|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_Информатика и Системы Управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_Программная инженерия\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

***\_*** **Система рекомендаций комиксов *\_***

Студент \_\_ИУ7-21М\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_** Д. М. Иванов**\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**Г. А. Щетинин**\_\_**

(Подпись, дата) (И.О.Фамилия)

*2022 г.*

Оглавление

[Введение 3](#_Toc115484266)

[Цель работы 3](#_Toc115484267)

[1. Аналитическая часть 4](#_Toc115484268)

[1.1 Существующие аналоги 4](#_Toc115484269)

[1.2 Функциональные требования 4](#_Toc115484270)

[1.3 Требования к отказоустойчивости 4](#_Toc115484271)

[1.4 Паттерн Circuit Breaker 5](#_Toc115484272)

[2. Конструкторская часть 7](#_Toc115484273)

[2.1 Описание архитектуры системы 7](#_Toc115484274)

[2.2 Архитектура приложения. 7](#_Toc115484275)

[2.3 Сценарии использования 8](#_Toc115484276)

[2.4 IDEF0-диаграмма 9](#_Toc115484277)

[2.5 Диаграммы потоков данных 10](#_Toc115484278)

[2.6 Схемы алгоритмов 11](#_Toc115484279)

[3. Технологический раздел 12](#_Toc115484280)

[3.1 Выбор языка программирования. 12](#_Toc115484281)

[3.2 Листинги кода 13](#_Toc115484282)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 17](#_Toc115484283)

Введение

Комиксы – хобби, объединяющее очень широкую аудиторию. Они есть на любой вкус, в очень больших количествах. Из-за этого бывает трудно найти подходящие конкретному человеку комиксы. Решением является рекомендательная система, подбирающая комиксы на основе предпочтений конкретного пользователя. Подобной системой могут заинтересоваться интернет-магазины, однако она очень полезна пользователям даже как самостоятельный инструмент.

Цель работы

Целью данной работы является реализация веб-приложения для подбора рекомендаций пользователем. Пользователь может выбрать понравившиеся и не понравившиеся комиксы среди каталога. Предпочтения можно изменить в любой момент. На основе текущих предпочтений приложение составляет упорядоченный список комиксов каталога, от наиболее рекомендованных к наименее рекомендованным.

Задачи работы:

1. Изучить предметную область.
2. Сформулировать функциональные требования к приложению.
3. Спроектировать архитектуру будущего приложения.
4. Реализовать приложение.
5. Настроить автоматическое развертывание приложения (деплой).
6. Аналитическая часть
   1. Существующие аналоги

Рекомендательные системы имеют, как правило, крупные маркетплейсы, такие, как Yandex Market, Avito и пр. Однако, их рекомендательные системы подбирают рекомендации среди всего ассортимента товаров, и плохо подходят для выбора нового комикса.

* 1. Функциональные требования

Для незарегистрированных пользователей система должна предоставлять следующие возможности:

* просмотр каталога;
* регистрация
* авторизация.

После регистрации пользователь, кроме просмотра каталога, может:

* отметить комикс как «понравившийся» или «не понравившийся»;
* просмотреть сохраненные на данный момент предпочтения;
* получить список рекомендаций на основе сохраненных предпочтений.
  1. Требования к отказоустойчивости

На всех запросах получения данных должен быть реализован паттерн CircuitBreaker.

Запросы, не требующие немедленного ответа, должны при недоступности целевого сервиса направляться в очередь, и выполняться в ее порядке после восстановления доступности сервиса. При этом возвращается статус успешного завершения запроса, даже если он был направлен в очередь.

Для проверки авторизации использовать JSON Web Token (JWT).

* 1. Паттерн Circuit Breaker

Если вызываемый ресурс недоступен из-за того, что он перегружен запросами, необходимо дать ему время, чтобы вернуться в нормальный режим. Кроме того, если служба занята, сбой в одной из частей системы может привести к лавинообразному накоплению сбоев. Например, операция, вызывающая службу, может быть настроена для реализации времени ожидания и будет отвечать сообщением об ошибке, если служба не отвечает в течение этого периода. Однако эта стратегия может привести к тому, что многие параллельные запросы к одной операции будут заблокированы до истечения периода ожидания. Эти заблокированные запросы могут содержать критические системные ресурсы, такие как память, потоки, подключения к базе данных и т. д. Следовательно, эти ресурсы могут закончиться, что приведет к сбою других, возможно, несвязанных частей системы, которым нужно использовать те же ресурсы. В этих ситуациях предпочтительно, чтобы операция немедленно завершалась с ошибкой и пыталась вызвать службу, только если такой вызов может быть успешно выполнен. [1]

