

## **Development Manual Strike Team Java**

### **For both Eclipse and Intelli J**

This project development requires the use of a database server that is on the CSU Computer Science network. In order to build/run you must first SSH into a CS department machine that will open and listen to a specific port to run any MySQL commands.

First open a terminal/command prompt and SSH to a CSU CS department machine

```
ssh -L 64148:faure.cs.colostate.edu:3306 eid@csmachine.cs.colostate.edu
```

- 64148 is the port that will be opened to send/listen for MySQL commands
- Eid is what you login with onto the cs department machines
- CS machine is any of the cs department machines available  
<https://www.cs.colostate.edu/~info/machines>

After you have successfully logged in the port is now open that we can use to connect to the database in our code

Clone the repo from

<https://github.com/mdcham/cs414-f20-Strike-Team-Java.git>

## For Eclipse

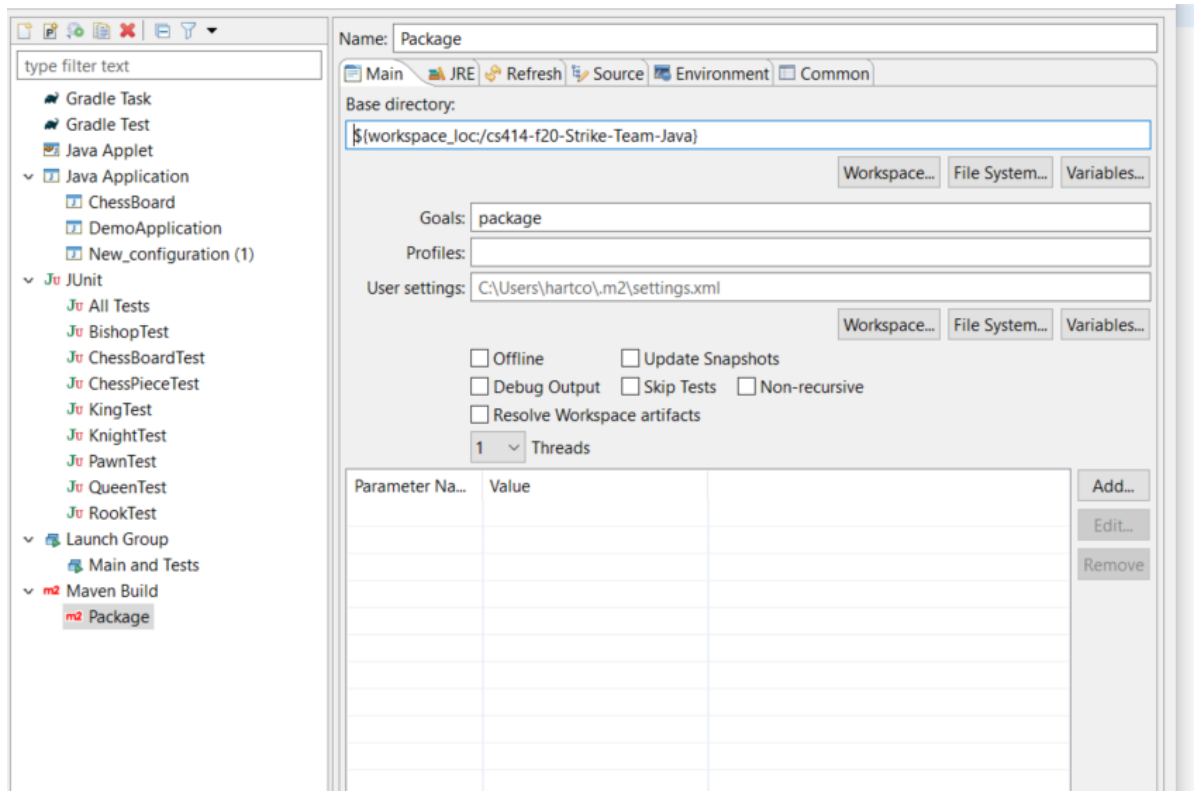
Cloning the repo follow the guide here [https://www.youtube.com/watch?v=uOpug3ie\\_h0](https://www.youtube.com/watch?v=uOpug3ie_h0)

He shows and explains how to do it better than we ever could, he also explains bringing in a git repository into a project that eclipse can then run.

