# Design Document for TowerDefense

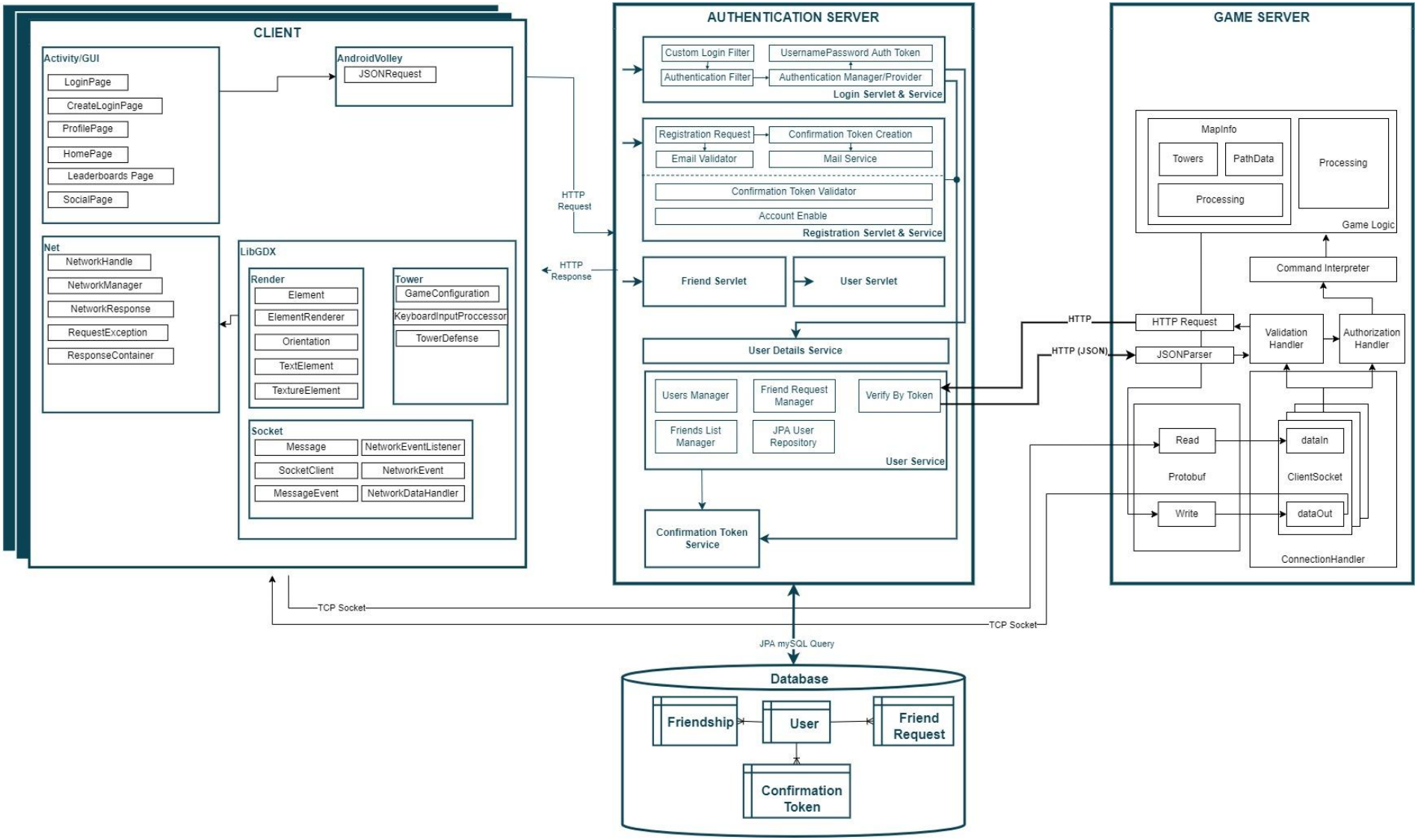
Group <2_do_7>

Ben Hall:  % contribution

Gavin Tersteeg: % contribution

James Minardi: % contribution

Jeremy Lewis: % contribution


CLIENT
Activity/GUI
LoginPage
CreateLoginPage
ProfilePage
HomePage
Leaderboards Page
SocialPage
AndroidVolley
JSONRequest
Net
NetworkHandle
NetworkManager
NetworkResponse
RequestException
ResponseContainer
LibGDX
Render
Element
ElementRenderer
Orientation
TextElement
TextureElement
Tower
GameConfiguration
KeyboardInputProccessor
TowerDefense
Socket
Message
NetworkEventListener
SocketClient
NetworkEvent
MessageEvent
NetworkDataHandler
HTTP Request
HTTP Response
AUTHENTICATION SERVER
Custom Login Filter
UsernamePassword Auth Token
Authentication Filter
Authentication Manager/Provider
Login Servlet & Service
Registration Request
Confirmation Token Creation
Email Validator
Mail Service
Confirmation Token Validator
Account Enable
Registration Servlet & Service
Friend Servlet
User Servlet
User Details Service
Users Manager
Friend Request Manager
Verify By Token
Friends List Manager
JPA User Repository
User Service
Confirmation Token Service
GAME SERVER
MapInfo
Towers
PathData
Processing
Processing
Game Logic
Command Interpreter
HTTP
HTTP (JSON)
HTTP Request
JSONParser
Validation Handler
Authorization Handler
Read
Protobuf
Write
dataIn
ClientSocket
dataOut
ConnectionHandler
TCP Socket
TCP Socket
JPA mySQL Query
Database
Friendship
User
Friend Request
Confirmation Token

## Login & Authorization System

Using an *almost* out of the box solution for authorization allows our application to fully secure our communication with users. Our spring security custom configuration defines a set of secured end points that may only be accessed by users with a certain priviledge. Unauthorized requests to secured end points result in an appropriate HTTP response, thus blocking the user from accessing that resource.

To authenticate, the user sends a login request containing their username and password. Our custom login filter converts this JSON request object into form-data that can be parsed by spring security. After the login credentials are checked against our password encryption service and authentication provider, the user is then sent to our login success handler to redirect them to the appropriate page. The user is now authenticated and can access previously unauthorized resources.

## Friends Frontend System

The friends system on the frontend uses pretty much exclusively android studio libraries such as android studio to send json requests to the backend server. In the case of this system, there are many different calls that can and are made everytime you go onto this page. To start when you first get to the page it will request the list of incoming friend requests and your current friends which are displayed at two different parts of the page.

Other request that you can perform is by adding users which is done through text input and sending a request to the backend which will let you know the response. You can also accept or decline invites from users that have sent you a request via another text entry and buttons, both of which will confirm if it was successful or not.

