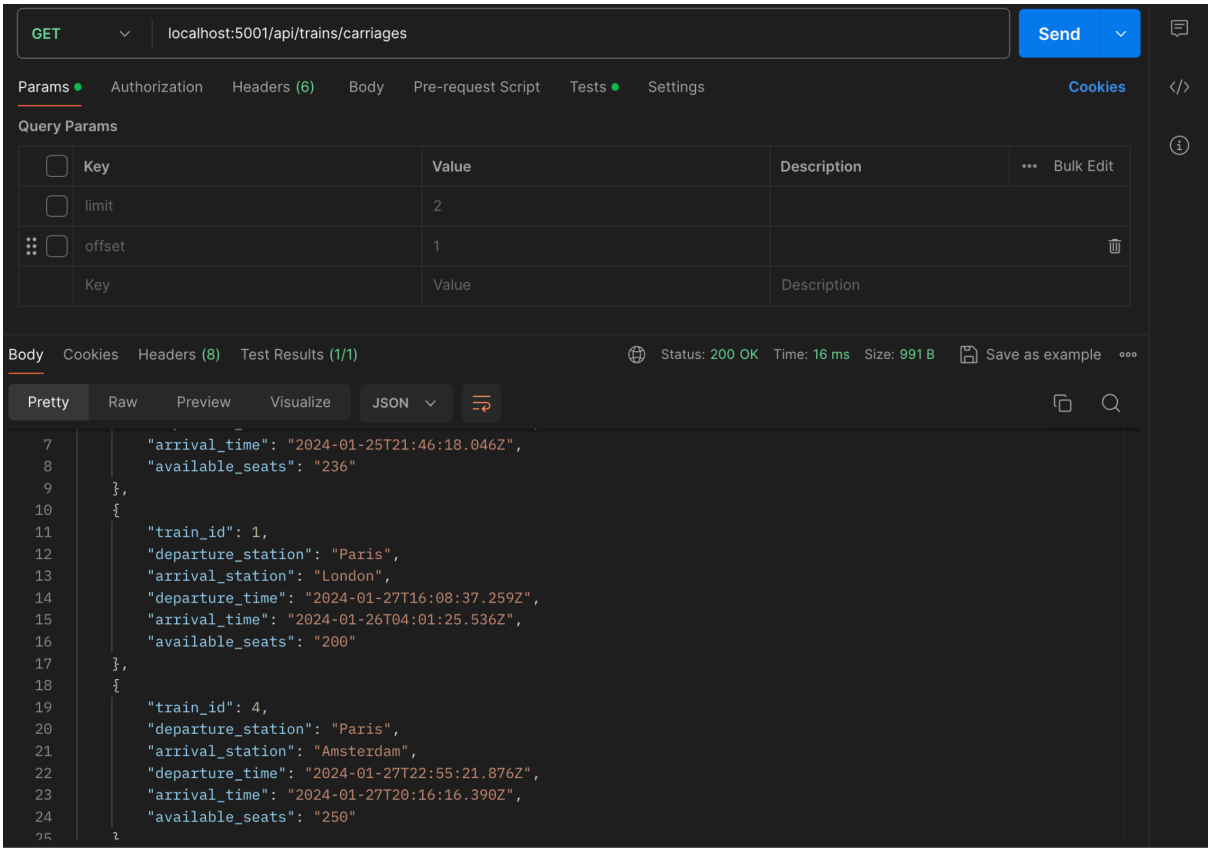
Service Oriented Computing	Cong Quang Nguyen Pyae Sone Kyaw

How to run the project.

