Оглавление

Описание проекта

Нам предстоит сделать веб приложение по отображению владельцев и принадлежавших им автомобилей. реализация требуется с разделением на back и front.

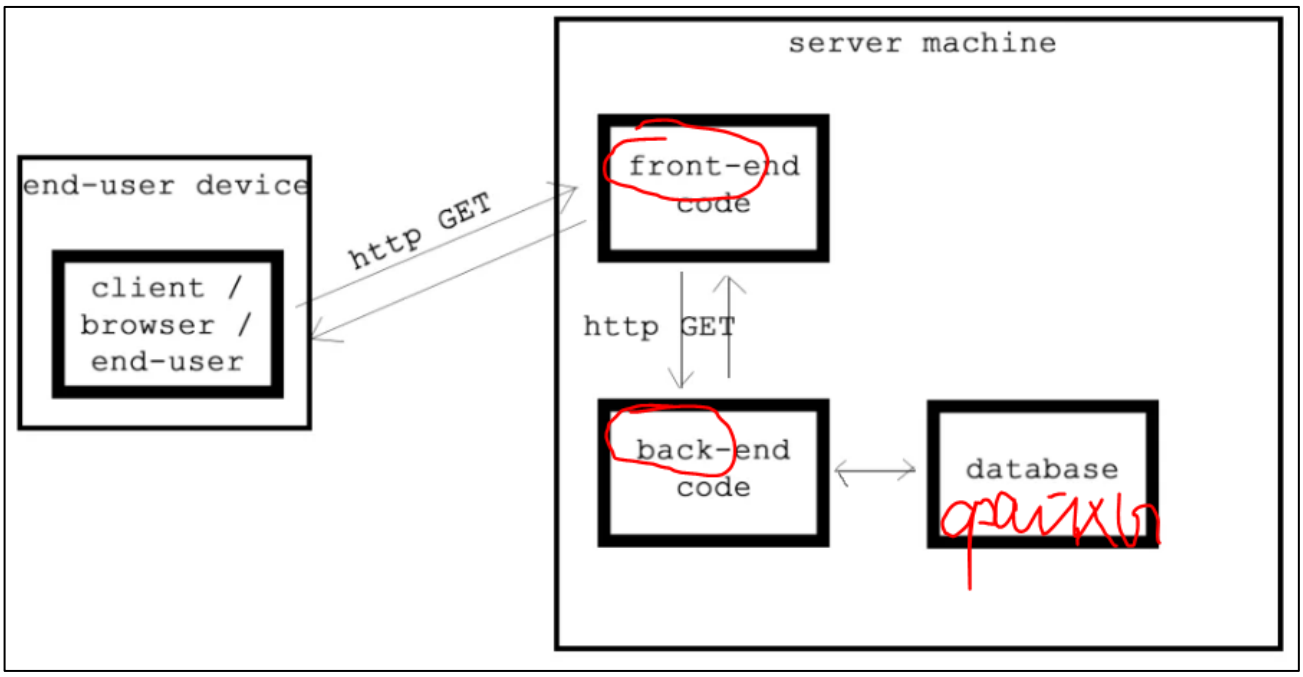

Рисунок 1. Принцип разделения кода на клиентскую часть и серверную.

Сделаем back на node.js Для программирования серверной части можно использовать почти любой язык, а вот для клиентской части практически только один язык – javascript. Node.js использует язык javascript в своей основе, поэтому нам будет удобнее, ведь достаточно уметь пользоваться одним языком программирования.

напиши в разных файлах классы User (id, фамилия, имя, телефон, почта, город) Auto (id, марка, модель, цвет, госномер), Owner(id_user, id_auto, дата регистрации, дата снятия с учета).

