

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ
Московский Авиационный Институт
(Национальный исследовательский университет)

Институт №8
«Компьютерные науки и прикладная математика»
Кафедра 806
«Вычислительная математика и программирование»

Курсовой проект по дисциплине «Базы данных»
Тема: «Разработка базы данных, API и клиентского приложения для информационной
модели "Онлайн сервис бронирования водного инвентаря"»

Студент: Третьяков Н.М.

Группа: М8О-313Б-22

Преподаватель: Крылов С.С.

Оценка:

Дата:

Москва, 2024

Содержание

| | |
|---|----|
| Введение | 3 |
| Структура базы данных..... | 4 |
| Функции..... | 11 |
| Авторизация | 12 |
| Клиент-серверное взаимодействие | 15 |
| Взаимодействие базы данных и серверной части приложения | 16 |
| Клиент | 19 |
| Компоненты клиента | 20 |
| Вывод | 24 |
| Список использованных источников..... | 25 |
| Приложение..... | 26 |

Введение

В рамках курсового проекта разработано веб-приложение, включающее серверную часть на языке программирования Golang и клиентскую часть на основе библиотеки React. Программа предоставляет пользователю инструмент для выбора и бронирования водного инвентаря, а продавцу возможность добавлять и удалять товары, редактировать уже созданные, управлять версиями СУБД, наблюдать за бронированием товаров.

Основные возможности приложения:

- 1) каталог товаров с возможностью фильтрации по заданным параметрам;
- 2) бронирование выбранных товаров;
- 3) система отзывов, позволяющая пользователям оставлять комментарии о забронированных товарах;
- 4) личный кабинет, обеспечивающий доступ к управлению профилем и просмотром истории бронирований;
- 5) страница администратора, предоставляющая возможность управления приложением.

Реализованы задачи:

- 1) спроектирована структура базы данных;
- 2) разработаны модели предметной области, соответствующие функционалу приложения и выбранной предметной области;
- 3) организованы различные роли пользователей (администратор, пользователь) и права доступам к данным, присутствует функционал доступа пользователя по логину и паролю;
- 4) реализована возможность создания администратором архивных копий базы данных и восстановления данных из клиентского приложения;
- 5) логика обработки данных реализована на стороне серверного приложения;
- 6) на стороне базы данных определены полезные триггеры и функции;
- 7) реализована SQL скрипт для инициализации структуры базы данных.

Структура базы данных

Для хранения и управлениеми данными используется PostgreSQL. База данных состоит из нескольких связных таблиц. Почти все из них представляют сущности предметной области.

