Spaghetti-Coders Jungle Sprint 2

EJ Lee
Sam Stobbelaar
Jeff Bradley
Vlad Stepanuga
John Miller

User Stories (overview)

- **Register a new account**

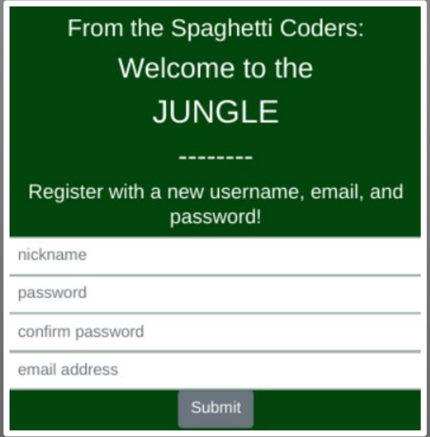
  <u>Acceptance Criteria</u>
  ```
  Test with all valid information
  Test with existing username
  Test with existing email
  Test with incorrectly formatted email
  Test with missing fields
  Test with username that includes special characters
  Test with a username that is too long
  Test with a password that is too short
  Test with a password that is too long
  ```

- **Login to application**

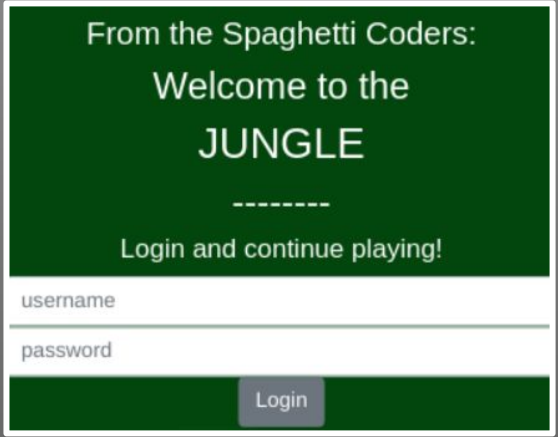
  <u>Acceptance Criteria</u>
  ```
  Test with valid registered user information
  Test with an invalid username
  Test with an invalid password
  Test with unregistered username
  Test with incorrect password
  Test with blank fields
  Test with all invalid fields
  Test with special characters
  ```

- **Play Jungle**

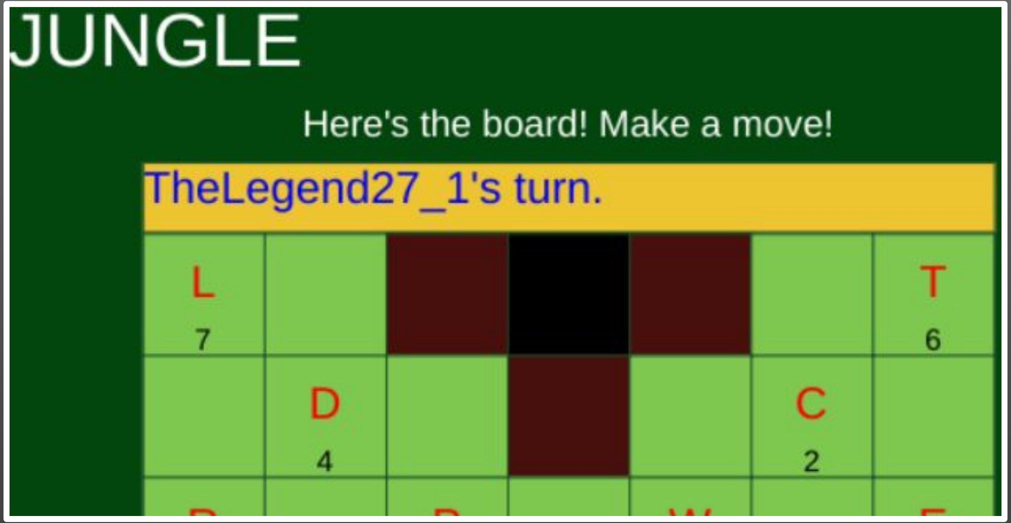
  <u>Acceptance Criteria</u>
  ```
  Test to see if the game board renders on the site.
  All pieces render on the board before the game starts.
  Test to see if game board is updated once a move is made.
  Test move prevention until opponent makes a move.
  Test rendered elements once game end has been reached.
  Test for JSON object validity once a move has been made.
  JSON object validity on move "locking" status by player.
  Test validity of the logic of each piece's movements.
  Test that end state of match can be reached validly.
  ```


