A Web-Based Ticketing System

Improving IT Support Efficiency and Employee Productivity

Bachelor's Thesis

Degree Programme in Computer Applications

Spring, 2025

Parth Patel

HAMK
Häme University
of Applied Sciences

Abstract

DP	Degree Programme in Computer Applications	
Author	Parth Patel	Year 2025
Subject	A Web-Based Ticketing System: Improving IT Support Efficiency and Employee Productivity	
Supervisors	Kevin Cheng	

This thesis describes the development of a web based ticketing system with the aim of enhancing IT support efficiency and employee productivity. The research design was based on an iterative agile-based prototyping methodology which more or less facilitated regular testing and improvement of the system features of each iteration based on the feedbacks received. The system enables users to safely login, open support ticket, track progress and connect with IT personnel based on their respective roles. Employee, Admin, or Super Admin.

A new modern architecture: full-stack was deployed, based on React.js for the user interface, Node.js and Express.js for backend logic and MongoDB as a robust and scalable database. The system as well incorporates critical aspects such as role based access control, ticket prioritization and activity tracking to make it responsive and accountability-based.

To verify the performance and usability of the system structured functional and user testing was carried out. These assessments showed that the system does indeed cut the time taken to resolve tickets and improve communication within IT teams.

This learnt a full stack development, secure authentication as well as user-centric development. The outcome is a practical application that can support efficient IT operations, reduced downtime and service delivery improvements across departments.

Keywords	Web-based ticketing system, IT support, React.js, Node.js, MongoDB, full-stack development, issue tracking, employee productivity, role-based access, system design.
Pages	44 pages and appendices 02 pages

Glossary

HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
API	Application Programming Interface
UI	User Interface
UX	User Experience
IT	Information Technology
NPM	Node Package Manager
IDE	Integrated Development Environment
DB	Database
JSON	JavaScript Object Notation
GIT	Distributed version control system
JWT	JSON Web Token
CRUD	Create, Read, Update, Delete

Table of Contents

1 Introduction 1
2 IT support systems and web-based ticketing 3
2.1 IT support systems overview 3
2.2 Problems in traditional IT support systems 3
2.3 Web-based ticketing systems 4
2.3.1 Definition and key features 4
2.3.2 Benefits of web-based ticketing system 5
2.4 Comparison of existing ticketing systems 6
2.5 Challenges in implementing ticketing systems 7
3 Full-stack development for IT solutions 8
3.1 Introduction to full-stack development 8
3.2 Chosen technologies for this project 9
3.2.1 Frontend: react.js 9
3.2.2 Backend: node.js and express.js 10
3.2.3 Database: mongoDB 11
3.3 Why these technologies were chosen 11
3.4 Benefits of full-stack development in this project 12
3.5 Challenges in full-stack development 13
4 Methodology & system design 15
4.1 Research methodology 15
4.2 Tools and technologies used 16
4.3 System design 16
4.3.1 Use case diagram 16
4.3.2 System architecture 17
4.3.3 Entity relationship (ER) diagram 19
4.4 Ethical considerations in system design 21
5 Implementation and features 23
5.1 System overview 23
5.2 Front-end implementation 24
5.3 Back-end implementation 29
5.4 Database schema and models 32
5.5 Security and authentication 34
5.6 UI and User experience 36

5.7 Notifications or ticket updates 38
6 Testing and evaluation 40
6.1 Testing methods 40
6.2 Evaluation against research questions 41
7 Results 43
8 Conclusion and future work 44
References 45

Figures

Figure 1 Functions Assigned to Each User Role in the Ticketing System 5
Figure 2 Use Case Diagram for Ticketing System 17
Figure 3 System Architecture of the Web-Based IT Support Ticketing System 19
Figure 4 Entity Relationship Diagram (ERD) of the IT Support Ticketing System 21
Figure 5 Login page interface 25
Figure 6 Ticket form interface for Employee 26
Figure 7 Ticket list for Admin 27
Figure 8 Ticket list for Super Admin 28

