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 user 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

Deactivate User

Deactivates a user. User will no longer appear in searches.

POST /user/deactivate?id={int}

Get User Profile

Gets a user's match history and profile information.

GET /user/profile?id={int}

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}

Reject a match invitation

Rejects an invaitation to a match. PlayerId must match the id of the black player of the match. Match status changes to "REJECTED".

PATCH /match/reject?matchId={int}&playerId={int}

Propose or accept a draw

Allows one player to propose a draw and another player to accept the draw. If both players accept the draw, then the match ends and match status changes to "COMPLETED" and the outcome to "DRAW".

PATCH /match/draw?matchId={int}&playerId={int}

Deny a draw request

Denies a draw request.

PATCH /match/draw/deny?matchId={int}&playerId={int}

Get all ongoing matches

Gets all ongoing matches for a player.

GET /matches?playerId={int}

Forfeit a match

Forfeits a match. Changes match status to “COMPLETED” and outcome to “FORFEIT”. Sets winning player to other player.

PATCH /match/forfeit?matchId={int}&playerId={int}

Message

The message API is responsible for sending messages to users and reading a user’s messages. Messages include match invitations, in addition to system messages.

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. Gets a list of Messages.

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/invite?playerId={int}

Send a message

Sends a message to a player.

POST /message?playerId={int}&contents={string}

Read a message

Marks a single message as read.

PATCH /message?messageId={int}

Read all messages for a user

Marks all messages as read for a single user.

PATCH /messages?playerId={int}

Game

The game API is necessary for playing a match and interacting with the chess engine.

Endpoints

Move a piece

Moves a piece on the board for the given game. Checks that it is the correct player's turn and that the move is a valid move based on the state of the board. Returns a Match.

PATCH /game/move?matchId={int}&toPosition={string}&fromPosition={string}

Get legal moves

Gets all the legal moves for a piece in the given game at the given position.

GET /game/legalMoves?matchId={int}&piecePosition={string}

Promote a pawn

Promotes a pawn in the given game at the given position to the given piece. Returns a Match.

PATCH /game/promote?matchId={int}&position={string}&piece={string}

Objects

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

Match: {
    "id": int,
    "whitePlayerId": int,
    "blackPlayerId": int,
    "whitePlayerNickname": string,
    "blackPlayerNickname": string,
    "winningPlayerId": int,
    "winningPlayerNickname": string,
    "turnCount": int,
    "status": int,
    "chessBoard": string
}

User: {

    "id": int,
    "nickname": string,
    "email": string
}

UserCredentials: {

    "nickname": string,
    "email": string,

"password": string
}

Profile: {
"userNickname": string,
"matchHistory": [
"id": int,
"startTime": timestamp,
"endTime": timestamp,
"opponentNickname": string,
"color": int,
"outcome": int,
"moveCount": int,
"winninPlayer": 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

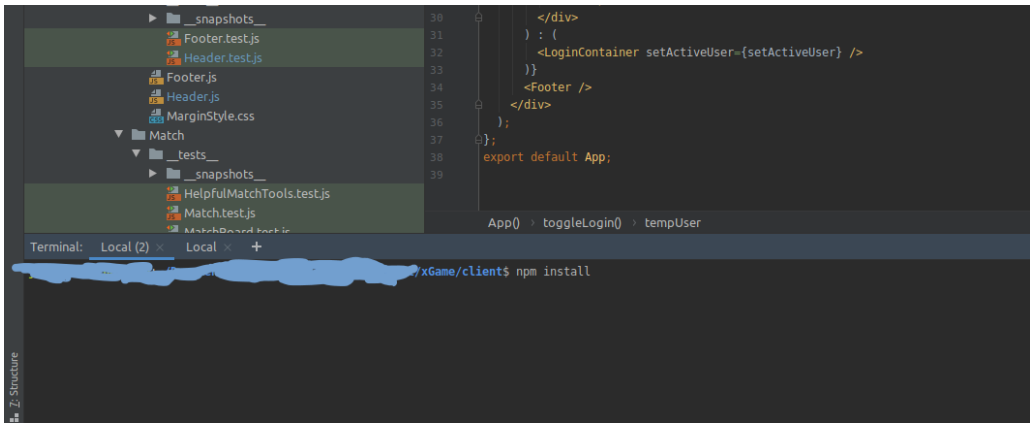
Node.JS if the engine that will allow us to run the JavaScript outside the browser