From the Spaghetti Coders:
Welcome to the
JUNGLE

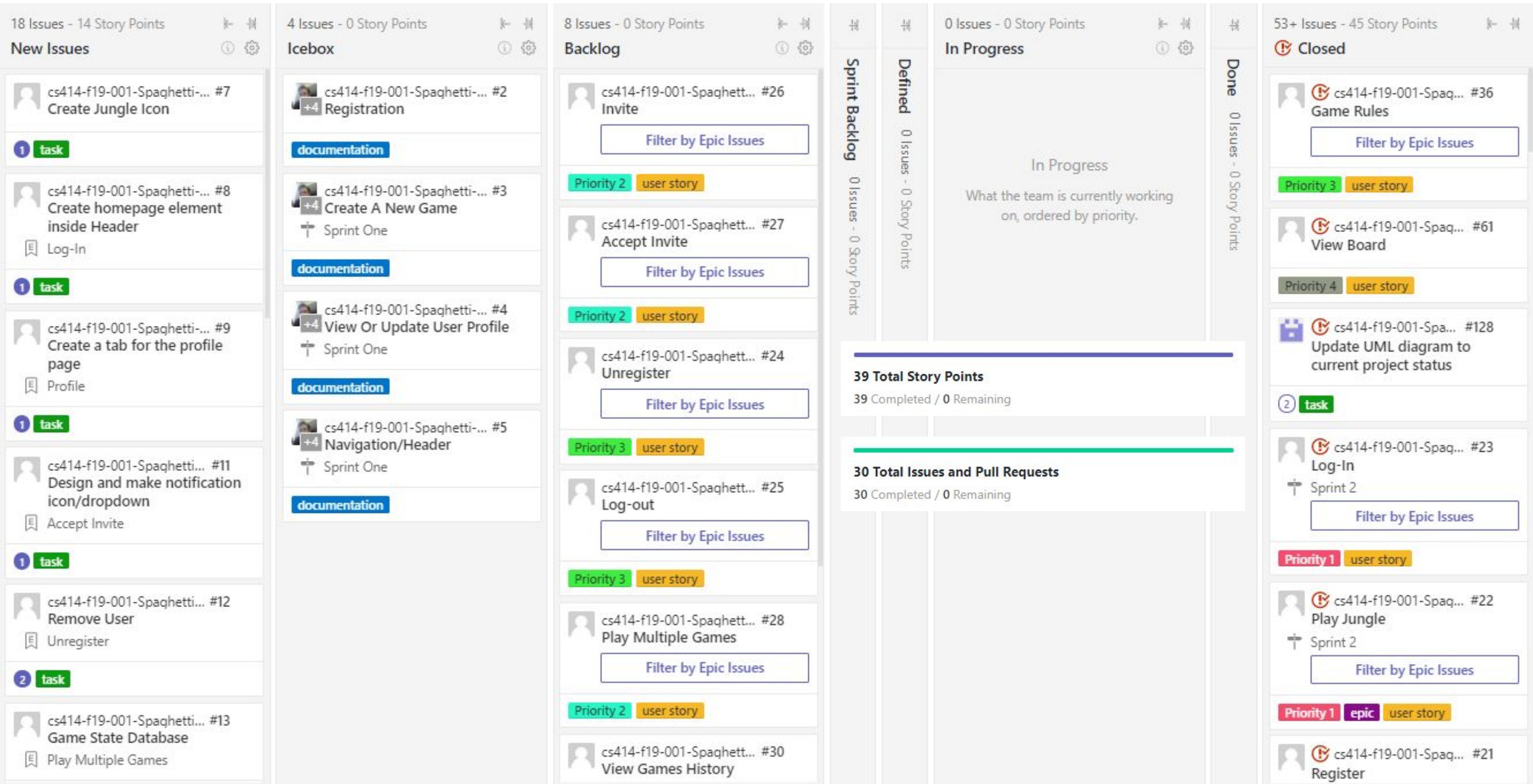
Register with a new username, email, and password!


From the Spaghetti Coders:
Welcome to the
JUNGLE

Login and continue playing!


JUNGLE
Here's the board! Make a move!
TheLegend27_1's turn.

Kanban Board



18 Issues - 14 Story Points
New Issues

cs414-f19-001-Spaghetti-... #7
Create Jungle Icon
1 task

cs414-f19-001-Spaghetti-... #8
Create homepage element inside Header
Log-In
1 task

cs414-f19-001-Spaghetti-... #9
Create a tab for the profile page
Profile
1 task

cs414-f19-001-Spaghetti... #11
Design and make notification icon/dropdown
Accept Invite
1 task

cs414-f19-001-Spaghetti... #12
Remove User
Unregister
2 task

cs414-f19-001-Spaghetti... #13
Game State Database
Play Multiple Games

4 Issues - 0 Story Points
Icebox

cs414-f19-001-Spaghetti-... #2
Registration
documentation

cs414-f19-001-Spaghetti-... #3
Create A New Game
Sprint One
documentation

cs414-f19-001-Spaghetti-... #4
View Or Update User Profile
Sprint One
documentation

cs414-f19-001-Spaghetti-... #5
Navigation/Header
Sprint One
documentation

8 Issues - 0 Story Points
Backlog

cs414-f19-001-Spaghett... #26
Invite
Filter by Epic Issues
Priority 2 user story

cs414-f19-001-Spaghett... #27
Accept Invite
Filter by Epic Issues
Priority 2 user story

cs414-f19-001-Spaghett... #24
Unregister
Filter by Epic Issues
Priority 3 user story

cs414-f19-001-Spaghett... #25
Log-out
Filter by Epic Issues
Priority 3 user story

cs414-f19-001-Spaghett... #28
Play Multiple Games
Filter by Epic Issues
Priority 2 user story

cs414-f19-001-Spaghett... #30
View Games History

Sprint Backlog 0 Issues - 0 Story Points

Defined 0 Issues - 0 Story Points

0 Issues - 0 Story Points
In Progress

In Progress
What the team is currently working on, ordered by priority.