Tables

Table 1.Comparative Overview of Jira, Freshdesk, and Zoho Desk (CRM.org, 2025; SourceForge, n.d.; ThriveDesk, 2024) 6
Table 2.Summary of Tools and Their Roles in the Project 16

Program codes

Program Code 1 Authentication check and role-based redirection logic (AdminLayout.js) 24
Program Code 2 Ticket object state initialization (TicketForm.js) 26
Program Code 3 Role-based ticket table column logic (ViewTickets.js) 26
Program Code 4 Super Admin user creation logic (EmployeeForm.js) 28
Program Code 5 User login and token generation 30
Program Code 6 Ticket creation logic 30
Program Code 7 Role validation middleware example 31
Program Code 8 Mongoose schema for UserType 32

Program Code 9 Mongoose schema for Ticket 33
Program Code 10 Mongoose schema for Priority 33
Program Code 11 Password hashing using bcrypt 34
Program Code 12 JWT verification middleware 35

Appendices

Appendix 1. Data management plan 47

1 Introduction

In today's modern business organizations heavily depend on effective IT support systems for their operational flow. Numerous firms still deal with ongoing difficulties in resolving their IT problems. Flow interruptions happen when technical help is slow in coming leading to reduced work efficiency of staff members who feel dissatisfied. The combination of old-fashioned and uncoordinated IT support systems creates multiple difficulties when managing issues which leads to unhelpful communication between teams and unresolved problems alongside longer-than-expected resolution durations.

Not only do these inefficiencies obstruct productivity, but finances can be lost and risks associated with data safety and system failure may be heightened. There is, therefore, an increasing demand for structured and automated support tools. Web based ticketing systems provide a promising solution to avoid this as users can send support requests, create priority levels and follow issue progress and update times. These systems enable IT departments to make (logs/lists) of requests and be able to follow them, therefore getting a better response time and overall user experience.

This thesis investigates how a web-based IT ticketing system was designed and installed in order to address these inefficiencies. A modern full-stack approach- React.js implemented for the front-end view, Node.js and Express.js for the back-end logic, and MongoDB as the database solution - are used for the builds of the proposed system. The idea is to develop the scalable user friendly solution which will improve IT service delivery and employee's satisfaction.

The research addresses the following questions:

- What are the key challenges in IT support that reduce efficiency and productivity? (Section 2.2)
- How can a web-based ticketing system improve IT support workflows and reduce issue resolution time? (Section 2.3 and 6.2)
- What essential features should a ticketing system include to enhance IT support efficiency? (Section 5.5 and 6.2)

By answering these questions, the project aims to provide a practical solution to improve IT support responsiveness and organizational productivity through modern web technologies.

2 IT support systems and web-based ticketing

Today, most organizations are driven by technology and require an effective IT support system to run their day-to-day activities. Due to the increasing reliance on digital infrastructure by businesses, a need for structured and effective support mechanisms has arisen. This chapter talks about the basics of IT support systems, their limitations and how a web-based ticketing system can help overcome these limitations. This thesis will also compare current ticketing solutions, thus providing a theoretical basis for the actual development of the system.

2.1 IT support systems overview

The services represented by IT support assist organizations in maintaining and developing as well as organizing their technology systems. It includes resolving technical problems, assisting with hardware and software issues, handling cybersecurity, and advising for system upgrades and improvements. To enable smooth business operations, protect sensitive information, reduce downtime, and improve productivity, organizations depend on IT support. In the absence of proper IT support, many businesses struggle with extended system downtimes, data security issues, decreased productivity, and poor customer service. Poor IT support contributes to persistent technological issues, delays in service, increased expenses, and ultimately diminish business efficiency and competitiveness (Electric, 2024).

2.2 Problems in traditional IT support systems

Old fragmented systems for servicing requirements involved receiving requests via emails, phone calls, and even sticky notes. Although such methods provided basic communication, they became increasingly inefficient as organizations grew. Losing or misplacing support tickets is a common problem with fragmented IT systems because there are no centralized computers, or hubs, designed to monitor and control unstructured systems. Other recurring system drawbacks included passive response activities, manual regulation of system interactions, and no methods for ranking and placing urgency so tasks could be completed, which created unnecessary stagnation. Most of these factors contributed to employees facing aggravated technical difficulties, which became long-standing problems, compelling

gaps, or prolonged periods of inactivity, forcing employees to remain idle. Such factors resulted towards insufficient productivity alongside creating frustration (Muntakim n.d.).

