

# Team We Them Boys' xGame Development Guide

## Overview

xGame is a platform for players to face off in head-to-head matches of Legan chess. Players can create an account to create matches with other players and play their matches over time. Matches can be quit and resumed at any time, picking up where they left off. Players will be notified of new match invites, if their match invite has been accepted or rejected, when a move has been made, and when a match has completed. Players can also view their match history and view other player's profile and match history.

## Architecture

xGame is a web platform consisting of a SPA React front end, a REST API backend, and a MariaDB database. The full list of technologies is below:

- Node.js
- React
- Java Spring Framework
- Maven
- JPA
- Hibernate
- MariaDB

The Java application is a REST API built in Spring Boot and makes use of N-tier architecture, with defined view, logic, and data access layers. The view layer contains all the REST API endpoint controllers that are used for communication with the front end React client. The logic layer consists of services that make up the business and system logic. Finally, the data access layer contains all the CRUD repositories and is solely responsible for all communications with the backend database. Dependency Injection is used for communication across the various layers of the application.

## REST API

### User

The User API can be used to search, create, soft delete, and retrieve users.

### Endpoints

#### *User Search*

Searches for a user by the param. Performs a fuzzy match on the user's nickname and an exact match. Returns a User.

**GET /user/search?param=searchtext**

#### *Login a User*

Checks that user credentials match the given information and returns the found User. Returns a User.

**GET /user/login**

#### *Request Body*

UserCredentials

### *Register a new User*

Creates a new user based on the given UserCredentials. Checks for unique nickname and email. Returns a User.

### **POST /user/register**

#### *Request Body*

UserCredentials

#### *Match*

The match API can create new matches, get all match invitations, and accept match invitations.

#### *Endpoints*

##### *Get match state*

Gets state of match with given id. Returns the Match.

### **GET /match?matchId=int**

#### *Create a match*

Creates a new PENDING match between the white player and the black player. The black player logically now has an invitation to a match. Returns the newly created Match.

### **POST /match?whiteId=int&blackId=int**

#### *Accept a match invitation*

Accepts a match by changing the match status to "INPROGRESS" and setting the turn count to 1. Returns a Match with the updated match state.

### **PATCH /match/accept?matchId=int**

#### *Message*

#### *Endpoints*

##### *Get all messages*

Gets all messages, including notifications and invitations for a single player. Returns a list of generic Message objects that are either a message or invitation. If the object is a message, the id is the messageId. If the object is an invitation, the id is the matchId of the corresponding match.

### **GET /message?playerId=int**

#### *Get match invitations*

Gets all match invitations for the player with the given id. Returns a list of MatchInvites.

### **GET /message/invitation?playerId=int**

#### *Objects*

```
Message: {  
  "id": int,  
  "type": string,  
  "content": string,  
  "timestamp": string
```

```

}
Match: {
    "id": int,
    "whiteId": int,
    "blackId": int,
    "whiteEmail": string,
    "blackEmail": string,
    "turnCount": int,
    "chessBoard": string
}
User: {
    "id": int,
    "nickname": string,
    "email": string
}

UserCredentials: {
    "nickname": string,
    "email": string,
    "password": string
}

```

## Environment Setup

### Client

#### Software and Tools

##### *IntelliJ IDEA – 2020-2.3*

IntelliJ is used as the integrated development environment for the client side of the application.