39 Total Story Points
39 Completed / 0 Remaining

30 Total Issues and Pull Requests
30 Completed / 0 Remaining

Done 0 Issues - 0 Story Points

53+ Issues - 45 Story Points
Closed

cs414-f19-001-Spaq... #36
Game Rules
Filter by Epic Issues
Priority 3 user story

cs414-f19-001-Spaq... #61
View Board
Priority 4 user story

cs414-f19-001-Spa... #128
Update UML diagram to current project status
2 task

cs414-f19-001-Spaq... #23
Log-In
Sprint 2
Filter by Epic Issues
Priority 1 user story

cs414-f19-001-Spaq... #22
Play Jungle
Sprint 2
Filter by Epic Issues
Priority 1 epic user story

cs414-f19-001-Spaq... #21
Register

CRC Cards (client)

Application	
• Handle login	• Welcome
• Render Welcome and Main Page	• Main

Login	
• Validates user credentials	• Application
• Sends user to main page upon successful login	• Welcome

Register	
• Inserts new users into database and creates profile	• Application
• Validates new credentials are correct format and unused	• Welcome
• Sends user to login page after successful registration	

GamePage	
• Display current state of the board	• Main
• Show selected piece	• Match
• Send move made to back end	

reactstrap

CRC Cards (client & restful api)

Main	
• Display current and past games from back end • Display current user logged in • Allow user to start a new game	• Match • Login

Welcome	
• Initial page user sees upon visiting website • Allows user to go to login or register page • Displays info about Jungle game and devs	

HTTPRestful	
• Handle register and login requests • Start server	• Login • Register

InitServer	
• Initialize the server using an HTTPRestful object	• HTTPRestful

CRC Cards (game logic)

JungleBoard	
• Create a new board • Initialize the starting pieces • Place pieces • Get piece at certain position • Make moves by updating piece locations • Hold current game state variables for front end communication	• Piece

Piece	Rat, Cat, Wolf, Dog, Leopard, BigCat, Lion, Tiger, Elephant
• Hold information for each piece like rank, color, row and column • Get and set piece positions • Check and return legal moves for each piece • Check if a move will trap the piece • Check if a move will win the game • Set winner and state of game for front end use	• JungleBoard • Child classes

Rat	Piece
• Create Rat pieces • Set Rat rank • Check legal positions as Rat has special moves	• Piece

Game Logic

CRC Cards (pieces)

Collaborators are the same as Elephant

Cat
• Create Cat pieces • Set Cat rank

Wolf
• Create Wolf pieces • Set Wolf rank

Dog
• Create Dog pieces • Set Dog rank

Leopard
• Create Leopard pieces • Set Leopard rank

BigCat	Piece Lion, Tiger
• Check legal moves for Lion and Tiger child classes as they have special moves	• Piece • Lion • Tiger

Lion	BigCat, Piece
• Create Lion pieces • Set Lion rank	• BigCat • Piece

Tiger	BigCat, Piece
• Create Tiger pieces • Set Tiger rank	• BigCat • Piece

Elephant	Piece
• Create Elephant pieces • Set Elephant rank	• Piece

CRC Cards (database)

RetrieveMatches	
• Communicate with database to pull all active and past matches for a user	• Match

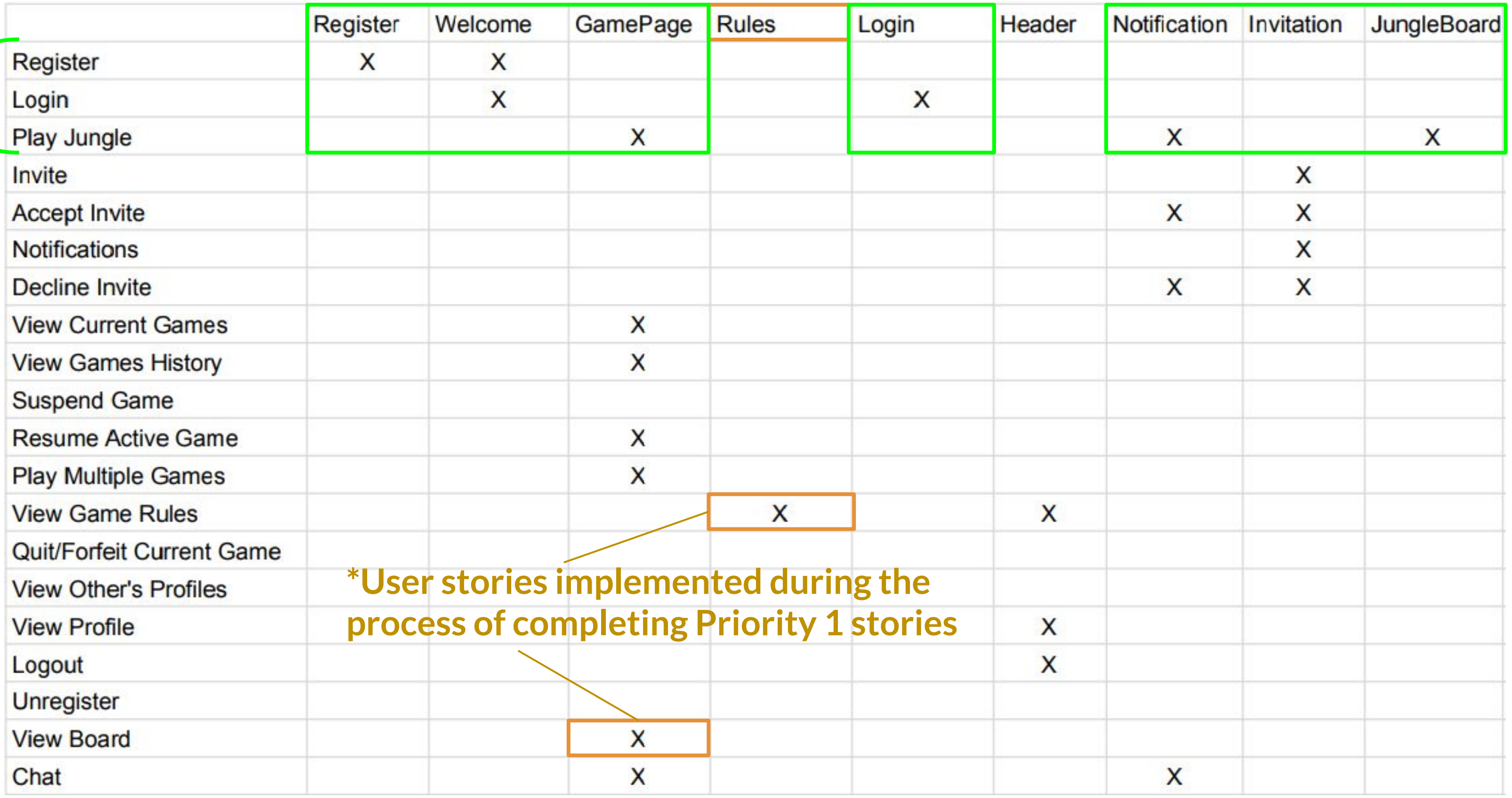
RetrieveProfile	
• Communicate with database to pull user profiles	• Profile