Автоматическое выключение действует в качестве посредника при выполнении операций, которые могут завершиться со сбоем. Он должен отслеживать количество недавних сбоев и использовать эту информацию, чтобы решить, разрешить ли продолжение операции или же немедленно вернуть исключение.[1]

Этот посредник имеет следующие состояния:

* **Закрытый.** Запрос приложения перенаправляется на операцию. Алгоритм ведет подсчет числа недавних сбоев, и если вызов операции не завершился успешно, алгоритм увеличивает это число. Если число недавних сбоев превышает заданный порог в течение заданного периода времени, прокси-сервер переводится в состояние «открытый». На этом этапе алгоритм запускает таймер времени ожидания, и по истечении времени этого таймера, состояние меняется на «полуоткрытый». Цель таймера времени ожидания — дать системе время на исправление ошибки, которая вызвала сбой, прежде чем разрешить приложению попытаться выполнить операцию еще раз.[1]
* **Открытый.** Запрос приложения немедленно завершается со сбоем, и исключение возвращается в приложение.[1]
* **Полуоткрытый.** Ограниченному числу запросов от приложения разрешено проходить через операцию и вызывать ее. Если эти запросы выполняются успешно, предполагается, что ошибка, которая ранее вызывала сбой, устранена, а автоматический выключатель переходит в состояние «закрытый»(счетчик сбоев сбрасывается). Если какой-либо запрос завершается со сбоем, автоматическое выключение предполагает, что неисправность все еще присутствует, поэтому он возвращается в состояние «открытый» и перезапускает таймер времени ожидания, чтобы дать системе дополнительное время на восстановление после сбоя.[1]

1. Конструкторская часть
   1. Описание архитектуры системы

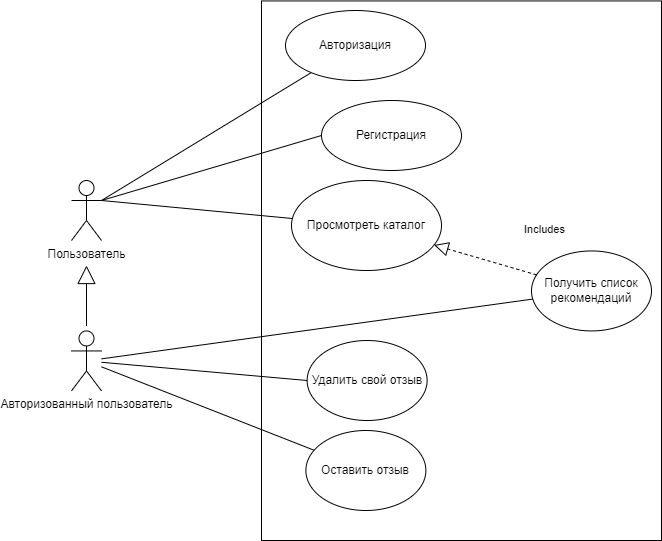
Серверная часть системы состоит из следующих микросервисов:

* Gateway – точка входа в систему, получающая все внешние запросы и взаимодействующая со всеми остальными микросервисами. Остальные микросервисы могут взаимодействовать только с gateway.
* Catalogue – сервис каталога. Отвечает за данные комиксов.
* Users – сервис контроля учетных записей. Отвечает за данные пользователей и их верификацию.
* Recommendations – сервис рекомендаций. Реализует непосредственно рекомендательную систему.
  1. Архитектура приложения.

Каждый микросервис имеет несколько слоев, в частности, handler (обработчики http-запросов), usecase (сценарии использования) и models (используемые модели данных). Обработчик принимает запрос и, с помощью usecase, формирует ответ. Usecase реализует логику обработки сценария использования. Он может взаимодействовать с репозиторием, если сервис обладает базой данных или прочими слоями сервиса.

