данном пакете возможно создавать абсолютно любые pdf файлы с различными метаданными документа, начиная с содержания, заканчивая указыванием автора написания документа. Библиотека способна также использовать различные картинки различных разрешений, позволяя менять их размер, цвет, прозрачность и другие параметры.

Пакет был взят в использование потому, что предоставляется разработчиками удобный и понятный интферфейс для собирания большого количество картинок в pdf файл. Также модуль отличается своей быстротой исполнения поставленных задач. Рисунок 1 может продемонстрировать простоту использования пакета.

```
1  func CreatePDFFromImagesDir(imagesDirPath string, outputPath string
       ) error {
2    imagesPath, err := GetImagesPathStr(imagesDirPath)
3    if err != nil {
4      return errors.Wrap(err, "something wrong with creating images
       path")
5    }
6    err = api.ImportImagesFile(imagesPath, outputPath, nil, nil)
7    if err != nil {
8      return errors.Wrap(err, "pdf creation filure")
9    }
10   return nil
11 }
```

Рисунок 1 – Демонстрация работы интерфейса pdfcpu

1.1.2 Goquery

Goquery [2] — пакет, написанный на ЯП GO, который способен выполнять задачи библиотеки jQuery [3].

Данный пакет используется для того, чтобы парсить DOM дерево полученного HTML файла. С ним возможно быстро и удобно получить нужную информацию с любого сайта. После реверс-инженеринга, благодаря этому пакету, быстро и просто достается информация о картинках, классифицируется на разделы и подразделы (например, группировка на главы).

Как видно из рисунка 2, используются команды jQuery для получения информации о главе. Впоследствии эта глава используется для того, чтобы дастать из неё картинки.

```
1  func (mlb mangaLibController) getChapterID(doc *goquery.Document)
     (string, error) {
2    result, ok := doc.Find("#comments").Attr("data-post-id")
3    if !ok {
4      return "", errors.Wrap(customerrors.ErrEmptyAttr, "mangalib:
     data-post-id")
5    }
6    return result, nil
7  }
8
```

Рисунок 2 – Демонстрация работы пакета goquery

1.1.3 Jet

Go-jet [4] — пакет, написанный на ЯП GO, являющийся конструктором SQL запросов в БД.

С помощью этой библиотеки возможно собирать любые SQL запросы, избегая уязвимость — SQL-инъекцию. Этот пакет подключается к БД и генерирует код на языке GO, при помощи которого можно собирать различные SQL запросы. От инъекции помогает избавиться тот факт, что абсолютно весь запрос делается при помощи конструктора, в процессе сбора которого есть постоянные проверки на соответствие данных, поступающих в процесс создания запроса. Также стоит заметить, что сама возможность собирать запрос из различных кусочков, может очень сильно помочь при составлении денамического запроса, который так или иначе зависит от поступивших на вход параметров.

Именно по этой причине был выбран данный пакет. Как можно видеть из рисунка 3, собирается SQL запрос в зависимости от пришедших на вход параметров. Таким образом, условная секция запроса может варьироваться.

```
1    func (perC personController) GetEmailByID(ctx context.Context,
     person domain.PersonInfo) (string, error){
2       selectStmt := table.Persons.SELECT(table.Persons.Email)
3       var whereStmt postgres.BoolExpression
4
5       personID, _ := uuid.Parse(person.PersonID)
6       switch {
7       case personID != uuid.UUID{} && personID != uuid.Nil:
8         whereStmt = table.Persons.ID.EQ(postgres.UUID(personID))
9       case person.TelegramID > 0:
10        whereStmt = table.Persons.TelegramID.EQ(postgres.Int64(
     person.TelegramID))
11      default:
12        return "", customerrors.ErrEmailsNotFound
13      }
14      stmt, args := selectStmt.WHERE(whereStmt).Sql()
15
16      ...
17
18      return email, nil
19    }
20
```

Рисунок 3 – Демонстрация работы пакета jet

1.2 Устройство сайтов

Предметом исследования диплома был ограниченный список сайтов, участвующих в сборе картинок с последующей их сортировкой и группировкой в pdf файл для последующего их удобного просмотра или прочтения. Суть изучения состояло в том, чтобы произвести реверс-инженеринг сайта, понять как он работает. Затем использовать полученные знания в поиске нужной мне информации, путем парсинга страницы сайта, разбиения полученной структуры сайта на составляющие, с последующей структуризацией в нужный мне формат для правильной отдачи итогового результата.

Абсолютно любой сайт состоит из 3-х компонент:

– HTML,
– CSS,

– JavaScript.

1.2.1 HTML

В данной компоненте описывается различная информация о странице. Будь то текст статьи, картинки и прочего. Затем, браузер, применяя свои движки визуализации, преображает инфорацию, содержащуюся в компоненте, в понятное для человека визуализированное представление. HTML имеет свою структуру, называемой DOM деревом. Эта структура по сути своей является набором различных тэгов. Для идеального примера можно сказать, что HTML похож на LATEX. Таким образом, HTML — это то, откуда будет получаться нужная информация.

Как было сказано ранее, HTML состоит из тэгов. Есть следующие виды тэгов:

– двойные,
– ординарные.

Двойной тэг отличается от ординарного тем, что у двойного тэга есть закрывающий тэг. Между открывающим тэгом и закрывающим пишется текстовая информация. Например, на рисунке 4 продемонстрирован двойной тэг, обозначающий параграф. Обычно такой тэг используется в различных статьях. На рисунке 5 продемонстрирован пример ординарного тэга, обозначающий картинку. Когда браузер видит этот тэг, он понимает, что перед ним картинка. За кадром происходит запрос на сервер по ссылке, расположенной в мета информации, с последующим отображением изображения.

```
1        <p class="paragraf">Текст параграфа</p>
2
```

Рисунок 4 – Пример двойного тэга

```
1        <img src="picture.img" alt="Cup of Kitty">
2
```

Рисунок 5 – Пример ординарного тэга

На рисунках 4 и 5 были продемонстрированы тэги, которые требуют от

браузера некоторых действий. Но, как было выше отмечено, HTML очень похож на LATEX. И похож он тем, что в основном страницы сайта нужно верстать. Поэтому люди придумали тэг, который не требует никаких действий со стороны браузера, но при этом будет как-то объединять информацию.

На рисунке 6 изображен тэг div. Этот тэг нужен исключительно для того, чтобы как-то объединять информацию, присваивая ей какую-то общую метаинформацию, например, класс.

```
1  <div class="pritty"></div>
2
```

Рисунок 6 – Тэг div

Из совокупности тэгов состоит страница сайта. А структура, которая в конечном итоге образуется, называется DOM деревом.

1.2.2 CSS

Эта компонента, в свою очередь, отвечает за то как будет выглядить информация, содержащаяся в HTML. По сути своей, CSS — оформление страницы. После появления данной компоненты, сайты в интернете стали выглядить так, как мы их сейчас видим.

Для формирования различных стилей, используется метаинформация тэгов. Удобнее всего для таких нужд оказалось использовать тэг, изображенный на рисунке 6, так как этот тэг как никто лучше служит для объединения информации, которую можно как-то по-особенному расположить и украсить.

Именно поэтому на большинстве сайтов можно увидеть именно этот тэг.

1.2.3 JavaScript

Эта компонента отвечает за логику на страницах. Логика может использоваться при анимировании сайтов и общении с сервером. Общение с сервером нужно для того, чтобы возможно было у него получить какую-то информацию.

1.3 Методика парсинга

Понимание сайта — важная часть абсолютно любого парсинга данных.

Как понятно из определения, необходимо собирать информацию. Для того, чтобы собирать информацию, необходимо знать как она расположена. Для того, чтобы понимать как расположена информация, необходимо исселодовать пути появления этой информации перед пользователем и выяснить откуда она берется.

Плавно мы подбираемся к тому, что знание устройства сайта, клиентской части и сервера — необходимы для ранее сказанного парсинга данных. Другими словами, для парсинга данных с сайта необходимо полностью понимать как тот устроен. Если же не получается полностью изучить, то необходимо понять, какие данные должны использоваться, чтобы в конечном итоге достичь нужного результата. Нужно цепляться буквально за любую крупицу данных, которая может так или иначе помочь.

