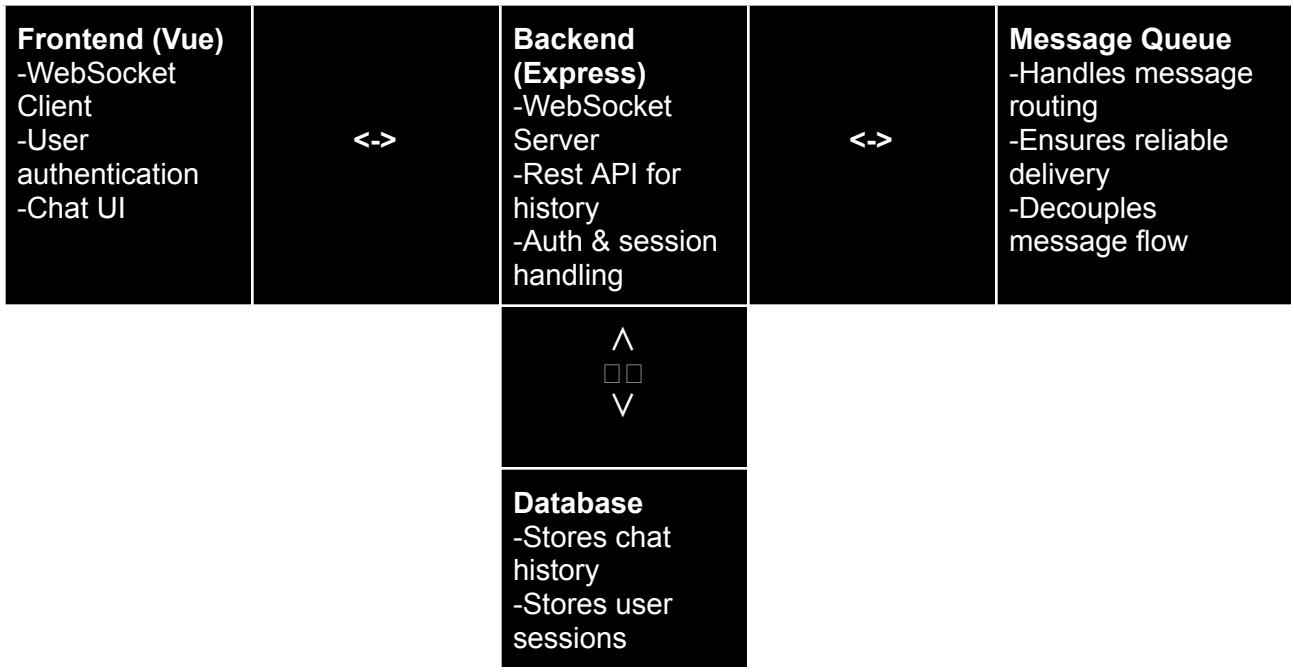
Hypervisor Enhanced Logistics Program (H.E.L.P)

Granjeon, Lucas
Ayub, Umer
Lebo, Drew

Chapter 1

Many businesses require a scalable and efficient help desk solution for customer support. The project covers critical topics such as WebSockets, message queues, and distributed deployment. Using a message queue allows for handling high volumes of messages efficiently. This project aims to develop a real-time chat application designed as a helpdesk platform. It will feature a responsive frontend using Vue.js, a backend powered by Express.js with WebSocket support, and a message queue system to ensure reliable communication.



The frontend is developed using Vue.js and runs inside a Docker container with an Nginx web server to serve the application. It includes a real-time chat interface, user authentication, and a responsive UI. The frontend communicates with the backend using WebSockets for real-time chat updates and REST API calls for retrieving historical data. This design allows the frontend to deliver dynamic user interaction while maintaining efficient communication with other services.

The backend, built with Express.js, is responsible for user authentication, session management, WebSocket connections, and RESTful APIs. It runs inside its own Docker container for isolated deployment and scalability. To support real-time message delivery and decouple services, the system also includes a message queue implemented using RabbitMQ or Kafka, hosted in a separate container. The message queue handles message routing, guarantees reliable delivery, and reduces the processing load on the backend. This combination ensures asynchronous

communication between components and improves system performance under heavy usage. The message queue would also run in a separate docker container

Finally, the system includes a database running on MongoDB, also in its own Docker container. It stores chat history, user sessions, and authentication data. Isolating the database ensures data persistence and security while enabling fast, indexed queries for user and message information. This setup provides a solid data layer for the application and supports the scalability of the overall system.

