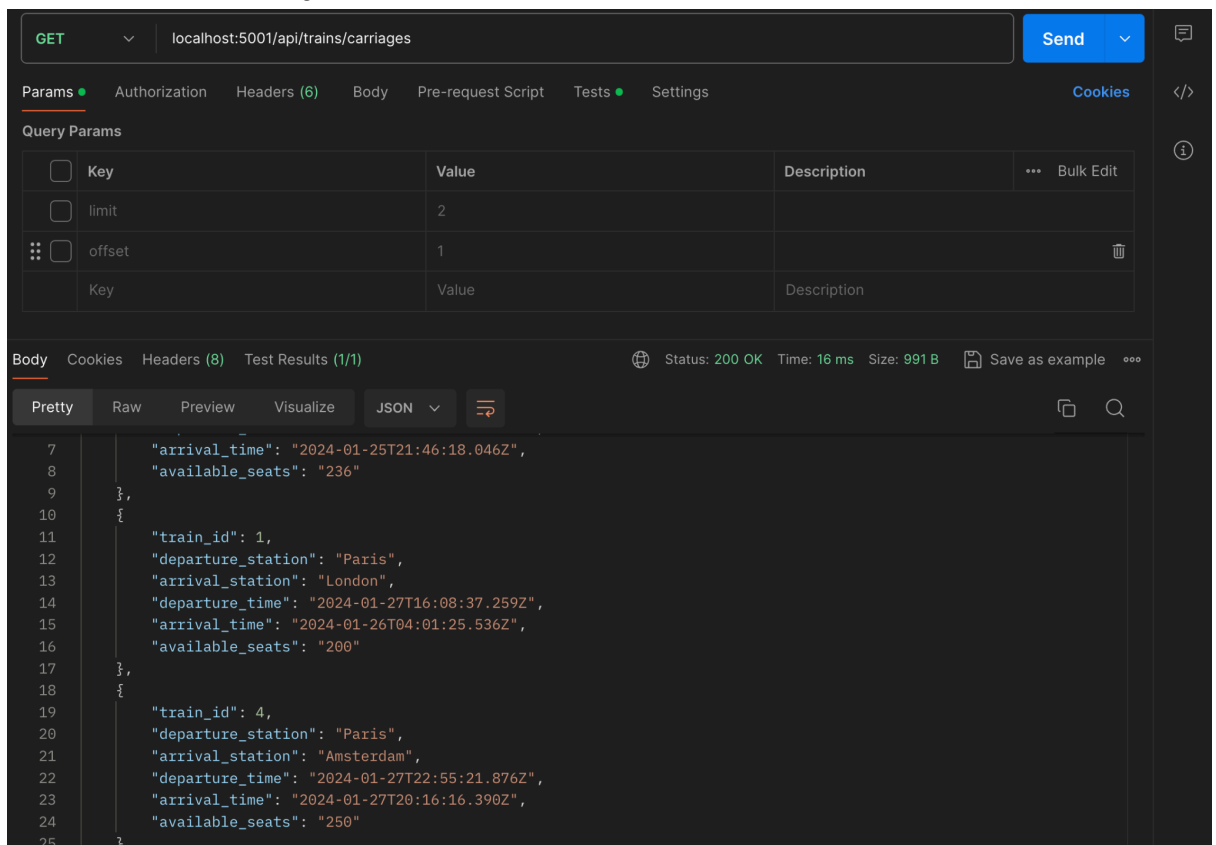


How to run the project.

- Install all project dependencies with command ***npm i***.
- Run db migration and seed with two commands: ***npx knex migrate:latest*** and ***npx knex seed:run***
- Start Rest and Soap services with: ***nodemon rest/index.js*** and ***nodemon soap/index.js***.
- Resting will be listening at 5001 and Soap at 8000.

Project Evaluation.

- Create REST Train Filtering service B was done, self evaluate: 6



- Create SOAP Train Booking Service B was done, self evaluate: 4

POST http://localhost:8000/wsd/ Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL XML Beautify

```

4   <<soapenv:Header/>
5   <<soapenv:Body>
6     <<web:BookTrainMessageSplitterRequest>
7       <<web:trainId>5</web:trainId>
8       <<web:className>B</web:className>
9       <<web:userId>3</web:userId>
10      <<web:seatNumber>50</web:seatNumber>
11      <<web:token>eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTESIm1hdCI6MTcwNjI2ODczMSwiZXhwIjoxNzA0ODYwNzMxZmF0.tZ6Zpz1KUvix96jpjYlPzJ4jgrzbZaI4Dd5hUdIVK4</web:token>
12    </web:BookTrainMessageSplitterRequest>
13  </soapenv:Body>
14 </soapenv:Envelope>

```

Body Cookies Headers (6) Test Results Status: 200 OK Time: 65 ms Size: 561 B Save as example