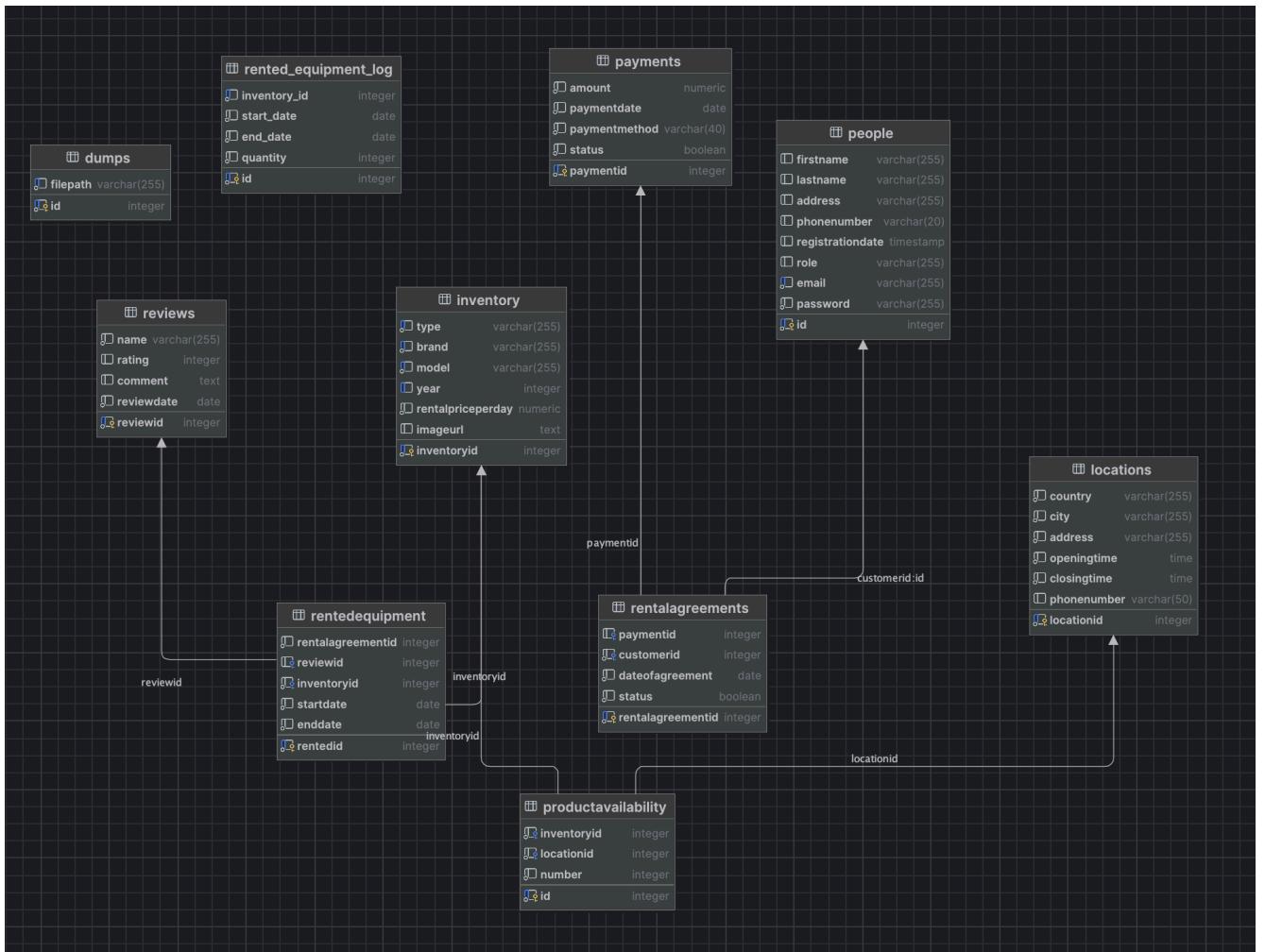


Рисунок 1. Связи в базе данных

Таблица `people` отвечает за хранение информации о пользователях, зарегистрированных в системе. Описание сущности представлено в таблице 1.

Таблица 1. Описание сущности user

| Поле | Комментарий | Тип | Ключ |
|------------------|----------------------------|--------------|------|
| id | идентификатор пользователя | serial | PK |
| firstname | имя пользователя | varchar(255) | |
| lastname | фамилия | varchar(255) | |
| address | адрес | varchar(255) | |
| phononenumber | номер телефона | varchar(20) | |
| registrationdate | дата регистрации | timestamp | |
| role | роль | varchar(255) | |

| | | | |
|----------|---------------------|--------------|--|
| email | почта | varchar(255) | |
| password | хешированный пароль | varchar(255) | |

Таблица locations хранит информацию о пункте аренды.

Таблица 2. Описание сущности inventory

| Поле | Комментарий | Тип | Ключ |
|-------------------|-----------------------|--|------|
| inventoryid | идентификатор товара | serial | PK |
| type | тип | varchar(255) | |
| brand | бренд | varchar(255) | |
| model | модель | varchar(255) | |
| year | год | int check (year >= 1990 and year <= extract(year from current_date)) | |
| rentalpriceperday | цена за день | decimal | |
| imageurl | ссылка на изображение | text | |

Таблица inventory хранит информацию о товарах.

Таблица 3. Описание сущности location

| Поле | Комментарий | Тип | Ключ |
|--------------|-----------------------|--------------|------|
| locationid | идентификатор локации | serial | PK |
| country | страна | varchar(255) | |
| city | город | varchar(255) | |
| address | адрес | varchar(255) | |
| openingtime | время открытия | varchar(20) | |
| closingtime | время закрытия | timestamp | |
| phonenumbers | номер телефона | varchar(255) | |

Таблица reviews содержит отзывы о товарах.

Таблица 4. Описание сущности review

| Поле | Комментарий | Тип | Ключ |
|------------|----------------------|---|------|
| reviewid | идентификатор отзыва | serial | PK |
| name | имя пользователя | varchar(255) | |
| rating | рейтинг | int check (rating >= 1 and rating <= 5) | |
| comment | комментарий | text | |
| reviewdate | дата публикации | date | |

Таблица payments хранит информацию о платежах.

Таблица 5. Описание сущности payment

| Поле | Комментарий | Тип | Ключ |
|---------------|-----------------------|-------------|------|
| paymentid | идентификатор платежа | serial | PK |
| amount | сумма платежа | decimal | |
| paymentdate | дата платежа | date | |
| paymentmethod | метод платежа | varchar(40) | |
| status | статус | boolean | |

Таблица rentalAgreements хранит информацию о заключенных договорах.