##### *Node.JS*

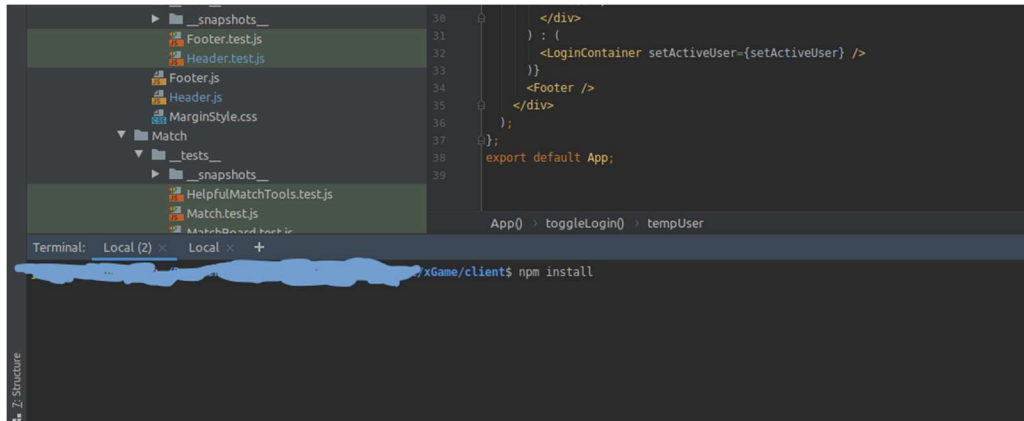
Node.JS is the engine that will allow us to run the JavaScript outside the browser

##### *NPM*

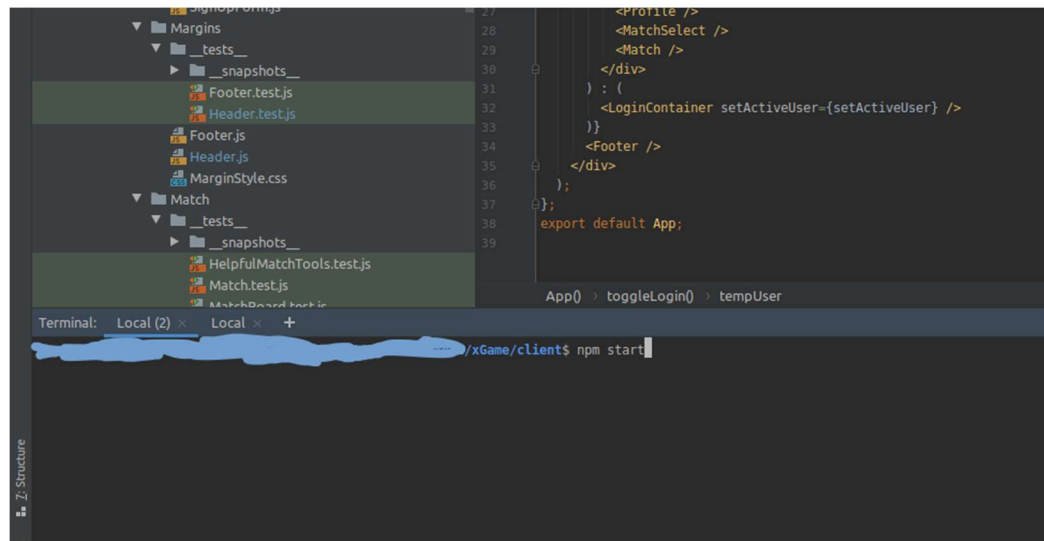
NPM is a node package manager that allows us to install packages to use in our code.