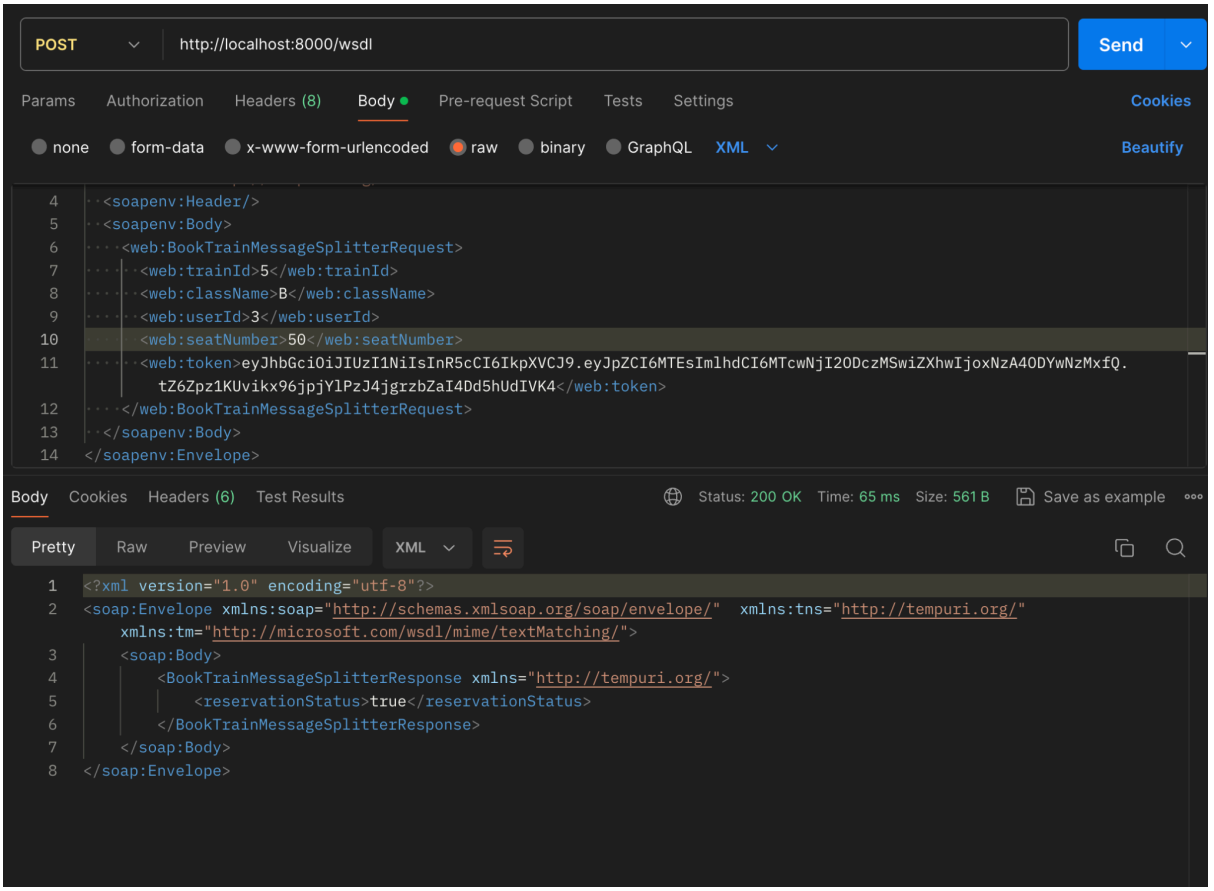
- Install all project dependencies with command ***npm i.***
- Run db migration and seed with two commands: ***npx knex migrate:latest*** and ***npx knex seed:run***
- Start Rest and Soap services with: ***nodemon rest/index.js*** and ***nodemon soap/index.js***.
- Resting will be listening at 5001 and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



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

```
async function handleFilterRequest(args) {
    let trainFilterUrl = `http://localhost:5001/api/trains/carriages`;
    let queryParams = {
        departureStation: args.departureStation !== null ? args.departureStation : undefined,
        arrivalStation: args.arrivalStation !== null ? args.arrivalStation : undefined,
        departureTime: args.departureTime !== null ? args.departureTime : undefined,
        arrivalTime: args.arrivalTime !== null ? args.arrivalTime : undefined,
        carriageClass: args.carriageClass !== null ? args.carriageClass : undefined,
        limit: args.limit !== null ? args.limit : undefined,
        offset: args.offset !== null ? args.offset : undefined,
    };

    queryParams = Object.fromEntries(
        Object.entries(queryParams).filter(([key, value]) => value !== undefined)
    );

    if (Object.values(queryParams).length) {
        trainFilterUrl = `${trainFilterUrl}?${Object.entries(queryParams).map(([key, value]) => `${key}=${value}`).join('&')}`;
    }
    try {
        const trainResponse = await axios.get(trainFilterUrl);
        if (trainResponse.data.length === 0) return NO_TRAIN_AVAILABLE_MSG;
        return trainResponse.data;
    } catch (error) {
        console.log(`filter train error in soap service!`);
        console.log(error);
    }
}
```