Таблица 6. Описание сущности rentalAgreement

| Поле | Комментарий | Тип | Ключ |
|-------------------|----------------------------|--------|---|
| rentalagreementid | идентификатор договора | serial | PK |
| paymentid | идентификатор платежа | int | FK ссылочная таблица: payments ссылочный столбец: paymentid |
| customerid | идентификатор пользователя | int | FK ссылочная таблица: people ссылочный столбец: id |

| | | | |
|-----------------|--------------------------|---------|--|
| dateofagreement | дата заключения договора | date | |
| status | статус | boolean | |

Таблица rentedEquipment хранит информацию об арендованных товарах.

Таблица 7. Описание сущности rentedEquipment

| Поле | Комментарий | Тип | Ключ |
|-------------------|------------------------------------|--------|---|
| rentedid | идентификатор арендованного товара | serial | PK |
| rentalagreementid | Идентификатор договора об аренде | int | FK ссылочная таблица: rentalAgreements ссылочный столбец: rentalagreementid |
| reviewid | идентификатор отзыва | int | FK ссылочная таблица: reviews ссылочный столбец: reviewid |
| inventoryid | идентификатор товара | int | FK ссылочная таблица: inventory ссылочный столбец: inventoryid |
| startdate | дата начала аренды | date | |
| enddate | дата окончания аренды | date | |

Таблица productAvailability хранит информацию о доступных на локациях товарах.

Таблица 8. Описание сущности productAvailability

| Поле | Комментарий | Тип | Ключ |
|-------------|---|--------|---|
| id | идентификатор товара | serial | PK |
| inventoryid | идентификатор товара | int | FK ссылочная таблица: inventory ссылочный столбец: inventoryid |
| locationid | идентификатор локации | int | FK ссылочная таблица: locations ссылочный столбец: locationid |
| number | количество доступного товара на локации | int | |

Таблица dumps хранит информацию о файлах для восстановления.

Таблица 9. Описание сущности dump

| Поле | Комментарий | Тип | Ключ |
|----------|---------------------------------|--------------|------|
| id | идентификатор копии | serial | PK |
| filepath | путь к файлу с резервной копией | varchar(255) | |

Таблица rented_equipment_log хранит информацию о всех заказанных товарах. Информация добавляется с помощью триггера.

Таблица 10. Описание сущности rented_equipment_log

| Поле | Комментарий | Тип | Ключ |
|-----------|--------------------|--------|------|
| id | идентификатор лога | serial | PK |
| startdate | дата начала аренды | date | |

| | | | |
|-------------|-----------------------|------|---|
| enddate | дата окончания аренды | date | |
| inventoryid | идентификатор товара | int | FK ссылочная таблица: inventory ссылочный столбец: inventoryid |

Взаимосвязи между таблицами базы данных обеспечивают структурированное управление информацией и формируют логически взаимосвязанную систему.

Между таблицами rentalagreements и people существует отношение «многие к одному». Каждый договор аренды ассоциирован с одним пользователем, выступающим в роли клиента. При этом один пользователь может заключить множество договоров аренды, что позволяет учитывать все его заказы.

Связь между rentalagreements и payments также организована как отношение «многие к одному». Для каждого договора аренды может быть зарегистрирован один платеж, однако каждый платеж строго привязан к единственному договору, что исключает дублирование.

Между таблицами rentedequipment и rentalagreements реализовано отношение «многие к одному». Каждая запись об аренде оборудования относится к одному договору, но один договор может включать несколько единиц арендованного оборудования.

Связь между rentedequipment и inventory отражает отношение «многие к одному»: каждая запись об аренде указывает на конкретную единицу из каталога оборудования. При этом одна и та же единица оборудования может быть арендована в рамках разных записей.

Таблицы rentedequipment и reviews соединены отношением «многие к одному». Каждая запись об аренде может быть связана с одним отзывом, что даёт пользователям возможность оставлять обратную связь о товарах, которые они арендовали.

Связь между productavailability и inventory представляет собой отношение «многие к одному». Каждая запись о доступности товара в конкретной локации ссылается на одну единицу из каталога инвентаря, фиксируя её наличие в определённом месте.

Между таблицами productavailability и locations установлено отношение «многие к одному». Каждая запись о доступности привязана к одному местоположению. В то же время, связь между locations и productavailability образует отношение «один ко многим», поскольку каждое местоположение может содержать записи о доступности различных товаров.

Функции

Хранимая функция — это набор инструкций SQL, которые выполняют некоторую операцию и возвращают единственное значение.