создай 2-3 записи в константах кода, напиши код для сохранения экземпляров в текстовый файл csv и json

1. csv – файл текстовый, для хранения структурированных данных (csv (читается «цээсвэ») **C**omma **S**eparated **V**alue – значения разделенные запятыми)

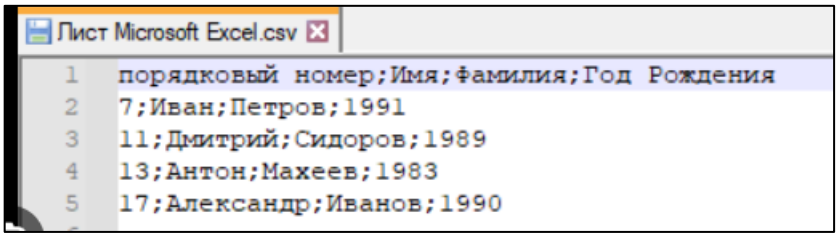

Рисунок 2. Пример данных в файле CSV

2. json – файл текстовый, для представления и передачи сложно структурированных данных. (читается «джейсон» (java script object notation) – Нотация, способ представления, описания данных для javascript

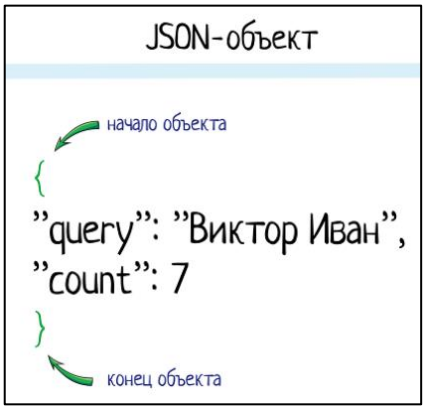

Рисунок 3. Пример содержимого JSON файла

Уровень 1. Реализация Backend (серверной части)

Структура проекта

Убедитесь, что у вас работает node.js

Создайте папку проекта, папки для модулей и файл index.js

```
mygibdd/
├── models/
|   ├── User.js
|   ├── Auto.js
|   ├── Owner.js
├── data/
|   ├── users.csv
|   ├── autos.csv
|   ├── owners.csv
|   ├── users.json
|   ├── autos.json
|   ├── owners.json
├── index.js
```

1. Класс User (файл models/User.js)

Реализуйте класс User, обратите внимание на номера строк

```
class User {
  constructor(id, lastName, firstName, phone, email, city) {
    this.id = id;
    this.lastName = lastName;
    this.firstName = firstName;
    this.phone = phone;
    this.email = email;
    this.city = city;
  }

  toCSV() {
    return `${this.id},${this.lastName},${this.firstName},${this.phone},${this.email},${this.city}`;
  }

  toJSON() {
    return {
      id: this.id,
      lastName: this.lastName,
      firstName: this.firstName,
      phone: this.phone,
      email: this.email,
      city: this.city,
    };
  }
}

module.exports = User;
```

2. Класс Auto (файл models/Auto.js)

```
class Auto {
  constructor(id, brand, model, color, licensePlate) {
    this.id = id;
    this.brand = brand;
    this.model = model;
    this.color = color;
    this.licensePlate = licensePlate;
  }

  toCSV() {
    return `${this.id},${this.brand},${this.model},${this.color},${this.licensePlate}`;
  }

  toJSON() {
    return {
      id: this.id,
      brand: this.brand,
      model: this.model,
      color: this.color,
      licensePlate: this.licensePlate,
    };
  }
}

module.exports = Auto;
```

3. Класс Owner (файл models/Owner.js)

Класс «Владелец_автомобиля»

```
class Owner {
    constructor(userId, autoId, registrationDate, deregistrationDate) {
        this.userId = userId;
        this.autoId = autoId;
        this.registrationDate = registrationDate;
        this.deregistrationDate = deregistrationDate;
    }
```

Опишите методы для вывода в строку для записи в файл самостоятельно

4. Основной файл index.js

Здесь создадим экземпляры классов, сохраним их в CSV и JSON файлы.

Подключим модули:

- fs (Файловая система)
- path (хранение пути к каталогу/файлу)

Подключим файлы наших моделей как модули проекта

```
const fs = require('fs');
const path = require('path');
const User = require('./models/User');
const Auto = require('./models/Auto');
const Owner = require('./models/Owner');