Chapter 2

System Architecture

The architecture of the H.E.L.P. platform is designed using a containerized microservices approach to support scalability, maintainability, and real-time communication. The frontend is built with Vue.js and runs inside a Docker container served by an Nginx web server. It performs REST API calls to fetch historical chat data and uses WebSockets to handle live chat updates. The backend, developed using Express.js, is also containerized and is responsible for handling user authentication, session management, RESTful API requests, and WebSocket connections.

To enable asynchronous communication and ensure reliable message delivery, the system uses a message queue running in a dedicated Docker container. This queue allows users to exchange messages without requiring a persistent direct connection between the frontend and backend. Finally, a MongoDB container is used as the primary data store for the platform. It stores user information, session data, message history, and support tickets. This separation of services into distinct containers allows each component to scale independently and simplifies deployment and maintenance.

Database Design

Utilizing MongoDB, a document-based structure, having efficient collections is necessary. The primary focus is to have indexes for quick lookups and securely stored data for efficiency.

Users Collection

The users table should contain a handful of information, such as the table below. The biggest concern about the table is a security concern, which is why storing the password as a hash is necessary. Utilizing **bcrypt**, a securely stored password can be achieved.

Another concern is quick lookups, which is why the username and email indexes should be utilized.

_id	**username**	**email**	**password_hash**	**role**	**created_at**	**updated_at**	**status**
ObjectId	string	string	string	string	ISODate	ISODate	string

- **Role** can be admin, support_agent, or customer
- **Status** can be active, inactive, or banned

Sessions Collection

This collection stores all the information for user session tokens. Quick lookup IDs are included for the user's id and the session token. The session token should be signed and stored securely

_id	**user_id**	**session_token**	**expires_at**
ObjectId	ObjectId	string	ISODate

Messages Collection

This collection is a crucial security point in the application. The primary concern of this table is input sanitization to prevent attacks via NoSQL Injection. For quick lookups we have the sender and receiver ids and the timestamp.

_id	**sender_id**	**~~receiver_id~~**	**content**	**timestamp**	**status**
ObjectId	ObjectId	~~ObjectId~~	string	ISODate	string

- The status string can be sent, delivered, or read

Tickets Collection

Like the messages collection, this collection is also a crucial part. Depending on the volume of data, it might be important to consider **message sharding**, but for the time being the following table should be sufficient. The quick lookup indexes are the customer id, the agent id, and the status.

_id	customer_id	agent_id	status	priority	messages	created_at	updated_at

ObjectId	ObjectId	ObjectId	string	string	[
					{
					
					“sender_id” : “ObjectId”,
					
					“content” : “string”,
					
					“timestamp” : “ISODate”
					}
					]
						ISODate	ISODate

- **Status** string can be open, pending, resolved, or closed
- **Priority** string can be low, medium, high, or urgent

Security Considerations

Authentication and Authorization

To handle the login, user roles, and session tokens with optimal security, a few technologies will need to be implemented.

- JWT-based authentication
    - Provides a stateless login
- Role-based access control (RBAC)
    - Restricts access
- Session Expiration Policy
    - No user should be capable of being signed in forever.

Data Protection

To handle data security, the following technologies are crucial.

- Password encryption
    - Using bcrypt
- Sanitizing data input
    - Prevents NoSQL injection

For future reference, using HTTPS for all communications and Rate-limiting are also ways to improve security.

WebSockets Security

The primary importance of security for WebSockets is connection authentication. Without a WebSocket connection, the session should not be established. The WebSocket should also ensure message size limits and message format validation are implemented to prevent abuse.

Message Queue

Considering the unsureness of which technology to use between Kafka and RabbitMQ, here's a breakdown comparing the two.

Feature	**RabbitMQ**	**Kafka**
Message Model	Queue-based (push)	Pub/Sub (pull-based)
Message Order	Guaranteed per queue	Ordered within partitions
Persistence	Durable Queues (if enabled)	Logs messages by default
Scalability	Good for small workloads	Better for high throughput
Use Case	Real-time chat, event-driven	Large-scale streaming

Given the comparisons, it seems RabbitMQ is better suited for the needs of this project. Since the project is real-time chat and event-driven, RabbitMQ makes more sense, especially since it seems to be easier to set up. RabbitMQ will be the choice for this project due to those reasons.

