# Web-based Chess

Patrick Lawrence, Trush Patel, Ishmail Koroma, Kevin Codd, Matt DiStefano

# Table of Contents

# Summary

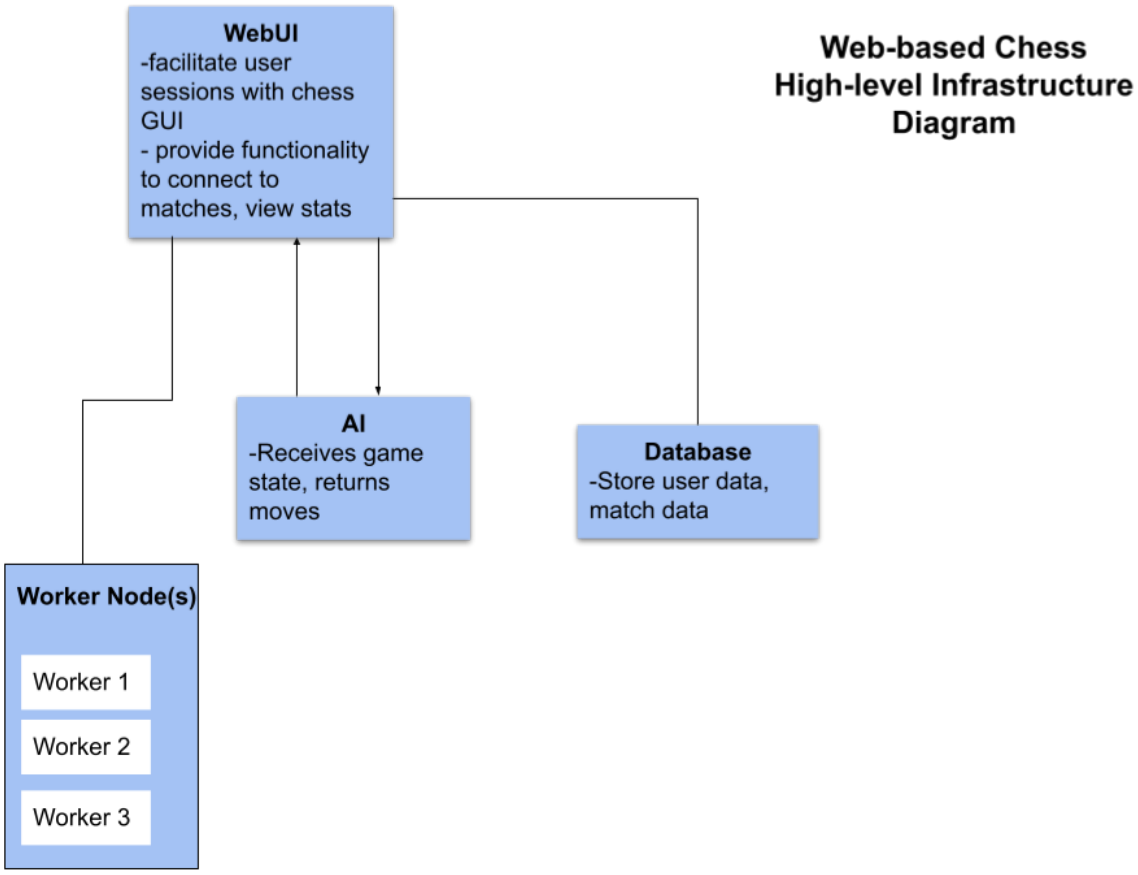
The task for this project is to come up with an original idea, design and implement the application, and carry out a full stack deployment with CI/CD services on CloudLab. We decided to build a web-based multiplayer chess application. Our goal is to build a cloud-supported application that supports multiple users and concurrent game sessions with the option to play against an AI at varying levels difficulty. Each user should be availed individual performance statistics to gauge their skill level.

# Team Description

We are a team of 5 computer science undergraduate students at West Chester University of Pennsylvania designing, implementing, and deploying a full-stack project for the Introduction to Cloud Computing course. We all come from various backgrounds that include software design as well as front- and back-end development. Each team member's individual qualifications are specified in the Resumes section.