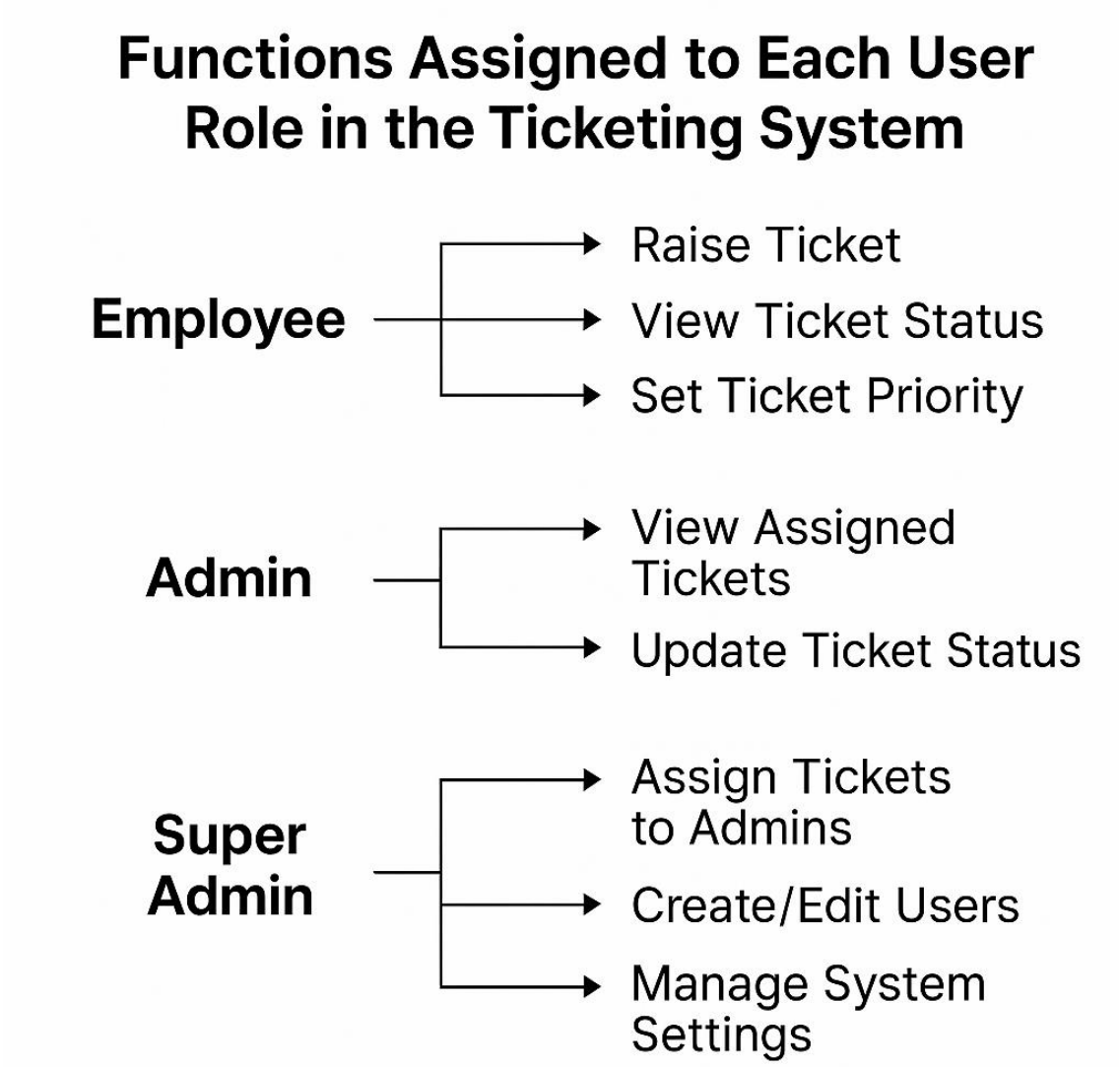
2.3 Web-based ticketing systems

This section gives a basic understanding of web-based ticketing systems and their operations coupled with their benefits and how they enhance IT support mechanisms. It outlines their main features and discusses real world cases in use, setting the stage for the system developed in this thesis.