### Setup Guide

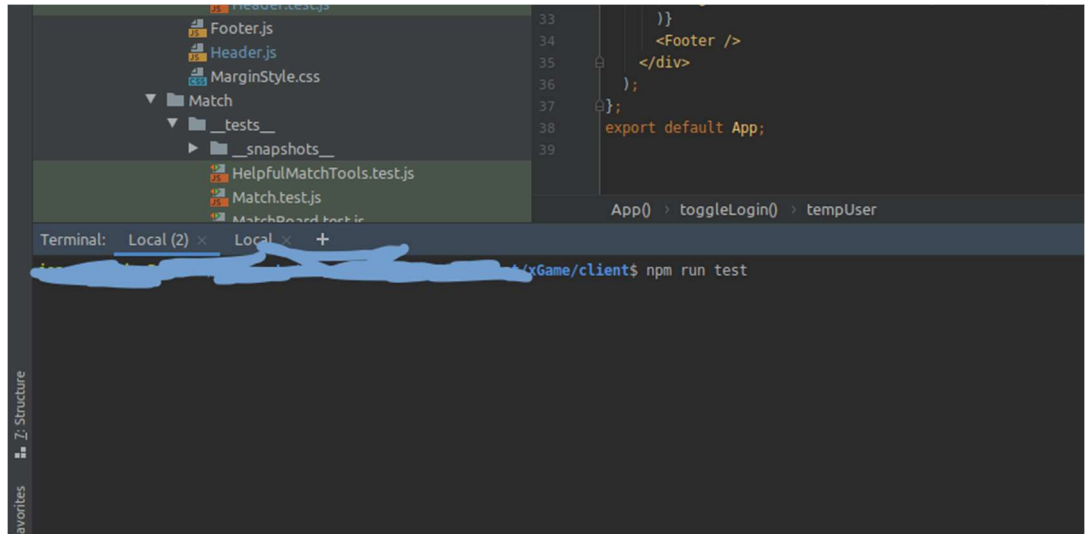
1. Install IntelliJ IDEA
  - a. Follow the steps in the wizard, keeping all settings to default
2. Install Node.JS
3. Install NPM.
4. Open IntelliJ and create a new project from existing version control.
5. In the IntelliJ terminal, navigate to the client directory.
6. Run 'npm install' to automatically install all the necessary packages.



7. From here the project is ready for development
  - a. To test the project type 'npm start' this will open a browser in development mode so you can see the changes you make in the code live



- b. To build the project for deployment type 'npm run build' this will create a build folder that will hold the files ready for deployment
  - c. To run unit test type 'npm test' this will find all of the test code in the project and run it



## Server

### Software and Tools

*Eclipse IDE for Java Developers – 2020-09*

Eclipse is used as the integrated development environment for the server side of the application.

Required extensions include:

- Spring Tools 4
- ObjectAid UML Explorer

*Maria DB 10.3.25*

MariaDB is a database management system. The server application uses MariaDB to store users, messages, and chess match information. It is necessary to stand up and instance for the development environment. A direct download link can be found below:

<https://mariadb.org/download/>

*HeidiSQL*

HeidiSQL is a database administration tool that we will use to manage and maintain the application database as necessary. HeidiSQL comes bundled with MariaDB and is highly recommended. Although any database administration tool that works with MariaDB or MySQL can be used.

*GitHub Desktop (Optional)*

GitHub Desktop is a tool for working with our GitHub repository. With it, you can pull down code, create branches, merge branches, and create pull requests. It is not required, and other tools can do the same job. A direct download link can be found below:

<https://central.github.com/deployments/desktop/desktop/latest/win32>

*Postman (Optional)*

Postman is a tool for testing REST API functionality. It is recommended to check that HTTP calls to the server are functioning properly. A direct download link can be found below:

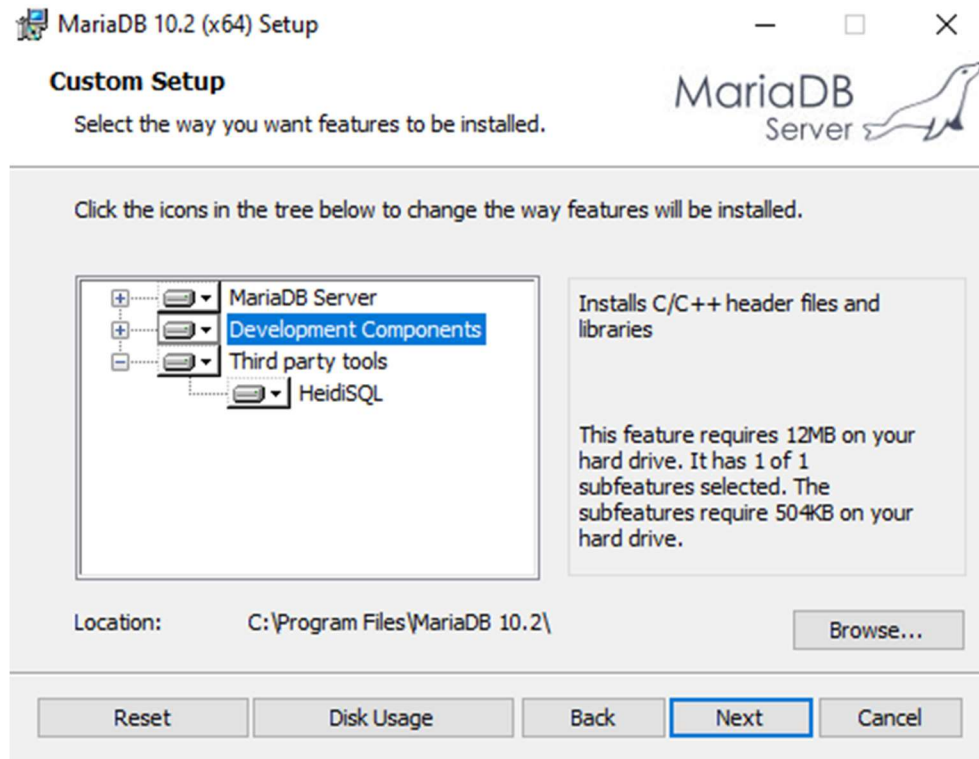
<https://dl.pstmn.io/download/latest/win64>