Profile	
• Hold information about each user • Display statistics about player	

Match	
• Move data between JungleBoard and GamePage • Communicate with database to pull current state of games • Set game state variables	• JungleBoard • GamePage

Traceability Link Matrix

Priority 1



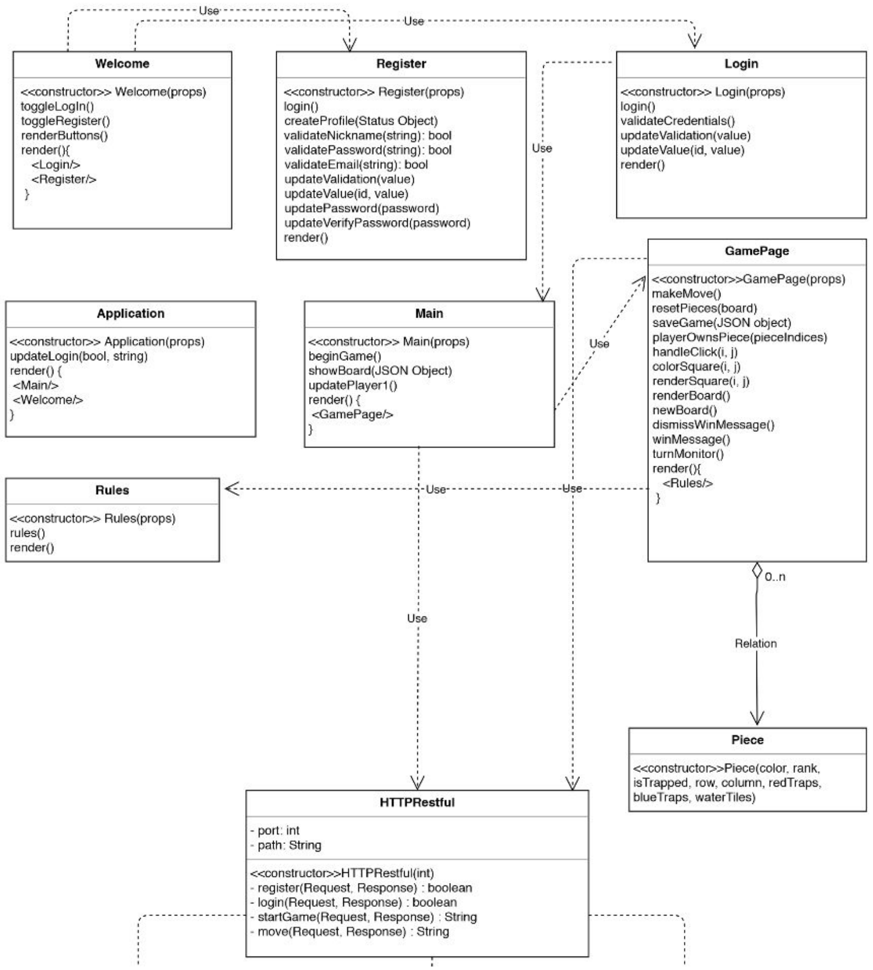
	Register	Welcome	GamePage	Rules	Login	Header	Notification	Invitation	JungleBoard
Register	X	X							
Login		X			X				
Play Jungle			X				X		X
Invite								X	
Accept Invite							X	X	
Notifications								X	
Decline Invite							X	X	
View Current Games			X						
View Games History			X						
Suspend Game									
Resume Active Game			X						
Play Multiple Games			X						
View Game Rules				X		X			
Quit/Forfeit Current Game									
View Other's Profiles									
View Profile						X			
Logout						X			
Unregister									
View Board			X						
Chat			X				X		

*User stories implemented during the process of completing Priority 1 stories

Traceability Link Matrix (cont.)

