

DONOSTIA, AZAROAK 7  
CARLOS SANTAMARIA ZENTROA

# IEB2024 ADIMEN ARTIFIZIAL SORTZAILEA

WWW.UEU.EUS

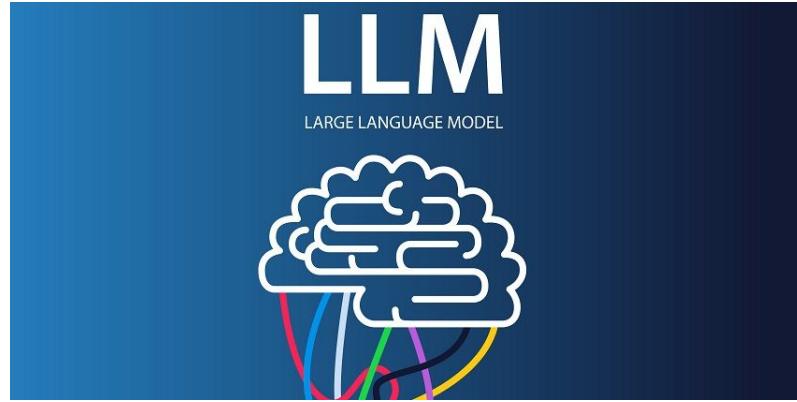


## Sofwarearen garapena LLmak erabiliz: 10 adibide praktiko

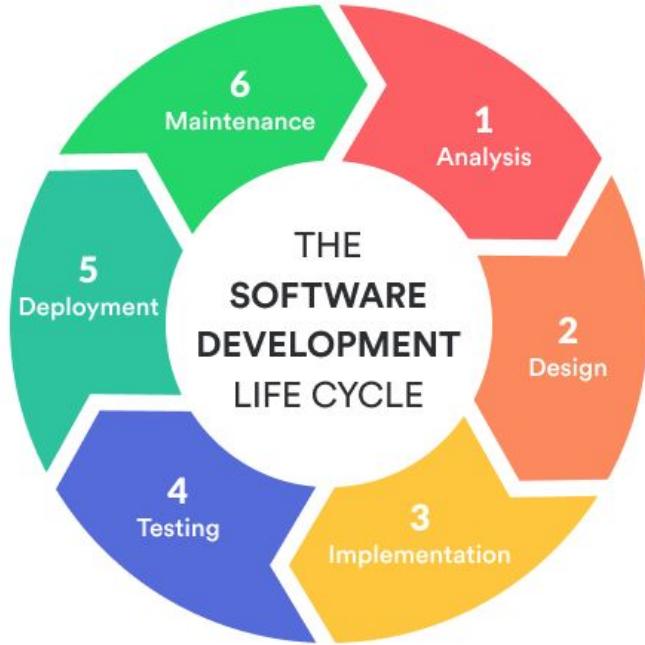


Juanan Pereira  
[juanan.pereira@ehu.eus](mailto:juanan.pereira@ehu.eus)





#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/uanganpe/RidesFX-hibernate>

# Hemendik jarraituko dugu....

<https://labur.eus/gjJO6>

# Implementazioa (GitHub Copilot)

The screenshot shows a GitHub Copilot interface integrated into a code editor. On the left, there's a sidebar with icons for Chat, Terminal, and a file browser. The main area has tabs for index.js, index.ejs, main.js, and scraper.js. The scraper.js tab is active, displaying the following code:

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();

  try {
    await page.goto('http://example.com');
    await page.waitForSelector('#networkidle2');
    await page.evaluate(() => {
      // Extract the data from .svg-wrapper elements
      const data = await page.evaluate(() => {
        const svgWrappers = document.querySelectorAll('.svg-wrapper');
        return Array.from(svgWrappers).map(wrapper => {
          const id = wrapper.getAttribute('data-id');
          const textElements = wrapper.querySelectorAll('text tspan');
          const text = Array.from(textElements).map(tspan => tspan.textContent).join(' ');
          return { id, text };
        });
      });
      // Save data to skills.json file
      fs.writeFileSync('skills.json', JSON.stringify(data, null, 2));
    });
    await browser.close();
  } catch (error) {
    console.error(`Navigation error: ${error}`);
  } finally {
    await browser.close();
  }
})();
```

Below the code, a note says: "In this example: How can I handle errors when using Puppeteer in Node.js?" A tooltip from GitHub Copilot suggests: "Ask Copilot @ Sarif". The bottom status bar shows "Node.js v21.7.3" and "skills/scripts". The bottom right corner shows the GitHub Copilot logo.

In the terminal pane, there is an error message:

