

DONOSTIA, AZAROAK 7
CARLOS SANTAMARIA ZENTROA

IEB2024 ADIMEN ARTIFIZIAL SORTZAILEA

WWW.UEU.EUS

Babeslea:



DONOSTIA
SAN SEBASTIÁN
Euzkara

laguntzailea:



Universidad
del País Vasco

Euskal Herriko
Unibertsitatea

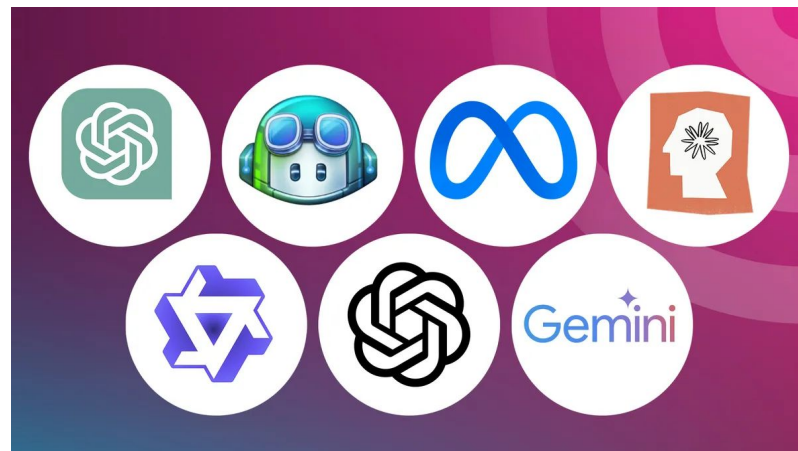
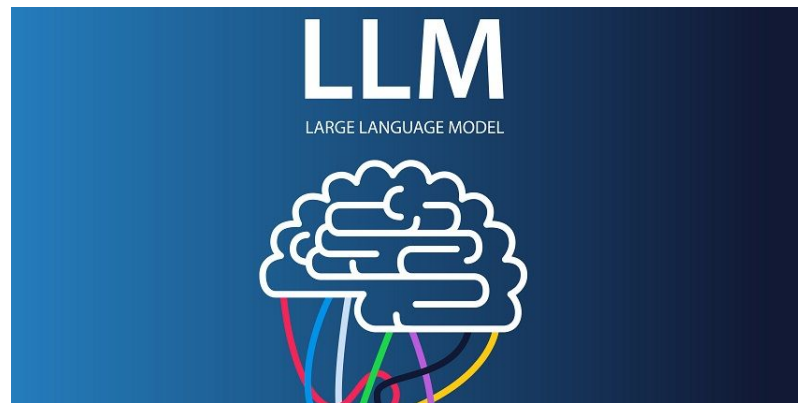


Softwarearen garapena LLMak erabiliz: 10 adibide praktiko

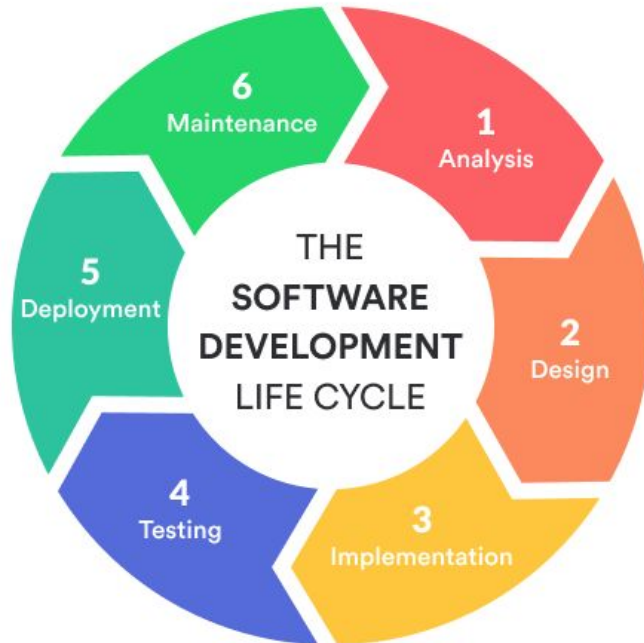


Juanan Pereira
juanan.pereira@ehu.eus

   @juanan



#2022...



ShareTrip: A Carpooling System - Project Kick-Off



Introduction

We want to create an application that allows carpooling between individuals.

When a driver wants to offer a ride he is going to make, s/he fills in a form indicating the origin and destination cities, the date, the number of seats and the price per passenger (e.g. Bilbao-Donostia, 04/04/2024, 5€,). Cities are preset, but if a city does not exist, you can always create a new one.

Any user of the application can consult the ride offer, indicating the towns of origin and destination, the date and the number of seats. Only the rides for which there are enough seats available will be shown. To carry out any other type of operation related to a ride, you must register in the system.