Right click pom.xml run as maven clean

Right click pom.xml run as -> run configurations -> bottom left select maven build -> then on the top left select new launch configuration

Click workspace and choose the current project and in goals type package, hit apply and run and it should build. Mine looked like this



If it complains about react-scripts not being an internal or external command you will have to delete the node\_modules folder(this will take a minute) and the package-lock.json. Then run the package build again.

After this has been made you can the create a Java Application. Choose the correct workspace and have the DemoApplication selected as the main class.

Running this will start up the server. You can then navigate to the link below to access the application.

<http://localhost:8080>

## **For IntelliJ J**

Cloning the repo follow this guide

<https://www.jetbrains.com/help/idea/set-up-a-git-repository.html>

Intelli J requires installing Node.js on your system to build and install using NPM

<https://nodejs.org/en/download/>

In demo folder right click pom.xml add as Maven Project (Near the bottom of the list)

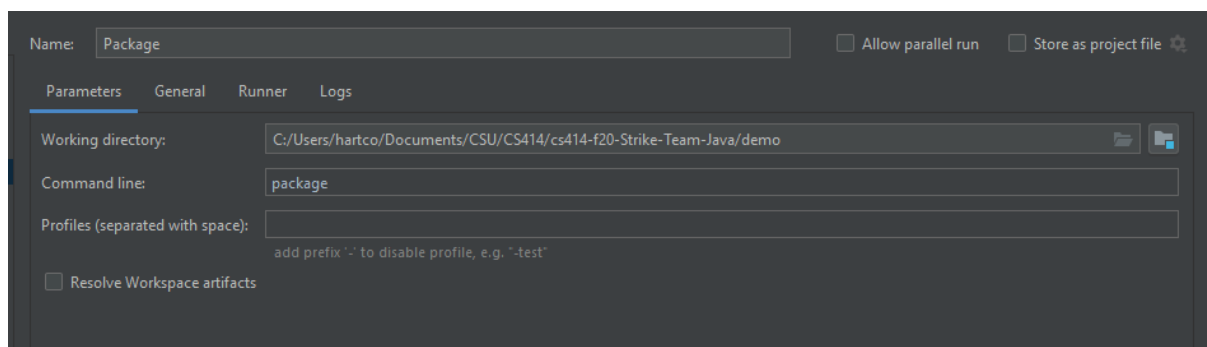
It will take a few minutes to resolve the dependencies and download plugins, after that has finished right click the pom.xml again and go to Maven then select "Generate Sources and Update Folders.

Ideally now the project should be imported as a Maven project and building it should be possible.

For this go to the top tab Run -> Edit configurations

Top left of this popup click the "+" sign to create a new config and choose Maven from this list

In this new screen you can name it whatever you would like but make sure the Working Directory is in the /demo folder and the command line contains package. Like this



After making this in the top right make sure this newly created config is select and click the green play button to start building the package. First time running this will take a few minutes to install and package all the dependencies.

After this has finished go back to run -> edit configurations -> click the "+" again but this time choose application. Again name it whatever you would like choose the main class that is in demo.Demoapplication. Make sure the classpath is set to demo and that a JRE is selected then click ok. Then in the top right select this newly made application cofig and run this. This will start up the server.