1.3.1 Взаимодействие с сервером

Для поиска подобной крупицы информации нужно понимать, что ничего не происходит из ниоткуда. Везде есть какие-то следы. Они могут быть зашифрованы, или спрятаны за тонной других запросов. Но информация откуда-то получается. Нужно сделать оговорку, потому что есть сайты, которые не используют сервер для получения какой-либо информации, а она уже сразу закодирована в верстке сайта. Такими сайтами называются «визиткой». Как понятно из навзвания, такие сайты нужны для того, чтобы красиво продемонстрировать род деятильности, прорекламировать компанию, которую представляет тот или иной сотрудник и указать там контакты для сотрудничества. Обычно такие сайты одностроничные, под собой не имеют никакой логики. Такие сайты очень удобно парсить. Но чаще всего человек сталкивается с сложными сайтами со сложной логикой, где есть большая база пользователей и контета. Такие сайты просто обязаны обращаться за какой-то логикой на сервер.

Можно заметить, что в разделе 1.2.3 упоминается JS — логическая сторона сайта. Почему не пользоваться этой замечательной компонетой сайта для различной логики? Дело, конечно же, в безопасности. Куда более надежно будет положить какую-то информацию в переменную, а потом ее в будущем отобразить на странице при помощи, например, jQuery. Никто не хочет, чтобы алгоритмы обработки личной информации лежали перед всеми на видном месте, чтобы их можно было запустить и все расшифровать. Или как еще

по-другому можно хранить миллионы миллиардов информации?

Итак, так как предмет исследования как раз идет за нужной информацией на сервер, необходимо понять из каких крупиц информации создаются следующие более сложные запросы на сервер. Для этого необходимо изучить выдаваемую информацию на сайте, запросы, которые исполняются в процессе загрузки страницы. Исследование нужно производить с конца, идя в самое начало. Например, если мне нужно достать нужную мне картинку, то мне нужно найти запрос, который эту картинку запрашивает. Далее нужно найти запрос, который делает предыдущий запрос и так до самого основание.

Таким образом мы можем понять как сайт взаимодействует с сервером при различном получении разных страниц.

1.3.2 Изучение данных

Необходимо сделать несколько оговорок:

- все примеры будут продемонстрированы в браузере Google Chrome, так как в нём имеется богатый инструментарий для программистов, а также поддерживается огромное колличество сайтов, ведь именно этот браузер является самым популярным [5],
- комбинации клавиш, продемонстрированные в примерах, актуальны для операционной системы Windows.

Для поиска информации необходимо пользоваться инструментом разработчика. Открывается данное меню разработчика при помощи нажатия клавиши F12. На рисунке 7 продемонстрировано меню.

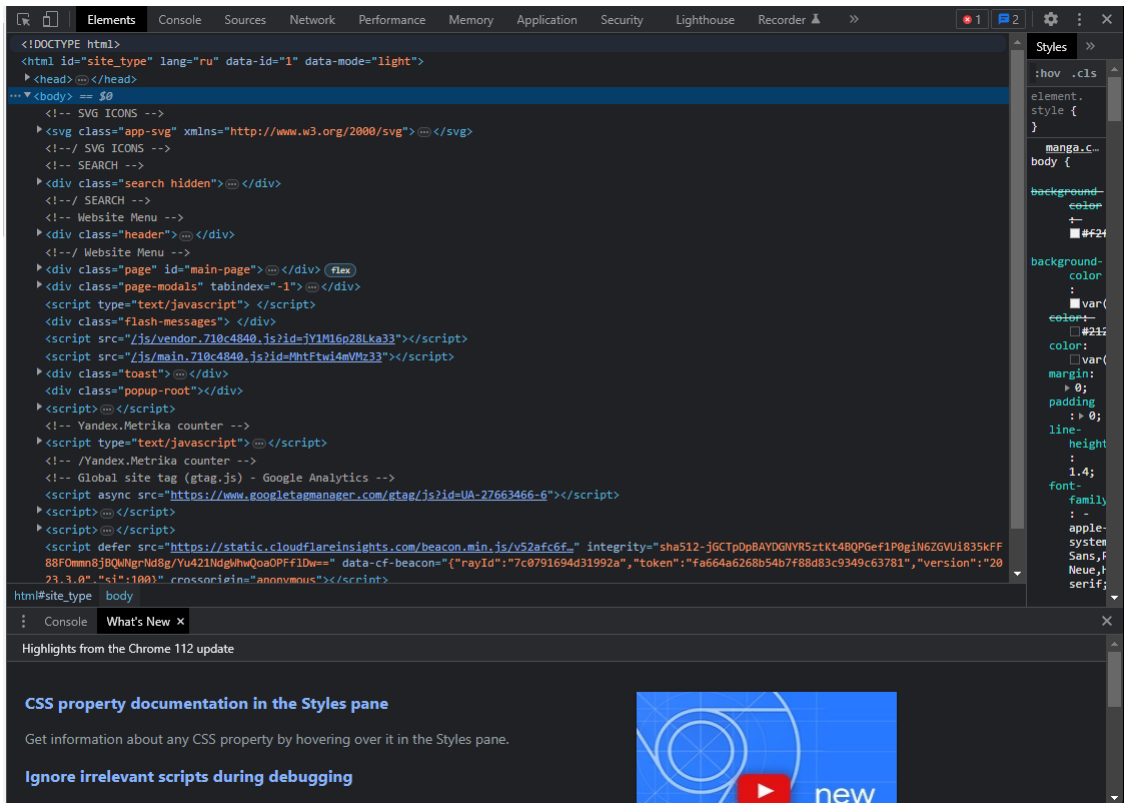

Рисунок 7 – Меню разработчика

У браузера есть свойство, при котором он исполняет все скрипты в тегах языка HTML. Как говорилось ранее, эти скрипты могут содержать запросы на сервер за получением информации на страницу. Также еще нужно знать, что вписывание ссылки в браузер и последующем нажиманием подтверждения ввода, по сути своей делает GET запрос на домен с соответствующим роутингом. Зная также тот факт, что на один запрос приходится один ответ, мы можем определенно точно сказать, что программа наша получит сайт с первичной информацией. То есть будет загружены скрипты, которые делают последующие запросы на сервер. Из этого мы можем сделать вывод, что просто так никакая информация из ниоткуда не берется и с большой вероятностью мы при помощи только одного запроса не получим весь сайт. Для тестирования запросов можно пользоваться удобным инструментом postman [6]. На рисунке 8 продемонстрирован пример одного GET запроса на сайт через инструмент postman. На рисунке А.1 приложения изображена команда, которую можно сделать, чтобы посмотреть полностью код HTML.