A registered user (from now on, traveler), can request a reservation for a ride, indicating the number of seats, whenever s/he wants or when s/he is consulting rides.

At that moment, the system will send an email to the driver of that ride, indicating that there is a reservation request. The driver can consult the reservation requests associated with his/her rides (applicant, ride and number of seats), and if deemed appropriate, accept or cancel the request. The traveler will be automatically notified of the decision in each case through an email.

<https://labur.eus/gjJO6>

Travelers can check the status of their reservations (request date, rides, number of seats, status (accepted/rejected/pending)). The driver can accept or cancel a reservation request directly, indicating the reservation code.

In case a traveler does not find a ride that matches her needs, she can create an alert, indicating the origin and destination cities, the date of the ride and the number of seats. Each time a new ride is added, the system will notify by mail the travelers who have an alert that matches the parameters of the new ride. At any time, travelers can check their alerts and unsubscribe them.

Finally, travelers can send messages to drivers requesting, for example, additional information about a ride, or requesting a pick-up. Drivers can also reply to messages. Both travelers and drivers can view all messages sent with the information of the message recipient, date, subject and text of the message, as well as those received, with the information of the message sender, date, subject and text.

Goals

We have been asked to implement a JavaFX GUI application for the ShareTrip company.

We will need to capture the requirements (requirements gathering), create the use case diagram (describing the event flow), generate the domain model, class diagram, sequence diagrams and finally implement and deploy the application, following software engineering best practices. We will follow the Scrum methodology with 3 sprints that we will need to schedule, plan and manage.

For the first iteration you are asked to complete the following tasks:

- * Requirements Analysis for this project, obtaining the Use Case Model (include one Event Flow for each actor, at your choice) and the Domain Model (we are not interested in **login** or **register** use cases, **don't include them**). In the Domain Model: name the relationships, add cardinalities, and specify the data type of each attribute.

- * Implement Sign-Up (Register) and Sign-In (Login) features. Initially just for the Driver actor. It is not explicitly asked in the wording of the exercise, but we will need those features in order to test the application.

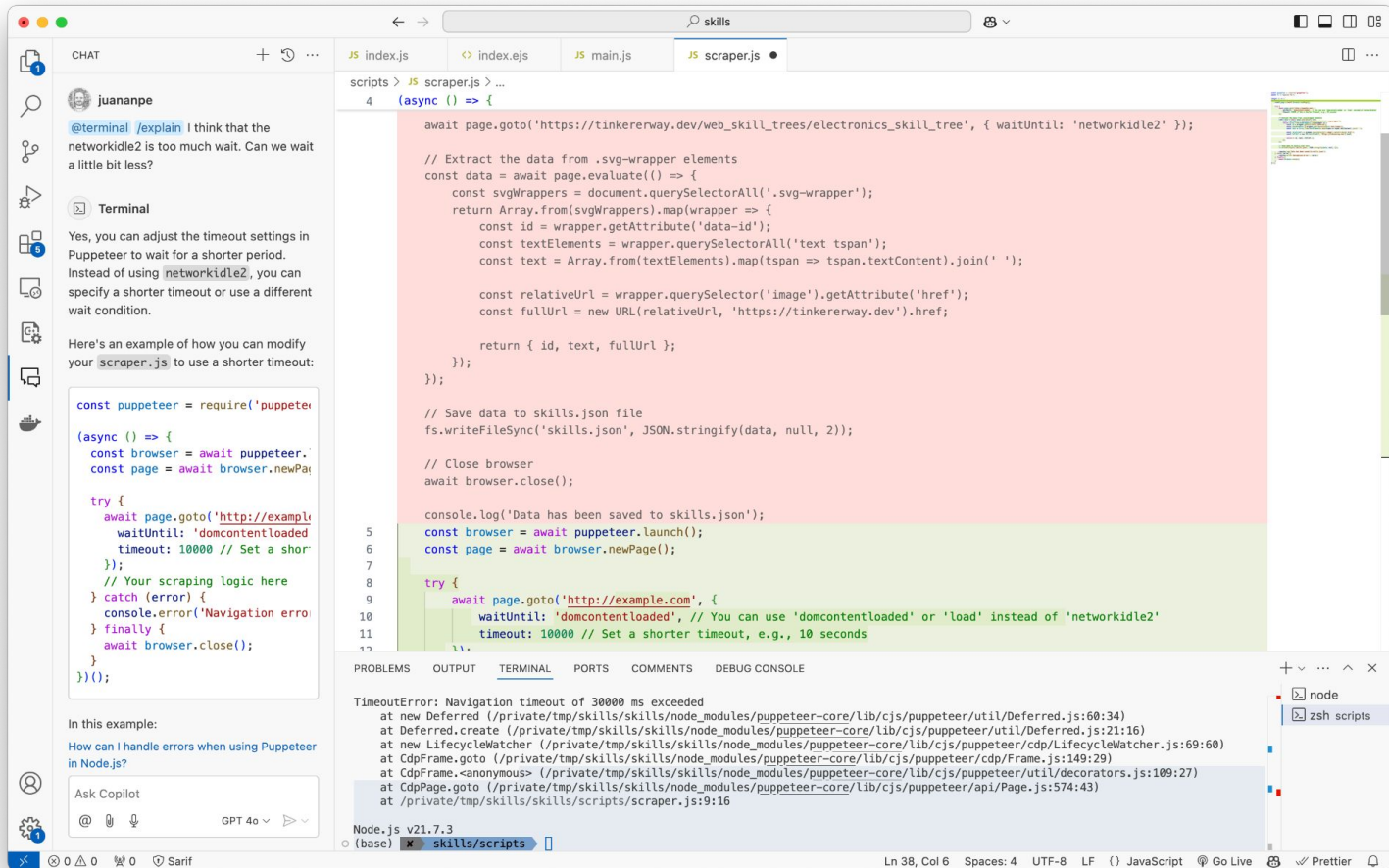
- * Adapt the "Create Ride" and "Query Rides" from iteration 0 to the new [Borderpane Layout](#).