NPM

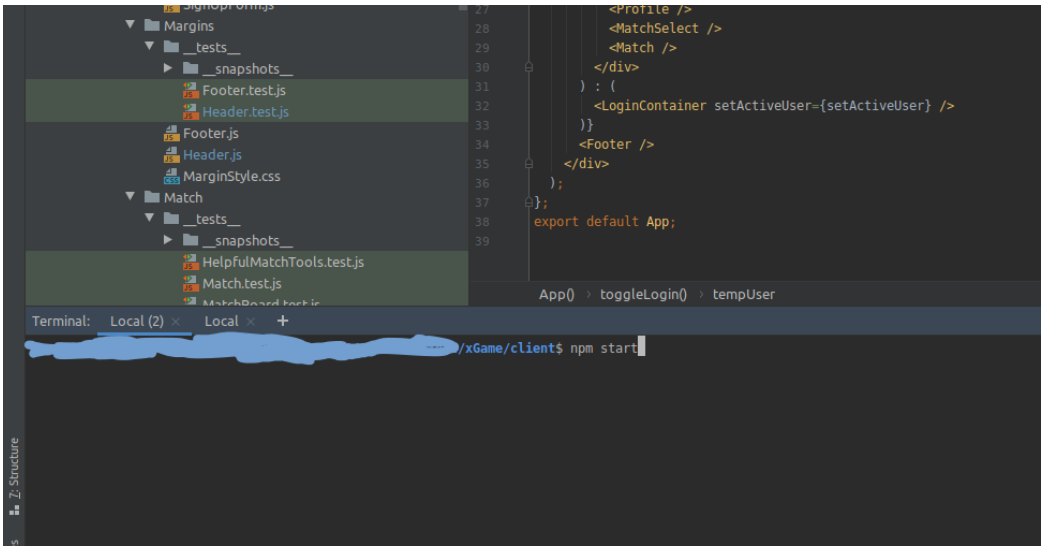
NPM is a node package manager that allows us to install packages to use in our code.

Setup Guide

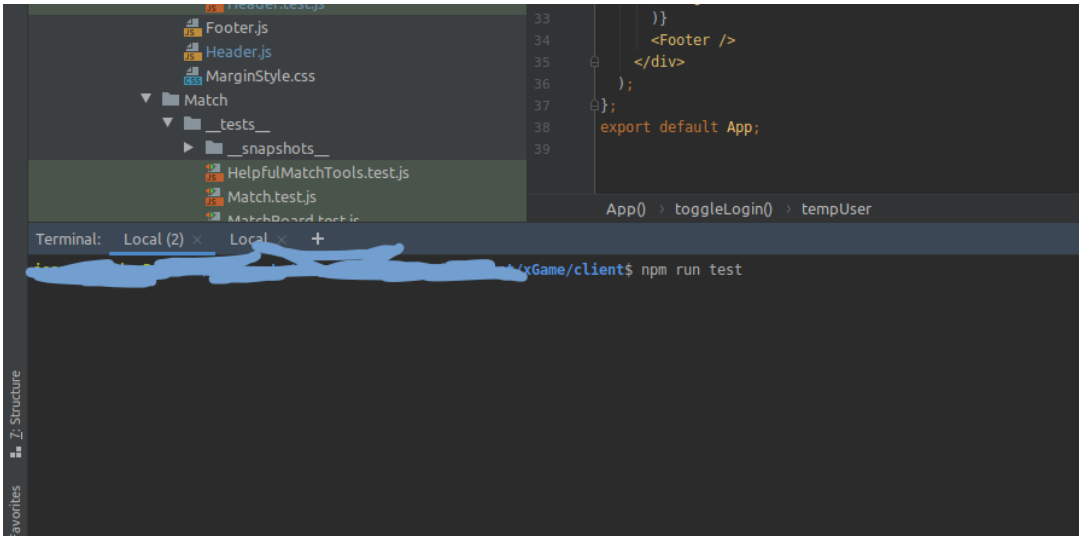
1. Install Intellij IDEA
    a. Follow the steps in the wizard, keeping all settings to default
2. Install Node.JS
3. Install NPM.
4. Open Intellij and create a new project from existing version control.
5. In the Intellij terminal, navigate to the client directory.
6. Run 'npm install' to automatically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