You can then navigate to <http://localhost:8080> to see the application running

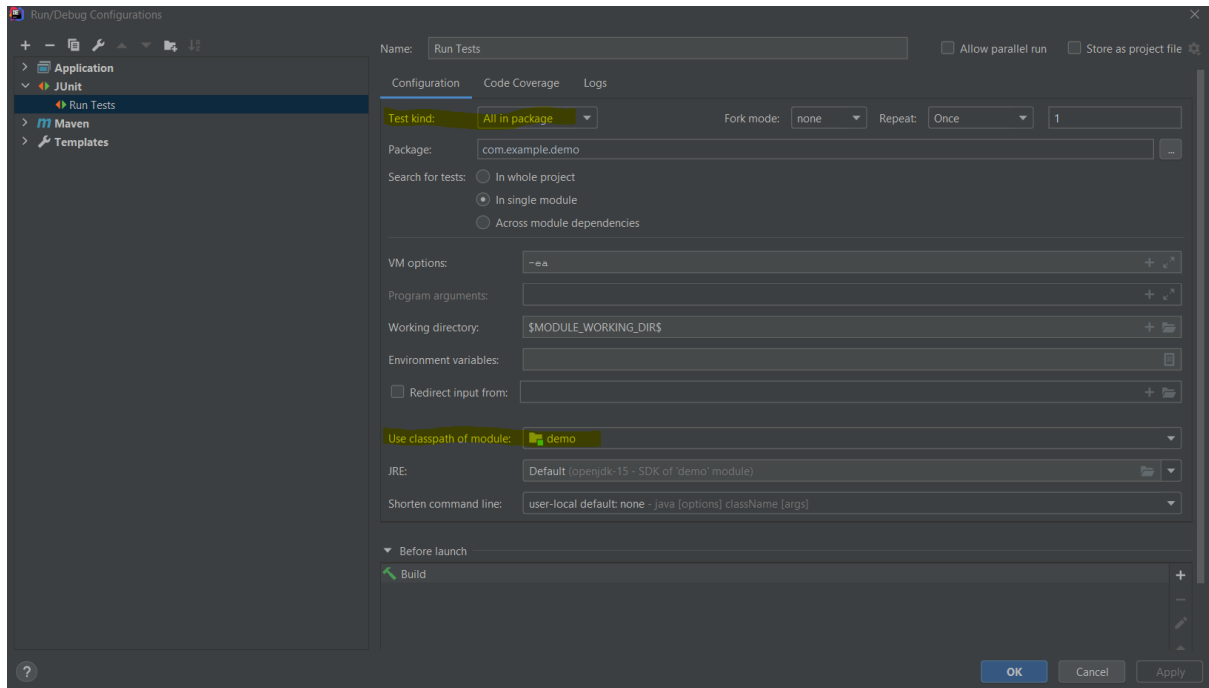
## For Running Tests

Maven runs all the JUnit tests we have setup during the packaging of the project.

You can also manually run them by creating a run config

### Intelli J

Top right select edit run configs, top left of popup select the “+” to create a new config. Choose JUnit and in this window give it a name, in “Use classpath of module” select demo, then in “Test kind” select All in package to have it select all tests. Then hit ok and run this new config for our tests.

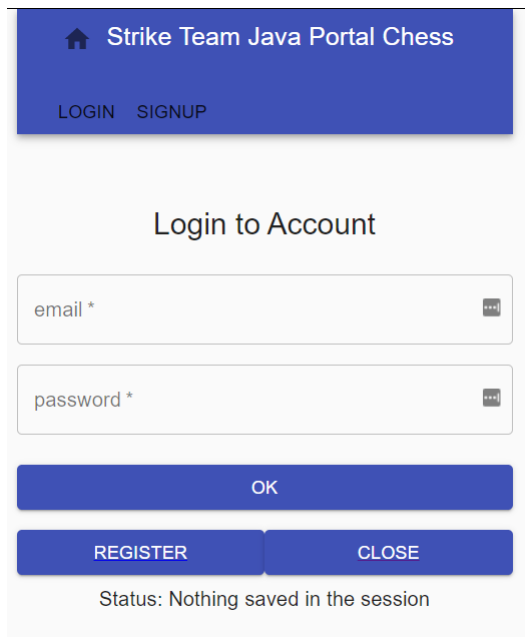


### Eclipse

In the project folder go to demo/src/test/java right click com.example.demo and Run as... > JUnit Test. This will run all tests in this directory as JUnit.