# Technical Report

Our overall vision for this project is to create a cloud-based chess game playable in the browser — essentially like a simplified version of chess.com or lichess.org. Core functionality would include a fullyimplemented GUI-based chess game with options to play against a human or AI opponent, creation of user accounts and persistent storage of user data, and matchmaking between users. We could use Jenkins for continuous integration and continuous deployment to prevent downtime and automate tasks when making updates to the application. Our primary revenue source will be from ads, i.e. banner ads placed throughout the user interface. We are also considering a wager feature that would bring in revenue by taking a percentage of deposits and/or wagers on games by the participating players. It is important to note that this project's constraints are limited to the extent of the CloudLab hardware available to us. Below is a high-level diagram of the rough infrastructure we envision.




We originally planned on including a "master" node, which would have facilitated player matchmaking and served as a broker between the WebUI sessions, players, and AI. Our revised plan tentatively scraps this component of the architecture for similification, opting to use the database as an intermediary between webUI sessions, and to have the AI code housed in the

webUI. We would also potentially have a number of “worker” nodes to provide extra compute to the webUI node when necessary, as it will be responsible for facilitating all webUI and AI

# Chapter 2 - Technical Implementation

## Web UI

The web UI’s key user-facing functionality will be GUI-based chess. The user will be able to play chess on a graphical chess board, most likely with a point-and-click control paradigm. The web UI will also include pages for login and account creation, and for viewing user stats. Additionally, it will serve to connect the user with the revenue framework we choose to implement, such as ads, a wagering system, or a premium subscription service. Some candidate frameworks for implementation include JavaFX, React.js, and Vue.js, but we must consider which will provide the most efficient rendering and convenience for the user.

### Key web UI Features

| **User Actions** | **Potential Implementation** |
|---|---|
| Create/update/delete account | When a user first connects to a webUI session, there will be a button linking to a sign-up window where the user can create a username and password. We can also have a “user settings” button somewhere in the primary window that brings up options to update username/password or delete an account. |
| Login | When a user first connects to the web UI, they will be presented with a window with options to enter their username and password or create an account via a button. |
| View stats | A button will link to a window where a user can pull up basic stats such as win/loss ratio, and their game history. This data would be read and calculated from the database. |
| Create invitations and join matches | Users can choose to create an invitation - either public or to a specific user. A list of invitations is |

| | |
|---|---|
| | then periodically published to all connected web UI sessions with permission to view them, and other users can accept these invitations. The user can also choose to create a game against an AI rather than play against other users. For all of these options, the user must also specify the starting amount of time for the match timers. (2, 5, 10 minutes) |
| Move pieces via point-and click/drag and drop | In the gameboard GUI, users can move pieces by dragging and dropping. A background rulechecking function will verify that the move is valid and allow it to be executed on the GUI if it is legal. |
| Game clocks | Clocks for both players will display how much time they have left. Time controls can be customized, perhaps via a drop-down when game invitations are first created. |
| Resign/leave a match | There will be a button to resign/leave a match, ending the game session in a win for the resigner's opponent. |

## Backend Functionality/Database

The database will store user account information including username, rating and game history. It will also serve as a broker to facilitate game invitations and the creation of game sessions. The database could be implemented using several different database frameworks, including mySQL, Redis, and MongoDB. However, we are leaning toward PostgreSQL because it is an object-relational database which makes it easier to instantiate and store game sessions. This could be useful for searching game history by player, interacting with the game AI, and restoring crashed sessions. Further, PostgreSQL is better-performing for concurrent reading/writing than our main alternatives (2), which could be critical for scaling to facilitate many matches at once.

To create matches, users will be able to send public invites to other users, or private invites to specific users. In our current architectural plan, these invites will be written to the database, perhaps with critical information about how to connect a game, such as the port number of the web UI session originating the invite. Web UI sessions will periodically query a list of invites from the database, which will refresh in the user's GUI. This system would be able to facilitate matches between several web UI, with the main constraint being the compute resources allocated to running web UI sessions (and AI players).

To facilitate games, we could potentially take a few different approaches. First, one web UI session could

"host" the game, storing the moves and game timer in memory, receiving moves from the opponent's web UI session, and also pushing updates to the opponent. We could also potentially have the database server as the host, with every move and the game clock being written to it while the game is in progress. The

viability of this approach would largely depend on latency. Finally, if we find that these approaches are infeasible or difficult to implement, we could switch to an event-driven networking model, utilizing a "master" node, as in our original architecture. The master would facilitate game sessions between users and store game state independently of web UI sessions, like the database as described above, but in the node's memory. The master would also track connected web UI users and game invites in memory, and be an alternative to serve as the "host".

When a game is finished, the game will be saved to persistent storage in the database, and player rating changes will be computed, likely in the WebUI backend. We currently also intend to house the AI code on the webUI backend.