## Setup Guide

### *Development Database*

The web application relies heavily on a database, so a local development database will need to be created on your machine.

1. Download and install MariaDB 10.3.25. Keep the configuration settings at their default values. This will also install HeidiSQL.



2. Modify the root password to: "teamwethemboys"

User settings

Default instance properties

MariaDB 10.2 (x64) database configuration

MariaDB Server

☒ **Modify password for database user 'root'**

New root password:  Enter new root password

Confirm:  Retype the password

☐ **Enable access from remote machines for 'root' user**

☐ **Use UTF8 as default server's character set**

Back Next Cancel

- Continue through each remaining setup screen, keeping the default values. MariaDB and HeidiSQL should now be installed on your machine.
- Open HeidiSQL. If the default connection to localhost:3306 is not there, create a new connection to the database by selecting the 'New' button in the bottom left corner of the screen. Enter "teamwethemboys" as the password, then click the 'Open' button.

Session manager

Filter ...

Session name ^	Host
Unnamed	127.0.0.1

Settings Advanced Statistics

Network type: MySQL (TCP/IP)

Library: libmariadb.dll

Hostname / IP: 127.0.0.1

☐ Prompt for credentials

☐ Use Windows authentication

User: root

Password:

Port: 3306

☐ Compressed client/server protocol

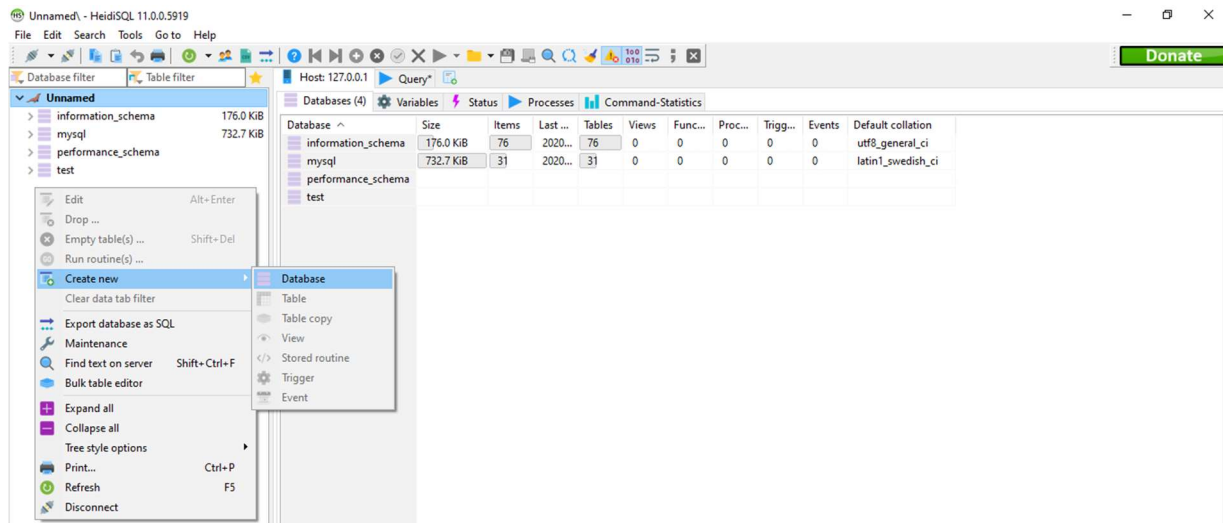
Databases: Separated by semicolon

Comment:

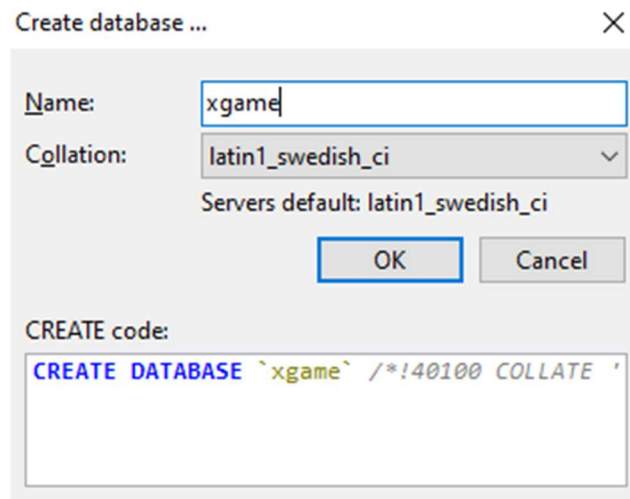
+ New Save Delete Open Cancel More



5. On the left pane, right-click and select Create new > Database.



6. Enter “xgame” as the name of the new database, then click “ok”. We have just created a development database. Spring Boot will initialize the schema once we run the Java project.



#### Integrated Development Environment

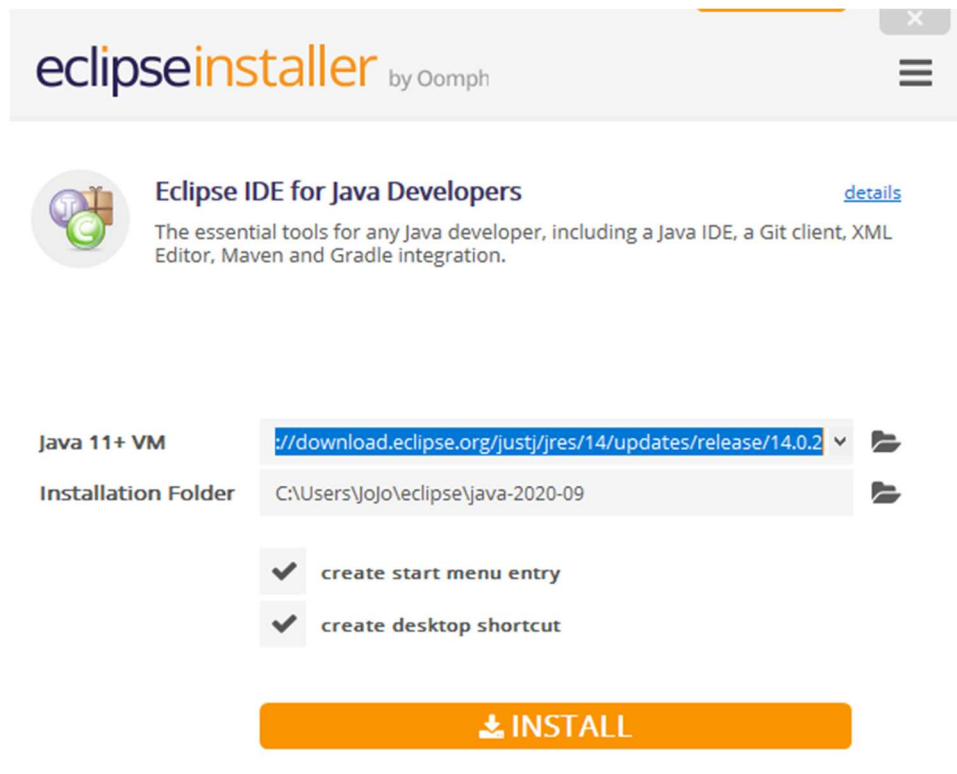
1. Download and install Eclipse using the following link:

[https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2020-09/R/eclipse-inst-jre-win64.exe&mirror\\_id=518](https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2020-09/R/eclipse-inst-jre-win64.exe&mirror_id=518)

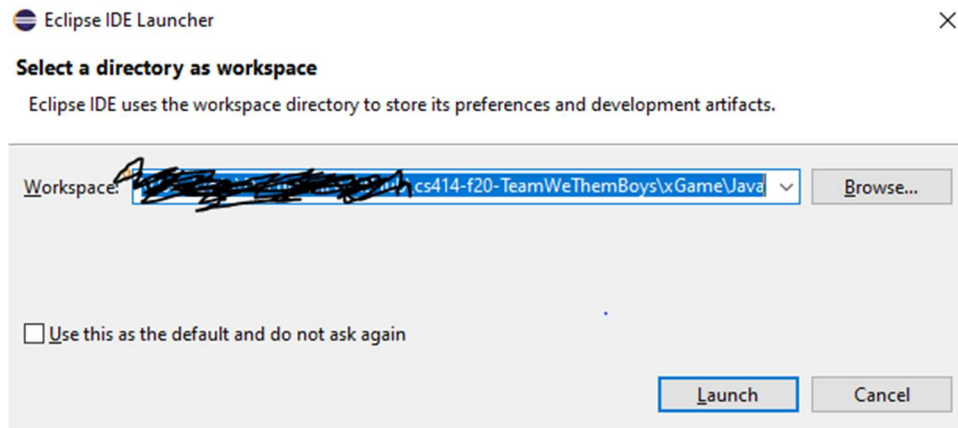
Run the Eclipse Installer. Select “Eclipse IDE for Java Developers”.



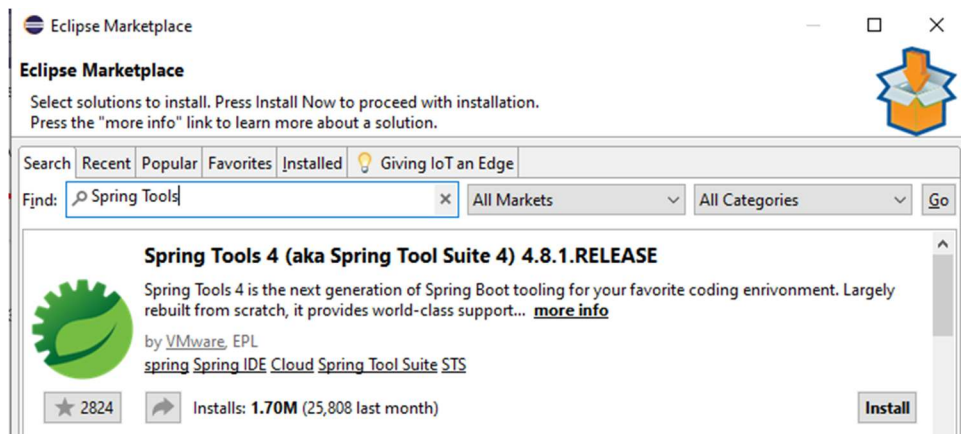
2. Keep the default install locations for the JVM and installation folder unless you have a good reason to put them somewhere else. Click 'Install' and accept the following agreement.