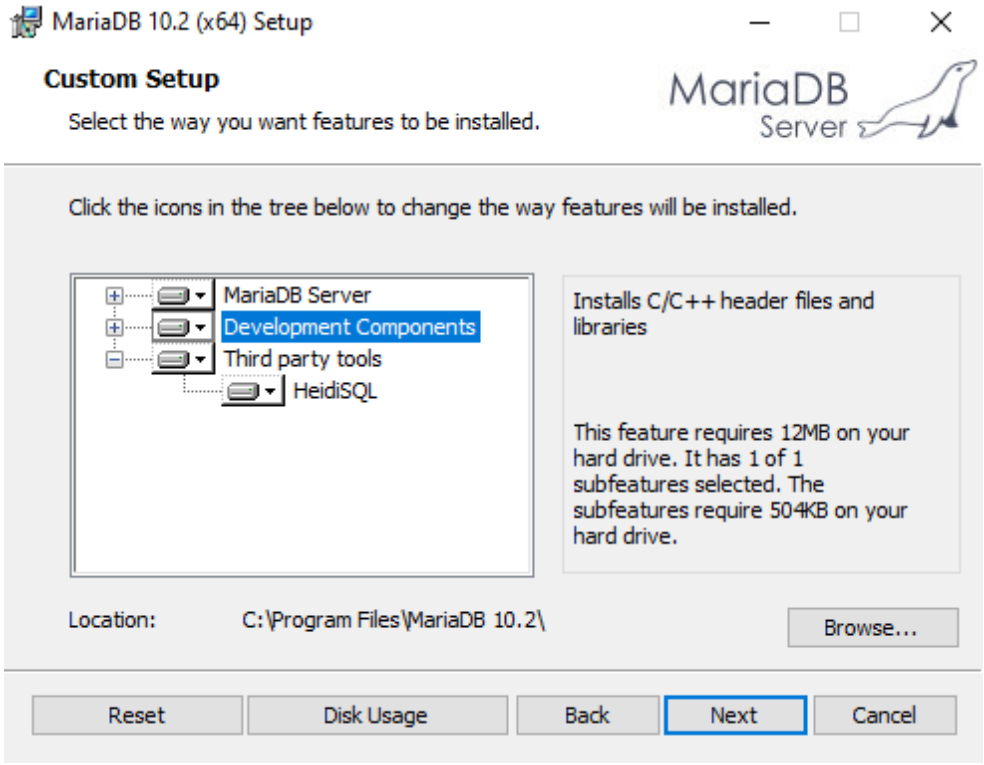
https://dl.pstmn.io/download/latest/win64