To provide the compute resources necessary to run multiple webUI sessions and AI players, we intend to allocate a number of "worker" nodes to the webUI. Ideally, we could combine multiple hardware nodes into one virtual machine for simplicity, perhaps using Kubernetes, Linux Virtual Server, or another option easily deployable on Cloudlab.

Most of our backend code could likely be implemented in Java and SQL.

### Key Backend Features

| **Functionality** | **Potential Implementation** |
|---|---|
| Check that moves are legal before allowing execution. | This legality check could run on the WebUI backend, which would be optimal for reducing lag. A function would take in the current game state and proposed move and ensure it is in compliance with all rules. |
| Send moves to database, opponent webUI session, AI. | Once a legal move is executed, it needs to be sent to the opponent through one of the means described above. |
| Send and receive game invites. | When a user creates a game invite, it must be pushed to all other users with viewing permissions. Similarly, the WebUI must be open to read/receive a list of open invites. Our current plan is to read/write these lists from the database. Entries could include fields specifying viewing permissions and webUI port numbers. |

| | |
|---|---|
| Keep registry of users connected to web UI sessions | The database will keep track of all users that have connected to the network with their login as well as the players that are connected and currently playing a game. |
| Push list of game invitations to users | When a game invitation is created, it will be |
| | written to a registry in the database, which other webUI sessions will regularly query. If the invite is public, all users will see the invitation. If the invite is to a specific user, only that user will see the invitation. |
| Track game state for in-progress games | The sequence of moves made so far in game will be written to the database in real-time, stored in the memory of a webUI session, or stored in the memory of a "master" node as in an event-driven model. |
| Timeout/auto-resign | Should a user have connection issues or disconnect from the application, there will be a small amount of time to allow the user to reconnect and resume play. When logging back in, the database will have a record of if the user is supposed to be in a game. The webUI can query this record upon login, and automatically connect the user if they are supposed to be in an inprogress game. Should the user not connect in time, they will automatically resign. |
| Initialize a game | When a game between two players is created via an accepted invite or a request to play against the AI, colors will be randomly assigned and the clock will be started. |
| Timers | The official time will be stored in the database or host WebUI session and pushed to players on a regular basis. |
| Write results and updated player stats to database | At the end of the game, the game could be encoded in a file format such as FEN or PGN and written to the database. Players' game history could also be updated. |

| | |
|---|---|
| Update player ratings (tentative) | We could implement an elo rating system where at the end of every game the master calculates rating updates and writes them to the database. |

### AI Chess Player

We'd like the user to have the option to play against an AI chess player should they want to. Implementing an AI is an important stepping stone for us in this project because it's very likely that there will not be any users to matchmake with at times. Having the AI player will allow these users to use the site during these slower times.

Behind the scenes, this AI will receive a game state and return what it believes to be the best possible move for each board position. A feature like this could be implemented in a couple different ways. We could bite the bullet and implement our own AI using the well-known minimax algorithm. This is a recursive or backtracking algorithm used in decision-making and game theory. It provides an optimal move for the player assuming that the opponent is also playing optimally. Alternatively, our other option is utilize open source options that already exist such as Stockfish to do this task for us.

Stockfish is one of the most well-known chess engines available to the public and it's considered one of the strongest as well. Many chess players consider the engine to be stronger than human players at the game. It would be able to provide better moves for board positions than our algorithms could and it would greatly increase the quality of the user experience.

In either case we would need to implement fine tuning the AI difficulty to allow users of all skill levels to play against the AI. This would likely involve giving it a probability of playing varying degrees of suboptimal moves.

We are currently leaning towards using Stockfish, as creating our own AI would be way too timeintensive for this project. We've decided to consider adding our own AI algorithms as a "stretch" goal for us to experiment with once other key functionality is complete.

In terms of deployment, we currently plan on hosting the AI in the webUI backend.

## Chapter 3 – Intermediate Milestones

As of April 17, we have implemented the foundation of our web application's infrastructure. The front-end consists of a chess user interface found on GitHub. Since we have also built out all the supporting files for the Kubernetes clusters, we are able to get the game interface up, running, and functional on the Kubernetes branch of our project. Additionally, the Stockfish AI image and Dockerfile are built out.