An example workflow for the message queue could look like: Frontend sends message via WebSocket -> Backend publishes message to a RabbitMQ exchange -> Consumer service (express.js application) pulls message from the queue -> Message is stored in MongoDB and forwarded to the recipient.

Deployment Strategy

The frontend (Vue) and the backend (Express.js + WebSockets + REST API), RabbitMQ, and MongoDB, each run inside of their own Docker container.

Chapter 3

Containerizing the application using Docker is a core focus of this project. By leveraging Docker, the system achieves a clean separation of concerns between the frontend, backend, message queue, and database components. This abstraction allows each service to operate in its own isolated environment, simplifying deployment, improving scalability, and ensuring consistent behavior across different development and production platforms.

Infrastructure

We use Docker for the isolation of components, while utilizing Docker Compose to streamline the deployment process and keep all containers within the same network and cluster. This setup ensures that the frontend, backend, message queue, and database can communicate efficiently while remaining modular. We mount volumes for the database, frontend, and backend to enable persistent data storage and support rapid deployment during development. These mounted volumes allow changes to be reflected immediately and ensure that important data, such as user information and message history, is retained across container rebuilds or restarts.

Deployment

We implemented executable scripts compatible with both Mac/Linux and Windows systems to simplify the deployment process. These scripts are designed to remove any previously built Docker images and then execute Docker Compose to build and run the containers. During development, an issue arose due to inconsistent line endings in the scripts created on Windows, which caused errors when deploying the project on macOS and Linux machines. This problem was identified and resolved by adjusting the line endings to ensure cross-platform compatibility.

Docker Image Build Process

Each component of the H.E.L.P. is containerized using Docker and organized using Docker Compose. The backend uses a custom Dockerfile for both development and production. In development, it uses the node:latest image, installs dependencies, sets environment variables using build arguments, and runs the development server on port 3033. For production, the backend uses node:23-slim, installs only production dependencies, and runs the server with npm run start. The frontend is also built differently for each environment. In development, the container uses node:latest to run the Vue development server on port 5173 with the local source code mounted into the container. In production, a multi-stage Dockerfile first builds the frontend with node:23-slim and then serves the static files using nginx:stable-alpine. It includes custom Nginx configuration files and an entrypoint script to support both HTTP and HTTPS. MongoDB uses the official mongo image with a mounted volume at /data/db and runs on port 27017. RabbitMQ uses the official rabbitmq image and exposes ports 5672 and 15672. In production, Certbot is included to automatically renew SSL certificates every 12 hours, and Watchtower is used to monitor and update containers. All containers are connected through a shared bridge network, with separate Compose files for development and production to manage service configuration, dependencies, and port mappings.

Data Collection/Creation

As described in Chapter 2, the project defines four MongoDB collections: users, tickets, sessions, and messages. These collections are responsible for handling user data, support ticket information, session details, and message logs. The users collection stores essential

information such as names, emails, hashed credentials, and account roles. The tickets collection manages support ticket details, including their current status, assigned agents, priority levels, and timestamps. The sessions collection logs user activity by tracking login and logout sessions along with associated expiration times. The messages collection stores the content of conversations exchanged between users or system notifications, along with metadata like sender, receiver, and delivery status. These collections ensure that all application data is well-structured, queryable, and persistently stored to support real-time communication and user management. For a more detailed breakdown of the collection fields and structure, refer to the data tables in Chapter 2. As of now, these collections have not yet been implemented in the codebase and are planned for the next phase of development.

Express.js and REST API for CRUD Operations

We are using Express.js to build our REST API, which manages the CRUD operations for the four MongoDB collections: user data, tickets, sessions, and messages. Each collection has dedicated API routes to handle its data, using standard HTTP methods—POST for creating, GET for reading, PUT for updating, and DELETE for removing records. These routes are implemented within the Express.js backend and are designed to ensure smooth, consistent communication between the frontend and MongoDB. This structure supports efficient data handling and keeps the application organized and modular.

Chapter 4: Final Results

The project successfully implemented nearly all components originally outlined in Chapter 2. The system includes a functional frontend, backend, and database, all containerized and integrated using Docker. The frontend is complete and displays a working webpage that allows users to register, log in, and interact with a help desk interface. It features a clean user interface with dynamic updates, along with implemented styling, routing, and form validation to support smooth interaction.

The backend is operational and provides REST API endpoints for handling data related to users, messages, and tickets. It is structured to support secure communication with the frontend and connection to the MongoDB database. However, user authentication and session management have not yet been fully built into the backend, though the system is prepared to support them once implemented. The backend’s current implementation lays the groundwork for these features through its route structure and database schema.