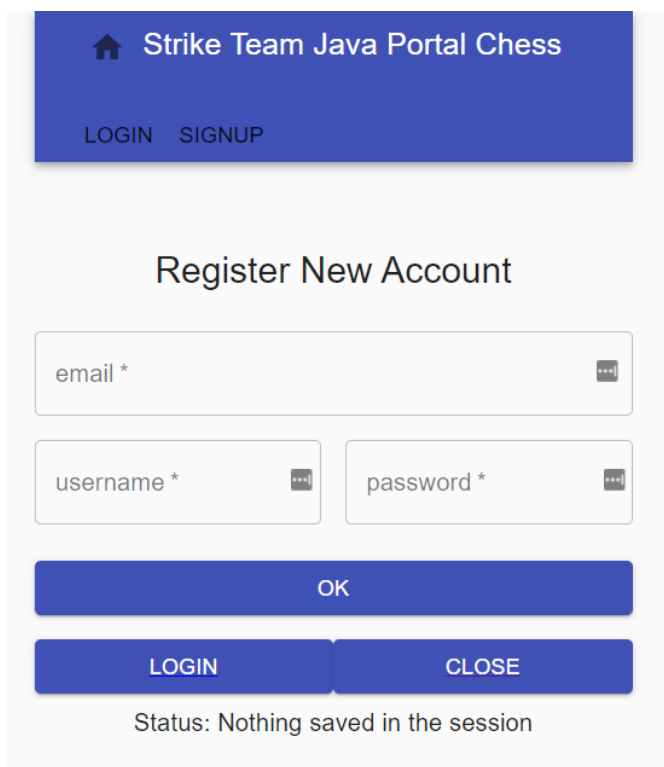
## Navigating the Application

Going to <http://localhost:8080> will bring you to the homepage



The screenshot shows the 'Login to Account' page of the 'Strike Team Java Portal Chess' application. At the top, there is a blue header bar with a home icon, the title 'Strike Team Java Portal Chess', and links for 'LOGIN' and 'SIGNUP'. Below the header, the page title 'Login to Account' is centered. There are two input fields: 'email \*' and 'password \*', both with password visibility toggles. Below these fields is a large blue 'OK' button. At the bottom, there are two smaller blue buttons: 'REGISTER' and 'CLOSE'. A status message at the very bottom reads 'Status: Nothing saved in the session'.

From here going to Register will bring you to the registration page



The screenshot shows the 'Register New Account' page of the 'Strike Team Java Portal Chess' application. It has the same blue header bar as the login page. The page title 'Register New Account' is centered. There are three input fields: 'email \*', 'username \*', and 'password \*', each with a password visibility toggle. Below these fields is a large blue 'OK' button. At the bottom, there are two smaller blue buttons: 'LOGIN' and 'CLOSE'. A status message at the very bottom reads 'Status: Nothing saved in the session'.

Where users can register their own account in the system. We have added some simple error checking to make sure the user doesn't give the application invalid info.

If the email address given doesn't match the form of [string@string.string](#), it returns the status

Status: The email address entered is not valid

If the username is left blank it returns

Status: The username entered is not valid

If the password is left blank or less than 4 characters long then it returns

Status: The password entered is not valid

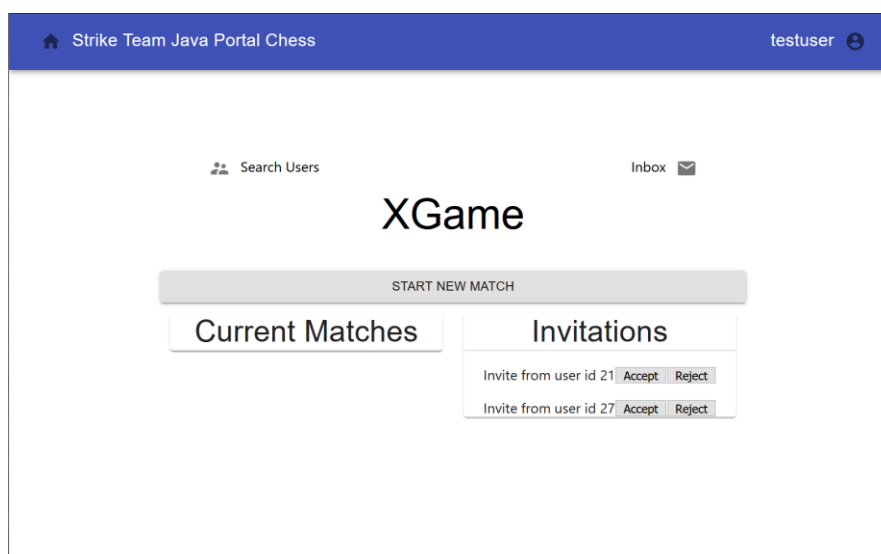
After giving valid info and hitting ok it returns

Status: Account created successfully

From here going back to the login screen we also implemented some error checking. If the email address given isn't in the valid form it gives the same error as above. If the email address is not in the system with the same password given it then informs the user of this.