We have run into a number of challenges with getting the front-end to work and in building out the continuous integration and continuous delivery with Jenkins. The user interface only worked on the Kubernetes branch if each command was executed one-by-one. When we ran a bash script, the same error consistently occurred when pushing the docker images: connection to port 30000 refused. We addressed this by adding sleep commands in the shell script to allow time for longer commands to execute. Challenges with the CI/CD revolve around configuring Jenkins to support our project's specifications. We found that the Golang server template provided requires significant modifications to accommodate our multi-pod Kubernetes cluster.

Our primary goal looking forward is to get the CI/CD pipeline functional with our project's specifications, allowing us to more rapidly and efficiently iterate in our development cycle. Next in priority is building out front-end features such as a menu system, and supporting chess games between multiple users and/or an AI agent. Once we have the barebones functionality of the application established, we can move onto more intricate features such as game clocks, pages for viewing players stats, ad placement, and other bells and whistles.

## Kubernetes Cluster

As of this deliverable, we've been successful in running our application on a Kubernetes cluster deployed on Cloudlab, and accessing it through our head node URL. We were able to implement Kuberntes .yml files for deployment of our pods and services by following the format in ram_coin. We were able to do the same for our docker-compose files. Although our app has three pods, docker-compose and Kubernetes allow for build and deployment with empty docker files, which we have leveraged to test the webUI without our other pods being ready for deployment yet. As the AI is not quite ready for integration with the front-end, we have not yet included it in our configuration files. Further, we have not yet implemented the database, so its Dockerfile is currently blank. The webUI is fully configured for the Kubernetes cluster. We were able to borrow and make slight modifications to the ram_coin Docker and Kubernetes configuration files for the webUI, as ours is also a Node.js application. We were first able to successfully test our cluster by following the commands in the Kubernetes lecture, replacing ram_coin's git repo and Kubernetes services with ours, and logging into the webui port on the Cloudlab head node. After successfully viewing a message we printed to the console, we pointed the web UI at an index.html file, and were able to begin building it out from there. We have since converted the commands necessary to deploy the cluster to a bash script.

## Jenkins CI/CD

Our work on configuring Jenkins for our application has mostly consisted of adapting the Jenkinsfile for the example golang server from "Kubernetes Application: CI/CD pipeline - Part II" to suit our Kubernetes cluster, as well as the pursuant changes to the setup in the Jenkins GUI. We were successful in deploying the application with Jenkins by porting over the commands from the aforementioned bash script for launching the cluster into the Jenkinsfile. However, this deployment only works once – once the services are up and running we don't yet do anything to reset them, thus running another Jenkins build tries to over-write our already up-and-running services, causing an error. We will have to adapt our commands and overall pipeline configuration to update the services with new source code rather than attempting to re-initialize them without properly killing them.

## WebUI

The user interface currently consists of a simple chess board with moving pieces, valid moves, and move suggestions. It is a node.JS application that we found here. Since an existing framework exists that has the essential functionality we require for a chess game, there was no need for us to build one from scratch. Once the database and AI are built, we will be able to post and get information to and from each container. Our goal is to add more features once the infrastructure is completely built out (i.e., game clocks, player stats, etc.). We would like the user to start at a login page, jump to a dashboard where they can create new game sessions and observe their performance statistics, and finally access a page for the live game session.

## AI

We currently have a Stockfish Docker image set up and a dedicated container for it. We've automated the compilation of the Stockfish source code and made it seamlessly run Stockfish once compiled. Although, right now we're working on a solution that will allow us to pipe input into the Stockfish console and a solution that will allow us to pipe the output to another one of our containers on our Docker network. We believe the most efficient way to fix this issue is to use HTTP get / post commands to communicate between containers. Once we've resolved this issue we can continue to add features that we originally wanted to implement. Features such as AI difficulty tuning and saving completed games to a database. Overall, we're making swift progress with Stockfish and are continuing to keep it a part of our project.

### Database

We have yet to configure the PostgreSQL database. Our current webUI framework stores game states in memory, so the database will tentatively only be necessary for persistent storage of games and player stats. Once implemented, we plan on storing players' game history, credentials, and elo rating. The Web UI will query and write to the database using HTTP get/post.