Pretty Raw Preview Visualize XML

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tns="http://tempuri.org/"
   xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/">
3    <soap:Body>
4      <BookTrainMessageSplitterResponse xmlns="http://tempuri.org/">
5        <reservationStatus>true</reservationStatus>
6      </BookTrainMessageSplitterResponse>
7    </soap:Body>
8  </soap:Envelope>

```

- Interaction between two services was done, self evaluate 4:

```

1  async function handleFilterRequest(args) {
2    let trainFilterUrl = 'http://localhost:5001/api/trains/carriages';
3    let queryParams = {
4      departureStation: args.departureStation !== null ? args.departureStation : undefined,
5      arrivalStation: args.arrivalStation !== null ? args.arrivalStation : undefined,
6      departureTime: args.departureTime !== null ? args.departureTime : undefined,
7      arrivalTime: args.arrivalTime !== null ? args.arrivalTime : undefined,
8      carriageClass: args.carriageClass !== null ? args.carriageClass : undefined,
9      limit: args.limit !== null ? args.limit : undefined,
10     offset: args.offset !== null ? args.offset : undefined,
11   };
12
13   queryParams = Object.fromEntries(
14     Object.entries(queryParams).filter(([key, value]) => value !== undefined)
15   );
16
17   if (Object.values(queryParams).length) {
18     trainFilterUrl = `${trainFilterUrl}?${Object.entries(queryParams).map(([key, value]) => `${key}=${value}`).join('&')}`;
19   }
20   try {
21     const trainResponse = await axios.get(trainFilterUrl);
22     if (trainResponse.data.length === 0) return NO_TRAIN_AVAILABLE_MSG;
23     return trainResponse.data;
24   } catch (error) {
25     console.log('filter train error in soap service!');
26     console.log(error);
27   }
28 }

```

- Test with Web Service Client was done, self evaluate: 1

```
1 import soap from 'soap';
2 var url = 'http://localhost:8000/wsdl?wsdl';
3
4 // Create client
5 soap.createClient(url, function (err, client) {
6   if (err) {
7     throw err;
8   }
9   /*
10    * Parameters of the service call: they need to be called as specified
11    * in the WSDL file
12    */
13   var args = {
14     message: 'Paris',
15     arrivalStation: 'London'
16   };
17   // call the service
18   client.TrainMessage(args, function (err, res) {
19     if (err)
20       throw err;
21     // print the service returned result
22     console.log(res);
23   });
24 });
```

- Working with complex data type: implemented with classes, functions, self evaluate: 2

```
1 class UserModel extends Model {
2   static tableName = 'user';
3   static async create(data) {
4     return super.insert({
5       ...data,
6       ...defaultUserRole
7     });
8   }
9   static async findByEmail(email) {
10     return this.table.where('email', '=', email);
11   }
12   static async findUserById(id) {
13     const [user] = await this.table.where('id', '=', id).returning(['name', 'email', 'id', 'role']);
14     return user;
15   }
16 }
17
18
19 export default UserModel;
```

```

1  export const getAllTrainCarriageAvailability = asyncHandler(async (req, res) => {
2    try {
3      const queryParams = req.query;
4      const trainAvailabilities = await TrainCarriageModel.getAllTrainAvailabilityByFilter(queryParams);
5      res.json(trainAvailabilities);
6    } catch (error) {
7      console.error(error);
8      res.status(500).json({
9        message: 'Internal Server Error!'
10     });
11   }
12 });

```

- Working with databases: postgres and used a db query builder tool for migration, seed and prevent sql injections, self evaluate:2

```

1  const environments = ['development', 'staging', 'production'];
2
3  const connection = {
4    host: '127.0.0.1',
5    port: process.env.DB_PORT,
6    database: process.env.DB_NAME,
7    user: process.env.DB_USER,
8    password: process.env.DB_PASSWORD
9  };
10
11 const commonConfig = {
12   client: 'pg',
13   connection,
14   pool: {
15     min: 2,
16     max: 10,
17   },
18   migrations: {
19     tableName: 'knex_migrations',
20     directory: './database/migrations'
21   },
22   seeds: {
23     directory: './database/seeds'
24   }
25 };
26
27 export default Object.fromEntries(environments.map((env) => [env, commonConfig]));
28

```