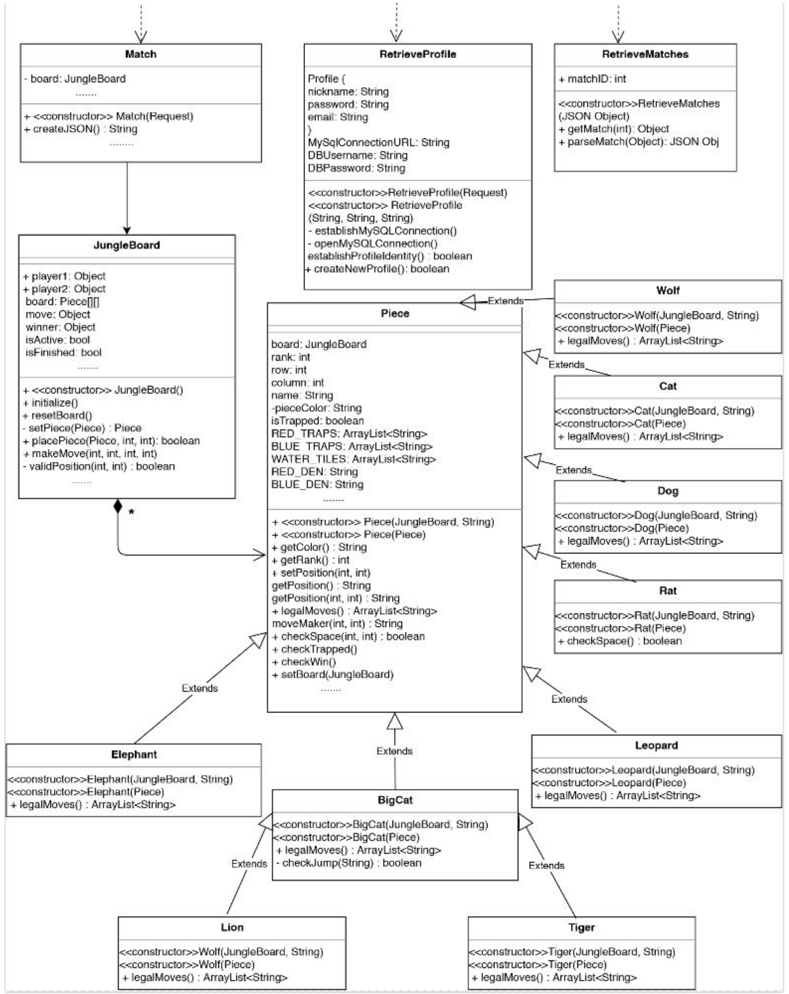
	Piece	BigCat	Match	RetrieveProfile	RetrieveMatches	Rat	Cat	Wolf	Dog	Leopard	Tiger	Lion	Elephant
Register				X									
Login													
Play Jungle	X	X	X		X	X	X	X	X	X	X	X	X
Invite													
Accept Invite													
Notifications													
Decline Invite													
View Current Games													
View Games History													
Suspend Game			X		X								
Resume Active Game			X		X								
Play Multiple Games					X								
View Game Rules													
Quit/Forfeit Current Game			X		X								
View Other's Profiles				X									
View Profile				X									
Logout													
Unregister				X									
View Board					X								
Chat													

Class UML Diagram Client Side


Use
Use
Welcome
<<constructor>> Welcome(props)
toggleLogIn()
toggleRegister()
renderButtons()
render(){
<Login/>
<Register/>
}
Register
<<constructor>> Register(props)
login()
createProfile(Status Object)
validateNickname(string): bool
validatePassword(string): bool
validateEmail(string): bool
updateValidation(value)
updateValue(id, value)
updatePassword(password)
updateVerifyPassword(password)
render()
Login
<<constructor>> Login(props)
login()
validateCredentials()
updateValidation(value)
updateValue(id, value)
render()
Use
GamePage
<<constructor>>GamePage(props)
makeMove()
resetPieces(board)
saveGame(JSON object)
playerOwnsPiece(pieceIndices)
handleClick(i, j)
colorSquare(i, j)
renderSquare(i, j)
renderBoard()
newBoard()
dismissWinMessage()
winMessage()
turnMonitor()
render(){
<Rules/>
}
Application
<<constructor>> Application(props)
updateLogin(bool, string)
render() {
<Main/>
<Welcome/>
}
Main
<<constructor>> Main(props)
beginGame()
showBoard(JSON Object)
updatePlayer1()
render() {
<GamePage/>
}
Use
Rules
<<constructor>> Rules(props)
rules()
render()
Use
Use
0..n
Use
Relation
Piece
<<constructor>>Piece(color, rank, isTrapped, row, column, redTraps, blueTraps, waterTiles)
HTTPRestful
- port: int
- path: String
<<constructor>>HTTPRestful(int)
- register(Request, Response) : boolean
- login(Request, Response) : boolean
- startGame(Request, Response) : String
- move(Request, Response) : String

