Smart Helmet Rental System Using Iot and Web Technologies

Surekha Byakod[1], Vaishnavi S[2], A S N Sreeya[3], Savya Shree S[4], Deepthi G B[5]

[1]Associate Professor, Department of Computer Science & Design, K S Institute of Technology, Bengaluru, Karnataka, India.

[2,3,4,5]Student, Department of Computer Science & Design, K S Institute of Technology, Bengaluru, Karnataka, India.

Abstract

In this paper, we present a smart helmet rental system based on web and IoT technologies to improve the safety of the road and for the convenience of the user. The system was developed using Angular on the front end and Spring Boot on the back end. Users can register using their phone number and rent helmets in a secure manner. When the user requests the helmet, a unique code is generated based on the requested time, and this code will unlock the lockers that are governed by ESP32's. When the user returns the helmet, the code will be reused. The ESP32 will internally track how long the user rented the helmet and automatically deduct the time taken from the user's wallet, which has a balance of ₹1000. Lastly, in this paper, we discuss the design, flow, and implementation of the proposed system.

Keywords: Helmet Rental, IoT, Angular, Spring Boot, ESP32, Web Application, Wallet System, Locker Management

1. Introduction

Many urban populations of two-wheeler riders do not use helmets and thus often suffer severe injuries in accidents. As education increases in road safety, access to easy and low-cost helmet solutions remains a problem. The Smart Helmet Rental System intends to introduce an applicable technology solution and assist a growing body of riders in helmet use while making the rental of helmets automated and more simplified. This Smart Helmet Rental System has combined IoT hardware with new front-end and back-end web technologies that allow for the rental of helmets in a simple and secure manner - in just a few clicks through the web application. The application is built using Angular for the front-end and Spring Boot for the back-end, which provides a wonderful responsive and sturdy framework for the application. The user logs in through their mobile phone number with the application, and when logged in, the user is presented with a dashboard for the rental process. Once the user presses the "Get Code" button to receive a code, the system generates a new code and opens a specific locker controlled by an ESP32 that the smart helmet is in. Only authenticated users can access the helmets in the locker, adding a system of security. The same code is reused while returning the helmet, and the system will then be able to account the duration of usage correctly. Rental charges are calculated based on the duration when the helmet is returned, and the needed funds will be deducted from the user's wallet. The wallet should contain at least

₹1000 to start the rental, so that the users have enough funds to cover the rental price. Overall, the smart helmet rental system is a scalable, real-time, and secure measure to promote safety, give better access to helmets, and use a modern approach for distributing these helmets to the public. The combination of hardware and software components reflects the growing trend of combining IoT within smart city applications.

2. SYSTEM OVERVIEW

The Smart Helmet Rental System employs a modular and scalable design that combines web technologies and embedded hardware. The frontend of the system is based on the Angular framework and is interactive, responsive, and user-friendly. The Angular framework also supports the creation of single-page applications where users can register for accounts, log in, and implement basic features such as viewing the wallet balance and generating codes to access a locker. The frontend communicates with the backend via RESTful APIs, also allowing for communication of data and interaction with various system components. The frontend is also responsible for session management, as well as generating error alerts and maintaining the user flow. The frontend maintains responsive use across devices, keeping the user experience consistent.

The system's backend is built with Spring Boot, a popular Java-based framework, to simplify the development of REST APIs and run enterprise-level applications. The backend contains essential functionality, such as user login and registration, code generation, rental session tracking, wallet balance verification, and billing calculations. Every user request effectively executes through a secure REST endpoint, which retrieves or stores user information, transaction logs, helmet availability, and numerous other types of information from the MySQL relational database. The ESP32 microcontroller serves as the hardware part of the system and executes the locker mechanisms connected to the backend server. To clarify, the ESP32 is batch programmed to conduct various commands in real time; receive the unlock/lock commands, monitor return status, and trigger the rental timer, all on Wi-Fi through the internet. The hardware and software are in harmony, so users can reliably know the helmet and locker will work securely in real time, promoting helmet use amongst urban two-wheeler riders.

3. SYSTEM FLOW

Smart Helmet Rental System operates via a structured user interaction flow mixed with backend processing flow, hardware control flow, and financial tracking. This user flow integrates a seamless user experience with reliability and automation within the system at each stage of the service.

The user flow commences with registration and login. To utilize the Smart Helmet Rental Service, each user is required to visit a web-based Angular application and enter their mobile phone number to validate authentication. User authentication through phone number reduces complexity and uses a familiar, recognizable, and globally available identifier. During registration, the user's details, including mobile phone number and initial wallet balance, were stored in a MySQL database via an encrypted API call to the Spring Boot backend. Following successful registration, the user can log in and is redirected to a dashboard representing their profile, wallet balance, current rental status, and complete actions.

After logging in, the next step is to use the "Get Code" functionality, which is at the core of the rental process. When the user selects the "Get Code" button, the backend system will generate a random numeric code, which will serve as a temporary access key to unlock the helmet locker. That temporary code will be stored in the database along with a timestamp and will be tied to the user and the locker that they will be assigned to. In the HTTP request, the code will also be transmitted to the ESP32 microcontroller, so the hardware system can prepare for user interaction.