Iteration 0: <https://github.com/juananpe/RidesFX-hibernate>

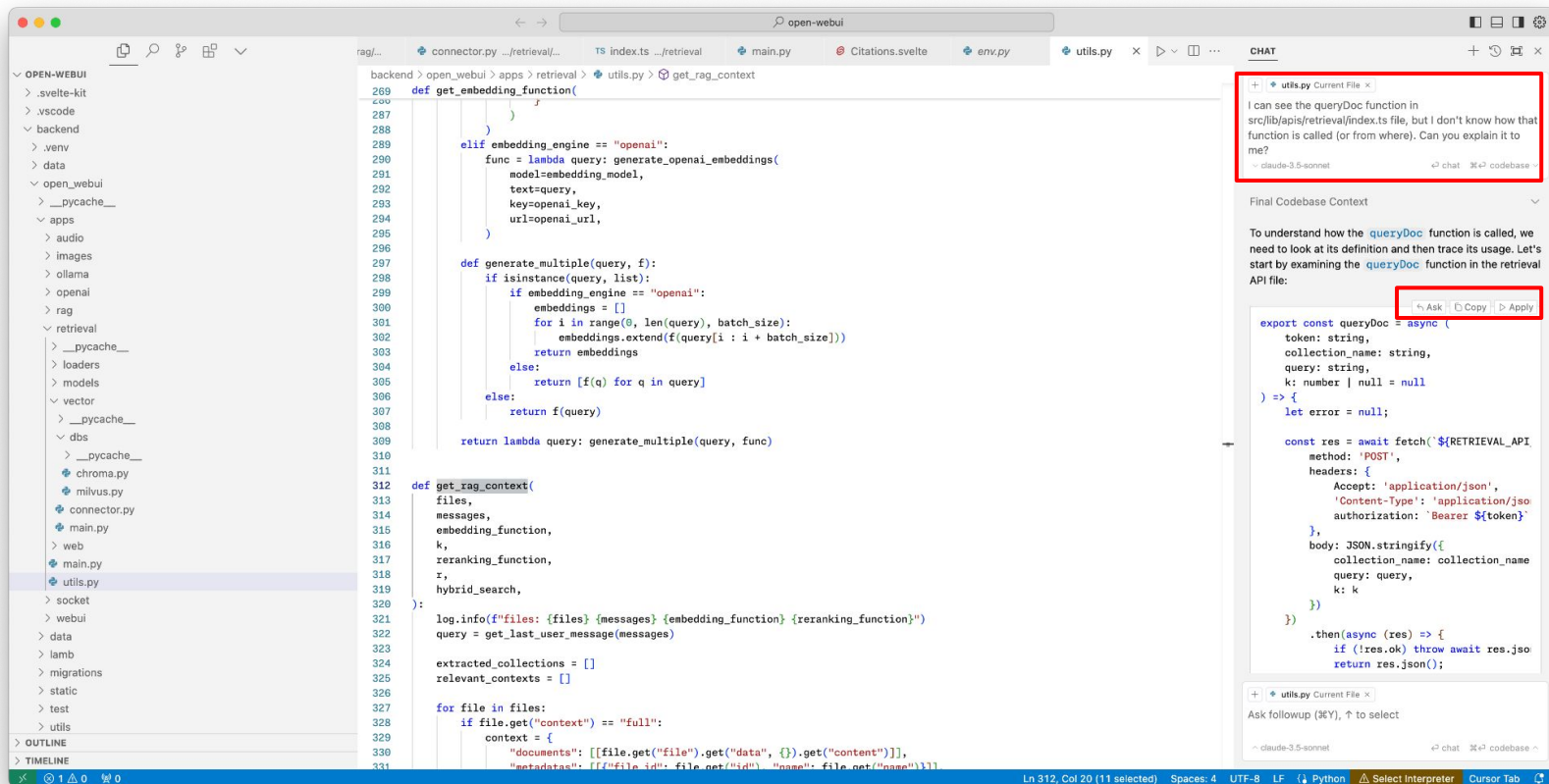
Hemendik jarraituko dugu....