## Chapter 4 - State of the Project

In summary, our project currently features:

- A React.js Web UI (http://155.98.37.68:30091) with login and user registration pages, and an interactive drag-and-drop chessboard with rules enforced. The user can play against an opponent that makes random moves. The chessboard is currently accessible locally, but in the UI deployed on the Kubernetes cluster only the login and registration pages are accessible, as full database functionality is required to get past these pages.
- A Node/Express.js Web UI (http://155.98.37.68:30088) with login and user registration pages, and an interactive point-and click chessboard GUI with gameclocks. This was our initial architecture, but we transitioned to a new React.js architecture using Chess.js and Chessboard.js to more effectively integrate multiplayer, a database, and communication with stockfish.
- A Jenkins server/Kubernetes cluster (http://155.98.37.68:30000). In our cluster we have the following pods/pipelines:
    - AI
        - Includes initial dockerfile for stockfish and Python script for sending and receiving moves to frontend via http – this is incomplete.
    - Backend
        - Includes the backend for the webui that interfaces with the database and handles user account creation and authentication.
    - Database
        - Placeholder for a database pod – not in use.
    - DB
        - Includes PostgreSQL database storing user data – designed to communicate via HTTP with "Backend" pod.
    - Frontend
        - Includes React.js user interface with login/registration pages and chess GUI - Designed to communicate with "Backend" via HTTP.
    - WebUI
        - Hosts initial Express.js/Node.js Architecture – does not interact with other pods.
    - WebUI2

- All-in-one version of “Frontend”, “Backend” and “DB”. We have since separated these into their own pods.

## Features from Chapter 2:

| **User Actions** | **Current State** |
|---|---|
| Create/update/delete account | Currently we have the ability to create an account that a user can use to login to the Kube Chess application in a local setting. We can also have this running in our pipeline, however; we are still having difficulties communicating from our webui to our database so when we go to save the account, no data is stored and the login page is shown again. Due to the lack of a stable database connection, updating and deleting accounts are still not functioning. |
| Login | Currently, when a user connects to the application, they are greeted with the login page. Since our database again is not functioning correctly with our webui, unfortunately we cannot fulfill a working login page. We have the ability to use the login page without the use of our database working as a proof of concept, so once we make the connection this feature should be working smoothly. |
| View stats | The ability to track stats and allow users to view them was definitely one of the later features we were looking to implement. At this time we have not given much thought to stats other than planning out our database to store game data from previous matches. |
| Create invitations and join matches | We looked at socket.io as a library to implement multiplayer sessions but did not make progress because we prioritized building the pipeline. |
| Move pieces via point-and click/drag and drop | The first iteration of our webui utilizes a point-and-click system. Our new React.js webui using Chessboard.js has a drag-and-drop system. |
| Game clocks | This feature is implemented in the first iteration of our frontend using Node and Express.js (http://155.98.37.68:30088). We have yet to implement this in our new React.js webui. |
| Resign/leave a match | This is not yet implemented in any of our UIs, as we did not get to the point of implementing network-based multiplayer. |
| Check that moves are legal before allowing execution. | This works through the chess.js framework. |

| | |
|---|---|
| Send moves to database, opponent webUI session, AI. | The database only stores user authentication information at this point. We need to figure out how to communicate between the pods to make the database robust enough to store gamestates. We have the ability to get moves from the Stockfish AI that we are planning to use, but at this time cannot pass its moves to interact in the gamestate. |
| Send and receive game invites. | Multiplayer is not currently supported. |
| Keep registry of users connected to webUI sessions. | Multiplayer is not currently supported. |
| Push list of game invitations to users. | Multiplayer is not currently supported. |
| Track game state for ini-progress games. | The database only stores user authentication information at this point. We need to figure out how to communicate between the pods to make the database robust enough to store gamestates. |
| Timeout/auto-resign. | This is not currently supported. We do not yet have game clocks in the react version of the application. |
| Initialize a game. | This works by clicking the Play Now button in the player dashboard. |
| Timers. | This feature is implemented in the first iteration of our frontend using Node and Express.js (http://155.98.37.68:30088). We have yet to implement this in our new React.js webui. |
| Write results and updated player stats to database. | The database only stores user authentication information at this point. We need to figure out how to communicate between the pods to make the database robust enough to store gamestates. |
| Update player ratings. | The database only stores user authentication information at this point. We need to figure out how to communicate between the pods to make the database robust enough to store player ratings. |