```
1  const tableName = 'train';
2
3  const trainsArr = [{
4    trainName: faker.string.alpha({ length: 4, casing: 'upper' }),
5    departureStation: 'Paris',
6    arrivalStation: 'London',
7    departureTime: faker.date.soon({ refDate: Date.now() }),
8    arrivalTime: faker.date.between({ from: Date.now(), to: '2024-01-26T00:00:00.000Z' })
9  },
10 {
11   trainName: faker.string.alpha({ length: 4, casing: 'upper' }),
12   departureStation: 'Paris',
13   arrivalStation: 'London',
14   departureTime: faker.date.soon({ days: 2 }),
15   arrivalTime: faker.date.soon({ days: 3 }),
16 },
17 {
18   trainName: faker.string.alpha({ length: 4, casing: 'upper' }),
19   departureStation: 'Paris',
20   arrivalStation: 'Milan',
21   departureTime: faker.date.soon({ refDate: Date.now() }),
22   arrivalTime: faker.date.between({ from: Date.now(), to: '2024-01-30T00:00:00.000Z' })
23 },
24 },
25 {
26   trainName: faker.string.alpha({ length: 4, casing: 'upper' }),
27   departureStation: 'Paris',
28   arrivalStation: 'Berlin',
29   departureTime: faker.date.soon({ days: 5 }),
30   arrivalTime: faker.date.soon({ days: 10 }),
31 },
32 {
33   trainName: faker.string.alpha({ length: 4, casing: 'upper' }),
34   departureStation: 'Paris',
35   arrivalStation: 'Amsterdam',
36   departureTime: faker.date.soon({ days: 7 }),
37   arrivalTime: faker.date.soon({ days: 14 }),
38 }
39 ];
40
41 export async function seed(knex) {
42   // Deletes ALL existing entries
43   await knex(tableName).del();
44   const trains = trainsArr
45     .map((train, index) => ({
46       id: index,
47       train_name: train.trainName,
48       departure_station: train.departureStation,
49       arrival_station: train.arrivalStation,
50       departure_time: train.departureTime,
51       arrival_time: train.arrivalTime
52     }));
53
54   await knex(tableName).insert(trains.map(train => ({ ...train })));
55 };
56
```

```

1 let query = this.table.select(selectedColumns)
2   .join('train_carriage_availability', 'train.id', '=', 'train_carriage_availability.train_id')
3   .join('carriage_class', 'carriage_class.id', '=', 'train_carriage_availability.carriage_class_id')
4   .sum('carriage_class.seating_capacity as available_seats')
5   .groupBy(selectedColumns)
6   .limit(limit)
7   .offset(offset);

```

- Tested with Postman was implemented, self evaluate:2

The screenshot displays the Postman interface for a GET request to `localhost:5001/api/users`. The **Tests** tab is active, showing three test scripts:

```

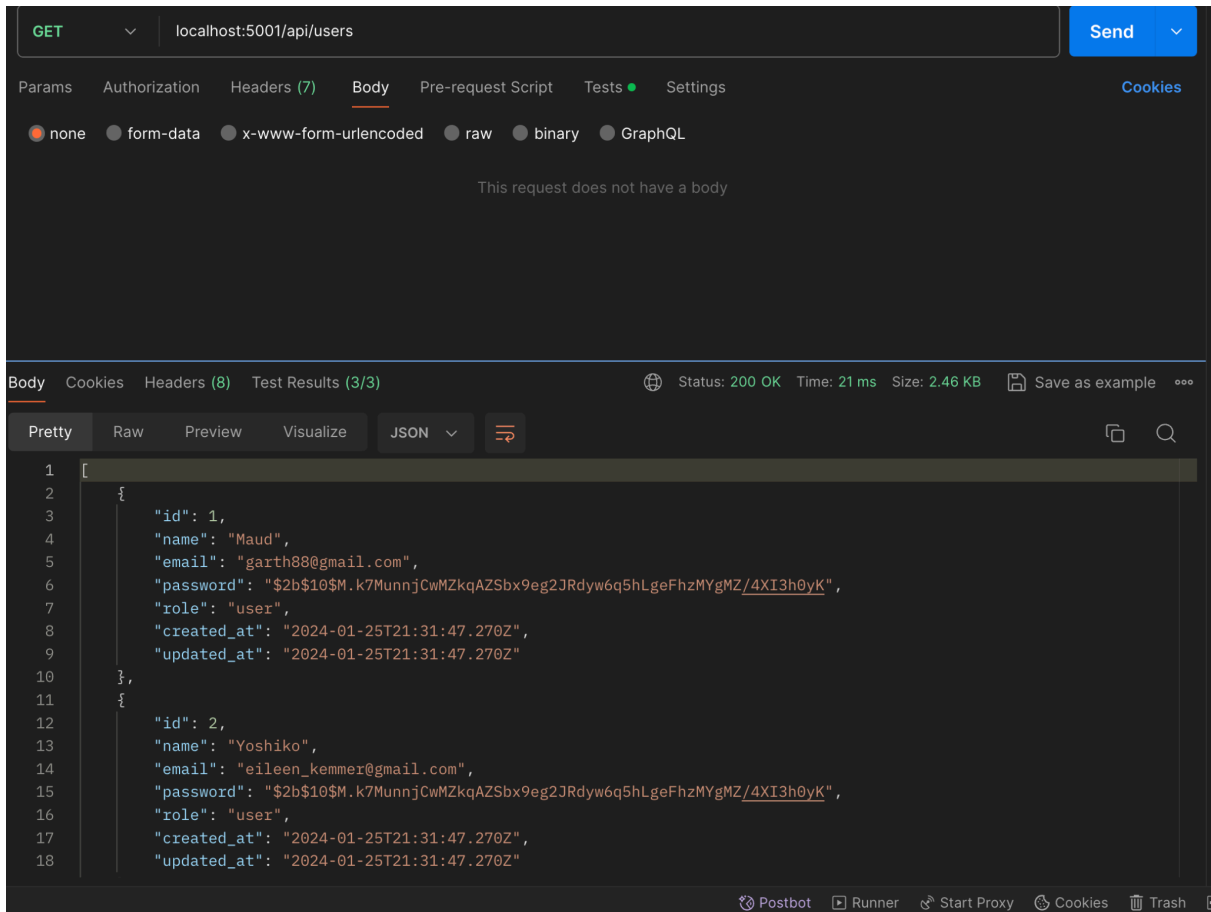
1 pm.test("Response status code is 200", function () {
2   pm.expect(pm.response.code).to.equal(200);
3 });
4
5
6
7 pm.test("Email is in a valid format", function () {
8   const responseData = pm.response.json();
9
10  responseData.forEach(function(user) {
11    pm.expect(user.email).to.match(/^[w-\.] +@([w-]+\.[w-]{2,4}$)/);
12  });
13 });
14
15
16 pm.test("Password should not be empty", function () {
17   const responseData = pm.response.json();
18   responseData.forEach(function(user) {

```

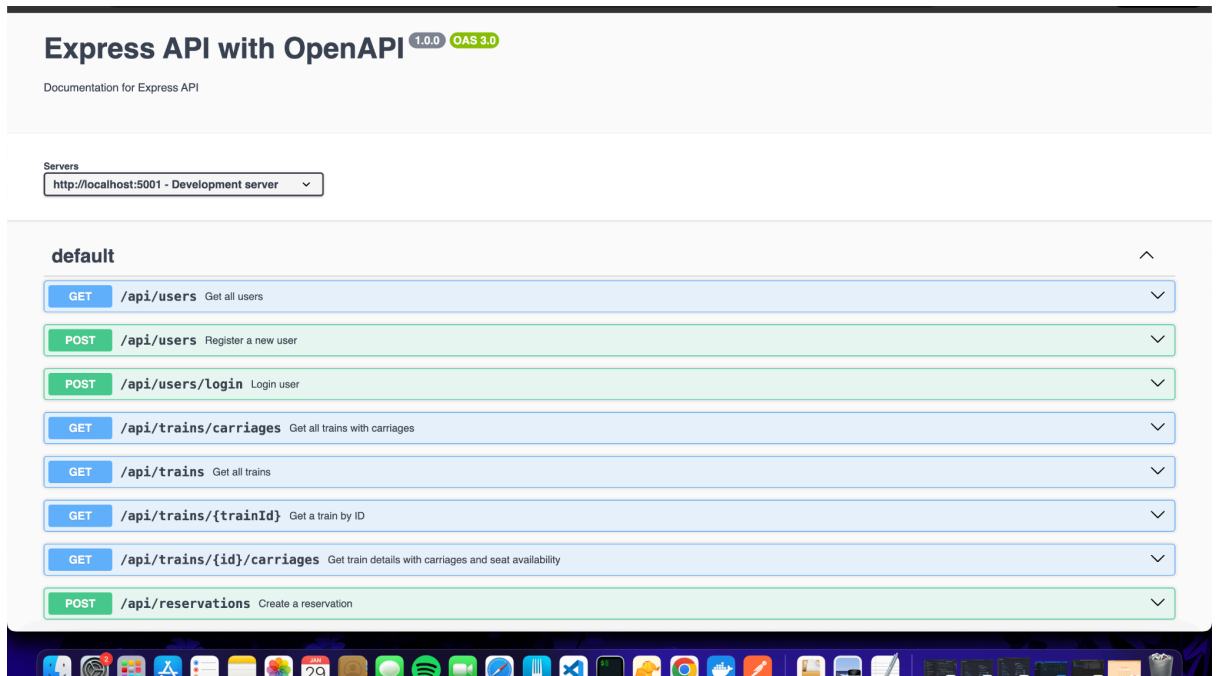
The **Test Results (3/3)** section at the bottom shows that all three tests passed:

- PASS** Response status code is 200
- PASS** Email is in a valid format

Additional details visible in the interface include the status `200 OK`, time `21 ms`, size `2.46 KB`, and a notification banner about API test templates.



- Document API with Open API was done, self evaluate: 2



- BPMS was partially implemented and to be continued, self evaluate: 3