<https://labur.eus/gjJO6>

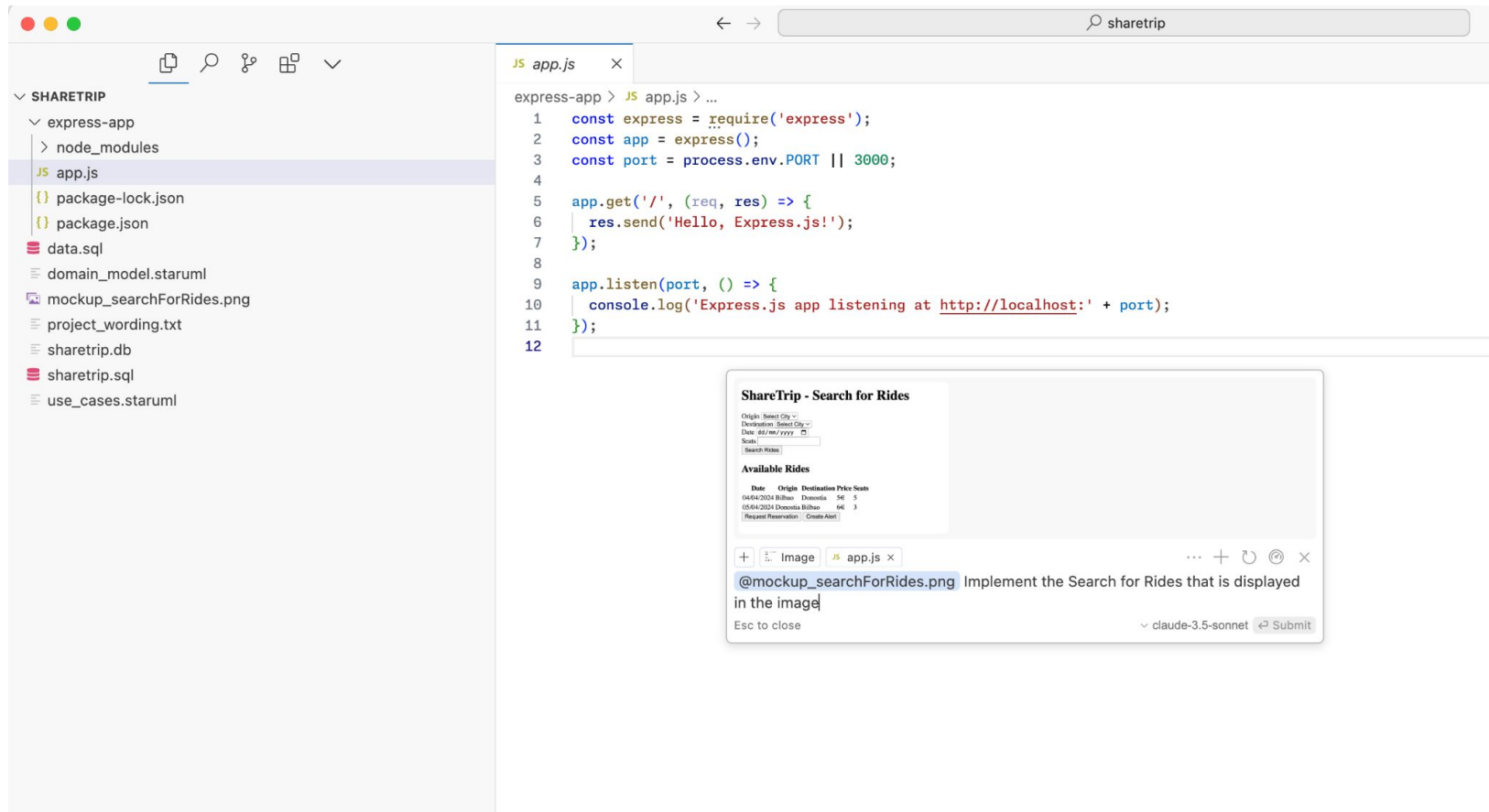
Implementazioa (GitHub Copilot)