Setup Guide

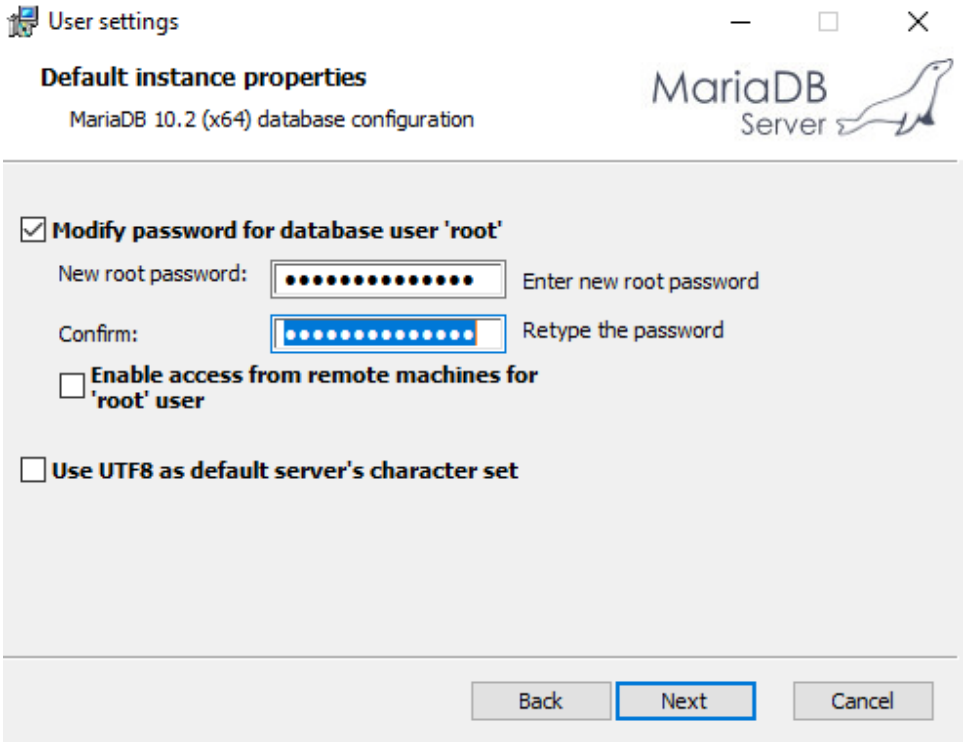
Development Database

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

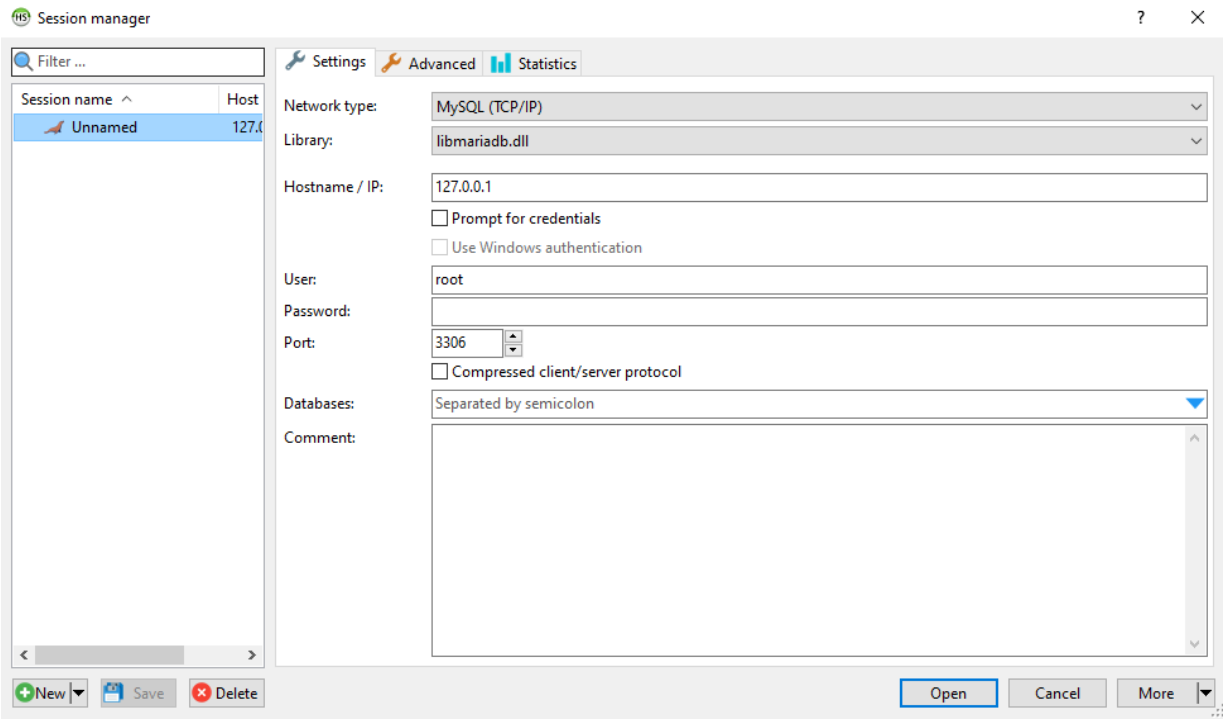
1. Download and install MariaDB 10.3.25. Keep the configuration settings at their default values. This will also install HeidiSQL.



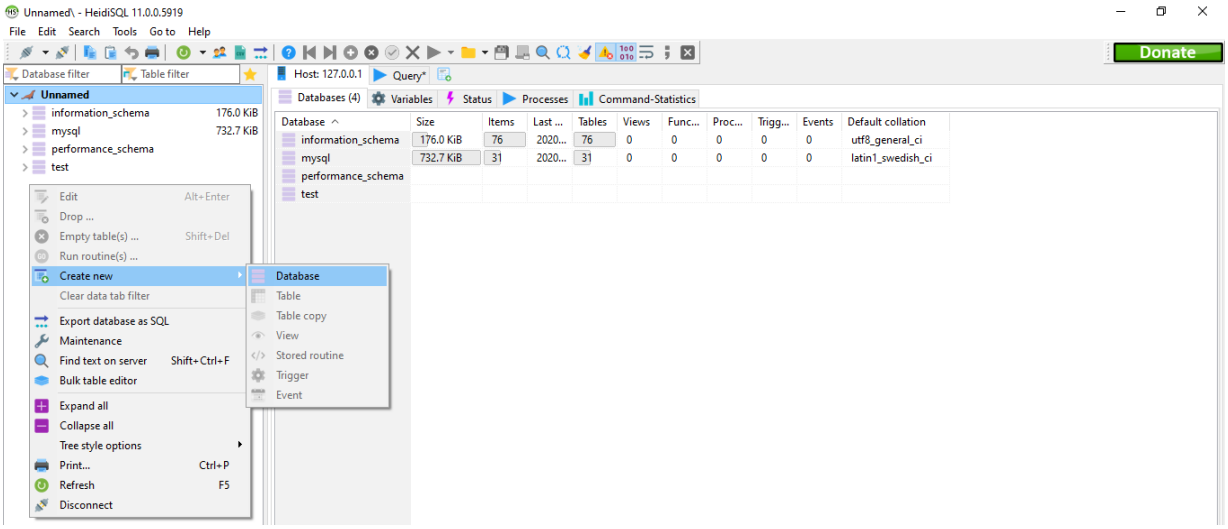
2. Modify the root password to: “teamwethemboys”