2.3.1 Definition and key features

A web-based ticketing system refers to an application that aids organizations in the management and automation of their IT support services. Unlike traditional methods of receiving requests via emails or phone calls, users now submit a “ticket” through a portal which describes the problem they need assistance with or what they want done. Each ticket is entered into the system, given a level of priority based on how urgent it is, and then allocated to the appropriate level of IT support. The system allows for tracking of a request from submission until the solution is provided so that there isn’t any request that is hidden or merely forgotten. Important components that make up a web based ticketing system are ticket handling, ticket prioritization, constant status monitoring, communication and tracking note reporting. As illustrated in Figure 1, each user role Employee, Admin, and Super Admin has clearly defined responsibilities within the ticketing system.

Figure 1 Functions Assigned to Each User Role in the Ticketing System



These systems enhance effectiveness, lowers the time required responses, and improves the clarity of interactions between users and IT supports, which in turn leads to quicker resolutions and higher employee ethical satisfaction.

2.3.2 Benefits of web-based ticketing system

The convenience of web-based ticketing systems can significantly improve the quality of IT support services. One of the most important advantages of these systems includes the enhancement of support request resolution speed. These systems streamline the

resolution processes by automating ticket assignment and prioritization, ensuring that all support requests are attended to in a timely manner and maximizing productivity (Verizon Business, n.d). Moreover, these systems allow for consolidated tracking and management of support tickets, thus enabling easier monitoring of issues and ensuring that all requests are attended to. The remote access capability of web-based ticketing systems make it possible for IT support teams to fully manage and attend to issues from any location which is advantageous in today's remote and scattered work settings. In addition, these systems ensure effective and uniform communication between employees and IT through the detailed documentation of all interactions pertaining to every ticket. Such openness enhances the quality of support provided and increases user satisfaction, as employees are constantly updated on the progress of their issues (Verizon Business, n.d.).

2.4 Comparison of existing ticketing systems

Table 1.Comparative Overview of Jira, Freshdesk, and Zoho Desk (CRM.org, 2025; SourceForge, n.d.; ThriveDesk, 2024)

Feature	**Jira Service Management**	**Freshdesk**	**Zoho Desk**
Primary Use Case	IT service management and project tracking	Customer support and help desk operations	Customer support with CRM integration
Key Features	- Advanced issue tracking - Customizable workflows - Integration with development tools - SLA management	- Multi-channel ticketing - Automation and AI features - Knowledge base - Reporting and analytics	- Multi-channel support - AI-powered assistance (Zia) - CRM integration - Customizable dashboards
Pros	- Robust for ITSM - Strong integration with Atlassian products - Suitable for agile teams	- User-friendly interface - Extensive integration options - Scalable for various business sizes	- Cost-effective - Seamless CRM integration - Flexible deployment options
Cons	- Steeper learning curve - Can be complex for non-technical users	- Some advanced features require higher-tier plans	- Limited third-party integrations compared to competitors - Some users report UI could be more intuitive

Pricing (Starting at)	$20/user/month	$15/user/month	$14/user/month
Free Trial	Yes	Yes	Yes
Ideal For	Medium to large IT teams requiring detailed project and issue tracking	Businesses seeking a comprehensive, user-friendly customer support solution	Organizations looking for an integrated customer support and CRM solution

2.5 Challenges in implementing ticketing systems

Implementing a ticketing system within an organization presents several challenges that can impact its effectiveness and adoption. One significant hurdle is user resistance, where employees may be reluctant to change established workflows or fear that automation could threaten their roles. This resistance can lead to inconsistent usage and hinder the system's success (DeskDirector, 2024). Additionally, the cost implications of adopting a new ticketing platform are considerable. Beyond the initial investment, organizations must account for expenses related to training, potential downtime during the transition, and productivity losses as staff acclimate to the new system (Supportbench, 2024).