## Game Server

The game server uses websockets to communicate with players in real time. To do this quickly and efficiently, we used protobuf, a Google library to serialize Java objects into a compressed byte format to send across socket data streams. Besides this, the game server handles game logic like collision detection, round starting and ending, and anything else related to the overall game logic.

To validate users, the client socket sends an authorization token that the game server passes to the authentication server to grab user details. After all users have connected, users can send request to manipulate the lobby, place towers, and receive updates about towers and enemy kills.

## Frontend Element Rendering System

In order to simplify how elements are created, manipulated, and displayed on the front end, a system was created that allows for different graphical objects abstracted from their LibGDX API calls. The system works by having many different “Element” objects, which can be added to or removed from a render queue. This render queue will automatically read the desired element properties, and render it accordingly. Element positions are abstracted from a “virtual” coordinate system into the actual LibGDX positioning system, allowing elements to be added in many different ways without having to do cumbersome math for each different element.

There are a number of different types of elements that can be rendered. The first, and most versatile, is the TextureElement. This element allows a texture to be rendered onto the screen with a number of properties. The X/Y position can be changed, width and height dynamically modified, and the orientation on the screen set to different locations. In addition, there is also the TextElement. This works a lot like the TextureElement, but instead displays a string of text instead of a texture. The width and height are automatically generated, and cannot be changed, but the font and the string can be swapped out for different text characteristics.

## Frontend Buffered Socket Communication

Due to the fact that HTTP requests are relatively slow for applications like games, a different form of networking connection was needed. This came in the form of a direct TCP socket server. For each different game lobby, or “server”, there is a certain IP and port number that the frontend can connect to in

order to enter the game lobby. The connection is all serial, so to make the programmers lives easier, an abstraction layer was created to allow for fast serialization of objects. This comes in the form of ProtoBuf, which can receive objects coming down the socket connection, or send objects through the socket connection.

The actual decoding of objects being received from the server needs to be done continuously, so this lives in an independent thread from the main game loop. The NetworkDataHandler will wait until information has been received from the socket, and then pass that information to ProtoBuf. Once that information has been deserialized, it passes that information on in the form of NetworkEvents. These events can be picked up by NetWorkEventListeners bound to the NetworkDataHandler, which can then go on to be processed by the game loop itself.

PUT THE TABLE RELATIONSHIPS DIAGRAM on this fourth page! (Create the picture using MySQLWorkbench)