As for the locker station, the user will go up to the smart locker that has the helmets, and the user will input the unique code into the hardware. This denotes the start of the Unlock Locker phase. At the time the user inputs the helmet code, the ESP32 microcontroller will validate the code by sending a validation request to the Spring Boot backend. If it is valid and not expired, the ESP32 will trigger a motor which in turn will physically open the locker door and grant the user access to the helmet. The first concept validates that the locker is secure from unauthorized personnel gaining access and allows helmets to only be accessed by legitimate users.

Upon collection of the helmet, the rental period is officially initiated. Technically, the system will enter the Rental Timer portion of the rental process. The back end will record the start time at the moment the user opens the locker. The ESP32 microcontrollers associated with the helmet station and return station work together with the backend to represent the session status, including elapsed rental time, in real time. Once the helmet is returned, the rental timer will stop.

To return the helmet, the user returns to any return station and enters the same unique code used to unlock the helmet. The ESP32 at the return station will again validate the unique code with the back end. Once validated, the locker door will be opened to enable the user to place the helmet in the locker. Once the user completes the return, back end can now record the end time of the rental.

Now that the system has both timestamps (start and end), the total rental period is calculated. This brings us to the final step—Billing. The backend calculates the charge per minute/hour for the rental. The determined charge is automatically deducted from the user's wallet. The user's wallet must have a minimum balance of ₹1000 to start a rental session

Each of the stages in the system flow is closely tied to logging and validating layers, which log every transaction, which could be, for example, an event in time when a user unlocks a locker, the time a user submits a return, the time the wallet is deducted, and so forth. Overall, this entire workflow presents a totally automated, user-friendly, secure helmet rental solution for the needs of modern city living.

4. METHODOLOGY

The Smart Helmet Rental System adopts a systematic, modular approach, which includes software engineering module development with hardware integration. The approach has three main elements: The technology, the data flow through the system, and the wallet-based transaction model. These three-part divisions give certainty about the behaviors of the system, allow for scalability, and allow for sensitive data transactions.

4.1 Technologies Used

The Smart Helmet Rental System developed encompasses multiple bleeding-edge frontend and backend technologies in conjunction with hardware interfacing through IoT components. The combination of technologies was selected with features such as platform independence, real-time interactions, and security in mind.

The frontend application was built using Angular. Angular is an open-source application framework, primarily maintained by Google, that offers a component-based approach for applications, leading to maintainable modular coding standards. The use of TypeScript supports strong typing and error checking while developing. The Angular application easily allows for near-real-time updates to the user interface with features such as two-way data binding and reactive forms. These features make Angular

an ideal choice for a rental application that provides users with near-instant feedback and state updates.
The backend runs on Spring Boot, a powerful and highly scalable framework for Java applications. Spring Boot is useful because it makes developing stand-alone, production-ready applications faster by providing a large number of plugins and integrated services, such as dependency injection, web services (via REST), and security modules. Spring Boot is the core of the application in terms of business logic, from registering users to authentication and managing locker control commands and rental sessions. Spring Boot also relies upon REST APIs to create a clean separation of the front-end and back-end and support stateless communication through easily scalable deployments.
The database layer is run on MySQL, an open-source relational database management system. MySQL was specifically chosen because it works well with Spring Boot, and MySQL has been known for reliable transactional data management. MySQL will store some basic credential information: user credentials, wallet balances, transaction histories, locker's availability status, and timestamp logs of the events when the helmet is picked up or returned.
The hardware implementation component used ESP32 microcontrollers. The ESP32 is a low-powered, low-cost system on a chip (SoC) with Wi-Fi and dual-mode Bluetooth built in. It can run real-time applications and enable communication with web services via HTTP wire protocols. Each helmet locker in the helmet rental system is controlled by an ESP32 unit. The ESP32 control units receive commands from the back end and send confirmations/status messages back to the back end. Thus, the noted two-layer communication between the helmet rental system is a technical connection between the digital and physical layers.
Together, the technologies used here make up a seamless and responsive system architecture in which each of the components of the architecture enables helmet rentals to be completed in a timely, secure, and semi-automated manner.

4.2 Data Flow

In the helmet rental system, the data flow is linear, trackable, and modular. It begins with the user engagement on the Angular frontend (e.g., signing up, finding the request for an access code, or checking the balance of their wallet) and then makes asynchronous calls (for example, via HttpClient in Angular) to the Spring Boot backend system as the main controller. For example, when registering a user, the mobile number and initial user-provided data are sent to the backend over a secure POST request. Spring Boot will check and save this data in the MySQL database as it is expected. In a similar fashion, if a user clicks "Get Code", this tells the backend to generate the access code, log a timestamp, and provide the code to the frontend (for user reference) in addition to sending a message to the ESP32, perhaps over REST.
Once at the locker station, the ESP32 microcontroller becomes involved at the rental site. The ESP32 sends the access code back to the backend to validate the code through an HTTP GET request. When the code is verified and permitted, the backend sends a command to the ESP32 to unlock the locker door, which the ESP32 responds to and opens the locker door. When opened, the ESP32 sends information back to the backend, updating the status, and the backend records the start time of the rental transaction.
The reverse data flow occurs when the helmet is returned, in this case the ESP32 at the return locker station receives the input code, validates it with the backend, opens the relevant locker for return and updates the status back to the system, the backend records the end time and rental duration and starts the billing module.
This interaction with the system, whether it is frontend to backend, or backend to hardware, is all statele-

ss and secure, and all tracked and logged in the MySQL database for auditing and credentials. This consistency in the data flow means that when a failure happens, it can be traced and responded to accordingly, which is critical for real-time systems operating in the public domain.