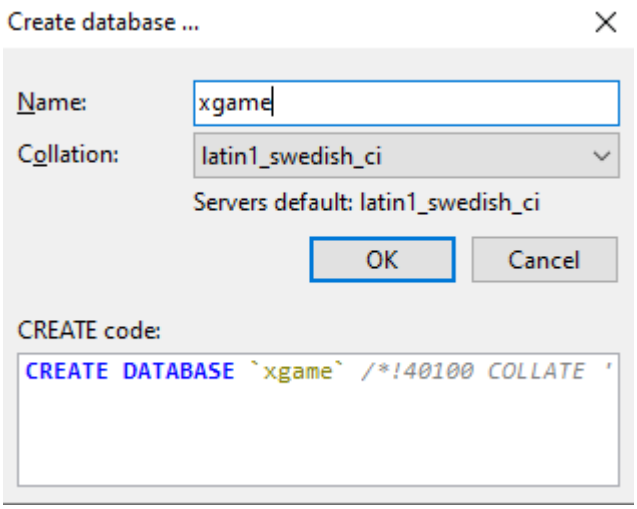
3. Continue through each remaining setup screen, keeping the default values. MariaDB and HeidiSQL should now be installed on your machine.
4. Open HeidiSQL. If the default connection to localhost:3306 is not there, create a new connection to the database by selecting the ‘New’ button in the bottom left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



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.