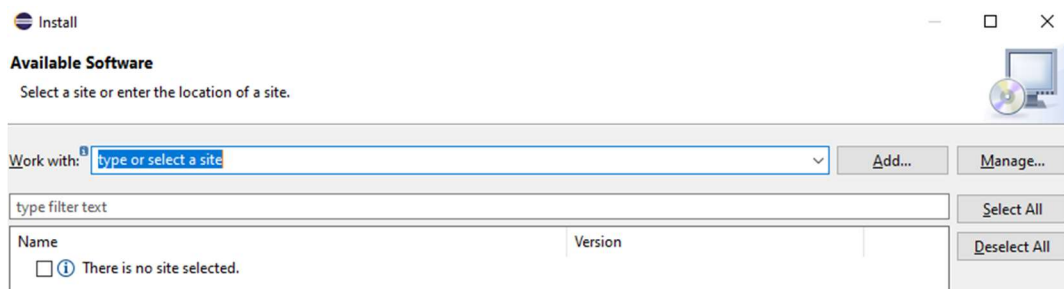
3. If you haven't already, use your favorite Git tool to pull down the repository code. Launch Eclipse and set the current workspace to the location of the code. Specifically target the "Java" folder of the repository. If eclipse does not automatically recognize the project, you may have to refresh the workspace, or select "File > Open Projects from File System..." and select the working directory.



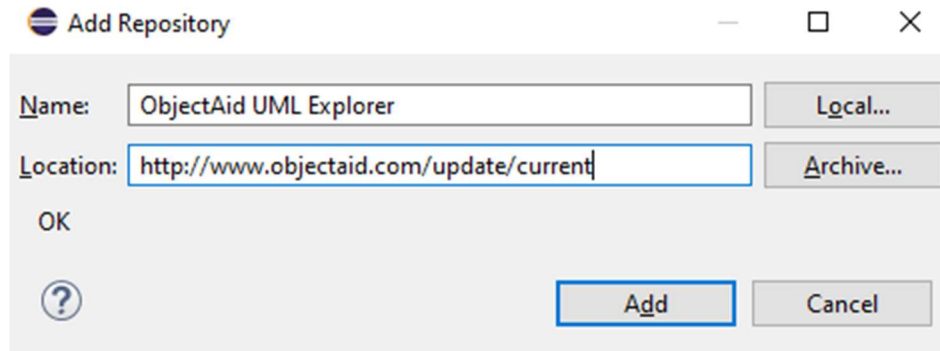
4. Now we need to install our extensions. The first will be Spring Tools 4. Select “Help > Marketplace”, then search “Spring Tools 4”. Click “Install”, install all features, and accept the terms of agreement. You will also have to restart Eclipse.



5. To install ObjectAid UML Explorer select “Help > Install New Software...”. Select “Add...” in the wizard.



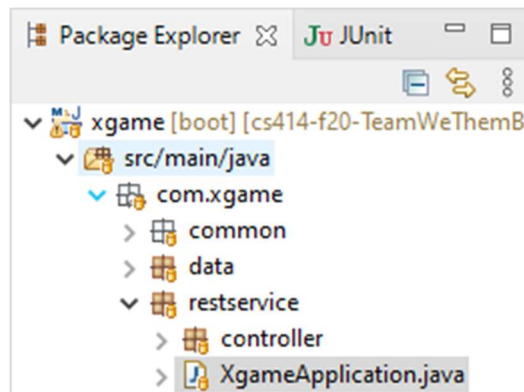
6. Enter the following as shown below, then click the “Add” button.



7. Select "ObjectAid Class Diagram". We don't need any of the other packages. Then click the "Next" button and agree to the terms. You may see a warning about untrusted add-ons. If you do, click "OK". You will have to restart Eclipse once installation is complete.

Name	Version
ObjectAid UML Explorer	
<input type="checkbox"/> Apache Batik (Required for Add-On)	1.6.0
<input checked="" type="checkbox"/> ObjectAid Class Diagram	1.2.4
<input type="checkbox"/> ObjectAid Diagram Add-On (License Required)	1.2.4
<input type="checkbox"/> ObjectAid Sequence Diagram (License Required)	1.2.4

8. In the Package Explorer, open "src/main/java/com.xgame.restservice.XgameApplication.java". This contains the main method for the application. Run it.