```

Для тестирования нашего проекта создадим фейковые данные, используйте константы для хранения:

```
const users = [
    new User(1, 'Иванов', 'Иван', '+79161234567', 'ivanov@example.com', 'Москва'),
    new User(2, 'Петров', 'Петр', '+79167654321', 'petrov@example.com', 'Санкт-Петербург')
];

const autos = [
    new Auto(1, 'Toyota', 'Corolla', 'Синий', 'А123ВС77'),
    new Auto(2, 'Lada', 'Vesta', 'Белый', 'О456ТУ78'),
];

const owners = [
    new Owner(1, 1, '2020-01-01', '2023-01-01'),
    new Owner(2, 2, '2021-05-15', null),
];
```

Напишем функцию для записи в файл CSV

```
// Функция для сохранения данных в CSV
function saveToCSV(data, fileName) {
    const header = Object.keys(data[0].toJSON()).join(',');
    const rows = data.map(item => item.toCSV()).join('\n');
    const content = `${header}\n${rows}`;

    fs.writeFileSync(path.join(__dirname, 'data', fileName), content, 'utf-8');
    console.log(`Данные сохранены в ${fileName}`);
}
```

Функция записи в файл JSON выглядит чуть иначе

```
// Функция для сохранения данных в JSON
function saveToJSON(data, fileName) {
    const content = JSON.stringify(data.map(item => item.toJSON()), null, 2);

    fs.writeFileSync(path.join(__dirname, 'data', fileName), content, 'utf-8');
    console.log(`Данные сохранены в ${fileName}`);
}

```

Здесь:

JSON.stringify() – функция построения JSON строки

map() – функция применения некоторой функции к КАЖДОМУ элементу массива. Функция map создает новый массив, с уже измененными данными.

Для самопроверки и закрепления новых знаний – используйте вывод в Console

5. Сохранение данных из проекта в файлы.

Впишите в код вызов функций для записи данных в файлы

```
// Сохраняем данные
saveToCSV(users, 'users.csv');
saveToCSV(autos, 'autos.csv');
saveToCSV(owners, 'owners.csv');

saveToJSON(users, 'users.json');
saveToJSON(autos, 'autos.json');
saveToJSON(owners, 'owners.json');
```

6. Проверка работы серверной части:

Вы должны перейти в папку с файлом **index.js**

Запустите проект командой в терминале VS Code или из командной строки:

```
bash
node index.js
```

Пример содержимого файлов:

users.csv

```
id,lastName,firstName,phone,email,city
1,Иванов,Иван,+79161234567,ivanov@example.com,Москва
2,Петров,Петр,+79167654321,petrov@example.com,Санкт-Петербург
```

users.json

```
[
  {
    "id": 1,
    "lastName": "Иванов",
    "firstName": "Иван",
    "phone": "+79161234567",
    "email": "ivanov@example.com",
    "city": "Москва"
  },
  {
    "id": 2,
    "lastName": "Петров",
    "firstName": "Петр",
    "phone": "+79167654321",
    "email": "petrov@example.com",
    "city": "Санкт-Петербург"
  }
]
```

Теперь у нас есть backend-часть, которая создает данные и сохраняет их в файлы. В следующем шаге можно добавить frontend для отображения этих данных.

Уровень 2. Создадим Frontend для нашего серверного модуля

Сделаем приложение в виде **Single Page Application (SPA)**. Всё отображение и изменение будет происходить на одной странице, без принудительного обновления страницы.

Для создания SPA с разделами "**Список владельцев**", "**Список авто**" и "**Регистрация авто**", создадим отдельную папку для **frontend**-части. В этой папке будет HTML, CSS и JavaScript файлы.

7. Структура frontend-проекта:

Создайте папку и файлы по схеме, реализуя независимую архитектуру нашей системы. Back – самостоятельная часть, и Front отдельно. Таким образом, эти части проекта могут делать разные разработчики. И связывать проект через систему контроля версий, например, github

```
mygibdd/
├── frontend/
|   ├── index.html
|   ├── styles.css
|   ├── script.js
├── backend/ (ваша предыдущая backend-часть)
```

8. HTML (файл frontend/index.html)

Сделайте структуру разметки. Подключите файл скрипта.

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>МояГИБДД</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <h1>МояГИБДД</h1>
    <div class="container">
        <!-- Список владельцев -->
        <section>⟺</section>

        <!-- Список авто -->
        <section>⟺</section>

        <!-- Регистрация авто -->
        <section>⟺</section>
    </div>

    <script src="script.js"></script>
</body>
</html>
```