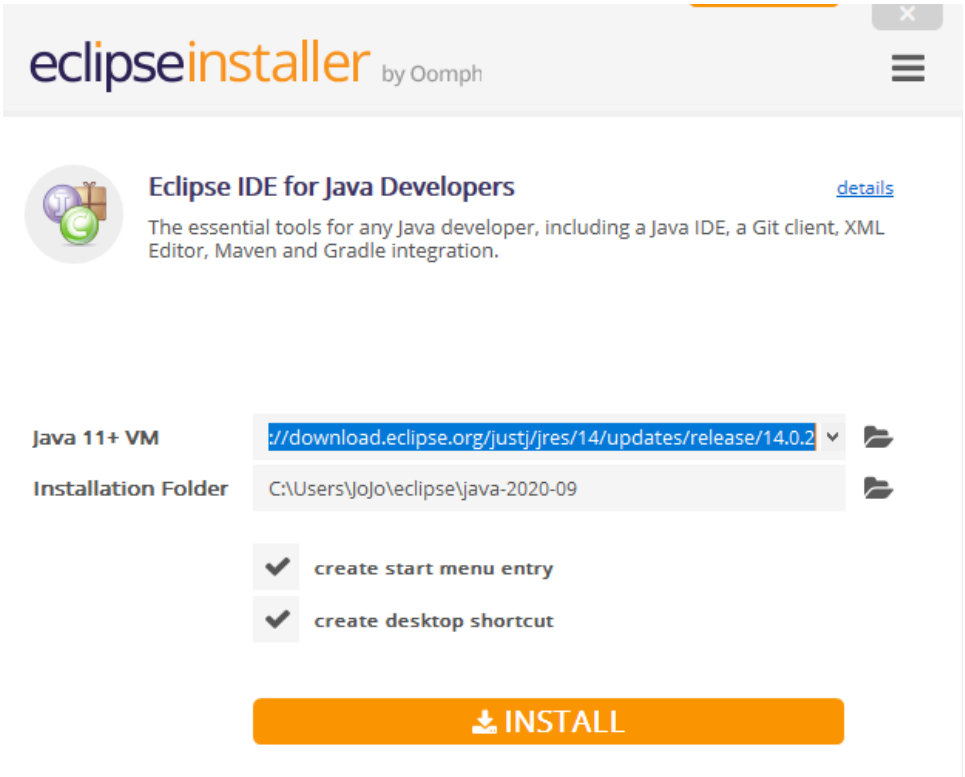
Integrated Development Environment

1. Download and install Eclipse using the following link:

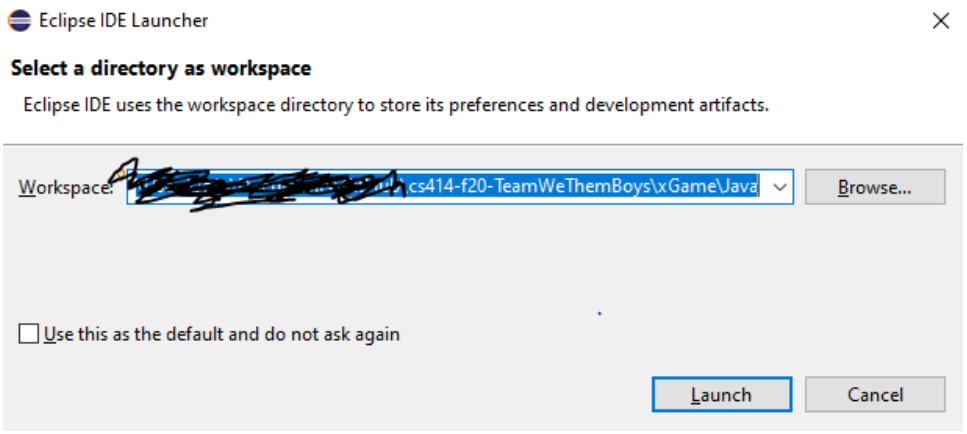
https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2020-09/R/eclipse-inst-jre-win64.exe&mirror_id=518

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

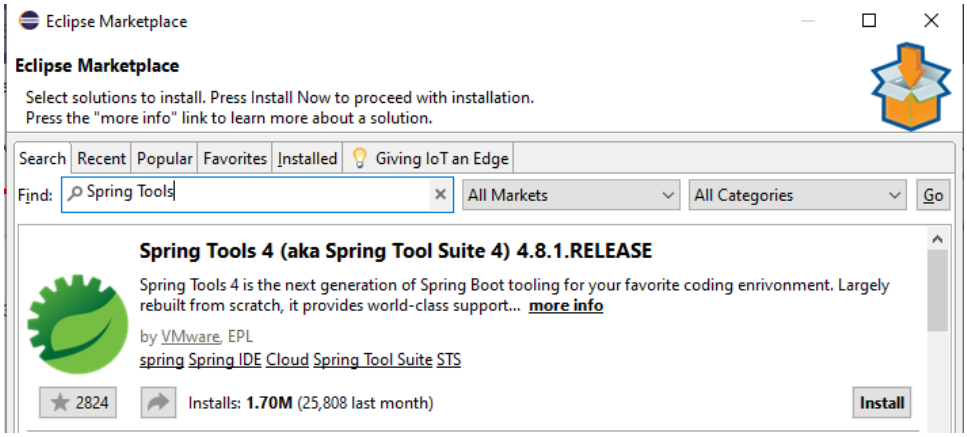
2. Keep the default install locations for the JVM and installation folder unless you have a good reason to put them somewhere else. Click 'Install' and accept the following agreement.



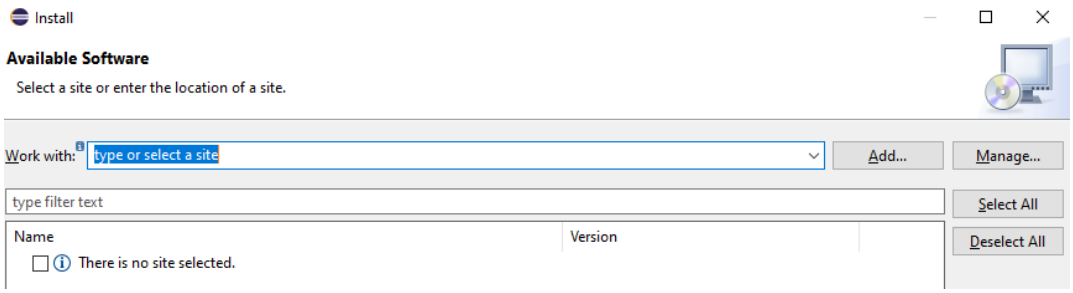
3. If you haven't already, use your favorite Git tool to pull down the repository code. Launch Eclipse and set the current workspace to the location of the code. Specifically target the "Java" folder of the repository. If eclipse does not automatically recognize the project, you may have to refresh the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