## Self-Evaluation

The project has not met the technical requirements we initially set out to achieve; however, the foundation is laid for the bells and whistles to be added down the line. We have a working Jenkins pipeline which allows us to modify the application without significant downtime. We also have our frontend and backend in separate K8 pods, but they do not yet communicate with each other effectively. Once the API requests are routed to the correct address, we can build out many of the features defined in the table. For example, storing game states is not possible without effective communication between the frontend game session and the database. This also prevents us from changing account information, deleting an account, logging in, and tracking user statistics. It is important to note that once a user is authenticated, they have access to the dashboard and the ability to initialize a game session, but this is not possible without effective communication between the frontend and backend pods. We do have banner ads implemented in the game session, and adding a game clock to a game session would require little technical effort. Multiplayer functionality is one aspect of this project we did not address. We researched using Socket.io and a server-client model to facilitate game sessions, but never got to focusing on this, as we were primarily concerned with building the Kubernetes cluster, Jenkins pipeline, and the app's fundamental functionality. Multiplayer temporarily became a "nice-to-have" feature for down the road. The same applied to Stockfish and integrating an AI agent – we prioritized fundamental functionality over this. The largest obstacle to implementing these additional features ended up being time. Further, we were all new to Javascript web development, so our first try at developing a webui ended up being unscalable. The initial architecture was not very adherent to the "open-closed" principle of software development, so extending it to support multiplayer via socket.io and communication of chess game states over network with another player or the AI would have been cumbersome. It was easier and allowed for more scalability of the architecture to restart with React. However, this also meant spending a considerable amount of time figuring out how to re-implement functionality in a new framework. Overall, we felt that our team made reasonable progress towards our goals given our time constraints.

# References

1. https://github.com/trushpatel/CSC468Group5.git

2. https://github.com/tudorfis/chess
3. SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Syst

# Kevin Codd

## Data Scientist // Software Engineer

kncodd@gmail.com
+1 (484) 947- 9711
github.com/KevinCodd
linkedin.com/in/kevin-codd
Greater Philadelphia

## EXPERIENCE

**Data Science Intern** — Jun 2021 – Present
CellCarta — Remote (Fremont, CA)

- Researched and improved upon multivariate time series outlier detection algorithm for Flow Cytometry data
- Developed data pipelines in Python and R for comparative evaluation of outlier detection algorithms on many datasets
- Contributed to R API toolkit for CellEngine data analysis software

**Data Science Intern** — Feb 2020 – May 2021
nth Solutions — Exton, PA

- Developed Java backend for IMU sensor data visualization application
- Modelled time series data using several Python and MATLAB libraries for state space modelling, deep learning, and frequency domain analysis
- Researched and implemented IMU sensor fusion algorithm for accurate orientation tracking
- Led team of interns responsible for data engineering and analysis tasks

**Undergraduate Research Assistant** — Spring, Fall 2021
West Chester University — West Chester, PA

- Analyzed break-in attempts to Computer Science department's servers using Linux authentication logs and PySpark in support of academic research paper
- Cleaned, parsed, and analyzed logs using PySpark to identify and quantify attacks
- Currently developing backend for data-streaming dashboard using Spark and Kafka

**Regulatory Affairs Intern** — May 2018 – Nov 2018
Lungpacer Medical — Exton, PA

- Organized clinical trial data and produced reports analyzing statistical trends
- Conducted scientific literature searches to substantiate efficacy of medical device treatment approach
- Proofread and edited technical documentation to be submitted to regulators

## EDUCATION

**B.S. in Computer Science & Minor in Mathematics** — Expected Dec 2022
West Chester University

- **GPA**: 3.95 / 4.0
- **Relevant Courses:** Big Data Engineering, Mathematical Statistics, Artificial Intelligence, Software Engineering, Data Structures & Algorithms, Computer Systems, Linear Algebra, Multivariable Calculus

## SKILLS

- **Languages:** Python, Java, R, C, MATLAB, SQL, SAS
- **Tools/Frameworks:** *Data analysis/engineering*: NumPy, SciPy, Pandas, Matplotlib, PySpark, Kafka. *Machine learning*: Scikit-learn, TensorFlow. *Development*: Git, Jupyter Notebook, Anaconda.

## ACTIVITIES

**Competitive Programming Club** — Aug 2020 – Present
West Chester University

- Club President (Aug 2021-Present) - Facilitated club meetings and competitions, created and curated practice material
- Algorithmic programming contests: member of team advancing to 2021 ICPC North American Division Championships, 3rd place team out of 24 in CCSC Eastern 2020 Programming Competition

# Matthew C DiStefano

13mdistefano@gmail.com

*805 Windridge Lane | Downingtown | PA | 19335 | tel: 610.247.8772*

## Education

**West Chester University of Pennsylvania**

Bachelor of Science in Computer Science

3.64 GPA | Expected to graduate May 2022

**Relevant Courses**

Computer Systems, Data Structures and Algorithms, Calculus, Database Management Systems, Data Communication and Networking, Edge Computing and Deep Learning, Software Testing, Intro to Cloud Computing, Digital Image Processing