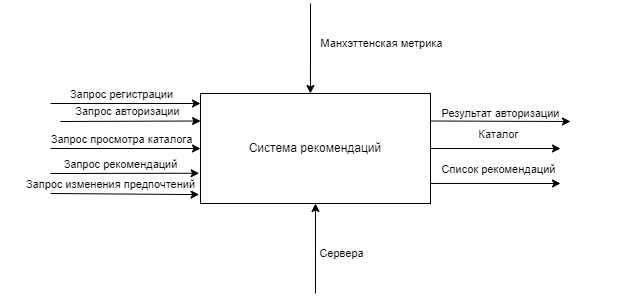
* 1. Сценарии использования

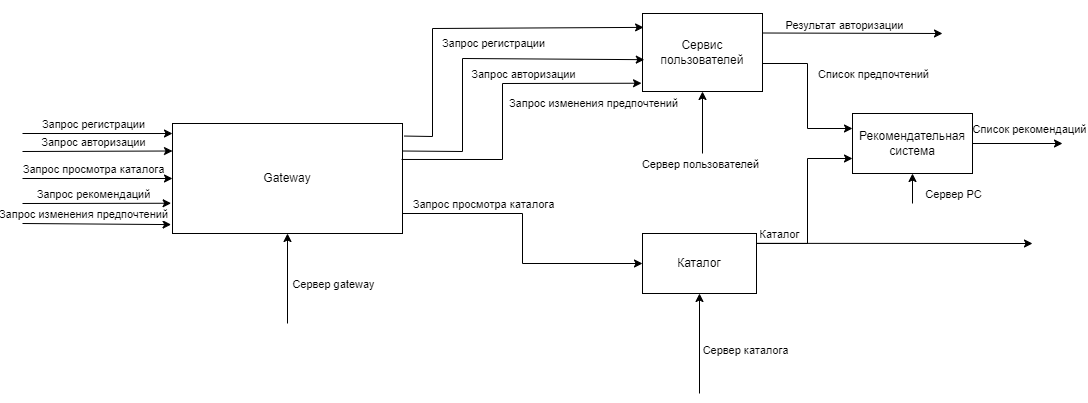
На рисунке 2.3.1 представлена диаграмма сценариев использования.

  
Рисунок 2.3.1 Usecase диаграмма.

* 1. IDEF0-диаграмма

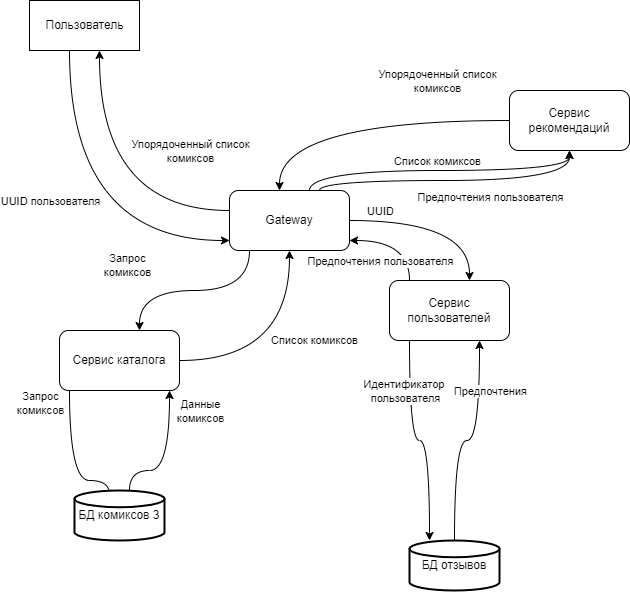
На рисунках 2.4.1-2.4.2 представлена IDEF0-диаграмма.

  
Рисунок 2.4.1. IDEF0, общий уровень

  
Рисунок 2.4.2. IDEF0, детализированный уровень.

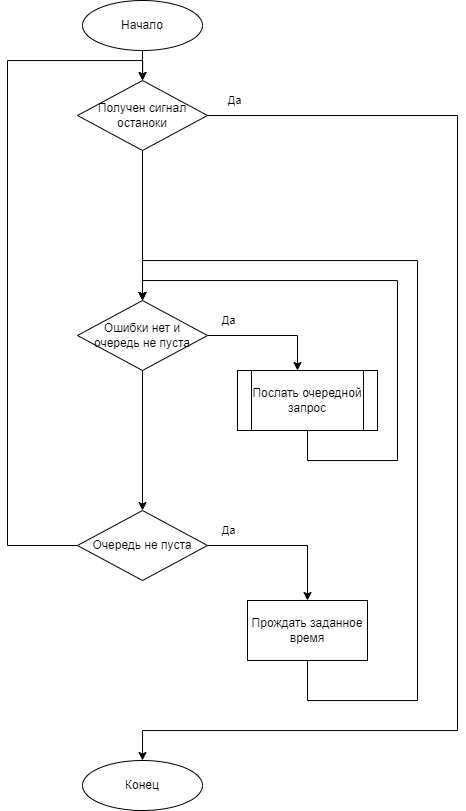
* 1. Диаграммы потоков данных

На рисунке 2.5.1 представлена диаграмма потоков данных получения рекомендаций.