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

```
import soap from 'soap';
var url = 'http://localhost:8000/wsdl?wsdl';

// Create client
soap.createClient(url, function (err, client) {
    if (err) {
        throw err;
    }
    /*
    * Parameters of the service call: they need to be called as specified
    * in the WSDL file
    */
    var args = {
        message: `Paris`,
        arrivalStation: `London`
    };
    // call the service
    client.TrainMessage(args, function (err, res) {
        if (err)
            throw err;
        // print the service returned result
        console.log(res);
    });
});
```

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

```
class UserModel extends Model {
    static tableName = 'user';
    static async create(data) {
        return super.insert({
            ...data,
            ...defaultUserRole
        });
    }
    static async findByEmail(email) {
        return this.table.where('email', '=', email);
    }

    static async findUserById(id) {
        const [user] = await this.table.where('id', '=', id).returning(['name', 'email', 'id', 'role']);
        return user;
    }
}

export default UserModel;
```

```
export const getAllTrainCarriageAvailability = asyncHandler(async (req, res) => {
    try {
        const queryParams = req.query;
        const trainAvailabilities = await TrainCarriageModel.getAllTrainAvailabilityByFilter(queryParams);
        res.json(trainAvailabilities);
    } catch (error) {
        console.error(error);
        res.status(500).json({
            message: `Internal Server Error!`
        });
    }
});
```

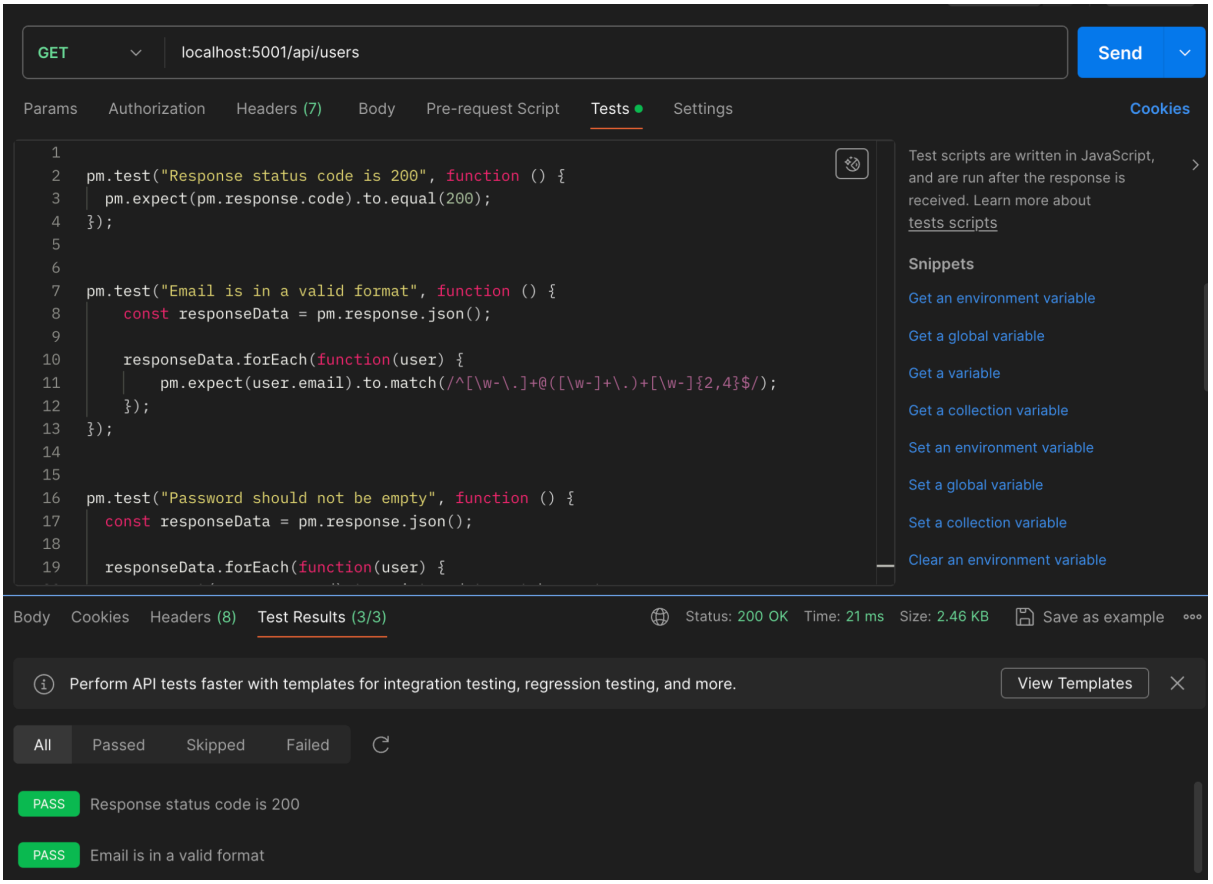
- Working with databases: postgres and used a db query builder tool for migration,seed and prevent sql injections, self evaluate:2