Листинг 1. Пример функции для обновления товара

```
CREATE OR REPLACE FUNCTION update_item(
    p_id INT,
    p_type VARCHAR(255),
    p_brand VARCHAR(255),
    p_model VARCHAR(255),
    p_year INT,
    p_price_per_day NUMERIC,
    p_image VARCHAR(255)
) RETURNS VOID AS
$$
BEGIN
    UPDATE inventory
    SET Type          = COALESCE(p_type, Type),
        Brand         = COALESCE(p_brand, Brand),
        Model         = COALESCE(p_model, Model),
        Year          = COALESCE(p_year, Year),
        RentalPricePerDay = COALESCE(p_price_per_day, RentalPricePerDay),
        ImageURL      = COALESCE(p_image, ImageURL)
    WHERE InventoryID = p_id;
END;
$$ LANGUAGE plpgsql;
```

Триггерные функции — это специальные функции, которые выполняются автоматически при срабатывании триггера. Они используются для выполнения действий до или после конкретного события в таблице или представлении.

Листинг 2. Триггер для логирования

```
CREATE OR REPLACE FUNCTION Log_rented_equipment()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO rented_equipment_Log (inventory_id, start_date, end_date)
    VALUES (NEW.InventoryID, NEW.StartDate, NEW.EndDate);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER after_insert_rented_equipment
AFTER INSERT ON rentedEquipment
FOR EACH ROW
EXECUTE FUNCTION Log_rented_equipment();
```

Авторизация

В приложении реализована авторизация с использованием JWT (JSON Web Token), что обеспечивает безопасный доступ пользователей к ресурсам. Для работы с токенами используется библиотека github.com/golang-jwt/jwt/v4. Токены содержат ключевую информацию о пользователе, такую как идентификатор и срок действия, и шифруются с использованием секретного ключа, заданного в файле окружения.

Механизм проверки токенов встроен в серверную часть в виде промежуточного слоя (middleware). Этот слой обрабатывает каждый входящий HTTP-запрос и выполняет следующие действия:

- 1) извлечение и проверка секретного ключа: секретный ключ берётся из переменных окружения. Если ключ отсутствует, сервер возвращает ошибку конфигурации, поскольку без него невозможно проверить токен;
- 2) анализ заголовка Authorization: токен извлекается из заголовка запроса, где он должен быть передан в формате Bearer <токен>. Если заголовок отсутствует или формат некорректен, доступ отклоняется;
- 3) верификация токена: токен проверяется на подлинность с использованием секретного ключа. Если токен недействителен, запрос блокируется, а клиент получает сообщение об отсутствии авторизации;
- 4) извлечение данных из токена: после успешной проверки из токена извлекается идентификатор пользователя. Эта информация добавляется в контекст запроса, чтобы использовать её на последующих этапах обработки;
- 5) передача запроса дальше: если токен прошёл проверку, запрос передаётся следующему обработчику с добавленной информацией о пользователе.

Для дополнительной безопасности пароли пользователей в базе данных хранятся в зашифрованном виде. Во время регистрации новый пароль хэшируется, а введённый при входе — сравнивается с сохранённым хэшем. Это обеспечивает защиту данных пользователей и минимизирует риски в случае утечки базы данных.

Листинг 3. Реализация Jwt middleware

```
func JwtAuthMiddleware() func(http.Handler) http.Handler {
    return func(next http.Handler) http.Handler {
        return http.HandlerFunc(func(w http.ResponseWriter, r
*http.Request) {
            secret := os.Getenv("ACCESS_TOKEN_SECRET")
            if secret == "" {
                http.Error(w, "error",
http.StatusInternalServerError)
                return
            }
            authHeader := r.Header.Get("Authorization")
            t := strings.Split(authHeader, " ")

```

```

        if len(t) == 2 {
            authToken := t[1]
            authorized, err := IsAuthorized(authToken, secret)
            if authorized {
                userID, err := ExtractIDFromToken(authToken,
secret)
                if err != nil {
                    http.Error(w,
errors.JsonError(err.Error()), http.StatusUnauthorized)
                    return
                }
                ctx := contextWithValue(r.Context(), "user-id",
userID)
                next.ServeHTTP(w, r.WithContext(ctx))
                return
            }
            http.Error(w, errors.JsonError(err.Error()),
http.StatusUnauthorized)
            return
        }
        http.Error(w, errors.JsonError("Not authorized"),
http.StatusUnauthorized)
    })
}
}

```

Листинг 4. Проверка на наличие авторизации

```

func IsAuthorized(requestToken string, secret string) (bool, error) {
    _, err := jwt.Parse(requestToken, func(token *jwt.Token)
(interface{}, error) {
    if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
        return nil, fmt.Errorf("Unexpected signing method: %v",
token.Header["alg"])
    }
    return []byte(secret), nil
})
    if err != nil {
        return false, err
    }
    return true, nil
}