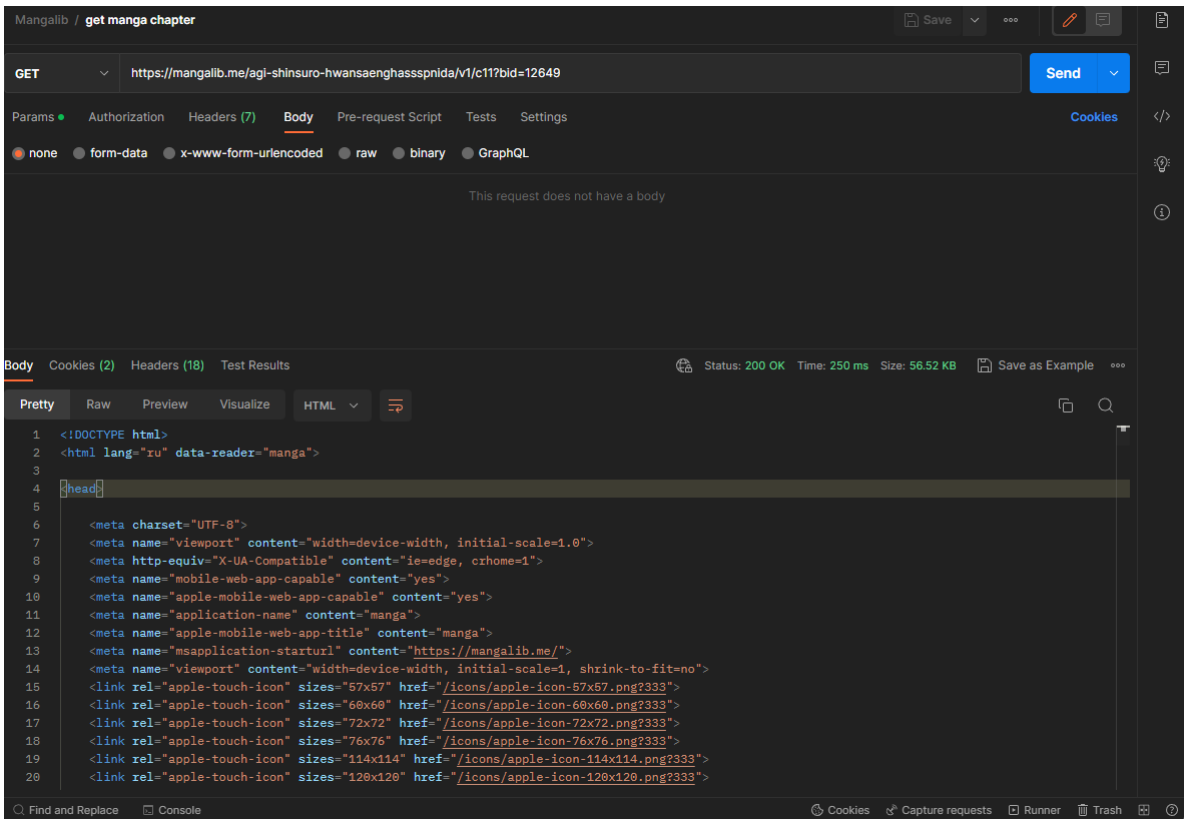

Рисунок 8 – GET запрос

Из кода можно увидеть, что HTML код очень сильно отличается от того, что изображен в меню разработчика. Это все потому, что мы не проделали все те запросы, что делает браузер, когда начинает прогрузку страницы. К счастью, нам этого делать не надо. Это лишь доказательство того, что до любой информации мы можем одним или несколькими путями добраться.

Для того, чтобы начать изучение данных, необходимо зайти на сайт и найти самостоятельно ту конечную информацию, которая нужна. В данном случае ведется поиск картинок. После того, как мы нашли нужную картинку, необходимо понять как выглядит ссылка на нее. Исходя из той логики, что разработчик сайта будет составлять ссылку из абсолютно случайного набора букв и цифр – крайне мала, так как это решение бесполезно и очень сильно влияет на быстроту разботки в худшую сторону, мы можем попробовать догадаться по какой логике конструируется ссылка на ресурс. Итак, чтобы посмотреть на ссылку картини, нам необходимо посмотреть код этой картинки в HTML. На рисунке 9 видно, как выделен тэг картинки, которую мы бы хотели получить.

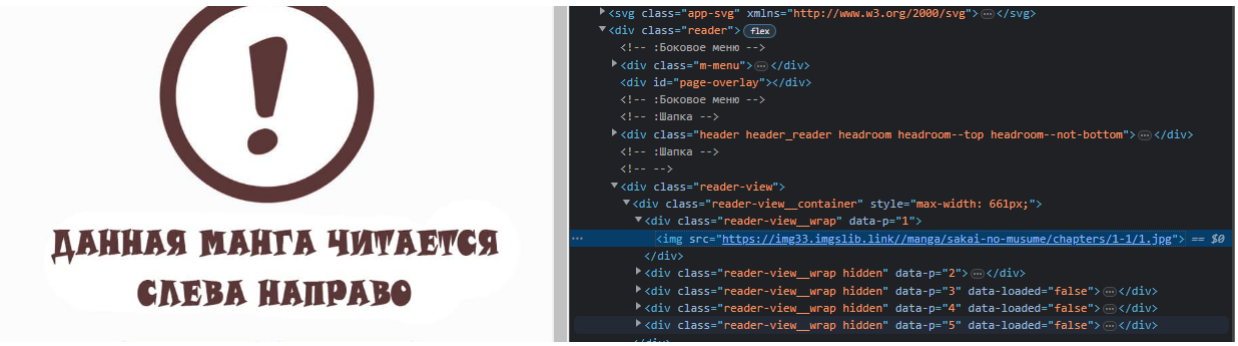

Рисунок 9 – Представление картинки в кода HTML

Посмотрев так на другие картинки, можно увидеть, что у них у всех есть один источник. А ссылка формируется по паттерну из рисунка А.2. Следовательно возможно эту ссылку конфигурировать своими руками в сервисе. Для этого нам нужны:

– название манги,
– правильных id главы,
– хеш картинки.

Теперь необходимо узнать где на странице еще лежит вышеописанная информация. Для этого нам нужно проанализировать запросы. Для этого нам пригодится вкладка Network в меню разработчика. На странице картинки открываем меню разработчика и перезагружаем страницу. Это нужно для того, чтобы во вкладке появились все абсолютно запросы, которые делает браузер для того, чтобы прогрузить полностью страницу. На рисунке 10 продемонстрирован весь список запросов, который делается при загрузке страницы определенного сайта. Очевидно этот список может отличаться от того, что продемонстрировано на рисунке 10.

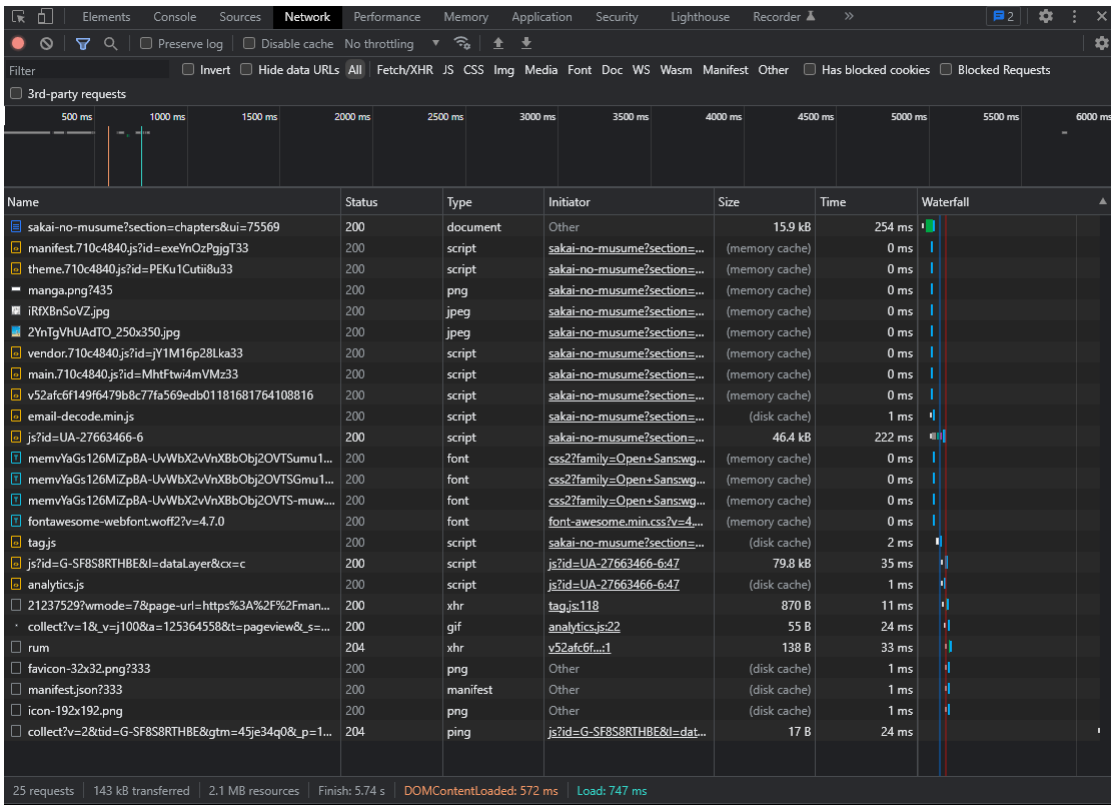

Рисунок 10 – Меню разработчика network вкладка

Далее необходимо найти эту картинку в запросах. В меню есть фильтр по типам файлов. Выбираем там картинки. На рисунке 11 мы видим, что не очень много запросов было на картинки. С легкостью можем найти наш файлик (на рисунке выделен). Можно еще также заметить, что разработчики постарались и заренее прогружают следующую страницу, чтобы было проще смотреть контент.

