RISK Bot Vs. Bot Website

Software Architecture

Specification

Overview

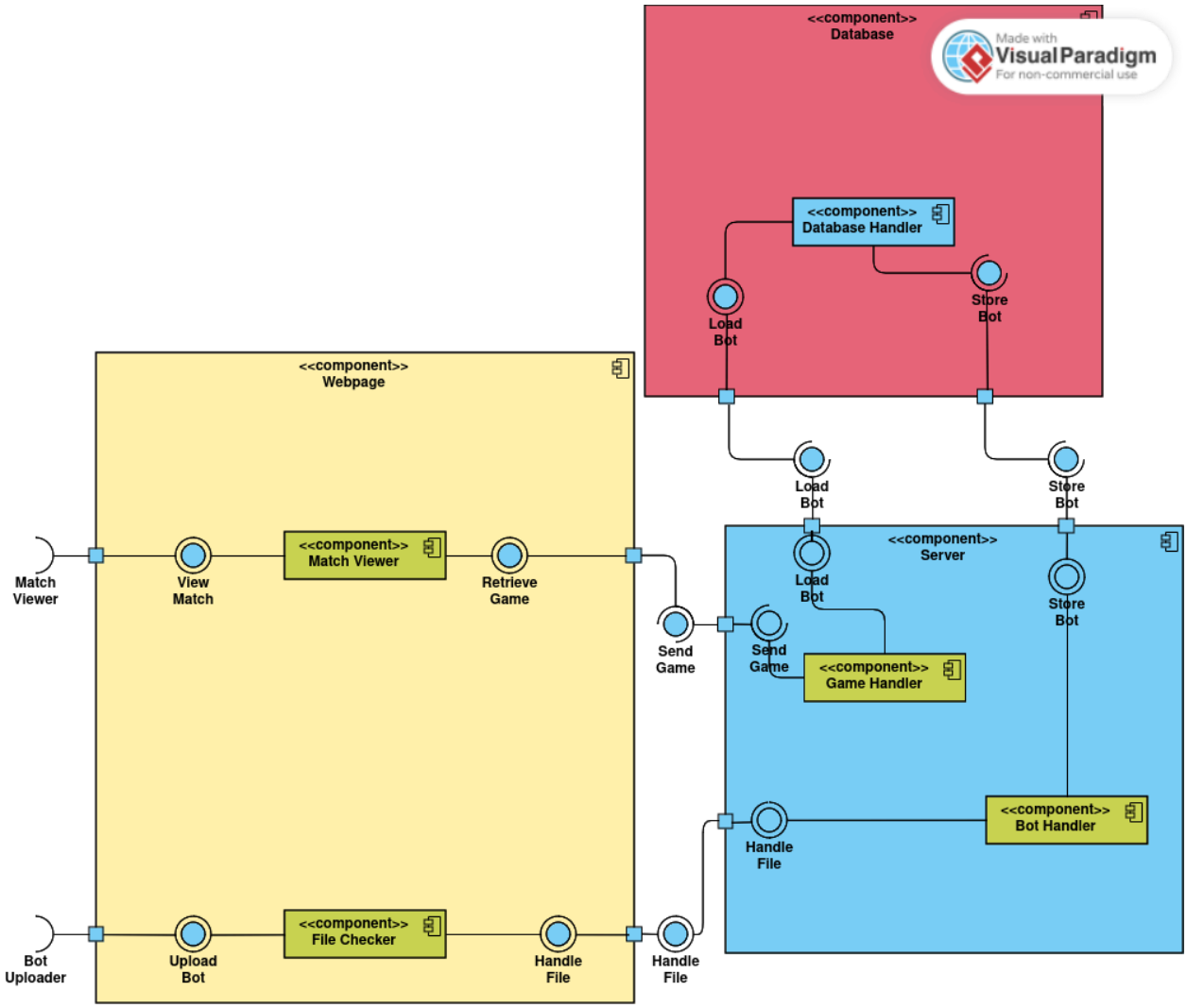
The basic structure of the applications software architecture system consists of three main components: the database, the webpage, and the server. Each component will be dockerized and run separately.

The server will consist of four components: a basic boilerplate set-up for all the essential processes and code necessary but not related to the project, the output system that connects to the webpage and ensures it's updated with the appropriate information when required, the game controlling portions that keep track of game logic like the board and players, and finally the system that will check and read the user code from the database to the game component ensuring that it’s safe and compilable.

The webpage component can be broken up into three key parts. It will need a segment that ingests and reads data from the server component, another that deals with necessary backend boilerplate code, and a third that handles what the user interfaces with. The UI can further be broken up into three separate pages. The first is the Home/Landing page, the second is the upload page where people put their bots onto the website, and the third is the place where people can see their bots battling on the website visibly in some manner.

The final main component is the database is fairly simple and cannot be broken up. The database won’t be storing much information beyond some simple time information on when a bot was submitted, the user who submitted it, a name for the bot, a value for the number of victories, a value for the number of losses and a file reference to where the bot’s code is stored. Further information is unnecessary so that the data can be placed within a single database table.