```
TimeoutError: Navigation timeout of 30000 ms exceeded
at new Deferred (/private/tmp/skills/skills/node_modules/puppeteer-core/lib/cjs/puppeteer/util/Deferred.js:60:34)
at Deferred.create (/private/tmp/skills/skills/node_modules/puppeteer-core/lib/cjs/puppeteer/util/Deferred.js:21:16)
at new LifecycleWatcher (/private/tmp/skills/skills/node_modules/puppeteer-core/lib/cjs/puppeteer/cdp/LifecycleWatcher.js:69:60)
at CdpFrame.goto (/private/tmp/skills/skills/node_modules/puppeteer-core/lib/cjs/puppeteer/cdp/Frame.js:149:29)
at CdpFrame.<anonymous> (/private/tmp/skills/skills/node_modules/puppeteer-core/lib/cjs/puppeteer/util/decorators.js:109:27)
at CdpPage.goto (/private/tmp/skills/skills/node_modules/puppeteer-core/lib/cjs/puppeteer/api/Page.js:574:43)
at /private/tmp/skills/scripts/scrapers.js:9:16
```

The status bar at the bottom also includes "Ln 38, Col 6" and "Spaces: 4".

# Implementazioa (Cursor AI)

# Implementazioa (Cursor AI)

The screenshot shows a code editor interface with a sidebar and a main workspace.

**Left Sidebar:** Shows the project structure under the 'SHARETRIP' folder:

- express-app
- node\_modules
- JS app.js** (selected)
- package-lock.json
- package.json
- data.sql
- domain\_model.staruml
- mockup\_searchForRides.png
- project\_wording.txt
- sharetrip.db
- sharetrip.sql
- use\_cases.staruml

**Main Workspace:** The code editor displays the contents of `app.js`:

```
express-app > JS app.js > ...
1 const express = require('express');
2 const app = express();
3 const port = process.env.PORT || 3000;
4
5 app.get('/', (req, res) => {
6   res.send('Hello, Express.js!');
7 });
8
9 app.listen(port, () => {
10   console.log('Express.js app listening at http://localhost:3000');
11 });
12
```

**Bottom Overlay:** A modal window titled "ShareTrip - Search for Rides" is displayed, showing a form and a table of available rides.

**Form Fields:**

- Origin: Select City
- Destination: Select City
- Date: dd/mm/yyyy
- Search
- Search Rides

**Available Rides Table:**

Date	Origin	Destination	Price	Seats
04/04/2023	Bilbao	Donostia	5€	5
05/04/2023	Donostia	Bilbao	6€	3

**Text Input:** An input field contains the instruction: "@mockup\_searchForRides.png Implement the Search for Rides that is displayed in the image".

# Implementazioa (Cursor AI)

The screenshot shows a code editor interface with a sidebar and a main workspace. The sidebar on the left lists project files: express-app, node\_modules, app.js, package-lock.json, package.json, data.sql, domain\_model.staruml, mockup\_searchForRides.png, project\_wording.txt, sharetrip.db, sharetrip.sql, and use\_cases.staruml. The main workspace has tabs for 'app.js' and 'index.html'. The 'app.js' tab contains JavaScript code for an Express application, including routes for cities and rides. A modal window titled 'Composer' is open over the code, displaying a task card:

**app.js**  
@mockup\_searchForRides.png Implement the Search for Rides that is displayed in the image

I'll help you implement the Search for Rides functionality. We'll need to create routes for searching rides and add some basic data storage. Here's how we can modify the Express app:

Now let's create the frontend HTML file:

index.html Generating → Tab to open

Generating

Cancel chat

+ app.js

Edit, refactor, or add code (/ for commands)

Esc to close

claudie-3.5-sonnet Submit

The code in 'app.js' includes:

```
express-app > JS app.js > ...
1 const express = require('express');
2 const app = express();
3 const port = process.env.PORT || 3001;
4
5 app.get('/', (req, res) => {
6   res.send('Hello, world!');
7 })
8
9 // Middleware to parse JSON bodies
10 app.use(express.json());
11 app.use(express.urlencoded({ extended: true }));
12
13 // Serve static files
14 app.use(express.static('public'));
15
16 // Mock database for rides
17 const rides = [
18   {
19     date: '2024-04-01',
20     origin: 'Bilbao',
21     destination: 'San Sebastián',
22     price: 5,
23     seats: 5
24   },
25   {
26     date: '2024-04-02',
27     origin: 'Donostia',
28     destination: 'Bilbao',
29     price: 6,
30     seats: 3
31   }
32 ];
33
34 // Route to get available cities (for dropdowns)
35 app.get('/api/cities', (req, res) => {
36   const cities = [...new Set(rides.flatMap(ride => [ride.origin, ride.destination]))];
37   res.json(cities);
38 });
39
40 // Route to search for rides
41 app.get('/api/rides/search', (req, res) => {
42   const { origin, destination, date, seats } = req.query;
43
44   let filteredRides = [...rides];
45
46   if (origin) {
47     filteredRides = filteredRides.filter(ride => ride.origin === origin);
48   }
49
50   if (destination) {
51     filteredRides = filteredRides.filter(ride => ride.destination === destination);
52   }
53
54   if (date) {
55     filteredRides = filteredRides.filter(ride => ride.date === date);
56   }
57
58   if (seats) {
59     filteredRides = filteredRides.filter(ride => ride.seats === seats);
60   }
61
62   res.json(filteredRides);
63 });
64
65 // Error handling
66 app.use((err, req, res, next) => {
67   console.error(err);
68   res.status(500).send(`Something went wrong! ${err.message}`);
69 });
70
71 // Start the server
72 app.listen(port, () => {
73   console.log(`Server is running on port ${port}`);
74 });
```

At the bottom of the screen, status indicators show: Ln 73, Col 1, Spaces: 2, UTF-8, LF, {} JavaScript, Cursor Tab.

PROBLEMS    OUTPUT    DEBUG CONSOLE    **TERMINAL**    PORTS

~ /sharetrip >

how to run the app.js?

Submit quick question gpt-4o ⌘K to toggle focus

PROBLEMS    OUTPUT    DEBUG CONSOLE    **TERMINAL**    PORTS

~ /sharetrip > node express-app/app.js

Follow-up instructions... ⌘K

Run Reject Esc to accept & close gpt-4o ⌘K to toggle focus

~ /sharetrip > PORT=3003 node express-app/app.js

use port 3003

Submit quick question gpt-4o ⌘K to toggle focus

## ShareTrip - Search for Rides

Origin

Destination

Date

Seats

### Available Rides

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

# Testing

ChatGPT 4o ▾

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

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 status;

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 Copy code

@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 do not match expected value.");
}

@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
├── domainModel.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.
```

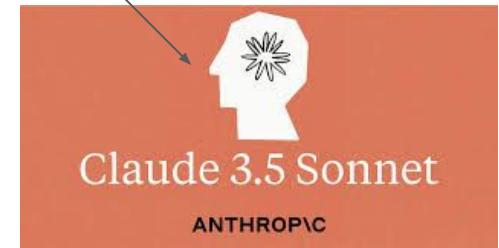
# Jarraipena eta kontrola

The image shows a digital planning board for a project titled "OctoArcade Invaders". The board is organized into four main columns: "Not Started", "Planning", "Building", and "Review". Each column contains a list of tasks or issues, each with a unique ID and a brief description. The tasks are categorized by status (e.g., "Not Started", "Planning", "Building", "Review") and priority (e.g., "bug", "demo", "enhancement", "epic", "backlog", "feature", "need help", "urgent", "web", "task"). The interface includes a sidebar with navigation links and a search/filter bar at the top.

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....

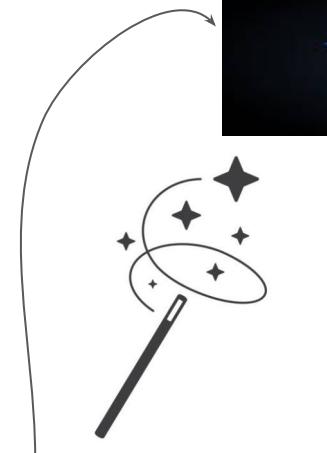
# Mantentze-lanak

A screenshot of a GitHub repository page for 'open-webui'. The left sidebar shows a tree view of files and folders. The main area displays the content of 'main.py'. The code is written in Python and includes imports from 'open\_webui' and 'openai'. It defines several functions and classes, such as 'check\_urlrequest', 'OpenAIConfigForm', 'KeysUpdateForm', and 'UrlsUpdateForm', which handle requests for configuration updates like enabling OpenAI API, updating URLs, and managing API keys.

```
open-webui / backend / open_webui / apps / openai / main.py
Code Blame 553 lines (444 loc) · 17.1 KB
63     async def check_urlrequest(request, call_next):
64         response = await call_next(request)
65         return response
66
67     @app.get("/config")
68     async def get_config(userDepends(get_admin_user)):
69         return {"ENABLE_OPENAI_API": app.state.config.ENABLE_OPENAI_API}
70
71     class OpenAIConfigForm(BaseModel):
72         enable_openai_api: Optional[bool] = None
73
74     @app.post("/config/update")
75     async def update_config(form_data: OpenAIConfigForm, userDepends(get_admin_user)):
76         app.state.config.ENABLE_OPENAI_API = form_data.enable_openai_api
77         return {"ENABLE_OPENAI_API": app.state.config.ENABLE_OPENAI_API}
78
79     class UrlsUpdateForm(BaseModel):
80         urls: list[str]
81
82     class KeysUpdateForm(BaseModel):
83         keys: list[str]
84
85     @app.get("/urls")
86     async def get_openai_urls(userDepends(get_admin_user)):
87         return {"OPENAI_API_BASE_URLS": app.state.config.OPENAI_API_BASE_URLS}
88
89     @app.post("/urls/update")
90     async def update_openai_urls(form_data: UrlsUpdateForm, userDepends(get_admin_user)):
91         await get_all_urls()
92         app.state.config.OPENAI_API_BASE_URLS = form_data.urls
93         return {"OPENAI_API_BASE_URLS": app.state.config.OPENAI_API_BASE_URLS}
94
95     @app.get("/keys")
96     async def get_openai_keys(userDepends(get_admin_user)):
97         return {"OPENAI_API_KEYS": app.state.config.OPENAI_API_KEYS}
98
99     @app.post("/keys/update")
100    async def update_openai_keys(form_data: KeysUpdateForm, userDepends(get_admin_user)):
101        app.state.config.OPENAI_API_KEYS = form_data.keys
102        return {"OPENAI_API_KEYS": app.state.config.OPENAI_API_KEYS}
103
104
105
106
107
108
109
110
111
112
113
114
```