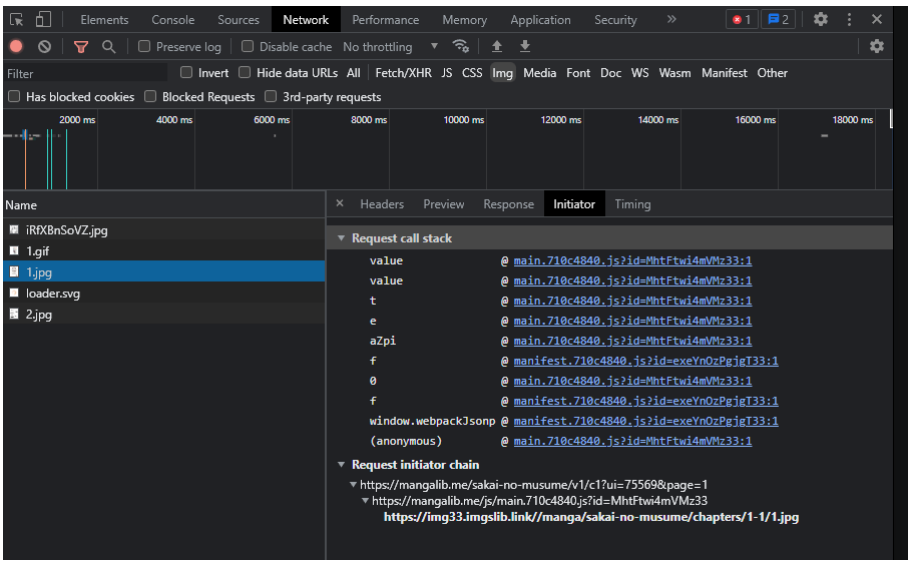

Рисунок 11 – Меню разработчика network вкладка с фильтром по картинкам

Теперь нам необходимо узнать как был получен запрос на эту картинку. Помним, что из-за особенности устройства браузера информация не появляется из ниоткуда. Следовательно мы можем с легкостью узнать откуда пришел запрос. Благо нам не нужно особо много, чтобы раздобыть подобного рода информацию. Меню разработчика предоставляет интерфейс для показа стека вызова. На рисунке 11 можно увидеть вкладку Initiator. Это и есть наш стэк вызовов.

На самом деле, если мы будем смотреть код каждого запроса, сможем легко закопаться в нем, так как чаще всего подобного рода сайты подключают разные сервисы для генерации логики. Этот сайт не исключение. Здесь добавлена защита от DDOS атак от компании Яндекс, также трекер использования сайта. Через подобного рода скрипты происходит процесс анализа запросов. Поэтому в середине запроса может быть сгенерированный код, который почти невозможно понять без нескольких дней его анализа. Благо в нашем случае его анализировать не надо. Можно начать смотреть код, который находится вокруг места, откуда произошел вызов того или иного запроса. Таким образом натыкаемся на самый первый вызов в стеке и замечаем, что это наш любимый HTML код. Изучив его, можно увидеть, что есть большой скрипт. В нем есть переменная «window.__DATA__» (при помощи вызова скрипта на рисунке А.1 можно увидеть этот скриат). Кажется это то, что нам нужно.

Переменная «window.__DATA__» — JSON, в котором явно написана

информация о манге. Исследовав этот объект, обнаруживаем, что там есть информация о названиях каждой страницы. Можно также заявить и сколько страниц в данной конкретной главе. На рисунке 12 можно посмотреть на получившуюся структуру, в которую можно парсить из JSON.

```
1     type pageList struct {
2             Page int    `json:"p"`
3             Url  string `json:"u"`
4         }
5
```

Рисунок 12 – Структры содержания информации о главах и картинках

В Url атрибуте хранится тот самый хэш картинки. ID главы можно узнать из атрибутов на той же странице. Название манги совсем просто найти. Оно находится буквально в каждой ссылке данного сайта. Поэтому можно просто скопировать название оттуда.

Примеры выше были для сайта mangalib.me, участвующий в итоговом дипломе. Анализ каждого сайта происходит в индивидуальном порядке, так как везде свои разные алгоритмы и подходы. Задача микросервиса сделать так, чтобы добавление новых сайтов было достаточно простым и требовало исключительно анализа и исполнении всех алгоритмов для автоматического поиска.

1.3.3 Сайты с API

Есть некоторые сайты, разработчики которых подумали о пользователях и о том, что эти пользователи возможно хотели бы автоматизировать процесс поиска и скачивания контента. Одним из таких сайтов является mangadex [7].

Для подобных сайтов нет необходимости заниматься реверс-инженерингом, как было представлено выше. Достаточно изучить документацию [8] и найти хендлеры, которые выполняют поставленную задачу и использовать их. Однако, это не означает, что предварительную обработку не нужно делать. Благо вышесказанный сайт имеет много примеров использования и свой SWAGGER [9] для тестирования.

Тестирование подобного API происходит через принцип «проб и ошибок», так как не всегда случается такое, что появляется на выходе

ожидаемая информация. В некоторых случаях нужно заниматься ее обработкой.

1.4 База данных в микросервисах

Для некоторых микросервиов база данных совсем не нужна. Такие микросервисы обычно не используют никакую информацию, например, сервисы по конвертированию данных. Там запускается какой-то разовый алгоритм и все. Часто сохранять информацию в один csv файл. Однако сервисы, которые в основе своей работают с данными пользователями, просто необходим. Собранную информацию потом возможно использовать для анализа, чтобы, например, делать на ее основе модели искусственного интеллекта.

Одной из основных причин использования баз данных в микросервисах является возможность распределения данных и обработки запросов между несколькими сервисами. Каждый сервис может иметь свою собственную базу данных или использовать общую базу данных, что позволяет сократить нагрузку на каждый сервис и повысить производительность всей системы.

Кроме того, базы данных обеспечивают целостность данных и защиту от потери информации. Они позволяют хранить информацию в структурированном виде, что упрощает ее поиск и обработку. Базы данных также обеспечивают безопасность данных, позволяя контролировать доступ к ним и устанавливать права доступа для каждого сервиса.

Базы данных также позволяют решать проблемы масштабирования и управления ресурсами. Благодаря возможности горизонтального масштабирования баз данных, можно легко расширять хранение данных и увеличивать производительность системы при необходимости. Кроме того, базы данных позволяют эффективно управлять ресурсами, например, памятью и дисковым пространством, что повышает эффективность работы всей системы.

Использование баз данных в микросервисах также упрощает разработку и тестирование сервисов. Каждый сервис может использовать свою собственную базу данных для тестирования и разработки, что позволяет избежать конфликтов между сервисами и обеспечить более быстрое и эффективное тестирование.

Также, базы данных помогают поддерживать констистентность данных.

Очень важно эту особенность учитывать и в сервисе по поиску картинок. На ее основе можно сохранять ранее полученную информацию и не беспокоить сайт при каждом новом запросе. Эта особенность очень важна в тех случаях, если сайт имеет защиту против DDOS атак. Если сервисом будет пользоваться даже сотни людей, то у сайта может возникнуть вопрос о появлении стольких запросов с одного IP. Именно поэтому намного лучше и проще будет сохранять всю информацию в базе данных.