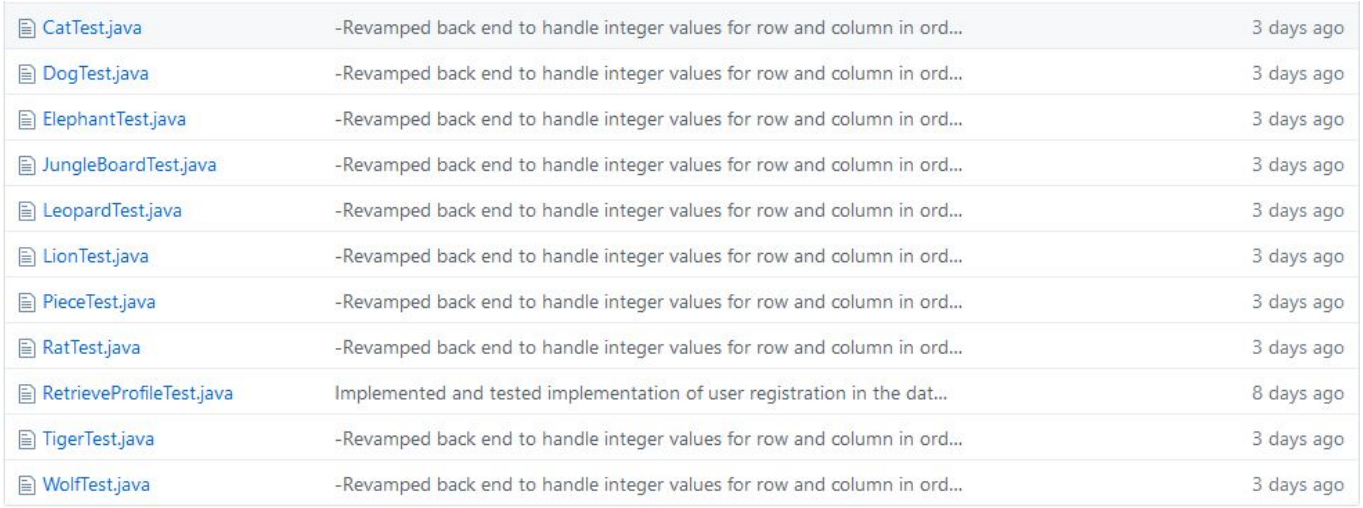
Spark

Class UML Diagram Server Side


Match
- board: JungleBoard
+ <<constructor>> Match(Request)
+ createJSON() : String
RetrieveProfile
Profile {
nickname: String
password: String
email: String
}
MySqlConnectionURL: String
DBUsername: String
DBPassword: String
<<constructor>>RetrieveProfile(Request)
<<constructor>> RetrieveProfile (String, String, String)
- establishMySQLConnection()
- openMySQLConnection()
establishProfileIdentity() : boolean
+ createNewProfile(): boolean
RetrieveMatches
+ matchID: int
<<constructor>>RetrieveMatches (JSON Object)
+ getMatch(int): Object
+ parseMatch(Object): JSON Obj
JungleBoard
+ player1: Object
+ player2: Object
board: Piece[][]
move: Object
winner: Object
isActive: bool
isFinished: bool
+ <<constructor>> JungleBoard()
+ initialize()
+ resetBoard()
- setPiece(Piece) : Piece
+ placePiece(Piece, int, int): boolean
+ makeMove(int, int, int, int)
- validPosition(int, int) : boolean
*
Piece
board: JungleBoard
rank: int
row: int
column: int
name: String
-pieceColor: String
isTrapped: boolean
RED_TRAPS: ArrayList<String>
BLUE_TRAPS: ArrayList<String>
WATER_TILES: ArrayList<String>
RED_DEN: String
BLUE_DEN: String
+ <<constructor>> Piece(JungleBoard, String)
+ <<constructor>> Piece(Piece)
+ getColor() : String
+ getRank() : int
+ setPosition(int, int)
getPosition() : String
getPosition(int, int) : String
+ legalMoves() : ArrayList<String>
moveMaker(int, int) : String
+ checkSpace(int, int) : boolean
+ checkTrapped()
+ checkWin()
+ setBoard(JungleBoard)
Extends
Wolf
<<constructor>>Wolf(JungleBoard, String)
<<constructor>>Wolf(Piece)
+ legalMoves() : ArrayList<String>
Cat
<<constructor>>Cat(JungleBoard, String)
<<constructor>>Cat(Piece)
+ legalMoves() : ArrayList<String>
Dog
<<constructor>>Dog(JungleBoard, String)
<<constructor>>Dog(Piece)
+ legalMoves() : ArrayList<String>
Rat
<<constructor>>Rat(JungleBoard, String)
<<constructor>>Rat(Piece)
+ checkSpace() : boolean
Elephant
<<constructor>>Elephant(JungleBoard, String)
<<constructor>>Elephant(Piece)
+ legalMoves() : ArrayList<String>
BigCat
<<constructor>>BigCat(JungleBoard, String)
<<constructor>>BigCat(Piece)
+ legalMoves() : ArrayList<String>
- checkJump(String) : boolean
Leopard
<<constructor>>Leopard(JungleBoard, String)
<<constructor>>Leopard(Piece)
+ legalMoves() : ArrayList<String>
Lion
<<constructor>>Wolf(JungleBoard, String)
<<constructor>>Wolf(Piece)
+ legalMoves() : ArrayList<String>
Tiger
<<constructor>>Tiger(JungleBoard, String)
<<constructor>>Tiger(Piece)
+ legalMoves() : ArrayList<String>

JUnit 5 Testing

- Pure JUnit 5 Methodology, no JUnit 4.
- 38 Unique Tests over 11 Test Classes.