Status: No account with matching credentials was found.

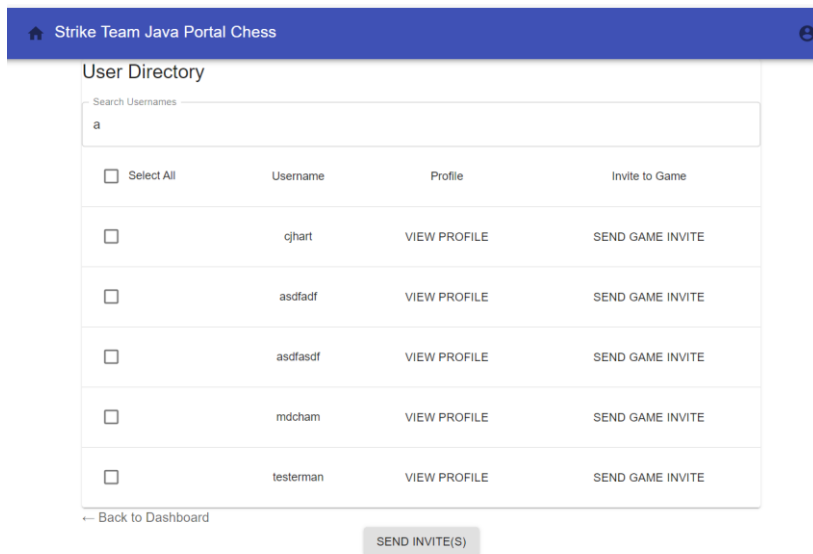
After logging in you are then redirected to the dashboard page. This acts as the main page for the user where they can navigate to other pages, search for other users, see the list of invitations from other users, and see the current list of active matches they are apart of.



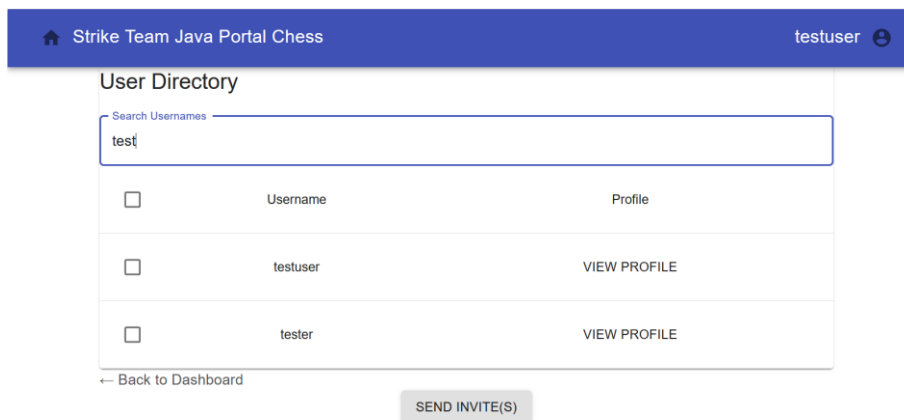
The home icon in the top left will bring the user to the login page

The person icon in the top right will allow the users navigate back to the dashboard page or logout.

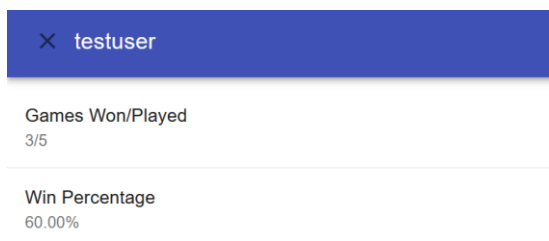
The start a new match button will navigate to the invite page where users can select other player(s) to invite to a match. The invite functionality is not completed yet but will be implemented in P3.