```

Листинг 5. Извлечение ID пользователя из токена

```

func ExtractIDFromToken(requestToken string, secret string) (string, error)
{
    token, err := jwt.Parse(requestToken, func(token *jwt.Token)
(interface{}, error) {
    if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
        return nil, fmt.Errorf("Unexpected signing method: %v",
token.Header["alg"])
    }
    return []byte(secret), nil
})
}

```

```
if err != nil {
    return "", err
}
claims, ok := token.Claims.(jwt.MapClaims)
if !ok || !token.Valid {
    return "", fmt.Errorf("Invalid Token")
}
id, ok := claims["id"].(string)
if !ok {
    return "", fmt.Errorf("ID not found in token")
}
return id, nil
}
```

Авторизация для всех пользователей, включая клиентов и продавцов, реализована одинаково — они входят в систему с использованием электронной почты и пароля. Однако для доступа к административным функциям предусмотрена дополнительная проверка через middleware (`AdminMiddleware`). Этот промежуточный слой анализирует данные пользователя из токена и определяет, обладает ли он правами администратора. Это позволяет разграничивать доступ к различным частям приложения в зависимости от роли пользователя, без необходимости использовать разные системы входа.

Клиент-серверное взаимодействие

Взаимодействие между сервером и клиентом в приложении организовано через REST API. Сервер обрабатывает запросы клиента и возвращает соответствующие ответы.

Реализованы такие типы запросов, как GET, POST и DELETE. GET-запрос отправляет клиенту данные, POST-запрос обрабатывает данные, переданные с клиента, DELETE – удаляет данные из таблиц. И, наоборот, клиент запрашивает и отправляет данные на сервер.

И клиент, и сервер используют асинхронные функции для взаимодействия друг с другом.

Листинг 6. Пример запросов на сервере

```
router.Route("/profile", func(r chi.Router) {
    r.Use(accessToken.JwtAuthMiddleware())
    r.Post("/update", h.UpdateUser)
    r.Get("/user", h.UserInfo)
})
```

Взаимодействие базы данных и серверной части приложения

На серверной части приложения используется паттерн «репозиторий». Представляет собой инструмент, который существенно упрощает работу с данными в программных приложениях. Его основное предназначение заключается в абстракции логики доступа к данным, что позволяет отделить взаимодействие с хранилищем данных от бизнес-логики. Такая структура делает код более читаемым, поддерживаемым и масштабируемым.

Одним из ключевых преимуществ данного подхода является возможность скрытия деталей реализации взаимодействия с базой данных или другими источниками данных за единым интерфейсом. Это позволяет разработчику сосредоточиться на бизнес-логике приложения, не углубляясь в технические аспекты извлечения или сохранения данных.[3]

Паттерн способствует устраниению дублирования кода за счет централизованного определения операций над данными. Это не только улучшает читаемость программы, но и упрощает внесение изменений: обновления в логике работы с данными требуют модификации только в одном месте — в репозитории.

Листинг 7. Слой Service

```
type Service struct {
    Authorization
    Agreement
    Equipment
    Payment
    Profile
    Review
    Location
    User
    Dump
}
func NewServices(repositories *repository.Repository) *Service {
    return &Service{
        Authorization: NewAuthService(repositories.Authorization),
        Equipment:     NewEquipmentService(repositories.Equipment),
        Agreement:     NewAgreementService(repositories.Agreement),
        Payment:       NewPaymentService(repositories.Payment),
        Profile:       NewProfileService(repositories.Profile),
        Review:        NewReviewService(repositories.Review),
        Location:      NewLocationService(repositories.Location),
        User:          NewUserService(repositories.User),
        Dump:          NewDumpService(repositories.Dump),
    }
}
```

Листинг 8. Слой UserService

```
type UserService struct {
    repo repository.User
}

func (u UserService) Create(c context.Context, user *entity.User) error {
    ctx, cancel := context.WithTimeout(c, time.Second*10)
    defer cancel()
    return u.repo.Create(ctx, user)
}
```

Листинг 9. Слой Repository

```
type Repository struct {
    Authorization
    Agreement
    Equipment
    Payment
    Profile
    Review
    Location
    User
    Dump
}

func NewRepository(db *sql.DB) *Repository {
    return &Repository{
        Authorization: NewAuthPostgres(db),
        Agreement:     NewAgreementPostgres(db),
        Equipment:    NewEquipmentPostgres(db),
        Payment:       NewPaymentPostgres(db),
        Profile:      NewProfilePostgres(db),
        Review:        NewReviewPostgres(db),
        Location:     NewLocationPostgres(db),
        User:          NewUserRepository(db),
        Dump:          NewDumpPostgres(db),
    }
}
```