Приведу пример, есть популярная манга, которая так или иначе пользуется спросом. При помощи поиска этой же манги и конкретной главы возможно сразу из памяти БД достать информацию, лишний раз не нагружая сайт запросами. Таким образом такое поведение очень хорошо скажется на:

- работоспособности приложения. Не придется каждый раз ждать ответа от конкретного сайта,
- внезапно не будет заблокирован наш микросервис за большое количество запросов на сайт,
- не придется в микросервисе обходить защиту от роботов (captcha).

Перед тем, как использовать БД, нужно также подумать о том что будет храниться там. Очень важно это сделать на этапе конструирования. Иначе может получиться так, что эффекта от такого мощного инструмента не будет.

Также при помощи БД возможно сделать подписку на обновление контента. С ее помощью можно организовать рассылку, достав большой объем нужных данных. С файлом csv это было бы очень трудно сделать.

Также преимуществом работы с БД является транзакции. При помощи транзакции можно изменять чувствительные данные, не боясь при этом гонки (race condition).

1.5 Чистая архитектура

Clean Architecture - это популярный подход к разработке программного обеспечения, который был предложен Робертом Мартином (также известным как Uncle Bob). Этот подход помогает разработчикам создавать гибкие, расширяемые и легко тестируемые приложения.

В Golang Clean Architecture часто реализуется с помощью пакетов и интерфейсов. Пакеты используются для организации кода в логически связанные блоки, а интерфейсы - для определения контрактов между различными компонентами приложения. На рисунке 13 представлена схема

такого взаимодействия пакетов.

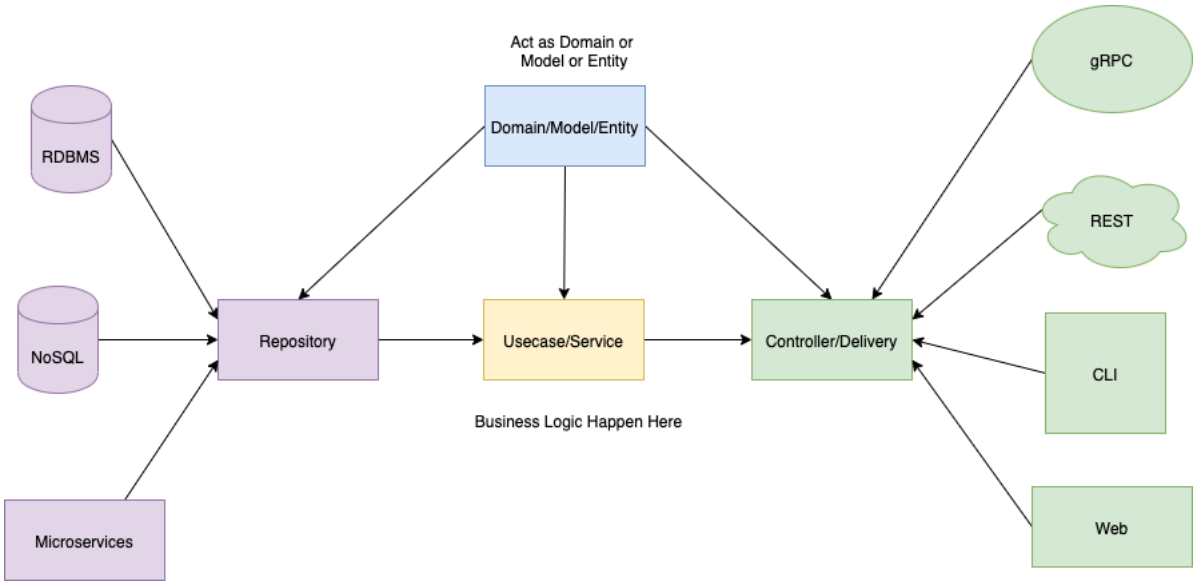

Рисунок 13 – Схема чистой архитектуры

В центре Clean Architecture находится принцип разделения зависимостей (Dependency Inversion Principle), который гласит, что зависимости должны быть направлены от абстракций к конкретным реализациям, а не наоборот. Это позволяет разработчикам легко заменять реализации компонентов приложения, не затрагивая другие части системы. Так, например, область, отвечающая за сохранение данных не должна содержать никакую логику. Она должна только выдавать или сохранять данные.

Стоит оговориться, что не стоит обязательно точно придерживаться этому концепту. Иногда в этом нет особой надобности. Но такое разделение лучше делать всегда и сохранять его в том или ином виде.

Одним из основных преимуществ Clean Architecture в Golang является возможность легко тестировать каждый слой независимо от других. Это достигается путем использования интерфейсов для определения контрактов между слоями. Это позволяет разработчикам легко заменять реализации компонентов приложения на макеты (mocks) для тестирования.

Кроме того, Golang Clean Architecture обеспечивает высокую гибкость и расширяемость приложения. Благодаря разделению зависимостей и использованию интерфейсов, программисты могут легко добавлять новые функции и изменять существующие, не затрагивая другие части системы.

В Golang Clean Architecture используется многослойная архитектура.

1.5.1 Представление (Presentation Layer)

Этот слой отвечает за взаимодействие с пользователем и обработку входящих запросов. Он содержит контроллеры, представления и другие компоненты, которые отображают данные для пользователя. В вышеизложенном сервисе расположены хендлеры, которые отвечают за обработку запросов по конкретному роутингу.

1.5.2 Использование (Use Case Layer)

Этот слой содержит бизнес-логику приложения. Он определяет, как приложение должно взаимодействовать с данными и какие операции должны быть выполнены. Он также содержит интерфейсы для доступа к данным. Например, в этом слое можно организовать логику парсинга картинок, организацию их по подгруппам и сбор в pdf. Затем данные отправлять в следующий слой.

1.5.3 Хранилище (Storage Layer)

Этот слой отвечает за хранение данных и доступ к ним. Он содержит реализации интерфейсов, определенных в слое использования. Эти реализации могут использовать различные источники данных, такие как базы данных, файлы или внешние API.

1.5.4 Инфраструктура (Infrastructure Layer)

Этот слой содержит код, который не относится к бизнес-логике приложения, но необходим для его работы. Это может быть код для обработки ошибок, логирования, аутентификации и т.д. Этот слой также содержит реализации интерфейсов, определенных в слое использования, для доступа к внешним сервисам и системам.

На рисунке Б.1 демонстрируется пакетное представление проекта.

2 ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ

2.1 Структура приложения

Приложение состоит из основной серверной части — API и клиентских частей. Клиентским сервисом считается любой сервис, например бот в мессенджере, способный отправлять запросы на серверную часть. Серверная часть выполнена в соответствии с концепцией чистой архитектуре, в соотвествии с тем, что описан в рзеделе 1.5. На рисунке Б.1 изображена схема проекта. Каждый пакет отвечает за свою определенную часть.

Пакет «repository» отвечает за взаимодействие между БД. В «cmd» находится main.go файл, который запускает сервер. «App» содержит в себе реализацию хендлер функций. Пакет «config» используется по всему проекту. В нем описаны функции, помогающие доставать различные секреты приложения, такие как пароли, из переменных окружения. В этом же пакете происходит парсинг тех из yaml файла. В пакете «gen» содержится сгенерированные структуры библиотекой jet из раздела 1.1.3. В «internal/pgk» содержатся все пакеты, которые так или иначе выполняют какую-то логигу приложения. Своего рода usecase слой. Пакет «parse» отвечает за логику парсинга описанного для конерктного сайта. Так можно увидеть там один из объектов исследования — сайт mangalib «Pdf_creator», как понятно из названия, занимается описанием логики сбора картинок в один pdf файл. Пакет «mailer» необходим для того, чтобы отправлять файлы через через какие-либо возможные способы передачи, например, отправки тех через электронную почту. В папке «migrations» описан SQL код всех миграций БД.

Внешние пакеты (те, что находится не в папке «internal») используются для того, чтобы к ним возможно было получить доступ мнешним клиентским сервисам. Так в «api» пакете описаны структуры request и response, отвечающие за боди запроса и ответа соответственно. Также в модуле описан роутинг API. Пакет «pgk» необходим для того, чтобы импортировать тот в клиентский сервис, так как там предполагается описание sdk. В этом пакете описаны кастомные ошибки, которые можно в последствии обработать.