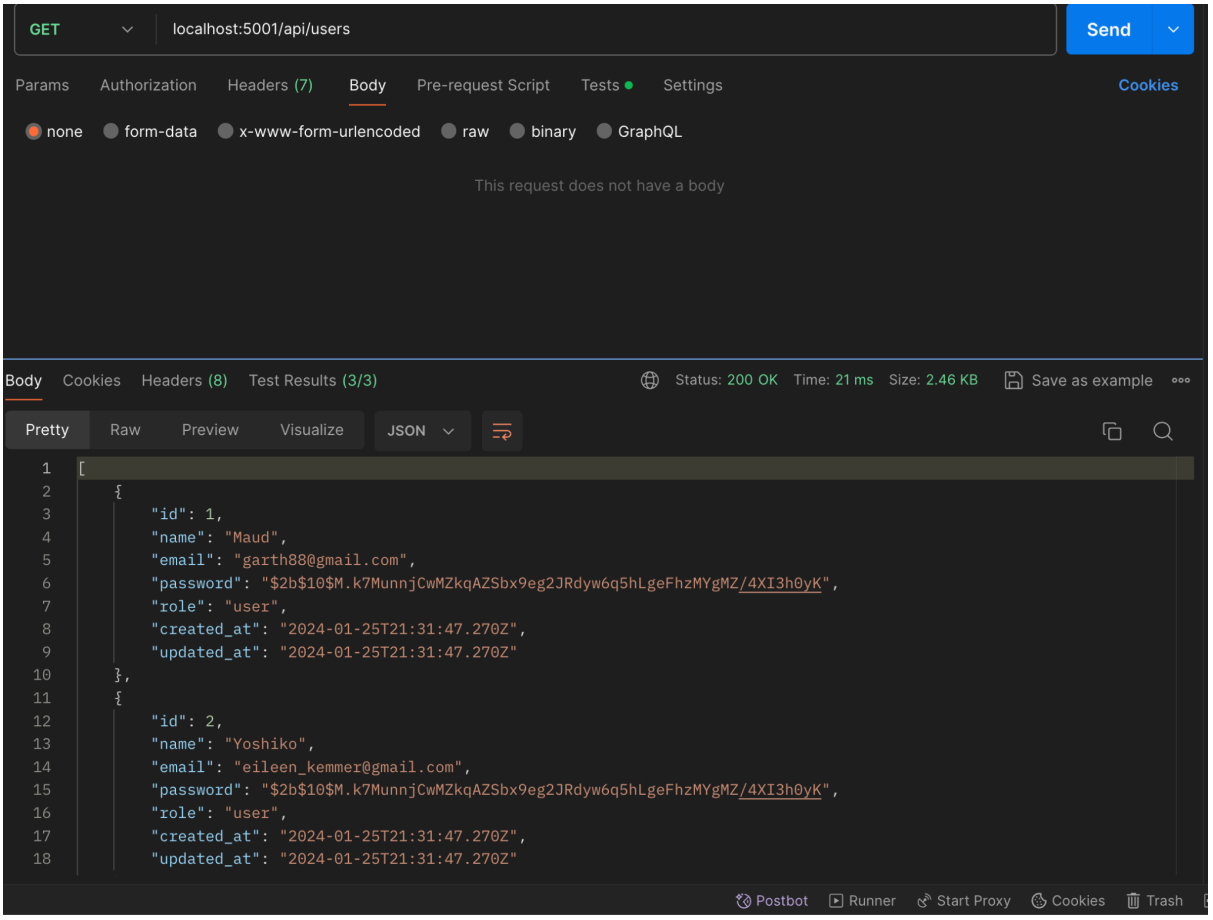
```
onst environments = ['development', 'staging', 'production'];

const connection = {
  host: '127.0.0.1',
  port: process.env.DB_PORT,
  database: process.env.DB_NAME,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD
};

const commonConfig = {
  client: 'pg',
  connection,
  pool: {
    min: 2,
    max: 10,
  },
  migrations: {
    tableName: 'knex_migrations',
    directory: './database/migrations'
  },
  seeds: {
    directory: './database/seeds'
  }
};

export default Object.fromEntries(environments.map((env) => [env, commonConfig]));