System configuration and integration pose another set of challenges. Many ticketing systems require detailed customization to align with an organization's specific processes, which can be time-consuming and demand technical expertise (DeskDirector, 2024). Furthermore, integrating the new system with existing tools and workflows is essential but often complex, potentially leading to disruptions if not managed carefully. Training and skill development are also critical; ensuring that all team members are proficient with the new platform necessitates comprehensive training programs, which can strain resources (Supportbench, 2024).

Lastly, data security and privacy concerns are paramount. Because ticketing systems deal with confidential data, they can attract hackers. To keep things under control, security measures must be applied and data protection regulation enforced on anyone that makes a request (OWASP, n.d.). In order to ensure success before, during and after reliance on, the ticketing system one must plan with experts and engage stakeholders.

3 Full-stack development for IT solutions

This chapter describe the full-stack technologies utilized to develop the IT support ticketing system. What is discussed are the front-end, back-end and database tools used, what each of them is responsible for and why they were selected. The discussion provides a technical ground for the subsequent design and implementation stages.

3.1 Introduction to full-stack development

Full-stack development is building the part of web apps that people see and use. The front-end consists of the actual visual elements the user engages with or interacts with. Other words, the back close by is the server-side logic, databases, besides application workflows that power the functionality behind the scenes. A full-stack developer is someone who has the competences and expertise to work on both the client and server sides of an application (GeeksforGeeks, 2024).

As the demand for quicker and more effective means of development arose, full-stack development evolved. At first, developers worked on a single project whether front-end or back-end. However, today's business wants experts who can work on both, it helps in process efficiency and reduces costs (MongoDB, n.d.-a).

In today's digital age, full-stack development is very significant. Organizations are looking for full-stack developers because they can work with many areas of a project which leads to more flexibility and faster delivery of the products. According to Lemonaki (2021), full-stack development is recommended for startups and agile teams due to the demand for generalist technical skills as a start-up and agile teams require a good deal of resourcefulness and generalist technical skills.

A successful full-stack developer requires a varied set of technical skills. The front-end and back-end technologies like React.js, Angular, Node.js, Django, MongoDB, MySQL, APIs, Git, servers, etc.

Full-stack development is a popular thing demonstrated by Facebook and Netflix. Their applications combine intuitive user interfaces with robust back-end systems to provide seamless experiences for millions of users.

Nonetheless, full-stack development has its drawbacks. It is challenging to master an entire technology stack. It is difficult for a single person to achieve, let alone maintain, deep knowledge in both the front-end and back-end fields (IBM, 2023). Yet, the efficiency and flexibility make full-stack development an essential part of modern-day IT solutions.

3.2 Chosen technologies for this project

To develop the IT support ticketing system successfully, a set of modern and widely used technologies has been chosen. These technologies are developed with performance and scaling in mind, equally smooth user experience. The front-end of the project is to be done in React.js. The server-side logic and API requests are efficiently handled using Node.js and Express.js. To manage and store data, have used MongoDB. MongoDB has chosen because it is NoSQL, and it is flexible. These combined technologies form a good-stack development environment for rapid development and real-time availability and easy maintenance. This combination allows the system to be highly adaptable, meeting the evolving needs of users and administrators in a modern IT support setting (GeeksforGeeks, 2023; Lemonaki, 2021).

3.2.1 Frontend: react.js

This IT support ticketing system uses React.js because it is efficient, flexible, and is popular technology used for creating web applications. React.js is a JavaScript library that was developed and is maintained by Facebook. By using a component-based architecture, developers can create fast and interactive user interfaces. (React, 2023). Components in React are pieces of code that can be reused and allow to controls its own state. Components enable the building of complex user interfaces from small and isolated pieces.

One important function of React is the Virtual DOM. The Virtual DOM speeds up the process of updating applications and a finished application will make the updates quickly and efficiently without refreshing the pages (GeeksforGeeks, 2023). React makes use of

one-way data binding. This offers a clear data flow system which is easy to debug and maintain as well.