Subsystem Decomposition

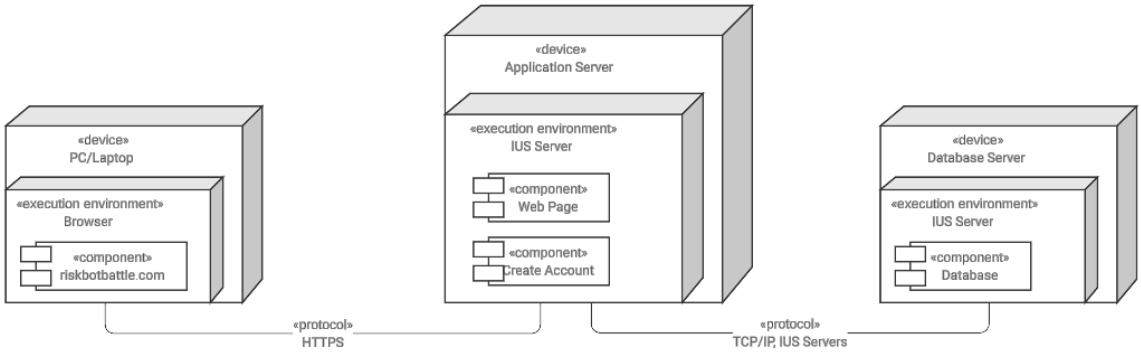


In this Diagram, there are three central systems with five main components. The webpage component handles all of the user interfacing and user display. The Database handles user code storage and information. The server handles running the game and parsing the bot to be compiled for the game.

The Match Viewer takes user input to decide what match to view, then retrieves the game from the Server and plays it for the User. The File Checker initially checks the uploaded bot file and passes user information to the Bot Handler. The Bot Handler prepares bots for being run by the Game Handler and passes any errors or issues to the User. The Game Handler runs the game by getting the loaded bot from the Database Handler and passing the finished game to the match viewer when it's finished. The database handler passes the database bots and user info and returns that information to the game handler when requested.

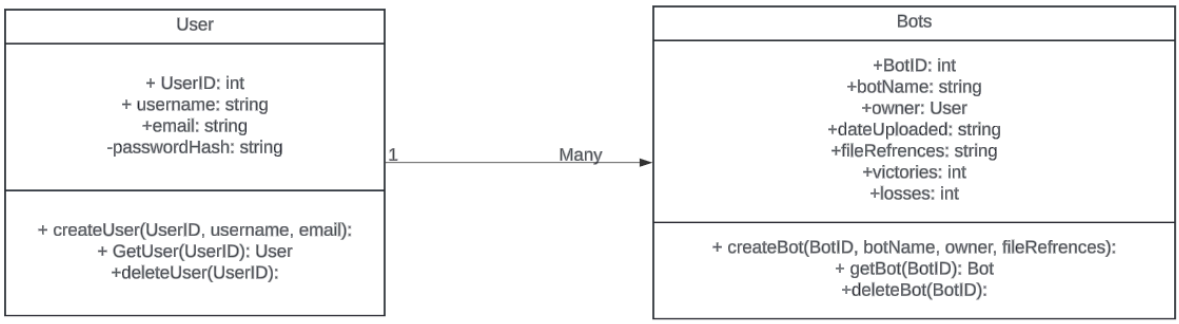
Hardware/Software Mapping

Data flow between the hardware and software is described in the diagram below.



Persistent Data Mapping

Users will need to register an account on the site to submit their A.I. bot for competition. Data stored about the users and the bots are described in the diagram below.

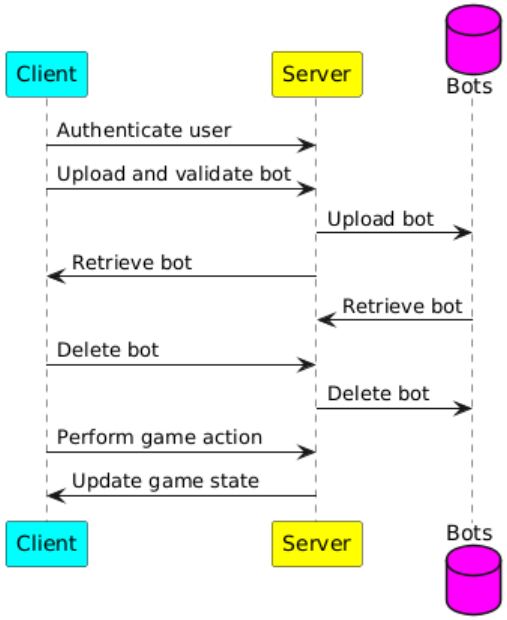


Access Control Management

Users will be capable of uploading their own bots to the website, retrieving bots they have previously uploaded, and deleting bots they have previously uploaded. System administrators will have the same permissions, but they will have access to bots uploaded from any user. OAuth will be used to authenticate users and ensure they will have access to only the operations they are permitted to perform. OAuth is an open-standard delegation protocol that allows websites to delegate authentication to other websites without giving the former websites access to the user's credentials.

Global Software Control

The website, both the game and the database access, will be event-driven. The control flow is described in the diagram below.



Boundary Control

The program will be started up and initialized using a docker file. To be able to migrate to another server, the docker file will need to be moved to the new server, then the docker build and docker run commands will be run. This will be all necessary to spin up a new instance of the program. Most exceptions should be caught and handled using C#'s try-catch feature. This will allow the program to go from a bad state back to a good state. For unexpected breaking errors, however, the program may need to be restarted to return to a working state.

Individual Contributions

Xavier Beatrice

- Implemented Formatting, Overview, and Subsystem Decomposition

Dale Morris

- Implemented Access Control Management, Global Software Control, and Boundary Control

Louis Wiley

- Hardware/Software Mapping, Persistent Data Mapping

Key Personnel

- Xavier Beatrice
    - Team-lead
    - Fill-in Developer
- Louis Wiley
    - Front-end Developer
- Dale Morris
    - Back-end Developer