4.3 Wallet System

One of the key parts of the Smart Helmet Rental System is the wallet payment method, which makes the user accountable for all helmet rental services. The system has a rule that we have made that for any helmet rental request to be made, the user should have a minimum of ₹1000 in wallet balance.

When a user registers for the first time, they are then prompted to top up their wallet via an online payment. The wallet balance is saved in the database and is retrieved whenever a user tries to rent a helmet. If the amount is below ₹1000, the system denies the request and asks for a wallet recharge.

For the rental process, once a helmet is successfully returned, the system receives the time both timestamps for the start and end of the helmet rental time usage. The total rental period timestamps are then deducted from a scheduled, predefined rate. The total due amount is deducted from their wallet balance then the balance gets updated in the database.

The wallet system has its own safeguards in terms of deduction only once per rental, session timeouts, and discrepancies if the ESP32 sends a return that is not recorded properly. Furthermore, wallet transactions are committed to a table that keeps the financial record for each user. This can be relevant for analytics purposes or when there are disputes.

From a user perspective, the wallet model provides convenience for the user, where users do not have to worry about payment after their first payment is made. Users also benefit from a better user experience by providing convenience, and they can access helmets quickly with service in a single click, provided that they meet the balance condition. From a system perspective, this is both a benefit to the user and a liability to the business, as users will provide payment in advance, minimizing the risk of non-payment or late payment and increasing the efficiency of transactions.

In summary, the wallet system is an improvement in terms of financial security, automation, and trust in the helmet rental process. Helmets are now more user-friendly and economically sustainable.

5. RESULTS

The testing and implementation of the Smart Helmet Rental System has resulted in positive outcomes and has proved the viability of using web technologies in conjunction with IoT-based hardware to create a completely automated helmet-sharing system. All primary functional components of the system were tested either individually or in integration, and the system successfully demonstrated the targets of design and delivery for functionality relative to user authentication and validation, locker control functionality, data communications using web technologies, session tracking, and payment billing security.

The module for user registration and validation reliably achieved positive outcomes in all testing scenarios. New users registered with a mobile phone number and were able to follow through a registration process that would allow existing users to log in with their existing credentials. The design set up validation checks for both the uniqueness of each mobile number, as well as indications of duplicate or incorrect inputs, with appropriate error messages. The MySQL backend of the system provided the appropriate storage and retrieval functionality for user data, while real-time feedback reporting was provided by the Angular frontend of the system. The simplicity of the mobile phone used during registration and login processes benefited users but also provided a level of protection for the Smart Helmet Rental System.

One of the most impressive aspects of the system is the ability to control who accesses the lockers through unique codes, digital keys. The "Get Code" feature always produced a unique numeric string with an expiration date that was individually assigned to each user. This string was successfully delivered to the front end (for user-facing visibility) and the ESP32 microcontrollers (locking and unlocking physical lockers). During the testing phases of the project, we validated that the ESP32 devices were able to access and confirm the access code via HTTP communication with the Spring Boot backend. Once the code was temporarily verified, the ESP32 could activate the electronic relay for the correct locker to be unlocked. After the lockers were opened and closed, we were confident in the updates the ESP was providing in real-time via the ESP's status updates. As long as the ESP32 was able to locate some sort of acknowledgement, the physical access could be tracked perfectly.

Another significant success for the system was the ability to track rental length using ESP32s. The rental session was automatically configured to begin as soon as the locker was opened and ended once the helmet had been returned via the same code.

Additionally, we put in place timeout logic to deal with edge-case situations for all users involved in a rental transaction. The logs were maintained in the backend, and confirmed the consistency of rental timing even when multiple users interacted with the system simultaneously.

The safety of deducting payments from users' wallets was crucial. After a helmet was returned, the system would automatically calculate the rental cost based on the duration and set rates. To ensure smooth transactions, users were required to maintain a minimum wallet balance of ₹1000 to start a rental. All wallet transactions were logged with timestamps, and deductions were made automatically without any need for human oversight. This pre-paid model eliminated the risk of unpaid rentals and boosted operational reliability.

Overall, the system exhibited excellent performance, responsiveness, and security during testing. All core functionalities—from registration to billing—were validated through controlled simulation and real user interaction. The integration between software layers and IoT hardware proved to be robust and efficient, laying a strong foundation for potential real-world deployment in smart cities or university campuses. The successful results of this system underline the potential of combining web development and embedded systems to solve real-life problems through automation and innovation.