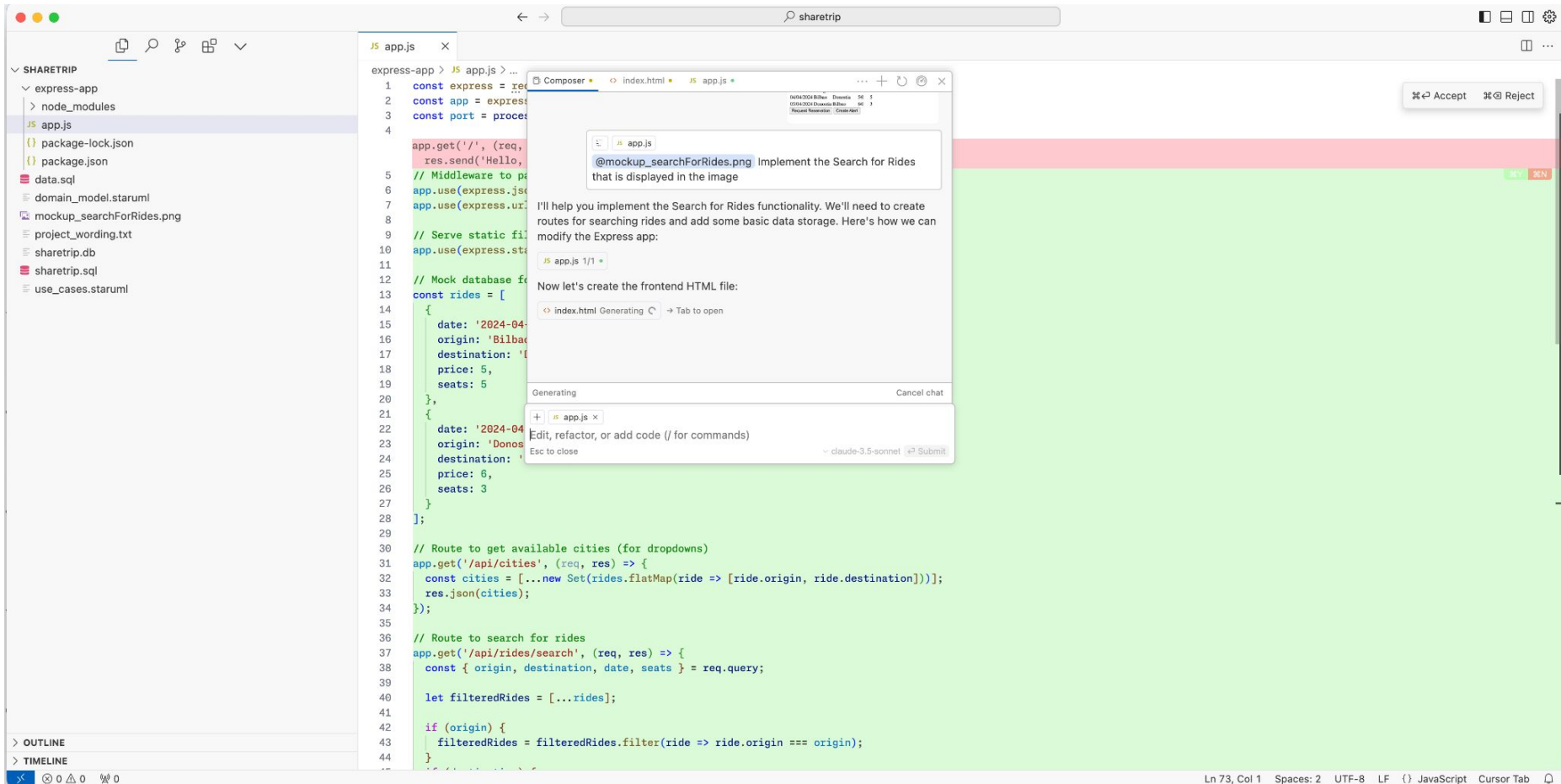
Implementazioa (Cursor AI)