Листинг 10. Слой UserRepository

```
type UserRepository struct {
    db *sql.DB
}

func (u UserRepository) Create(c context.Context, user *entity.User) error {
    tx, err := u.db.BeginTx(c, nil)
    if err != nil {
        return err
    }
    query := "SELECT create_user($1, $2, $3)"
    row := tx.QueryRowContext(c, query, user.Email, user.Password,
    user.RegistrationDate)
    if err := row.Err(); err != nil {
        tx.Rollback()
    }
}
```

```
        return err
    }
    if err := row.Scan(&user.Id); err != nil {
        tx.Rollback()
        return err
    }
    return tx.Commit()
}
```

Транзакция в программировании — это последовательность операций над данными, которая выполняется как единое целое. Если хотя бы одна из операций внутри транзакции не выполняется успешно, все изменения, произведенные до этого момента, отменяются. Это гарантирует целостность данных даже в случае возникновения ошибок.

Основные свойства транзакций описываются концепцией ACID:

- 1) atomicity (атомарность): транзакция выполняется либо полностью, либо не выполняется вовсе;
- 2) consistency (согласованность): транзакция переводит систему из одного согласованного состояния в другое;
- 3) isolation (изоляция): результат выполнения транзакции не зависит от параллельно выполняющихся транзакций;
- 4) durability (надежность): результаты завершенной транзакции сохраняются даже в случае сбоя системы.

Клиент

Клиент представляет собой приложение, созданное на основе React, с использованием React Router для маршрутизации.

Клиент поддерживает следующие маршруты:

- 1) главная страница;
- 2) страница регистрации/авторизации;
- 3) страница с контактной информацией;
- 4) страница с товарами;
- 5) страница товара;
- 6) профиль пользователя;
- 7) страница администратора;
- 8) управление товарами;
- 9) управление локациями;
- 10) управление пользователями;
- 11) управление восстановлением базы данных и логированием;
- 12) управление отзывами.

Листинг 10. Маршрутизация

```
import React from "react";
import { Routes, Route } from "react-router";
import Header from "./components/Header";
import Footer from "./components/Footer";
import HomePage from "@/pages/HomePage";
import ProductsPage from "@/pages/ProductsPage";
import ProductDetailsPage from "@/components/ProductDetailsPage.tsx";
import UserProfile from "@/components/UserProfile.tsx";
import Auth from "@/components/Auth.tsx";
import AdminPage from "@/components/AdminPage.tsx";
import ItemsPage from "@/components/ManageItems.tsx";
import LocationsPage from "@/components/ManageLocations.tsx";
import UsersPage from "@/components/ManageUsers.tsx";
import ReviewsPage from "@/components/ManageReviews.tsx";
import "./index.css"
import DumpsPage from "@/components/ManageDumps.tsx";
import Contacts from "@/components/Contacts.tsx";

const App: React.FC = () => {
  return (
    <React.Fragment>
      <Header />
      <main>
        <Routes>
          <Route path="/" element={<HomePage />} />
          <Route path="products" element={<ProductsPage />} />{" "}
          {/* Фиксированный маршрут */}
          <Route path="products/:id" element={<ProductDetailsPage />} />
        </Routes>
      </main>
    </React.Fragment>
  );
}

export default App;
```

```

<Route path="profile" element={<UserProfile />} />
<Route path="auth" element={<Auth />} />
<Route path="admin" element={<AdminPage />} />
<Route path="/" element={<h1>404 - Page Not Found</h1>} />
<Route path="admin/items" element={<ItemsPage />} />
<Route path="admin/Locations" element={<LocationsPage />} />
<Route path="admin/users" element={<UsersPage />} />
<Route path="admin/reviews" element={<ReviewsPage />} />
<Route path="admin/dumps" element={<DumpsPage />} />
<Route path="contacts" element={<Contacts/>} />
</Routes>
</main>
<Footer />
</React.Fragment>
);
};

export default App;