While RabbitMQ integration was explored and partially configured, it was not fully completed and has been excluded from the final version. Similarly, WebSocket functionality, though started, remains non-functional and is not included in the current deployment. These represent the project's missed milestones but remain well-documented for future development.

Conclusion

Throughout the development of the H.E.L.P. platform, the group learned how to successfully containerize multiple components of the system using Docker and Docker Compose, allowing all services to run smoothly and in coordination. We also gained experience with tools such as Watchtower, for automated container updates, and Certbot, for managing SSL certificates—both of which added real-world functionality and security to the deployment. A fully functional frontend was created using Vue.js, complete with routing, form validation, and a clean interface. While the backend is operational and supports CRUD operations, user authentication and session management are not yet fully implemented and remain a key area for future work. The major feature missing is the WebSocket integration, which was started but not completed; adding this will enable real-time messaging, which is central to the platform's intended functionality. Additionally, although the RabbitMQ container is present and configured, it is not currently incorporated into the system and remains unused. Integrating RabbitMQ would enhance message handling and scalability, especially when paired with WebSocket communication.

Github Link

https://github.com/HELP468/HELP.git

LUCAS GRANJEON

CERTIFICATIONS

Occupational Skills Award for Information Systems
December 2022

Occupational Skills Award for Infomation Security
December 2022

Google Data Analytics Certificate (ongoing)

SKILLS

Hard:
Java - Python - C++ - VS Code - Github

Soft:
Communication - Problem-Solving - Motivation

LANGUAGE

French: Native
English: Fluent
Spanish: Conversational

HOBBIES

Fly Fisher
Car enthousiast
PC builder

Computer Science Student at West Chester University

@ granjeonl@gmail.com / Linkedin: Lucas Granjeon / West Chester, PA

SUMMARY

Motivated Computer Science student at West Chester University with a passion for Artificial Intelligence. I have a solid foundation in programming, algorithms, and data structures. Proficient in languages such as Python, Java, and C++, with hands-on experience in software development, problem-solving, and teamwork. I am eager to apply my academic knowledge to real-world challenges.

PROJECTS

CSC240 - Text Processing Project

-Developped a text processing tool in Java to analyze and extract key information from large text datasets.
-Designed the tool to efficiently handle diverse text formats, optimizing performance for large-scale datasets.
-Implemented document summarization and content-based categorization, reducing time required for text analysis.
-Created a user-friendly interface to navigate through the data gathered.

CSC301 - AI in Cybersecurity (Research Paper and Presentation)

-Researched and analized the role of AI and machine learning techniques in improving cybersecurity measures
-Developed a research paper exploring the use of AI for threat detection, anomaly detection, and intrusion prevention in modern systems.
-Created a presentation summarizing the findings, focusing on practical applications and emerging trends in AI-driven cybersecurity
-Delivered the presentation to an academic audience, effectively communicating complex technical concepts and their real-world implications.

EDUCATION

Bachelor in Computer Science — 2020-2021
Moorpark Community College — Moorpark,Ca

Associate in Computer Information System — 2021-2023
McLennan Community College — Waco,Tx

Bachelor in Computer Science — 2023-Ongoing
West Chester University — West Chester,Pa

Drew Lebo

(They/He)

215-539-7122 | andrewlebo2@gmail.com | linkedin.com/in/andrewlebo | github.com/ALebo5193

Education

West Chester University of Pennsylvania — West Chester, PA
Bachelor of Science in Computer Science — *August 2021 – May 2025*

- **GPA**: 3.349
- **Relevant Courses**: Data Structures & Algorithms, Software Engineering, Software Security, Data Communications & Networking, Computer Security, Programming Language Concepts & Paradigms, Introduction to Cloud Computing, Modern Malware Analysis, Artificial Intelligence

Experience

Spray Park Attendant — May 2023 – August 2024
Bristol Township Parks and Recreation — *Bristol, PA*

- Maintained water chemistry by keeping ORP between 750-770 mV and the pH between 7.4-7.6 to keep in ordinance with the department of health.
- Followed a structured maintenance schedule for routine filter upkeep and strainer cleanings, enhancing the team's workflow efficiency and resulting in maintenance operations being completed more efficiently.
- Consistently delivered timely updates to coworkers and department leadership, ensuring effective communication and swift information flow.