  
Рисунок 2.5.1 Диаграмма потоков данных получения рекомендаций.

* 1. Схемы алгоритмов

На рисунке 2.6.1 представлена схема алгоритма обработки очереди запросов.

  
Рисунок 2.6.1. Обработка очереди запросов

1. Технологический раздел
   1. Выбор языка программирования.

В качестве языка программирования для разработки бэкенда был выбран Golang, так как он имеет встроенную библиотеку для обработки http-запросов, а также множество сторонних библиотек для прочей перфиерии бэкенда (например, для JWT).

Для реализации фронтенда был выбран язык TypeScript и фреймворк Angular, так как он имеет крупнейший функционал для верстки веб-приложений.

* 1. Структура приложения.

Помимо сервисов, приложение обладает внутренними пакетами:

* database;
* utility;
* config.

Пакет database содержит интерфейс доступа к данным в зависимости от СУБД и его реализацию для взаимодействия с PostgreSQL.

Пакет utility содержит функцию генерацию случайной строки (для JWT ключа), а также функции преобразования данных из последовательности байт в переменные определенных типов.

Пакет config содержит конфигурацию приложения.

Внутри сервиса Gateway реализован паттерн CircuitBreaker. Внутри Gateway также имеются пакеты gateway\_error для описания ошибок уровня gateway, connector для взаимодействия с остальными сервисами и request\_queue, содержащий интерфейс очереди запросов и его реализацию. Также в Gateway находится пакет middlewares, содержащий реализацию прокси проверки авторизации с помощью JWT.

* 1. Листинги кода

На листингах 1-3 представлена реализация рекомендательной системы.

Листинг 1. Вычисление расстояния манхэттенской метрикой.

func manhattanMetric(a attributes, b attributes) float64 {  
 dist := 0.0  
 for i := 0; i < len(a); i++ {  
 dist += math.Abs(a[i] - b[i])  
 }  
 return dist  
}  
  
func difference(id int, lib []attributes) []float64 {  
 n := len(lib)  
 res := make([]float64, n)  
 for i := 0; i < n; i++ {  
 res[i] = manhattanMetric(lib[id], lib[i])  
 }  
 return res  
}

Листинг 2. Формирование списка атрибутов.

func get\_attributes(lib []models.Book) []attributes {  
 tags := make([]string, 0)  
 maxyear := lib[0].Year  
 minyear := lib[0].Year  
 maxrating := lib[0].Year  
 minrating := lib[0].Year  
  
 for \_, book := range lib {  
 tags = append(tags, book.Tags...)  
 if book.Year > maxyear {  
 maxyear = book.Year  
 }  
 if book.Year < minyear {  
 minyear = book.Year  
 }  
 if book.Rating > maxrating {  
 maxrating = book.Rating  
 }  
 if book.Rating < minrating {  
 minrating = book.Rating  
 }  
 }  
  
 res := make([]attributes, 0)  
 for \_, book := range lib {  
 attr := make([]float64, 0)  
 attr = append(attr, float64(book.Year-minyear)/float64(maxyear))  
 attr = append(attr, float64(book.Rating-minrating)/float64(maxrating))  
 if book.Status {  
 attr = append(attr, 1)  
 } else {  
 attr = append(attr, 0)   
 }  
 for \_, tag := range tags {  
 var i int  
 for i = 0; i < len(book.Tags) && book.Tags[i] != tag; i++ {  
 if book.Tags[i] == tag {  
 attr = append(attr, 1)  
 } else {  
 attr = append(attr, 0)  
 }  
 }  
 }  
 res = append(res, attr)  
 }  
 return res  
}

Листинг 3. Получение списка рекомендаций.