Search users in the top left will bring you to a page with a search bar, where the user can then search for other registered users.



From here selecting view profile will show this user's stats



We currently do not have the ability to launch a game yet but in the future accepting a game invite from a user would add that game into the current matches list and launch the game. That way they can participate in the game they just accepted the invite from. As well as leave the current game and come back to it later. We would also like to create an inbox where users can see if someone has accepted/declined their game invite and potentially send messages back and forth.



## Development Files Description

In demo/src/main/java/com/example/demo contains all the Server Java files

### Account

- Responsible for creating an Account object, has several methods for setting and getting all the components of this object

### AccountController

Responsible for receiving requests from the client

- Path /createAccount will check information passed from client for creating an account. It validates the information given and if it passes all checks it will then pass this information to accountService.
- Path /account is a request to get all the accounts that are stored in the database. Immediately sends this request to accountService.
- Path /getInviteList/{accountID} is responsible for receiving an account id and passing this to account service to get the list of invites for this specific user.
- Path /login receives the email address and password from the client, validates the information is valid, and then sends it to accountService.

### AccountService

Is an Interface file that is extended by AccountServiceImpl

### AccountServiceImpl

Receives information from AccountController and passes everything onto AccountDAO

### AccountDAO

Is an Interface file that is extended by AccountDAOImplementation

### AccountDAOImplementation

Receives information from AccountService and interacts with the database to retrieve information, or validate information given to it. Then sends a response back to AccountController with either of these.

- List<Account> get creates a list of accounts that it gets from the database
- Account get(int id) returns an account object that matches this id from the database
- List<Notification> getNotificationList(Integer accountID) gets the list of notifications from the account with the ID it was given
- Account get(String username) gets an account from the database that matches the given username
- Boolean checkAccount(Account account) checks the database to see if an account exists with the email address given. Returns true if it does, false if not
- Boolean checkUser(Account account) checks the database to see if an account with this username exists. Returns true if it does, false if not
- Account getAccount(account) gets the entire account object from database that matches the email and password given.

### CustomGlobalExceptionHandler

Responsible for extending `ReponseEntityExceptionHandler`. Receives Exceptions and packages them into a HTTP status with a message that was given. It then sends this response back to the client

### InvalidRequest

Receives `RunTimeExceptions` with a String message attached to it

### Notification

Responsible for creating a Notification object, has several methods for setting and getting all the components of this object

### ValidateAccount

Responsible for validating the information that is passed with creating an account or logging in.

- `ValidateEmail` takes the email address that was given and makes sure its in the form of an email address. [String@String.String](#)
- `ValidateUsername` makes sure the username isn't an empty string or null value
- `ValidatePassword` makes sure password isn't an empty string or null and at least 4 characters long.

In `demo/frontend/src` contains files for the React javascript Client

### App.js

The backbone of the client that holds and sets the state of the logged in user (if they have done so)

Holds the Router Switch to be able to switch between pages

`/components` holds the components for each page

### UserLogin.js

Responsible for creating a login page with a username and password field. Hitting submit sends a request to the server with this info, then waits for a response. If they successfully login it then has `App.js` set this state and navigates to the Dashboard page. Displays error on why they couldn't login otherwise.

### Adduser.js

Creates an account creation page that has the user input an email address, username, and password. Hitting submit sends a request to the server to create an account. Displays successful if it was made, otherwise shows an error stating why if they did not successfully create an account.

### AccountMenu.js

This renders an icon button that contains the ability to logout. It may have more features later.

### Dashboard.js

This is responsible for loading the main page after a user has logged into an account. This is the central hub where players can see their account information, lookup other users, invite other players to a game, see a list of invites from other players, and see a list of currently active games.

### Header.js

This header is on top of every page, it contains the LogInOutButton.

### HomePage.js

Originally this was going to be a button to navigate to different pages, however the header replaced this. We may use this down the road.

### LogInOutButton.js

Contains a button to either log in or out depending on the state of the user being logged in or not. Also contains link to navigate to the register an account page and the ability to logout if you are logged in,

### Table.js

Responsible for loading a table of registered users, allows the ability to search for a specific user.