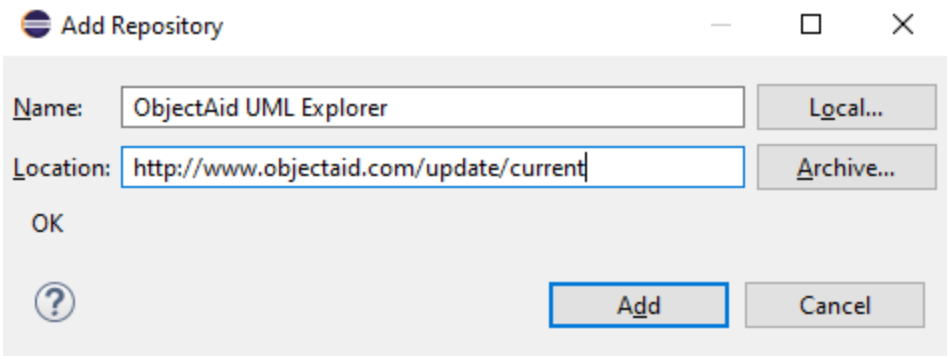
4. Now we need to install our extensions. The first will be Spring Tools 4. Select “Help > Marketplace”, then search “Spring Tools 4”. Click “Install”, install all features, and accept the terms of agreement. You will also have to restart Eclipse.



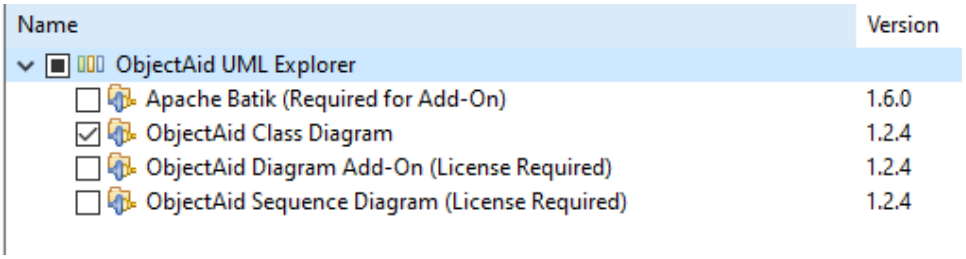
5. To install ObjectAid UML Explorer select “Help > Install New Software...”. Select “Add...” in the wizard.



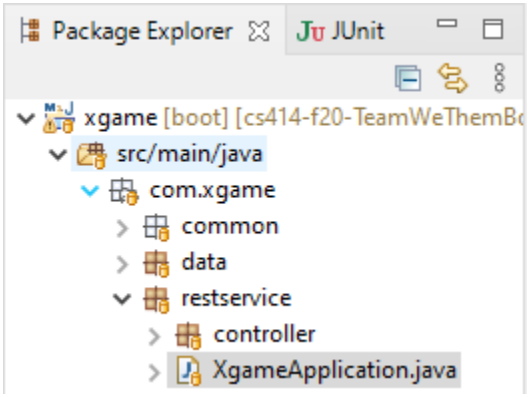
6. Enter the following as shown below, then click the “Add” button.



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.



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.

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.4.RELEASE)