2.2 БД в парсинге

Важной составляющей микросервиса является база данных. В ней

хранится совсем вся информация о пользователях и о структуре манги. Главной особенностью базы в данном проекте является то, что можно контролировать количество пользователей, которые используют сервис. Таким образом возможно принимать решение о том нужно ли заниматься оптимизацией или улучшением компонент сервера. БД также позволяет легко получить доступ к данным, ограничить его же другим пользователям и легко перенести информацию от одного сервера в другой.

Однако, выше были перечисленные общие преимущества для микросервисов. Главным же преимуществом базы данных в данном проекте заключается в том, что можно сохранять структуру глав к любой манге. Таким образом не придется делать большое количество запросов на каждого пользователя. Возможно ограничиться только запросами на загрузку картинок с сервера. При использовании клиентского сервиса в виде мессенджера телеграм, возможно загружать мангу, создавая статьи. Каждой статье в телеграме предоставляется своя ссылка. Так при генерировании статьи пользователем, ссылка может сохраниься в базе данных. В случаях, когда какая-то манга является популярной, возможно лишь выдавать в последствии ссылку уже подготовленную статью. Преимущество статьи выражается в том, что телеграм подготовил весь необходимый удобный интерфейс. Поэтому просмотр комикса выглядит локанично.

Если к API подключить сервис по облачному хранилищу или же прямо непосредственно на сервере хранить все уже загруженные и сгенерированные главы в виде pdf, то вполне возможно загружать в БД пути к определенным файлам, чтобы не делать заново генерацию.

БД может помочь в дальнейшем развитии проекта, а именно организации подписки на рассылку выхода новой главы конкретной манги. Так можно сделать воркер, способный в фоне раз в какое-то время обновлять структуру глав манги. При обновлении этой структуры, будет происходить рассылка пользователям, которые подписались на обновление комикса. Структуру глав манги можно хранить в виде документа JSON.

Особенностью конструирования схемы БД является то, что обновление ее происводится при помощи миграций [10].

2.2.1 Описание сущностей БД

На рисунке 14 представлена схема БД, которая используется на момент

написания ВКР.

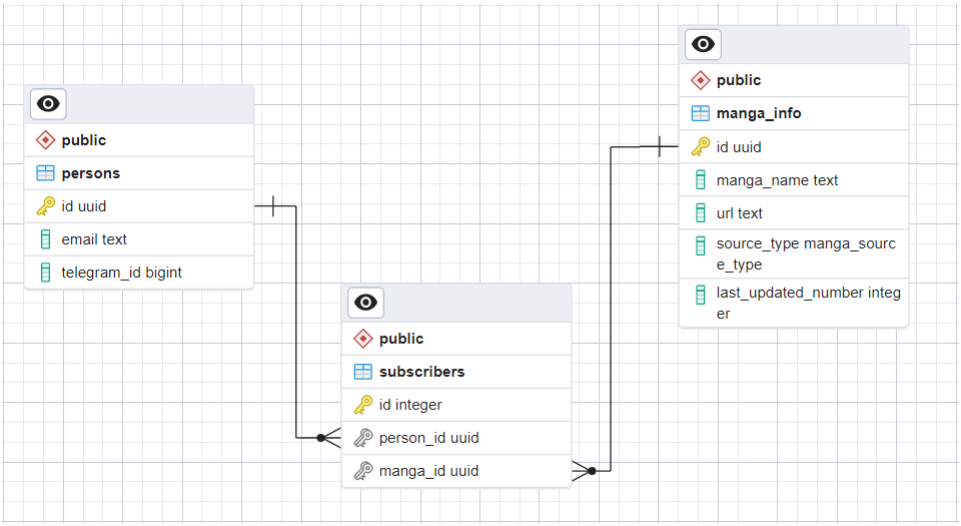

Рисунок 14 – Схема БД проекта

В этой схеме можно увидеть несколько таблиц:

– persons,
– manga_info,
– subscribers.

Опишем предназначение каждой из таблиц.

2.2.1.1 Persons

Таблица persons нужна для того, чтобы записывать информацию о пользователе. Информация может быть о его логине, ID в каком-то мессенджере или любая другая информация, которая может помочь идентефицировать пользователя. Также возможно хранить тут контактую информацию, по которой возможно уведомлять о рассылке. Сейчас эта таблица нужна для того, чтобы сохранить email, на который нужно присылать мангу, чтобы синхронизировать ту с электронной книгой. Таким образом происходит защита от спам рассылки на другие email. Сначала пользователю обязательно нужно залогиниться в API, подтвердить свой email, а уже затем возможно отправлять сгенерированные файлы себе.

Регистрация будет происходить таким образом для массового пользователя, что ему необходимо получить код, который будет в файле, отправленный на электронную книгу. Так возможно добиться подтверждения от пользователя, что почта точно принадлежит нужному человеку.

2.2.1.2 Manga_info

В таблице manga_info, как понятно из названия, хранится информация о структуре манги. В этой таблице в дальнейшем можно хранить ссылки на статью, которая сгенерируется для клиентского сервиса в телеграме. Когда множество пользователей захочет скачать себе одну главу, повторная информация будет браться из этой таблицы, дабы не нагружать сайт запросами лишний раз. Таблица также имеет идентефикатор типа источника информации. Эта информация нужна для того, чтобы понять какой тип алгоритма расшифровки структуры глав манги нужно применять. Сделано это было потому, что структура сайтов и, соответственно, организация манги на них может быть совершенно разной. Например, на многих сайтов есть поддержка нескольких языков. Соответственно базовая структура глав манг изменится, путом добавления строчки языка.

2.2.1.3 Subscribers

Таблица subscribers является связующим звеном между таблицей manga_info и persons. Она, как понятно из названия, создана для того, чтобы можно было отслеживать людей, которые подписались на какую-то конерктную мангу. Так можно делать рассылки сразу нескольким людям, используя в коде лишь один запрос с соединением данных. Подписка и отписка от рассылки будет происходить совсем просто: достаточно добавить строчку с ID пользователя и желаемой манги.

2.3 Построение роутинга API

Перед тем, как рассказывать о роутинге, необходимо пару слов сказать о конструкции ссылки на сайте или любом интернет ресурсе.

На рисунке 15 написана структура абсолютно любой ссылки в интернете. Как видно из рисунка, ссылка состоит из нескольких частей:

- http-type,
- domain,
- route.

```
1    <http_type>://<domain>/<route>
2
```

Рисунок 15 – Структура ссылки в интернете

Http-type — это тип передачи HTTP пакета. Их есть 2 штуки:

– http,
– https.

Обычная HTTP передача данный предполагает в себе открытую передачу данных. То есть абсолютно любой человек, который сможет поймать трафик, сможет увидеть все, что вы присылаете. Очевидно, с таким способом передачи информации очень много проблем с безопасностью.

HTTPS — это тот же HTTP, но S означает здесь secure. Как понятно из определения, это означает, что при https передача HTTP пакета каким-то образом шифруется. Для того, чтобы зашифровать данные, необходимо где-то найти сертификат и его переодически обновлять. С помощью этого сертификата и шифруются данные. Есть множество платных сетрификтов. Они чаще всего используются компаниями. Однако, для обычного использования подойдет и бесплатный. Современные браузеры научились предупреждать о любом http соединении, как о небезопасном и чаще всего могут запрещать их посещение.

Domain — это домен. То есть по сути IP адрес сервера, на котором хостится сайт или API. Домен обычно в интернете скрыт под каким-то именем — доменным. Этот домен нужно воспринимать, как IP нашего сервера.

Итак, теперь мы подошли к части с route. Эту часть нужно воспринимать, как файловую систему компьютера. Только за этим путем стоит какая-то функция, которая исполняется на фоне. Сложность составления роутинга состоит в том, чтобы сделать эту файловую систему так, чтобы возможно было в ней ориентироваться и не потеряться.