9. In the Console, you should see something like below. The final message "Started XgameApplication..." means that the web application is now running locally on port 8080.



```
2020-10-29 15:40:39.840 INFO 11244 --- [
2020-10-29 15:40:39.843 INFO 11244 --- [
2020-10-29 15:40:40.428 INFO 11244 --- [
2020-10-29 15:40:40.505 INFO 11244 --- [
2020-10-29 15:40:41.218 INFO 11244 --- [
2020-10-29 15:40:41.229 INFO 11244 --- [
2020-10-29 15:40:41.230 INFO 11244 --- [
2020-10-29 15:40:41.375 INFO 11244 --- [
2020-10-29 15:40:41.376 INFO 11244 --- [
2020-10-29 15:40:41.631 INFO 11244 --- [
2020-10-29 15:40:41.696 INFO 11244 --- [
2020-10-29 15:40:41.885 INFO 11244 --- [
2020-10-29 15:40:42.014 INFO 11244 --- [
2020-10-29 15:40:42.271 INFO 11244 --- [
2020-10-29 15:40:42.305 INFO 11244 --- [
2020-10-29 15:40:43.593 INFO 11244 --- [
2020-10-29 15:40:43.600 INFO 11244 --- [
2020-10-29 15:40:44.087 WARN 11244 --- [
2020-10-29 15:40:44.222 INFO 11244 --- [
2020-10-29 15:40:44.469 INFO 11244 --- [
2020-10-29 15:40:44.478 INFO 11244 --- [
```

```
main] com.xgame.restservice.XgameApplication : Starting XgameApplication on JoJo-PC with PID 11244 (C:\Users\JoJo\Docu
main] com.xgame.restservice.XgameApplication : No active profile set, falling back to default profiles: default
main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 67ms. Found 4 JPA reposito
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.38]
main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1470 ms
main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
main] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.4.21.Final
main] o.hibernate.annotations.common.Version : MCANN000001: Hibernate Commons Annotations {5.1.0.Final}
main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.MySQL55Dialect
main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.trai
main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database quer
main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
main] com.xgame.restservice.XgameApplication : Started XgameApplication in 5.052 seconds (JVM running for 6.028)
```

### Checking Database Schema Creation

Running the Java project should have created the schema in the xgame database. To confirm:

1. Open up HeidiSQL and reconnect to 127.0.0.1.
2. Select the xgame database
3. You should now see a list of tables in the database. Feel free to explore the database.

### Testing the Environment

1. To test the application properly, ensure that Postman is installed.
2. Fire up Postman and create a POST request with the url "localhost:8080/test" and set some value for the parameter "content". If you get a 200 OK response, then the API call was successful and a new test message has been saved to the database.

POST http://localhost:8080/test?content=this is a test Send

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

Query Params

KEY	VALUE	DESCRIPTION
content	this is a test	
Key	Value	Description

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 245 ms Size: 168 B Save

Pretty Raw Preview Visualize JSON

1 "OK"

3. Now try a GET request on the same URL with no parameters to retrieve all of the test messages from the database. You should see the test message you just created in the list of messages.

GET http://localhost:8080/tests Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results Status: 200 OK Time: 150 ms Size: 201 B Save

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 1,
4     "content": "this is a test"
5   }
6 ]
```

4. Since we got a message back, we know that the application can successfully receive HTTP requests and communicate with our local database. To confirm there is a message in the database, check back in with HeidiSQL.

Unnamed\xgame\test\ - HeidiSQL 11.0.0.5919

File Edit Search Tools Go to Help

Database filter Table filter Host: 127.0.0.1 Database: xgame Table: test Data Query\*

Database filter

- Unnamed
  - information\_schema 176.0 KiB
  - mysql
  - performance\_schema
  - test
  - xgame 176.0 KiB
    - chess\_match 64.0 KiB
    - hibernate\_sequence 16.0 KiB
    - message 32.0 KiB
    - test 16.0 KiB
    - user 48.0 KiB

xgame.test: 1 rows total (approximately)

id	content
1	this is a test