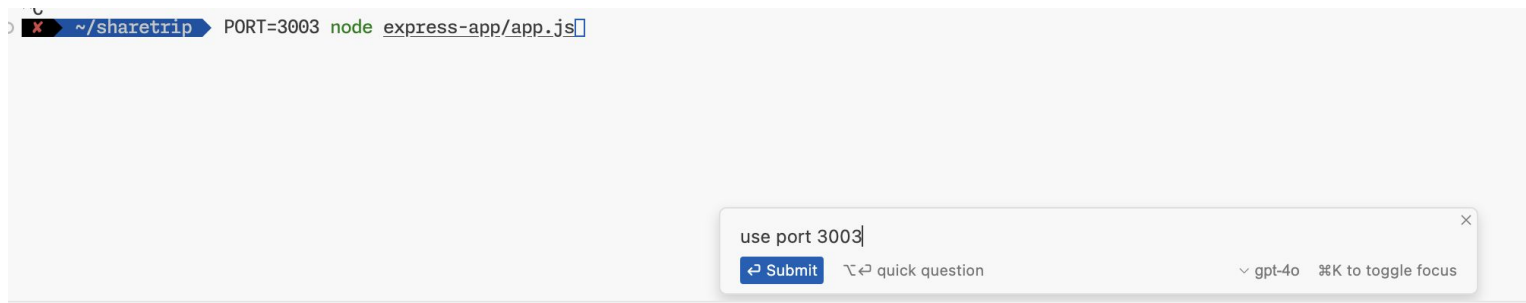
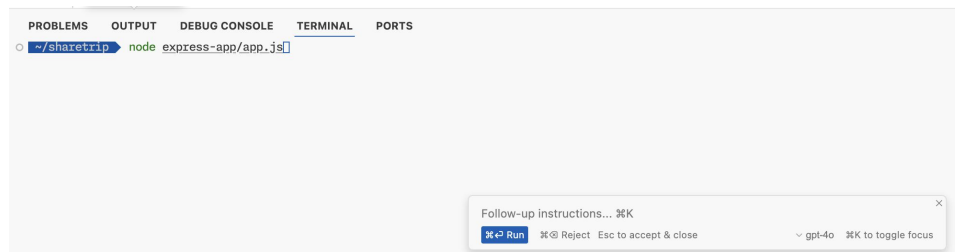
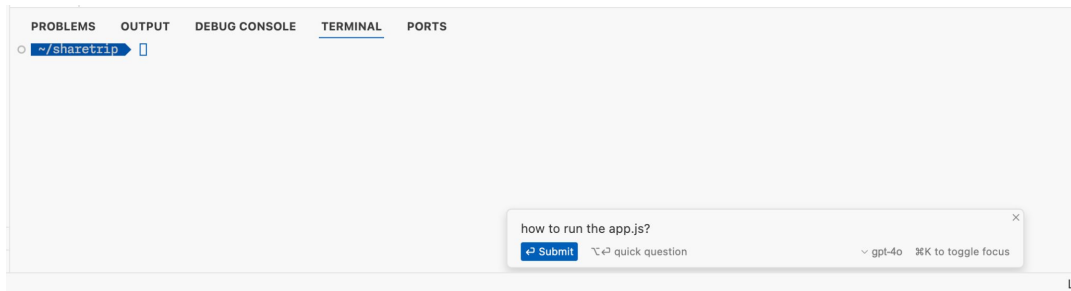


Implementazioa (Cursor AI)



Implementazioa (Cursor AI)





ShareTrip - Search for Rides

Origin

Destination

Date 📅

Seats

Available Rides

Date	Origin	Destination	Price	Seats	Actions
------	--------	-------------	-------	-------	---------

Testing

ChatGPT 4o

Please go ahead and share the source code of the three main classes of your Flight Booking application's first version. I'm ready to review them and help you with your request.

This is the first class (eus.ehu.presentation.FlightBookingController) code:

```
package eus.ehu.presentation;

import eus.ehu.business_logic.AeroplofFlightBooker;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.control.*;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Locale;

import javafx.event.ActionEvent;

import eus.ehu.business_logic.FlightBooker;
import eus.ehu.domain.ConcreteFlight;

public class FlightBookingController {

    @FXML
    private Label departure;
```

Please, implement a couple of tests for the bookSeat() method (just the tests, no need to include imports, I've already included them)

Certainly! Here are two tests for the bookSeat() method of the AeroplofFlightBooker class. These tests focus on verifying the functionality of booking seats on a flight under different conditions.

```
java

@Test
void testBookSeatSingleEconomyClassSuccessfully() {
    // Assume a flight exists and has Economy seats available
    String departure = "Donostia";
    String arrival = "Iruña";
    Date flightDate = new Date(); // Use a relevant date
    String fare = "Economy";
    int numSeatsToBook = 1;

    // Fetch a flight to book seats on
    List<ConcreteFlight> availableFlights = flightBooker.getMatchingConFlights(departure, arrival, flightDate);
    assertFalse(availableFlights.isEmpty(), "No flights found for booking.");

    ConcreteFlight flightToBook = availableFlights.get(0);
    int remainingSeatsBefore = flightToBook.getAvailableSeats(fare);
    int bookingResult = flightBooker.bookSeat(flightToBook, fare, numSeatsToBook);

    assertTrue(bookingResult >= 0, "Booking should be successful.");
    assertEquals(remainingSeatsBefore - numSeatsToBook, bookingResult, "Remaining seats should be correct.");
}

@Test
void testBookSeatFailureDueToInsufficientSeats() {
```

Message ChatGPT

Hedapena (Open Interpreter)

\$ **interpreter --auto**

> Generate a basic expressJS web application. Install everything that you need.