Как в обычной файловой системе в директории мы можем увидеть только файлы или другие директории, так и тут мы не можем обращаться к промежуточным путям. Для примера давайте посмотрим на рисунок 16. Тут можно увидеть, что «get_address» идет после api. Это значит, что по пути api мы не сможем повесить какой-либо хендлер.

```
        \---api
        |   +---get_address
        |   +---set_address

```

Рисунок 16 – Пример структуры роутинга

На рисунке B.1 можно увидеть структуру роутинга api на момент написания ВКР. Видно, что все хендлеры API расположены в одной структуре «api». Все хендлеры, что относятся к взаимодействию с мангой расположены в директории «manga». Хендлеры, отвечающие за регистрацию пользователей и прочую авторизацию расположены в директории «login».

«Docs» и «docs-local» созданы для того, чтобы можно было получить доступ к swagger. На рисунке 17 можно увидеть пример хендлеров, относящиеся к взаимодействию с мангой.

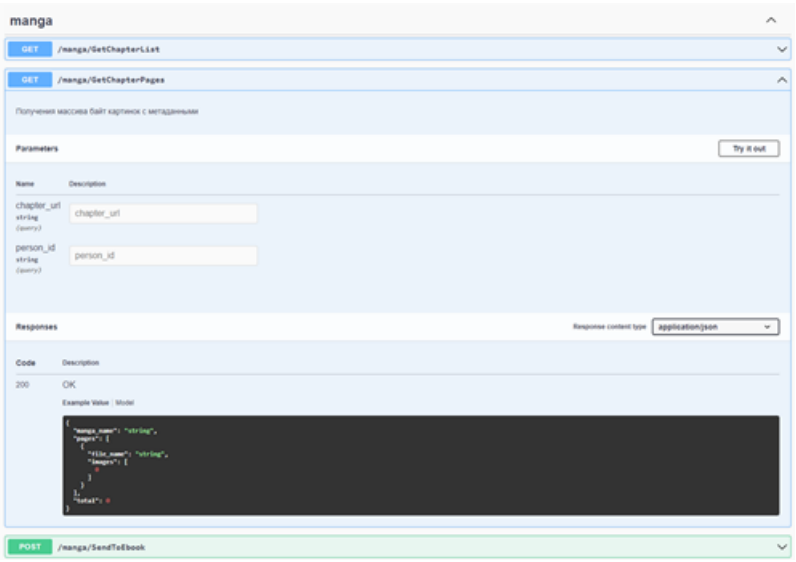

Рисунок 17 – SWAGGER представление хендлеров

2.3.1 Описание хендлеров

Основные хендлеры расположены в директории «manga». Как можно увидеть из рисунка 16 их на данным момент 4 штуки:

- GetChapterList,
- GetChapterPages,
- GetChapterPagesPDF,
- SendToEbook.

Хендер «GetChapterList» отвечает за то, чтобы выдать структурировано информацию о доступных главах конкретной манги в виде ссылок. Ссылки были выбраны потому, что с ними удобнее далее взаимодействовать на клиентских сервисах.

Хендлер «GetChapterPages» выдает картинки в отсортированной структуре. В структуре же находится байтовое представление картинки и вторым полем название файлы, чтобы удобнее потом было на клиентском сервисе собрать картинки вместе в нужном порядке. Такое подход был выбран потому, что предполагается, что клиентские сервисы смогут самостоятельно отобразить картинки так, как это было бы удобнее всего для пользователя.

Хендлер «GetChapterPagesPDF» делает все то же самое, что и «GetChapterPages», за исключением того, что в ответе получается файл pdf с уже собранной мангой.

Хендлер «SendToEbook» выполняет отправку сгенерированной манги на электронную книгу через электронную почту. Обязательно присутствие электронной почты в базе данных.

2.4 Пайплайн работы сервиса

У микросервиса есть свой пайплайн запуска некоторых процессов при активации хендлера. Такой подход к программированию позволяет делать встраивание различных функциональностей по пути и масштабировать сервис вширь. В данном разделе будет описан пайплайн работы сервиса для парсинга картинок.

Для начала сервис необходимо инициализировать различного рода переменными, которые содержат в себе секреты такие, как пароли. Данная информация хранится в ENV переменных.

На рисунке Г.1 изображен снипет кода, в котором написана функция, при помощи которой происходит забор всех необходимых секретов из yaml файла. На рисунке Г.2 представлен пример подобного yaml файла.

Далее происходит инициализация роутинга, после чего инициализируются объекты класса, которые выполняют всю работу.

Код всего парсинга начинается с пакета parser. Напомню, что схема всего проекта находится на рисунке Б.1. Информация из parser, ответ которой являются ссылки, далее ведет к пакету image-getter. Этот пакет начинает в асинхронном режиме скачивать все картинки. Для обработки ошибок в

асинхронном режиме, пришлось придумать систему, которая могла бы записать ошибки, если таковые были, а затем после окончания работы всех горутин сообщить о том, что где-то возникла ошибка. На рисунке 18 изображен пример кода, который выполняет вышеизложенную фукнцию.

```
1       // IsNul проверяет, что нет ошибок.
2       // Если ошибки есть, то возвращает эти ошибки и, ВНИМАНИЕ,
    обнуляет массив ошибок.
3       // После повторного вызова IsNul будет всегда возвращаться
    nil.
4       func (e *ErrController) IsNul() []error {
5           e.mx.Lock()
6           defer e.mx.Unlock()
7
8           if e.errors == nil {
9               return nil
10          }
11          result := make([]error, len(e.errors))
12          copy(result, e.errors)
13          e.errors = nil
14          return result
15      }
16
17      // PutError кладет ошибки в стэк ошибок для дальнейшей их
    обработки
18      func (e *ErrController) PutError(err error) {
19          e.mx.Lock()
20          defer e.mx.Unlock()
21          e.errors = append(e.errors, err)
22      }
23
```

Рисунок 18 – Пример кода с контролированием ошибок

Как видно, в данном коде присутствует мьютекс. Он нас сможет защитить от гонки.

После того, как файлы скачались в определенное место на диске, мы должны либо передать на выход хендлера байтовое представление готовых файлов. Либо мы идем в пакет pdf-builder, где идет сбор всех картинок в один большой pdf.

На рисунке 19 изображен описанный выше пайплайн, где в фигурках диаграммы написаны названия пакетов.

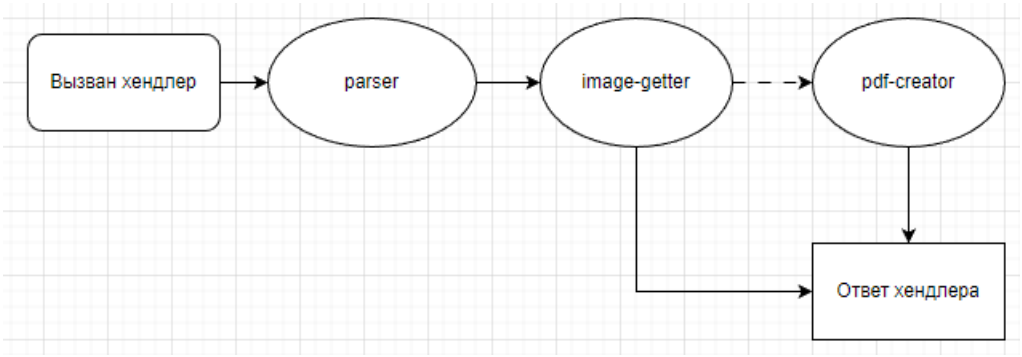

Рисунок 19 – Диаграмма пайплайна

2.5 Подписка на картинки

Было бы очень удобно пользователем автоматически получать обновление на свою электронную книгу своей любимой манги. Поэтому было придумано сделать подписку на определенную мангу на сайте, чтобы можно было не взаимодействуя с API, получить нужную главу манги, если та вышла. Подписка необходима для тех людей, которые имеют огромное количество манги и им просто сложно бывает уследить за тем, что они читали.

Используя все ранее сделанное, мы можем организовать рассылку при помощи базы данных и собранной информацией о главах манги. Так у нас хранится информация в таблицах о структуре манги и есть список пользователей, которые так или иначе захотели подписаться на определенную мангу.