```

```
const tableName = 'train';

const trainsArr = [{
  trainName: faker.string.alpha({ length: 4, casing: 'upper' }),
  departureStation: `Paris`,
  arrivalStation: `London`,
  departureTime: faker.date.soon({ refDate: Date.now() }),
  arrivalTime: faker.date.between({ from: Date.now(), to: '2024-01-26T00:00:00.000Z' })
},
{
  trainName: faker.string.alpha({ length: 4, casing: 'upper' }),
  departureStation: `Paris`,
  arrivalStation: `London`,
  departureTime: faker.date.soon({ days: 2 }),
  arrivalTime: faker.date.soon({ days: 3 }),
},
{
  trainName: faker.string.alpha({ length: 4, casing: 'upper' }),
  departureStation: `Paris`,
  arrivalStation: `Milan`,
  departureTime: faker.date.soon({ refDate: Date.now() }),
  arrivalTime: faker.date.between({ from: Date.now(), to: '2024-01-30T00:00:00.000Z' })

},
{
  trainName: faker.string.alpha({ length: 4, casing: 'upper' }),
  departureStation: `Paris`,
  arrivalStation: `Berlin`,
  departureTime: faker.date.soon({ days: 5 }),
  arrivalTime: faker.date.soon({ days: 10 }),
},
{
  trainName: faker.string.alpha({ length: 4, casing: 'upper' }),
  departureStation: `Paris`,
  arrivalStation: `Amsterdam`,
  departureTime: faker.date.soon({ days: 7 }),
  arrivalTime: faker.date.soon({ days: 14 }),
}
];

export async function seed(knex) {
  // Deletes ALL existing entries
  await knex(tableName).del();
  const trains = trainsArr
    .map((train, index) => ({
      id: index,
      train_name: train.trainName,
      departure_station: train.departureStation,
      arrival_station: train.arrivalStation,
      departure_time: train.departureTime,
      arrival_time: train.arrivalTime
    }));

  await knex(tableName).insert(trains.map(train => ({ ...train })));
};