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 Juananpe opened this issue on May 27 - 0 comments

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

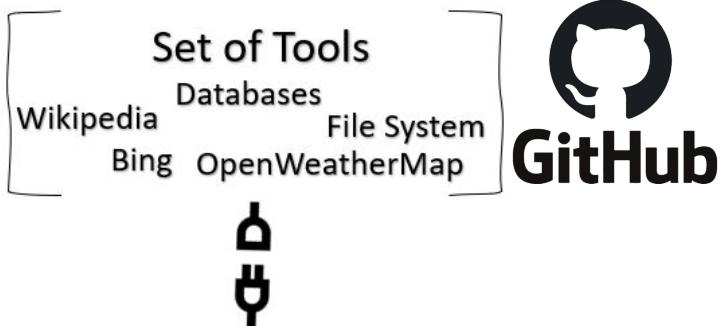
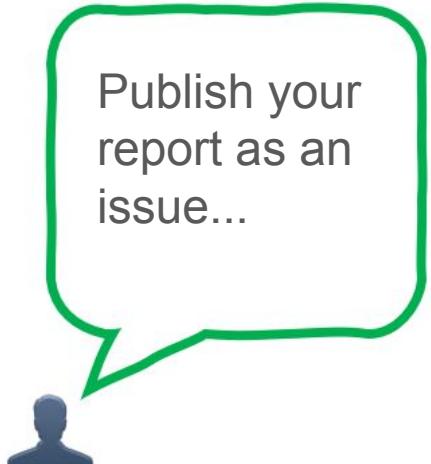
1. Use fully capitalized acronyms in names like "BL" instead of "B" for better readability. Rename the BLFacade and BLFacadeImplementation to BLFacade and BLFacadeImplementation.
2. Add a base entity class that contains the id field and any common methods. Make the domain classes Student, Teacher, Subject, 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 id for equality checks. This will ensure proper behavior when entities are added/moved from collections.
4. Extract some of the complex logic in methods like isEligible() and enrol() in the BLFacadeImplementation 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.

```
@MappedSuperclass
public abstract class BaseEntity {
    ...
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    ...
    // getters and setters
}

@Override
public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if (o instanceof BaseEntity) {
        BaseEntity other = (BaseEntity) o;
        return id.equals(other.id);
    }
    return false;
}

@Override
public int hashCode() {
    return Objects.hash(id);
}
```

```
public class DatabaseConfig {
    ...
    private EntityManagerFactory entityManagerFactory;
    ...
    public EntityManagerFactory getEntityManagerFactory() {
        if (entityManagerFactory == null) {
            entityManagerFactory = Persistence.createEntityManagerFactory("myEntityName");
        }
        return entityManagerFactory;
    }
}
```



Certainly!

A yellow bracket groups the GitHub screenshot on the left and the AI icon on the right. The GitHub screenshot shows a pull request titled "Suggestions for improving the Java project structure and code #3" with 2 comments. It lists several suggestions for code improvement, such as using final variables and avoiding static methods. The AI icon is a white robot head with a blue outline.

```
#Project class
private String name;
private String version;
private Long id;
// getters and setters

3. Overuse equals() and hashCode() methods in the domain classes, especially since they are used in tests. Use the equals() and hashCode() methods in the domain classes. This will avoid proper behavior when entities are addressed from collections.
4. Design the domain classes to be immutable. This will make the code more readable and easier to unit test.
5. Use a connection pool like Hikari for managing database connections instead of creating EntityManagerFactory directly. This will also allow you to use the EntityManager API to handle database operations.
```



# Ondorioak



+ Tools = Lagunzaile Birtualak

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