func (rc \*RecommendationsUsecase) GetRecommendations(lib []models.Book, prefs \*models.PreferencesList) []models.Book {  
 n := len(lib)  
 like\_res := make([]float64, n)  
 unlike\_res := make([]float64, n)  
 len1 := 1  
 len2 := 1  
  
 booksAttr := get\_attributes(lib)  
 likes, dislikes := getPreferences(lib, prefs)  
  
 if len(prefs.Likes) != 0 {  
 for \_, id := range likes {  
 for i, dif := range difference(id, booksAttr) {  
 like\_res[i] += dif  
 }  
 }  
 len1 = len(prefs.Likes)  
 }  
 if len(prefs.Dislikes) != 0 {  
 for \_, id := range dislikes {  
 for i, dif := range difference(id, booksAttr) {  
 unlike\_res[i] += dif  
 }  
 }  
 len2 = len(prefs.Dislikes)  
 }  
  
 m := 0.0  
 for i := 0; i < n; i++ {  
 like\_res[i] /= float64(len1)  
 unlike\_res[i] /= float64(len2)  
 if unlike\_res[i] > m {  
 m = unlike\_res[i]  
 }  
 }  
  
 dists := make([]float64, n)  
  
 for i := 0; i < n; i++ {  
 dists[i] = like\_res[i] - unlike\_res[i] + m  
 }  
  
 sort.Slice(lib, func(i, j int) bool { return dists[lib[i].Id-1] > dists[lib[j].Id-1] })  
  
 return lib  
}

В листинге 4 приведен код проверки JWT авторизации.

Листинг 4. Проверка авторизации

func AuthMiddleware(next http.Handler) http.Handler {  
 return http.HandlerFunc(func(w http.ResponseWriter, r \*http.Request) {  
 ctx := r.Context()  
  
 data := strings.Split(r.Header.Get("Authorization"), " ")  
 if len(data) == 2 {  
 authToken := data[1]  
 algorithm := jwt.HmacSha256(config.JWTKey)  
  
 claims, err := algorithm.DecodeAndValidate(authToken)  
 if err == nil {  
 uuid, \_ := claims.Get("UUID")  
 if err == nil {  
 ctx = context.WithValue(ctx, "UUID", uuid)  
 }  
 username, \_ := claims.Get("Username")  
 if err == nil {  
 ctx = context.WithValue(ctx, "Username", username)  
 }  
 }  
 }  
 r = r.WithContext(ctx)  
 next.ServeHTTP(w, r)  
 })  
}

В листинге 5 приведен код обработчика запроса авторизации на сервисе Users

Листинг 5. Авторизация пользователя

func (uh \*UsersHandlers) LoginUser(w http.ResponseWriter, r \*http.Request) {  
 defer func(Body io.ReadCloser) {  
 err := Body.Close()  
 if err != nil {  
 fmt.Println("Failed to close response body")  
 }  
 }(r.Body)  
  
 userData := &models.UserAuthData{}  
 err := json.NewDecoder(r.Body).Decode(userData)  
 if err != nil {  
 writeError(w, "Bad input given to user creation", http.*StatusBadRequest*)  
 }  
  
 user, err := uh.uc.LoginUser(userData)  
  
 if err != nil {  
 fmt.Println("Failed to authenficate the user")  
 writeError(w, "Failed to authenficate the user", http.*StatusInternalServerError*)  
 return  
 }  
  
 if user == nil {  
 fmt.Println("User failed authenfication")  
 writeError(w, "Authenfication failed", http.*StatusBadRequest*)  
 return  
 }  
  
 algorithm := jwt.HmacSha256(config.JWTKey)  
 claims := jwt.NewClaim()  
 claims.Set("UUID", user.UUID)  
 claims.Set("Username", user.Username)  
  
 token, err := algorithm.Encode(claims)  
 if err != nil {  
 fmt.Println(err)  
 http.Error(w, `{"message":"cannot create jwt-token"}`, http.*StatusInternalServerError*)  
 return  
 }  
 user.JWT = token  
  
 w.Header().Set("Content-Type", "application/json")  
 err = json.NewEncoder(w).Encode(user)  
 if err != nil {  
 fmt.Println("Encoding json error: ", err)  
 http.Error(w, "Failed to encode data to json", http.*StatusInternalServerError*)  
 return  
 }  
}

Проверка логина и пароля выполняется в usecase функции LoginUser.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Circuit Breaker Pattern – Azure Architecture Center | Microsoft Learn [Электронный ресурс] режим доступа: https://learn.microsoft.com/ru-ru/azure/architecture/patterns/circuit-breaker