```

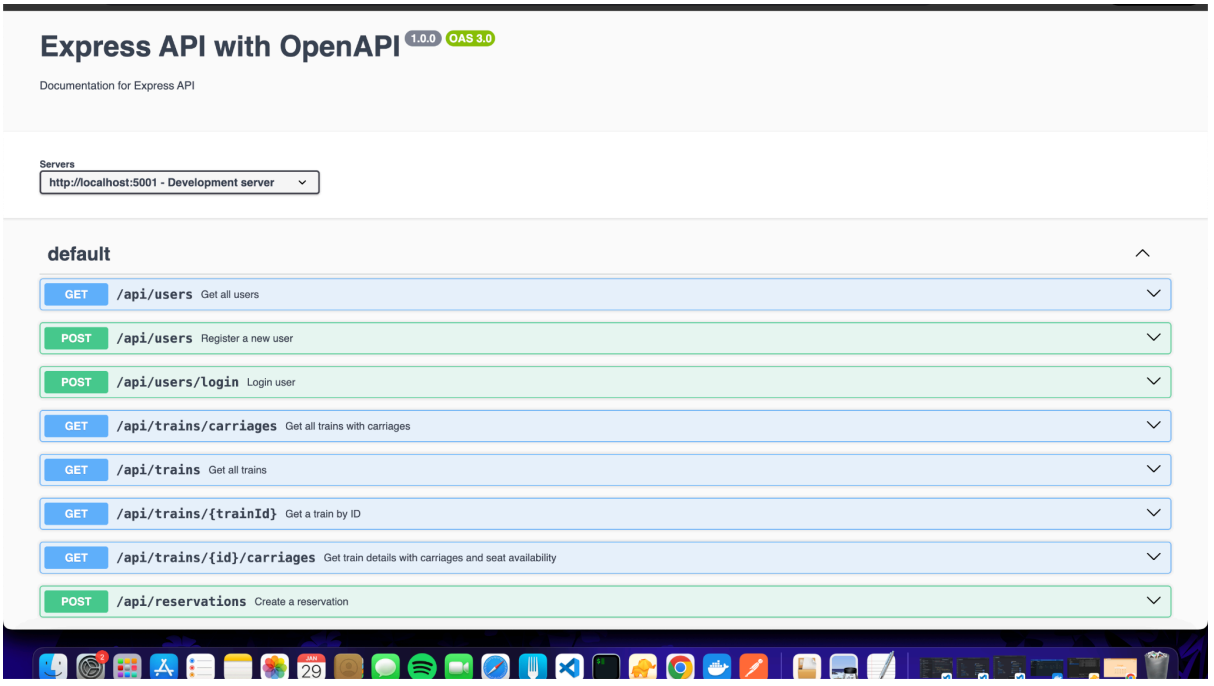
```
let query = this.table.select(selectedColumns)
            .join('train_carriage_availability', 'train.id', '=', 'train_carriage_availability.train_id')
            .join('carriage_class', 'carriage_class.id', '=', 'train_carriage_availability.carriage_class_id')
            .sum('carriage_class.seating_capacity as available_seats')
            .groupBy(selectedColumns)
            .limit(limit)
            .offset(offset);
```

- Tested with Postman was implemented,self evaluate:2

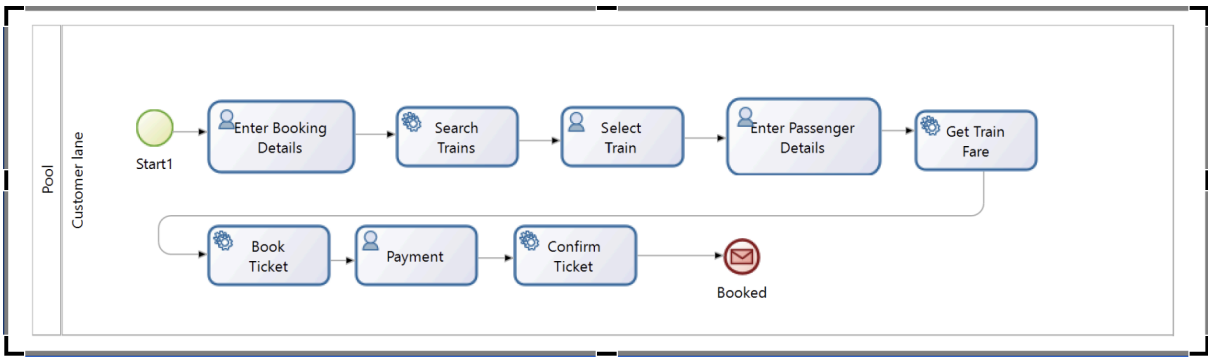




- Document API with Open API was done, self evaluate: 2



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

- 
Pool
Customer lane
Start1
Enter Booking Details
Search Trains
Select Train
Enter Passenger Details
Get Train Fare
Book Ticket
Payment
Confirm Ticket
Booked