## Professional Experience

**Law Clerk | June 2014 – August 2017**

**Dalton & Associates, LLC | Wilmington, DE**

– Managed and maintained the systems for closing of files and server storage

– Maintained and kept track of client's medical bills and records in accordance with HIPPA standards for ease of access by attorneys and clients alike

– In charge of maintaining files within our cloud system

**Law Clerk | June 2019 - Present**

**Potamkin ARM, LLC | Downingtown, PA**

– Keep over 170 entities up to date and manage documents that are to be kept in the minute books

– Maintained timely communication with fellow employees to better identify and resolve issues

## Computer Science Projects

**CSC496 Final Project | West Chester University | December 2021**

– Lead a team to the creation of a graph based shortest path algorithm used to determine the earliest semester a specified Computer Science course could be taken at WCU

## Skills

– Languages: Java, JavaScript, C#, Python, SQL, HTML, OCaml

– Tools/Frameworks: Git, Node.js, React.js

# Patrick Lawrence

West Chester, PA
(610)-675-8651
patrickjameslawrence@icloud.com

## Education

West Chester University of Pennsylvania
*Bachelor of Science in Computer Science* | May 2022

- 3.38 Cumulative GPA

## Related Experience

*IT Helpdesk Consultant* | December 2020 – Present
West Chester University of Pennsylvania | West Chester, PA

- Identify user issues and dispatch them to appropriate teams when needed
- Educate fellow consultants on internal IS&T functions and solutions to issues users face
- Maintain the IS&T knowledge base; adding articles to pass down my expertise when needed
- Maintain cordial communications and de-escalate situations with bad-tempered users

*Software Engineer Intern* | June 2021 – July 2021
Universal Health Services (UHS) | King of Prussia, PA

- Built new FTP connections to transfer patient records, test results, and other data between UHS network hospitals
- Utilized SQL and C# to create applications in .NET
- Maintained adherence to all HIPPA privacy standards and protected patient data
- Shadowed UHS software engineers in their meetings and projects
- Reviewed standard UHS operating procedures, all internal IS&T teams and their functions, and the software development lifecycle
- Learned office etiquette and best practices from other employees in the office

## Skills Summary

- *Programming Languages:* Java, C++, JavaScript, C#, SQL
- *Development Tools:* Version control & bug tracking
- Heavy interest in advanced mathematics

## Projects

*Synth* | September 2020 – March 2021
Multi-function Discord bot | Node.js and MySQL

- Created a robust command and event handler
- Designed a local MySQL database to store various configuration information from servers and users

*Keychain* | October 2020
Social media sharing service | Android Studio and MongoDB

- Designed an Android application using Android Studio
- Utilized database services to create a secure account login solution

# Trush Patel

(267)-261-9675 | trushp3@gmail.com | linkedin.com/in/trushpatel | github.com/trushpatel

## Education

**Georgia Institute of Technology** — Atlanta, GA
*Master of Science in Computer Science* — *Expected August 2024*

**West Chester University of Pennsylvania** — West Chester, PA
*Bachelor of Science in Computer Science* — *May 2022*

- GPA: 3.99/4.00

## Experience

**Software Engineer Intern - Product Team** — February 2022 – Present
*humanID* — *New York, NY*

- Developing anonymous authentication solutions for secure single sign-on
- Leading Scrum teams to consistently deliver new features and upgrades
- Constructed concise documentation to accelerate client integration by 50%
- Building a developer dashboard to manage client integrations and aggregate relevant data

**Tutor - Learning Assistance & Resource Center** — December 2021 – May 2022
*West Chester University of Pennsylvania* — *West Chester, PA*

- Organized one-on-one meetings to assess comprehension and teach fundamental concepts
- Instructed the development of effective time management and test preparation strategies
- Tutored undergraduate students in Introduction to Statistics, Foundations of Computer Science, and Data Structures and Algorithms
- Acquired CRLA Level I certification for professional tutors and peer educators

**Undergraduate Research Assistant - Dr. Jongwook Kim** — October 2021 – May 2022
*West Chester University of Pennsylvania* — *West Chester, PA*

- Researched software design patterns and refactoring paradigms
- Assessed program transformations that augment software reliability and extensibility
- Conducted tests to determine the efficacy of refactoring tools in programs with 10 to 10,000 lines of code
- Refactored a program that reduces the execution time of existing refactoring tools by 35%