Dining Room Busser — June 2020 – March 2021
King George II Inn and Tavern — *Bristol, PA*

- Independently managed all bussing operations until March 2021, when a new team member was trained and integrated.
- Championed the safe handling and disposal of 200+ pounds of kitchen waste weekly.
- Streamlined operations to reduce table wait times and improve customer service efficiency.

Projects

Skill Tree Web Application | *React, Bootstrap, Node.js, Express.js, MongoDB/Mongoose* — February 2025

- Develop a full-stack web application with a dynamic front-end
- Build a RESTful API backend to support the use of a React front-end.

Hug Nugs | *Godot, GDScript* — Fall 2024

- Collaborated with a team to create a platformer video game
- Developed the player script.
- Patched player interaction bugs in the project during merges.

Sons of Nug | *Godot, GDScript* — Fall 2024

- Collaborated with a team to create a multiplayer video game
- Developed the targeting system for the towers.
- Handled multiplayer synchronization between clients and server.

Portfolio Website | *CSS, HTML, Javascript* — June 2024

- Created a visually appealing website with a reactive user interface that adapts to screen sizes.
- Implemented a navigation header bar with contents that contain hover conditionals.

Technical Skills

Languages: Java, HTML/CSS, JavaScript, MongoDB
Frameworks: ReactJS, Node.js, RESTful API
Developer Tools: VSCode, Docker, Git, Cloudflare, Linux
Algorithms: Merge sort, Breadth-First Search, Depth-First Search, Uniform Cost Search

Umer Ayub

623 Saxony Drive, Fairless Hills, PA, 19030
267-981-9752 // ayubumer79@gmail.com // LinkedIn: umer-ayub

Summary

Computer science student at West Chester University of Pennsylvania. Proficient in Java, Python, and C, with experience in software development, data analysis, and machine learning applications. Developed projects including a book recommendation system, an email spam detection classifier, and command-line pattern generation using object-oriented principles. Experience as a Computer Tech at West Chester University, maintaining classroom technology and providing IT support while also having strong leadership and communication skills as a Shift Manager at 7-Eleven managing inventory, cash flow, and staff training.

Education

West Chester University, West Chester, Pennsylvania — Expected May 2026
Bachelor of Science, Computer Science
3.93 GPA
Relevant Courses: Data Structures and Algorithms, Computer Systems, Intro to Cloud Computing

University of Maryland, College Park, Maryland — Sep 2021 - 2023
Bachelor of Science, Undecided

Pennsbury High School, Fairless Hills, Pennsylvania — June 2021

Skills

Programming Languages: Java, Python, C
Software Packages: PyCharm, JDK Java Compiler, VSCode
Languages: Proficient in German, Hindi and Urdu

Projects

Book Recommendation System — Jan 2025
Personal Project
- Developed a book recommendation system using python which compiled multiple statistics in order to deliver personalized book recommendations towards the user
- Managed book data using JSON files and implemented scripts for recommendations, focusing on recommendation logic

Email Spam Detection Classifier — Dec 2024
Object Oriented Programming, West Chester University
- Developed a program to determine if an email is spam by the criteria of duplicate words, usage of URLs and word count using logistic regression
- Created multiple classes such as EmailDataset and Spam, using object-oriented principles for data-management and extensibility

Calculator June 2024

Personal Project

- Created a Java based calculator which is capable of performing addition, subtraction, multiplication and division
- Implemented object oriented principles as well as use of multiple classes
- Incorporated exception handling in cases of division by zero and invalid inputs

Command Line Pattern Art Feb 2024

Computer Science Principles, West Chester University

- Developed a program to output a specific pattern in terminal
- Utilized java abstraction and for loop logic to simplify the design

Experience

Computer Tech September 2024 - Present

College of Education and Social Work Tech Center, West Chester University

- Conducted maintenance and troubleshooting on classroom technology, ensuring smooth operation of devices such as projectors, computers, and other equipment
- Maintained inventory of thousands of technology items for the college and distributed by request to staff and students
- Assisted staff with technical issues, providing support for both software and hardware problems for both instructors and students

Shift Manager December 2020 - Present

7-Eleven

- Audited thousands of dollars' worth of stock to make sure everything was properly accounted for in the system
- Conducted daily cash reports in order to handle the thousands of dollars of daily sales
- Managed and trained multiple employees: ensuring proper use of the POS system and how to handle customer needs
- Increased sales from this location by 5.7% from the years 2021 to 2022