```

Компоненты клиента

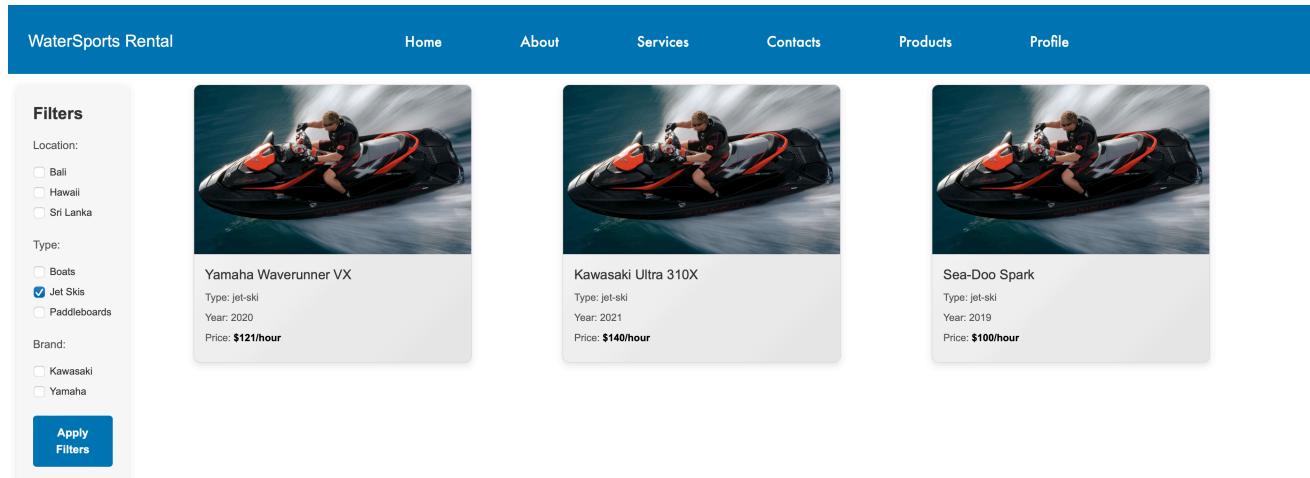


Рисунок 2. Страница с товарами



Sea-Doo Spark

Type: jet-ski
Year: 2019
Price: \$100/hour
Available at:

- 456 Beach Blvd, Miami (1 available)
- 123 Surf St, LA (1 available)

[Book Product](#)

Reviews:
 No reviews yet.
[Leave a Review](#)

Your Review (max 400 characters)

Rating (1-5) ▼

[Submit](#)

Рисунок 3. Страница товара

[Book Product](#)

Location:
 456 Beach Blvd, Miami ▼

Start Date:
 2024/12/31

End Date:
 2024/12/28

Payment Method:
 Card

| December 2024 | | | | | | |
|---------------|----|----|----|----|----|----|
| Su | Mo | Tu | We | Th | Fr | Sa |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| 29 | 30 | 31 | 1 | 2 | 3 | 4 |

Reviews:
 No reviews yet.
[Leave a Review](#)

Рисунок 4. Бронирование товара

Store Locations

| Country | City | Address | Opening Time | Closing Time | Phone Number |
|-----------|-------------|-----------------------------|--------------|--------------|---------------|
| USA | Los Angeles | 123 Surf St, LA | 08:00:00 | 18:00:00 | 123-456-11231 |
| USA | Miami | 456 Beach Blvd, Miami | 09:00:00 | 19:00:00 | 987-654-3210 |
| Australia | Sydney | 789 Ocean Dr, Sydney | 07:00:00 | 17:00:00 | 555-0123 |
| Australia | Cairns | 258 Resort Rd, Cairns | 08:30:00 | 20:00:00 | 444-0231 |
| Australia | Gold Coast | 369 Surf Avenue, Gold Coast | 09:00:00 | 18:00:00 | 777-4567 |
| Korea | Los Angeles | 123 Surf St, LA | 08:00:00 | 18:00:00 | 123-456-7890 |

Рисунок 5. Пункты аренды

User Profile

First Name: dfdfgfdgfdh

Last Name: Grechin

Address: Mirny

Phone Number: 89331234431

Email: 133@qmail.com

[Edit Profile](#)[Logout](#)

Order History



Рисунок 6. Профиль пользователя

Manage Locations

[Hide Locations](#)[Create New Location](#)[AdminPage](#)

| | | | | |
|---|---|---|--|--|
| USA, Los Angeles ID: 1 Address: 123 Surf St, LA Opening Time: 08:00:00 Closing Time: 18:00:00 Phone: 123-456-11231 | USA, Miami ID: 2 Address: 456 Beach Blvd, Miami Opening Time: 09:00:00 Closing Time: 19:00:00 Phone: 987-654-3210 | Australia, Sydney ID: 3 Address: 789 Ocean Dr, Sydney Opening Time: 07:00:00 Closing Time: 17:00:00 Phone: 555-0123 | Australia, Cairns ID: 4 Address: 258 Resort Rd, Cairns Opening Time: 08:30:00 Closing Time: 20:00:00 Phone: 444-0231 | Australia, Gold Coast ID: 5 Address: 369 Surf Avenue, Gold Coast Opening Time: 09:00:00 Closing Time: 18:00:00 Phone: 777-4567 |
| Korea, Los Angeles ID: 6 Address: 123 Surf St, LA Opening Time: 08:00:00 Closing Time: 18:00:00 Phone: 123-456-7890 | | | | |