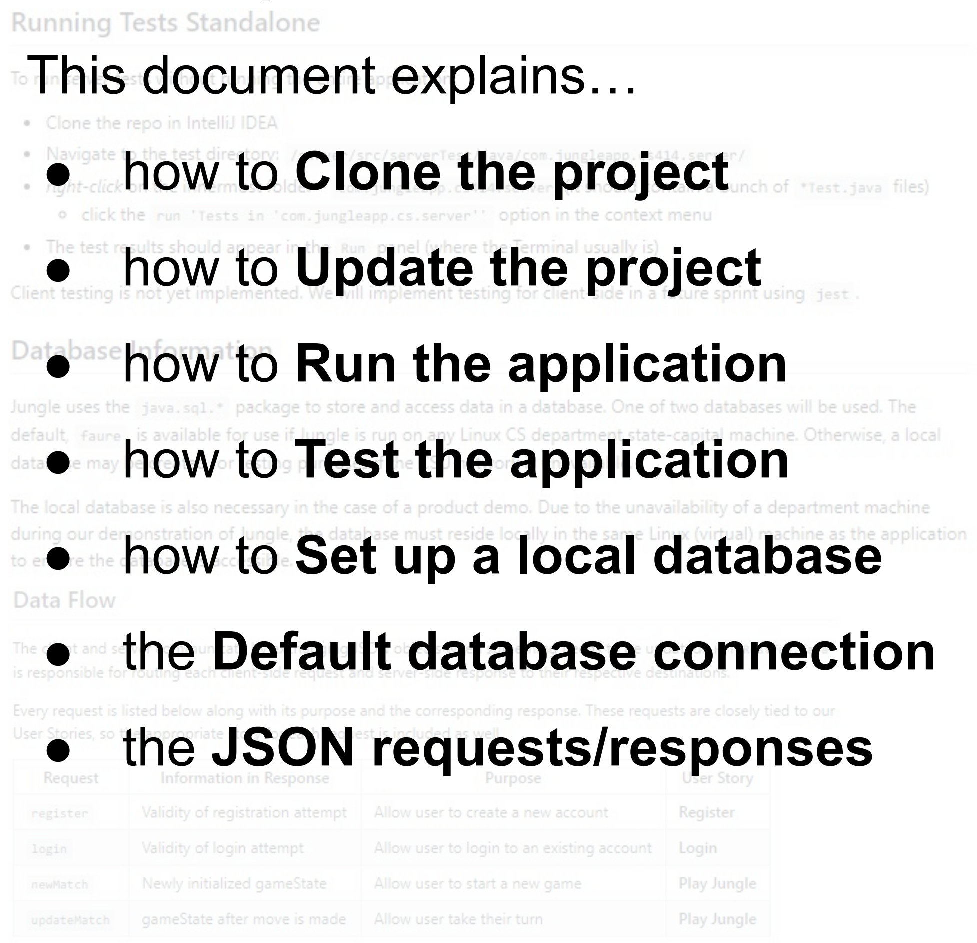


CatTest.java	-Revamped back end to handle integer values for row and column in ord...	3 days ago
DogTest.java	-Revamped back end to handle integer values for row and column in ord...	3 days ago
ElephantTest.java	-Revamped back end to handle integer values for row and column in ord...	3 days ago
JungleBoardTest.java	-Revamped back end to handle integer values for row and column in ord...	3 days ago
LeopardTest.java	-Revamped back end to handle integer values for row and column in ord...	3 days ago
LionTest.java	-Revamped back end to handle integer values for row and column in ord...	3 days ago
PieceTest.java	-Revamped back end to handle integer values for row and column in ord...	3 days ago
RatTest.java	-Revamped back end to handle integer values for row and column in ord...	3 days ago
RetrieveProfileTest.java	Implemented and tested implementation of user registration in the dat...	8 days ago
TigerTest.java	-Revamped back end to handle integer values for row and column in ord...	3 days ago
WolfTest.java	-Revamped back end to handle integer values for row and column in ord...	3 days ago

Development Manual

This document explains…

- how to **Clone the project**
- how to **Update the project**
- how to **Run the application**
- how to **Test the application**
- how to **Set up a local database**
- the **Default database connection**
- the **JSON requests/responses**



Running Tests Standalone

To [illegible]

- Clone the repo in IntelliJ IDEA
- Navigate [illegible] the test directory: `/[illegible]/src/serverTest/java/com.jungleapp.[illegible]414.server/`
- *right-click* [illegible] `*Test.java` files)
  - click the `run 'Tests in 'com.jungleapp.cs.server''` option in the context menu
- The test results should appear in the `Run` panel (where the Terminal usually is)

Client testing is not yet implemented. We will implement testing for client side in a future sprint using `jest`.

Database Information

Jungle uses the `java.sql.*` package to store and access data in a database. One of two databases will be used. The default, `faure`, is available for use if Jungle is run on any Linux CS department state-capital machine. Otherwise, a local data[illegible] may [illegible]