2020-10-29 15:40:39.840  INFO 11244 --- [           main] com.xgame.restservice.XgameApplication   : Starting XgameApplication on JoJo-PC with PID 11244 (C:\Users\JoJo\Doc
2020-10-29 15:40:39.843  INFO 11244 --- [           main] com.xgame.restservice.XgameApplication   : No active profile set, falling back to default profiles: default
2020-10-29 15:40:40.428  INFO 11244 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2020-10-29 15:40:40.505  INFO 11244 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 67ms. Found 4 JPA repositor
2020-10-29 15:40:41.218  INFO 11244 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2020-10-29 15:40:41.229  INFO 11244 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2020-10-29 15:40:41.230  INFO 11244 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.38]
2020-10-29 15:40:41.375  INFO 11244 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2020-10-29 15:40:41.376  INFO 11244 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1470 ms
2020-10-29 15:40:41.631  INFO 11244 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2020-10-29 15:40:41.696  INFO 11244 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.4.21.Final
2020-10-29 15:40:41.885  INFO 11244 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.0.Final}
2020-10-29 15:40:42.014  INFO 11244 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2020-10-29 15:40:42.271  INFO 11244 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2020-10-29 15:40:42.305  INFO 11244 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.MySQL55Dialect
2020-10-29 15:40:43.593  INFO 11244 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.tra
2020-10-29 15:40:43.600  INFO 11244 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2020-10-29 15:40:44.087  WARN 11244 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database que
2020-10-29 15:40:44.222  INFO 11244 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2020-10-29 15:40:44.469  INFO 11244 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2020-10-29 15:40:44.478  INFO 11244 --- [           main] com.xgame.restservice.XgameApplication   : Started XgameApplication in 5.052 seconds (JVM running for 6.028)
```

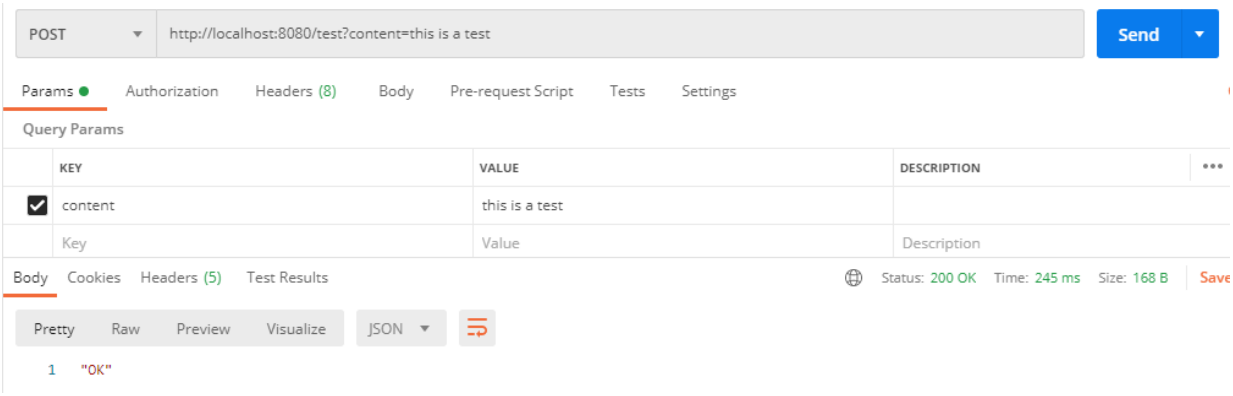
Checking Database Schema Creation

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

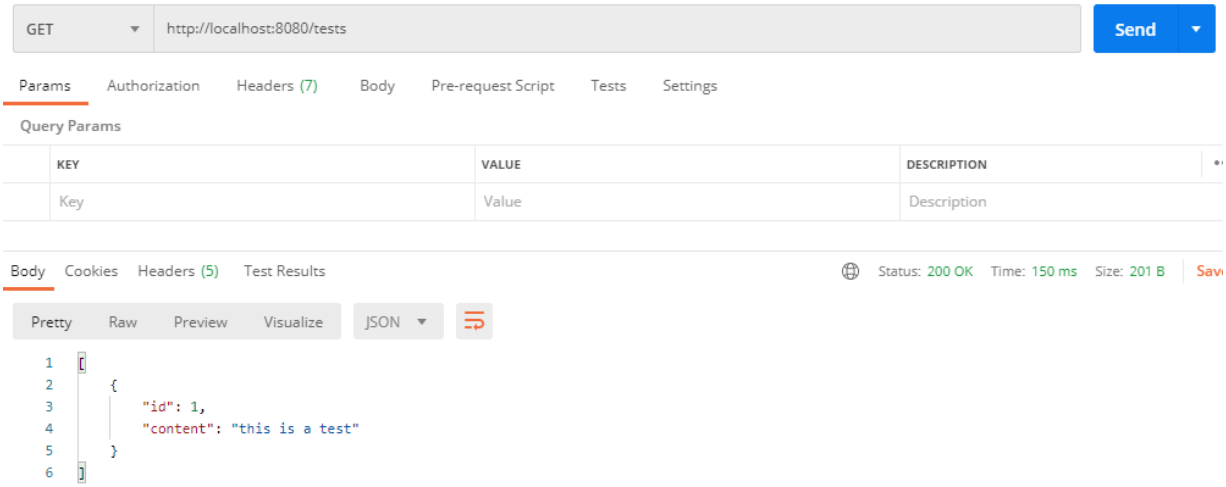
1. Open up HeidiSQL and reconnect to 127.0.0.1.
2. Select the xgame database
3. You should now see a list of tables in the database. Feel free to explore the database.

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.



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.



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.