Так как процесс обхода каждой манги необходимо делать без дополнительной помощи со стороны пользователя, было принято решение о том, чтобы запустить некий воркер, способный проходить по всем мангам, которые есть в таблице и обновлять эту структуру. Однако, сразу может прийти в голову мысль о том, что имея при себе 1000 пользователей, которые подписались на 1000 или более комиксов, нам придется сделать не меньше 1000 запросов на сайты с мангой, чтобы обновить структуру глав и отследить обновление. Поэтому нам нужно каким-то образом уменьшить нагрузку на сайт, чтобы нас случайно не заблокировали.

Была придумана схема которая позволяло бы смотреть время последнего обновления структуры манги и задавать не одинаковое время

проверки структуры главы. Есть смысл обновлять структуру не чаще раза в сутки, так как новая глава выходит, самое быстрое, раз в неделю.

Есть более сложный способ, который предполагает смотреть на статистику выхода главы и динамически задавать диапазон проверки на выход следующей главы через примерно похожий промежуток времени. Например, есть комиксы, главы которых выходят не чаще, чем раз в месяц. Поэтому смысла проверять обновления структуры глав совершенно не стоит.

Для всех вышеизложенных идей нам поможет собранная информация о мангах в таблицах БД. Соответсвенно в таблицу об информации о манге добавится еще одно значение — время, после которого стоит проверить структуру глав на обновление.

Далее должен присутствовать воркер, который раз в сутки, а можно и даже в разы чаще, проверять время в таблице с информацией о манге. Так, если мы заметим, что в столбце со временем значение окажется меньше, чем настоящее время, мы пойдем делать запросы на обновления структуры глав комикса.

Проблема, которая может возникнуть внезапно, что комиксы могут выходить в одно время, но вероятность этого крайне мала, а несколько десятков запросов сайт сможет выдержать.

ЗАКЛЮЧЕНИЕ

В результате выполненной выпускной квалификационной работы бакалавра был сделан API, способный предоставлять быстро и просто офлайн доступ к любой манге для дальнейнего прочтения в удобных мессенджерах или электронной книге. Оценка полноты решений показала, что данное направление позволяет настроить формат чтения под свои потребности и сохранять любимые серии для повторного чтения без необходимости загрузки каждый раз заново. Также была организована база данных для сбора информации о пользователях, скаченной манги и отслеживание ее обновления.

Рекомендуется использовать результаты выполненной выпускной квалификационной работы бакалавра для продвижения электронных книг в качестве основного способа чтения манги, а также для разработки новых функций и возможностей, которые могут улучшить процесс чтения.

Оценка технико-экономической эффективности внедрения показала, что использование электронных книг для чтения манги может сэкономить время и деньги, которые ранее были затрачены на поиск и загрузку серий в интернете.

Сравнение выполненной ВКР бакалавра с лучшими достижениями в этой области показало, что использование электронных книг для чтения манги является инновационным и перспективным направлением, которое может улучшить процесс чтения и повысить удовлетворенность пользователей.

Репоизторий с результатами работы [11].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *GitHub*. Pdfcpu. — 2023. — URL: https://github.com/pdfcpu/pdfcpu (дата обращения 28.04.2023).

2. *GitHub*. Goquery. — 2023. — URL: https://github.com/PuerkitoBio/goquery (дата обращения 28.04.2023).

3. *JQuery*. documentations. — 2023. — URL: https://jquery.com/ (дата обращения 28.04.2023).

4. *GitHub*. Go-jet. — 2023. — URL: https://github.com/go-jet/jet (дата обращения 29.04.2023).

5. *Similarweb*. Доля рынка топ браузеров. — 2023. — URL: https://www.similarweb.com/ru/browsers/ (дата обращения 01.05.2023).

6. *Postman*. Getting started. — 2023. — URL: https://www.postman.com/ (дата обращения 01.05.2023).

7. *Mangadex*. Главная страница. — 2023. — URL: https://mangadex.org/ (дата обращения 01.05.2023).

8. *Mangadex*. API documentation. — 2023. — URL: https://api.mangadex.org/docs/ (дата обращения 01.05.2023).

9. *Youtube*. Что такое Swagger и OpenAPI за 3 минуты. — 2023. — URL: https://youtu.be/aaFDBgPdXw4 (дата обращения 01.05.2023).

10. *Хабр*. Миграции баз данных — обзор библиотеки и ее использование. — 2011. — URL: https://habr.com/ru/articles/133312/ (дата обращения 09.05.2023).

11. *GitHub*. Репозиторий с результатом работы. — 2023. — URL: https://github.com/whitewolf185/mangaParser (дата обращения 03.05.2023).

ПРИЛОЖЕНИЕ А

Изучение данных

На рисунке А.1 демонстрируется bash скрипт, который можно исполнить для получения HTML кода анализируемой страницы сайта.

```
#!bin/bash
curl --location \
    'https://mangalib.me/agi-shinsuro-hwansaenghassspnida/v1/c11'
```

Рисунок А.1 – Пример HTML кода получения страницы сайта

На рисунке А.2 демонстрируется шаблон, по которому формируется ссылка на картику сайта mangalib.me.

```
        https://img33.imgslib.link//manga/{manga-name}/chapters/{
    chapter-hash}/{image-hash}

```

Рисунок А.2 – Шаблон конфигурируемой ссылки для конкретной картинки

ПРИЛОЖЕНИЕ Б

Чистая архитектура

На рисунке Б.1 демонстрируется файловая схема проекта.

```
+---api
|   +---domain
|   +---middleware
|   \---router
+---cmd
+---docs
+---internal
|   +---app
|   +---config
|   |   \---flags
|   +---gen
|   |   \---manga_parser
|   |       \---public
|   |           +---enum
|   |           +---model
|   |           \---table
|   +---pkg
|   |   +---err_controller
|   |   +---mailer
|   |   +---parse
|   |   |   +---http_requester
|   |   |   \---mangalib
|   |   |       \---mock
|   |   +---pdf_creator
|   |   |   +---mock
|   |   |   \---pdf_merger
|   |   \---utils
|   \---repository
+---migrations
\---pkg
    \---custom_errors
```

Рисунок Б.1 – Схема пакетов проекта

ПРИЛОЖЕНИЕ В

Роутинг проекта

На рисунке В.1 изображен роутинг API в проекте на момент написания ВКР.

```
+---api
|   \---manga
|       +---GetChapterList
|       +---GetChapterPages
|       +---GetChapterPagesPDF
|       +---SendToEbook
|   \---login
|       +---Register
+---docs
+---docs-local
```

Рисунок В.1 – Схема пакетов проекта

ПРИЛОЖЕНИЕ Г

Пайплайн проекта

На рисунке Г.1 изображен снипет кода, в котором написана функция, при помощи которой происходит забор всех необходимых секретов из yaml файла.

```
        type Env struct {
            Name string `yaml:"name"`
            Value string `yaml:"value"`
        }
        type Configs struct {
            Envs []Env `yaml:"env"`
        }
        const defaultEnvFileName = "values_local.yaml"
        var (used bool = false
            currentFileDir string)
        func InitServiceFlags() {
            envFileName := os.Getenv("ENV_FILE_NAME")
            ...
            var yamlResult Configs
            yamlData, err := os.ReadFile(currentFileDir + "/" +
    envFileName)
            if err != nil {
                panic(err)
            }
            //... unmarshaling
            for _, v := range yamlResult.Envs {
                os.Setenv(v.Name, v.Value)
            }
            used = true
        }
```

Рисунок Г.1 – Парсинг секретов в env переменные

На рисунке Г.2 yaml файла, в котором содержатся секреты сервиса для инициализации проекта.

```
env:
  - name: email_account
    value: example@examp.exe
  - name: email_password
    value: password
  - name: db_dsn
    value: postgres_credit_info

  - name: http_retry_duration
    value: 1s

  - name: listen_port
    value: 80
  - name: server_ip
    value: localhost
```

Рисунок Г.2 – YAML файл с секретами сервиса