- Check if Node.js and NPM (node package manager) are installed on your machine.
- If Node.js and NPM are not installed, install them.
- Create a directory for the new web application.
- Initiate a new Node.js application in the directory.
- Install Express.js module via NPM.
- Create a simple Express.js application.

```
~/sharetrip > tree
├── data.sql
├── domain_model.staruml
├── mockup_searchForRides.png
├── project_wording.txt
├── sharetrip.db
├── sharetrip.sql
└── use_cases.staruml

1 directory, 7 files
~/sharetrip > interpreter --auto
> Generate a basic expressJS web application. Install everything that you need.

Perfect! I would recommend using "Node.js" and "Express.js" as the base to generate a basic web application. Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

The detailed plan would be as below:

• Check if Node.js and NPM (node package manager) are installed on your machine.
• If Node.js and NPM are not installed, install them.
• Create a directory for the new web application.
• Initiate a new Node.js application in the directory.
• Install Express.js module via NPM.
• Create a simple Express.js application.

Firstly, let's check if Node.js and NPM are already installed on your system. I will do this by running the commands node -v and npm -v. These commands will output the versions of Node.js and NPM if they are installed. If not, an error message will be displayed.

node -v && npm -v

v21.6.1
10.8.1

Great! Node.js (v21.6.1) and NPM (v10.8.1) are already installed on your machine.

Next, let's create a new directory for your web application and navigate into it. Let's call this directory express-app. Please note that you can replace express-app with any name you wish. I will then initiate a new Node.js application in this directory.

mkdir express-app && cd express-app && npm init -y

Wrote to /Users/juanan/sharetrip/express-app/package.json:

{
  "name": "express-app",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

Hedapena (Open Interpreter)

\$ `interpreter --auto`

> Generate a basic expressJS web application. Install everything that you need.

```
npm install express --save
```

```
added 65 packages, and audited 66 packages in 2s
13 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
```

Express.js has been successfully installed and added to your `package.json` file. Now, we will create a basic Express.js application.

This application will have just one route (`/`) that will be accessed via a GET request. When this route is accessed, the text "Hello, Express.js!" will be sent to the client.

Express.js applications start by creating an instance of an express application, then defining routes, and finally starting the server.

Let's create a file named `app.js` and write the code for this in it. We'll write a JavaScript(JS) code to build a simple app. I'll write the code for you:

```
echo "const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.send('Hello, Express.js!');
});