## Projects

**KubeChess** | *PostgreSQL, Express.js, React.js, Node.js, Docker, Kubernetes, Jenkins, Ansible* — May 2022

- Collaborated with 4 developers to build an online playground for chess players
- Implemented a React.js front-end and RESTful backend that uses Express.js, Node.js, and PostgreSQL
- Deploying with Kubernetes to scale for multiple users and concurrent sessions
- Implementing a Jenkins pipeline to support CI/CD

**Yelp Dataset** | *Python, SQL, Apache Spark* — December 2021

- Built an algorithm that implements PageRank to identify influencers in a 10-gigabyte dataset
- Computes the top 5% of influential users from quantity and quality of reviews and connections
- Extracted each user's review data to identify individual preference in type of places reviewed
- Pinpointed reviewed business locations to cluster users geographically

## Honors & Awards

**Upsilon Pi Epsilon** | *International Honor Society for the Computing and Information Disciplines* — May 2022
**3rd Place** | *PACISE Regional Programming Competition* — December 2021
**Dean's List** | *West Chester University of Pennsylvania* — December 2020 - December 2021

## Technical Skills

**Languages**: Java, Python, C/C++, JavaScript, HTML, CSS, OCaml, Bash, SQL
**Tools/Frameworks**: Apache Spark, Bootstrap, Django, Docker, Git, Jenkins, Kubernetes, Node.js, React.js
**Coursework**: Artificial Intelligence, Big Data Engineering, Computer Systems, Data Structures and Algorithms, Introduction to Cloud Computing, Linear Algebra, User Interfaces, Software Engineering, Software Testing

# Morlai Ishmail Koroma

106 Norma Road, Yeadon PA 19050 ishmailkoroma1@gmail.com (267) 515 2702

## EDUCATION

**West Chester University** Expected Graduation Date: May 2022
700 S High St, West Chester, PA 19383
Major: Computer Science

## TECHNICAL SKILLS:

- Proficient in JavaScript, C++, Python, HTML, Linux, and Microsoft Office

## PROFESSIONAL EXPERIENCE:

**U.S. Navy** **Naval Research Enterprise Intern**
Philadelphia, PA October 2021 – December 2021

- Researched and provided recommendations and new methods for increased monitoring, reducing maintenance, and repair of critical naval machinery systems on Navy ships.
- Provided a possible application of LoRa sensors on the Lube Oil System on Navy ships to detect water leaks, maintain temperature pressure and reduce the maintenance cycle for sailors.
- Developed a cost analysis framework for the materials used in the implementation of the LoRa sensors on the Lube Oil System.

**U.S. Navy** **Naval Research Enterprise Intern**
Philadelphia, PA June 2020 – August 2020

- Researched methods to enforce LoRa (long-range) technologies on Naval platforms without compromising security.
- Assessed the LoRaWAN (long-range, wide area network) vulnerabilities and cyber security risks involved in implementing long range sensors on Naval platforms

**Avista Healthcare** **Dietary Aide**
Cherry Hill, NJ August 2019 – March 2020

- Monitored inventory and stock for kitchen ingredients
- Maintained kitchen equipment and appliances via guidelines of the facility's dietary manager and kitchen cook
- Oversaw take-down of dining areas, collecting all silverware, discarding of leftovers and removal of garbage and recycling items.

**Sky is the Limit (Moving Company)** **Moving Assistant**
Greater Philadelphia June 2018 – May 2019

- Tracked payments for services
- Kept detailed inventory of customer items to ensure safekeeping and integrity of items moved.
- Kept track of communication and interaction with all customers via emails, texts, and calls

### COMPUTER SCIENCE PROJECTS:

**CSC 472 Final Lab Project** **Lead Member**
West Chester University November 2021 – December 2021

- Performed a successful multi-stage exploit attack on a file to get and reveal content inside the target flag.
- Completed format string, ROP, GOT overwrite, and stack overflow attacks on the program to leak specific information.

### CLUBS & VOLUNTEER EXPERIENCE

**WCU Weekly** **Videographer/Editor**
West Chester University August 2020- December 2020

- Filmed highlights at West Chester University sports games and helped edit weekly sport segments

**Christ The King Prayer Chapel.** May 2018 to August 2018
Philadelphia, PA

- Filmed services.
- Helped organized the church and collected donations/offerings from members

### INTERESTS:

- Cybersecurity; Network and Computer Systems; Artificial Intelligence; Computer Graphics; Video Games; Basketball; Football