Рисунок 7. Управление локациями

| Manage Users | | | | |
|---|-------------------|--|-------------------|--|
| Hide Users | | AdminPage | | |
| ID: 15 Email: gff51441@google.com Address: Phone: Registration Date: 21.12.2024 Role: Change Role: User | X | ID: 18 Email: gff514461@google.com Address: Phone: Registration Date: 21.12.2024 Role: Change Role: User | X | ID: 19 Email: gff5614461@google.com Address: Phone: Registration Date: 21.12.2024 Role: Change Role: User |
| ID: 20 Email: gf1231f@.com Address: Phone: Registration Date: 21.12.2024 Role: Change Role: User | X | ID: 21 Email: gf1231f@google.com Address: Phone: Registration Date: 21.12.2024 Role: Change Role: User | X | ID: 22 Email: gf51231f@google.com Address: Phone: Registration Date: 21.12.2024 Role: Change Role: User |
| ID: 22 Email: gf51231f@google.com Address: Phone: Registration Date: 21.12.2024 Role: Change Role: User | X | ID: 24 Email: gf51hh2hh361f@google.com Address: Phone: Registration Date: 21.12.2024 Role: Change Role: User | X | ID: 27 Email: gf51hh2hh361f@google.com Address: Phone: Registration Date: 21.12.2024 Role: Change Role: User |
| ID: 28 Email: 1@gmail.com Address: Phone: Registration Date: 21.12.2024 Role: Change Role: User | X | ID: 29 Email: 1@.com Address: Phone: Registration Date: 21.12.2024 Role: Change Role: User | X | ID: 30 Email: 1@gmail.com Address: Phone: Registration Date: 21.12.2024 Role: Change Role: User |

Рисунок 8. Управление пользователями

| Manage Dumps | | | | |
|---|---------|-----------------------------|---------------------------|---------------------------|
| Hide Dumps | | Create Dump | AdminPage | Hide Logs |
| Filename | | | | |
| internal/dumps/backup_2024-12-28_05-32-11 | | | | |
| internal/dumps/backup_2024-12-28_05-34-43 | | | | |
| internal/dumps/backup_2024-12-28_06-02-26 | | | | |
| internal/dumps/backup_2024-12-28_07-15-16 | | | | |
| internal/dumps/backup_2024-12-28_09-52-15 | | | | |
| ID | Item ID | Start Date | End Date | |
| 1 | 8 | 2025-01-01T00:00:00Z | 2025-01-04T00:00:00Z | |
| 2 | 8 | 2024-12-11T00:00:00Z | 2024-12-18T00:00:00Z | |
| 3 | 3 | 2024-12-28T00:00:00Z | 2024-12-29T00:00:00Z | |

Рисунок 9. Восстановление базы данных и просмотр логирования

Register

| | |
|--------------------------|--------------------------|
| Email | <input type="text"/> |
| Password | <input type="password"/> |
| First Name | <input type="text"/> |
| Last Name | <input type="text"/> |
| Address | <input type="text"/> |
| Phone Number | <input type="text"/> |
| Register | |

[Already have an account? Sign In](#)

Рисунок 10. Форма регистрации

Вывод

В ходе разработки курсового проекта реализовано приложение для онлайн-сервиса аренды водного инвентаря с серверной частью на Golang и клиентской частью на React. Приложение позволяет пользователям просматривать каталог товаров, бронировать товары управлять профилем, оставлять отзывы, а администраторам — работать с резервными копиями данных, добавлять, редактировать товары и локации, удалять отзывы и пользователей, выдавать им роли.

Список использованных источников

- 1) Web-стандарты и спецификации для работы с REST API. REST API Tutorial. [Электронный ресурс]. URL: <https://restfulapi.net/> (дата обращения: 09.12.2024).
- 2) Документация по языку программирования Go. The Go Programming Language. [Электронный ресурс]. URL: <https://golang.org/doc/> (дата обращения: 12.12.2024).
- 3) Паттерны проектирования в разработке на Go. Design Patterns in Go. [Электронный ресурс]. URL: <https://refactoring.guru/design-patterns/go> (дата обращения: 13.12.2024).
- 4) Донован А., Керниган Б. Язык программирования Go. — М.: Вильямс, 2016. — 400 с.

Приложение

Репозиторий на GitHub: <https://github.com/JedX4rl/WaterSportsRental.git>