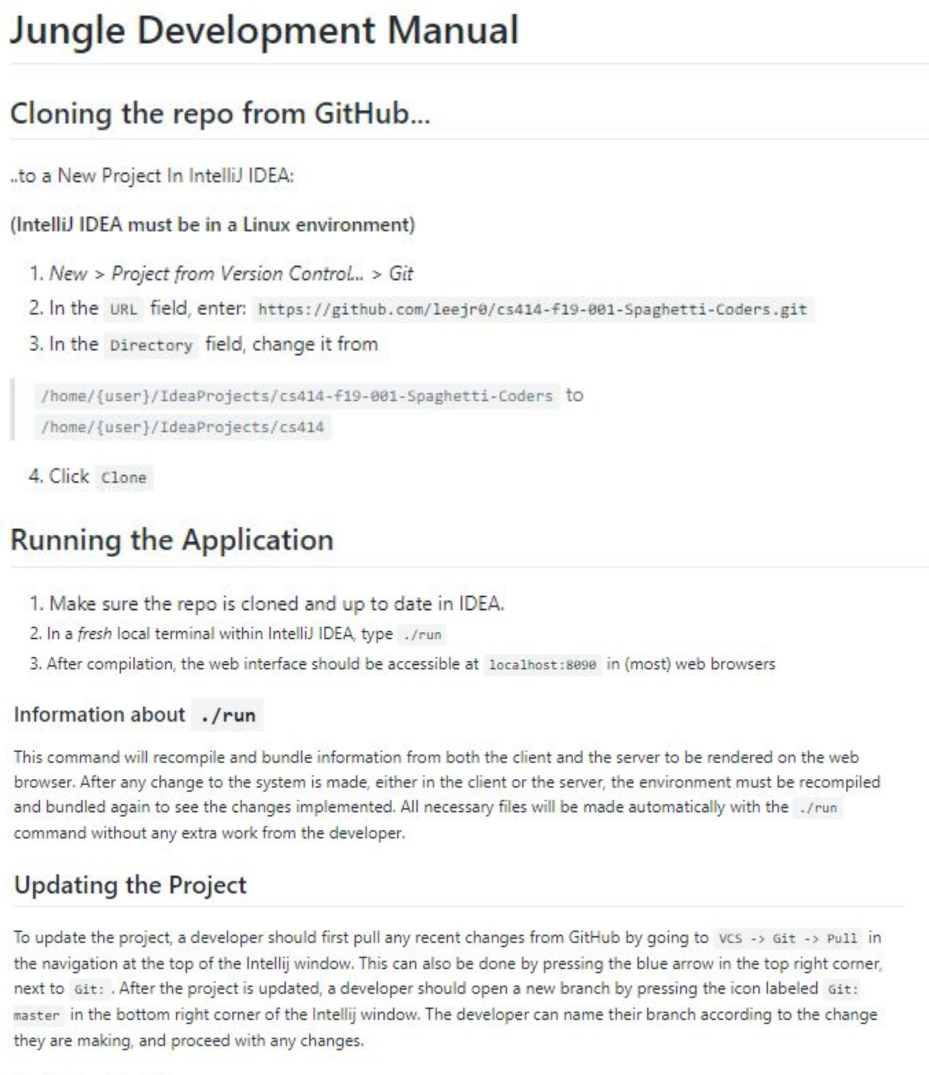
The local database is also necessary in the case of a product demo. Due to the unavailability of a department machine during our demonstration of Jungle, the database must reside locally in the same Linux (virtual) machine as the application to e[illegible] the [illegible]

Data Flow

The [illegible] and se[illegible]
is responsible for routing each client-side request and server-side response to their respective destinations.

Every request is listed below along with its purpose and the corresponding response. These requests are closely tied to our User Stories, so [illegible] request is included as well.

Request	Information in Response	Purpose	User Story
`register`	Validity of registration attempt	Allow user to create a new account	Register
`login`	Validity of login attempt	Allow user to login to an existing account	Login
`newMatch`	Newly initialized gameState	Allow user to start a new game	Play Jungle
`updateMatch`	gameState after move is made	Allow user take their turn	Play Jungle



Jungle Development Manual

Cloning the repo from GitHub…

..to a New Project In IntelliJ IDEA:

(IntelliJ IDEA must be in a Linux environment)

1. *New > Project from Version Control... > Git*
2. In the `URL` field, enter: `https://github.com/leejr0/cs414-f19-001-Spaghetti-Coders.git`
3. In the `Directory` field, change it from

> `/home/{user}/IdeaProjects/cs414-f19-001-Spaghetti-Coders` to
> `/home/{user}/IdeaProjects/cs414`

4. Click `Clone`

Running the Application

1. Make sure the repo is cloned and up to date in IDEA.
2. In a *fresh* local terminal within IntelliJ IDEA, type `./run`
3. After compilation, the web interface should be accessible at `localhost:8090` in (most) web browsers

Information about `./run`

This command will recompile and bundle information from both the client and the server to be rendered on the web browser. After any change to the system is made, either in the client or the server, the environment must be recompiled and bundled again to see the changes implemented. All necessary files will be made automatically with the `./run` command without any extra work from the developer.

Updating the Project

To update the project, a developer should first pull any recent changes from GitHub by going to `VCS -> Git -> Pull` in the navigation at the top of the Intellij window. This can also be done by pressing the blue arrow in the top right corner, next to `Git:` . After the project is updated, a developer should open a new branch by pressing the icon labeled `Git: master` in the bottom right corner of the Intellij window. The developer can name their branch according to the change they are making, and proceed with any changes.

Jungle Demo

Image Sources:
http://www.sitalgames.com/project/animal-chess/
https://dev.to/danijelajs/react-bootstrap-frameworks-review-1n13
https://jrebel.com/rebellabs/spark-java-is-an-amazing-java-web-framework-do-you-really-need-it/
https://www.iconfinder.com/icons/1954836/brain_digital_electronic_processor_icon
https://www.pinterest.com/pin/632263235162988772/?lp=true
https://seeklogo.net/mysql-in-eps-format-5459.html