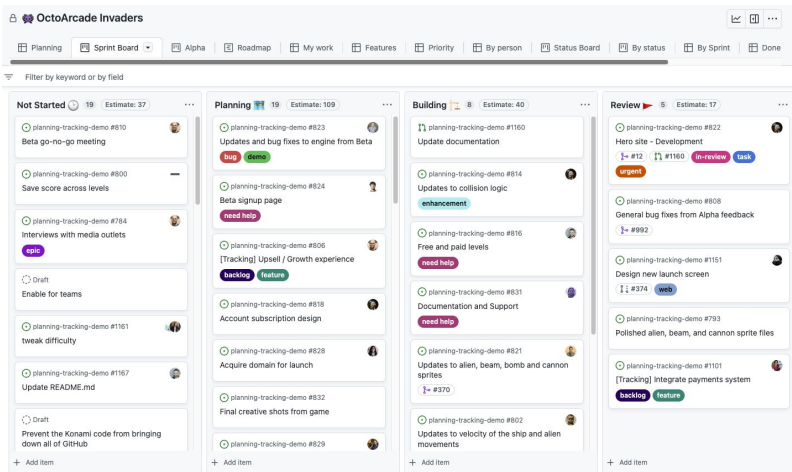
app.listen(port, () => {
  console.log('Express.js app listening at http://localhost:' + port);
});" > app.js
```

Our basic Express.js application is set up. The application is written in the `app.js` file.

To start the application, you can use the command `node app.js`, and then you should see a message in the terminal telling you that your application is listening at `http://localhost:3000`. When you navigate to this URL in your browser, you should see the message "Hello, Express.js!".

Please let me know if you'd like to continue with any further enhancements or features.

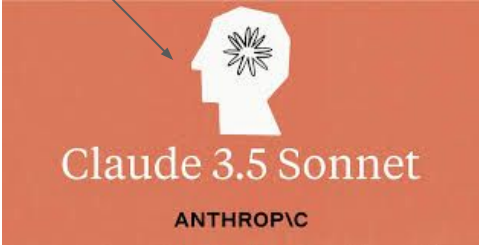
Jarraipena eta kontrola



1. Explain the project to the LLM

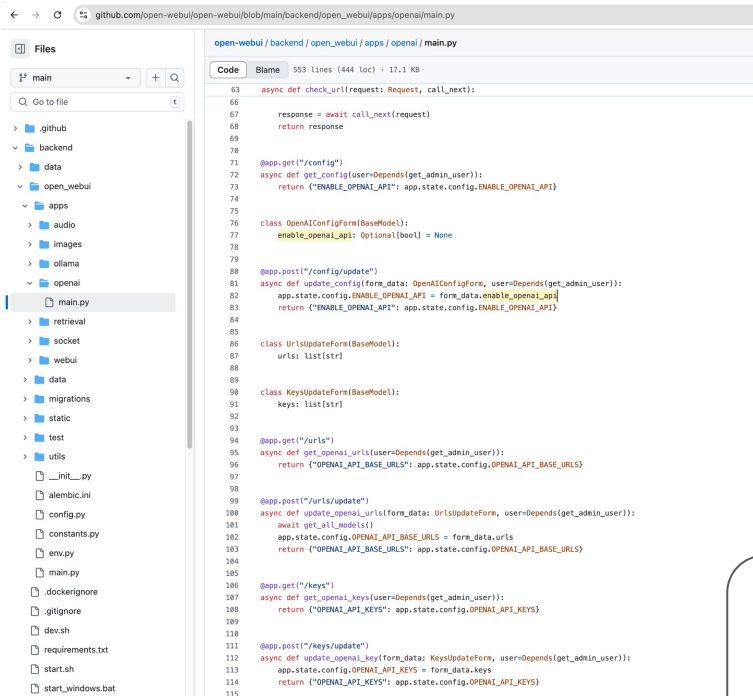
2. This is the planning and work done by my students for the 2nd sprint. The 3rd sprint will start in 24 hours.

3. Let the AI some time to think



Effort Estimation 🕒	Progress and Productivity 📈	Potential Issues ⚠️
Your estimation for the issue X is too optimistic... I'd recommend...	Good progress overall, but the pace could be improved by	There's a risk of missing deadlines ... I'd suggest....

Mantentze-lanak



1) files-to-prompt

2. "Identify potential bugs.
Suggest some Java code
improvements, best
practices, design patterns..."



3.

Suggestions for improving the Java project structure and code #3

[Closed](#) juanape opened this issue on May 27 · 0 comments

Here are some suggestions to improve the Java project structure and code:

1. Use fully capitalized acronyms in names like "BL" instead of "Bf" for better readability. Rename the BfFacade and BfFacadeImplementation to BLFacade and BLFacadeImplementation.
2. Add a base entity class that contains the id field and any common methods. Make the domain classes Student, Teacher, Subject, AcademicRecord extend this base class. This will reduce code duplication.
3. Override equals() and hashCode() methods in the domain classes, especially since they are used in Sets. Use the entity for equality checks. This will ensure proper behavior when entities are added/removed from collections.
4. Extract some of the complex logic in methods like isEligible() and enroll() in the BfFacadeImplementation into separate private methods. This will make the code more readable and easier to unit test.
5. Use a connection pool like HikariCP for managing database connections instead of creating EntityManagerFactory directly. Move the connection creation logic to a separate DatabaseConfig class.

```
public class DatabaseConfig {  
    private EntityManagerFactory emf;  
  
    @PostConstruct  
    public void init() {  
        HikariConfig config = new HikariConfig();  
        config.setJdbcUrl("jdbc:mysql://localhost:3306/mydatabase");  
        config.setUsername("root");  
        config.setPassword("password");  
  
        HikariDataSource ds = new HikariDataSource(config);  
  
        LocalContainerEntityManagerFactoryBean factory = new LocalContainerEntityManagerFactoryBean();  
        factory.setDataSource(ds);  
        factory.setPackagesToScan("faculty.project.domain");  
        factory.setJpaVendorAdapter(new HibernateJpaVendorAdapter());  
        factory.afterPropertiesSet();  
  
        this.emf = factory.getObject();  
    }  
  
    public EntityManagerFactory getEntityManagerFactory() {  
        return emf;  
    }  
}
```




Publish your report as an issue...

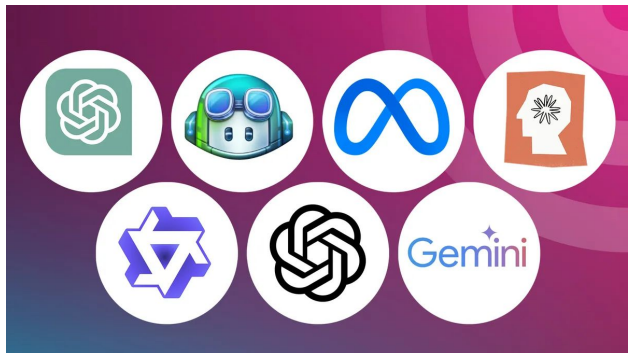


Certainly!

Questions earned this hour on May 22: 2 correct

[illegible]

Ondorioak



+ Tools = Laguntzaile Birtualak

- ez gaituzte **ordezkatzen**, lagundu egiten dute
- **gainbegiratzea** behar dute
- ez dira nekatzen (24/365)
- ideiak **aztertzen** laguntzen dute