Furthermore, React provides an extensive selection of tools and libraries. A particular example is Redux, which helps with state management, and React Router, which assists with navigating between various sections of the application. Also, React's strong community support ensures continuous updates, extensive documentation, as well as access to a wide range of third-party resources.

Using react.js this project aims to provide a responsive and dynamic user-friendly interface which will ensure smooth interactions between employees and IT administrators managing support tickets.

3.2.2 Backend: node.js and express.js

Node.js and Express.js have been chosen to build a powerful server-side architecture which produces an efficient and scalable back-end development for the IT support ticketing system. The execution environment of Node.js will run JavaScript code outside browsers through the V8 engine from Chrome. Node.js is recognized as an exceptional development environment for fast scalable networking applications because it relies on event-driven non-blocking architectural features (Node.js, 2023). Through this method the server system can perform multiple requests concurrently without performance reduction which proves crucial for real-time IT support operations.

The Node.js technology receives additional functionality through Express.js which delivers a basic yet adaptable system for building web platforms and APIs. Through Express developers can easily streamline their routine backend operations which include routing functions along with handling HTTP demands and responses and middleware implementation to regulate authentication and error management (GeeksforGeeks, 2023). The framework's basic structure enables developers to build powerful APIs rapidly because it gives complete oversight of architectural elements.

This Node.js and Express.js integration provides plenty of benefits including quick development speed, efficient operation with heavy traffic and access to an energetic and powerful open-source community for ongoing enhancement and extensive platform

support. The selected technologies serve this project to properly link the client-side React.js application to the database so support tickets can be processed and stored and retrieved without issues.

3.2.3 Database: mongoDB

The database layer of the IT support ticketing system selects MongoDB because of its adaptability combined with scalability features along with seamless compatibility with JavaScript programming languages. MongoDB functions as a well-known NoSQL database which organizes data through the flexible BSON (Binary JSON) format so it serves applications that need swift handling of unstructured or semi-structured information (MongoDB, n.d.-b).

The flexible database structures of MongoDB serve developers better because this system avoids the need for fixed schemas thus enabling database schema alterations during application evolution. Such adaptability proves critical in IT support systems since user request data can take different structural formats. The database solution from MongoDB gives users access to outstanding features including automatic sharding as well as horizontal scaling plus replication for high performance and availability during heavy traffic (GeeksforGeeks, 2025).

The main strength of MongoDB in this project stems from its native compatibility with Node.js through Mongoose libraries that make data modeling and validation process simpler. Due to its developer-friendly query language MongoDB enables smooth execution of complex operations for ticket information including sorting and searching and updating. MongoDB meets the requirements of modern IT support systems because of its characteristics that support dynamic scalability.

3.3 Why these technologies were chosen

React.js, Node.js, Express.js, and MongoDB have been chosen for the IT support ticketing system as they are compatible and high-performance and are popular choices in modern full-stack development. React.js was selected for the front end because it follows a component-based architecture that allows for code reusability, faster development, and

dynamic UI. Its Virtual DOM enhances the performance of the application by optimizing the rendering activities. It is ideal for applications that require frequent updates and a responsive user experience (React Official Documentation, 2023).

Node.js was chosen for the back end because it offers a non-blocking, event-driven model that allows the server to efficiently manage many simultaneous requests. This is especially important in case of a ticketing application where a lot of users may interact with the platform simultaneously (Node.js, 2023). Express.js is a minimal framework that simplifies the development of server-side logic on node. js. It helps with the development of back end applications super-fast without being opinionated (GeeksforGeeks, 2023).

MongoDB was selected as the database because it is NoSQL Database. This allows data to be stored without requiring a fixed schema. Being able to change quickly to handle differences in all the tickets and related information. MongoDB works better with Node.js libraries like Mongoose which makes handling the data more efficiently easier (MongoDB, n.d.-c). These technologies work together to create a solid, scalable, and high-performing stack that enables the system's rapid development cycle, easy and maintainable future scalability.

3.4 Benefits of full-stack development in this project

Full-stack development is very useful for making IT support ticketing system. Therefore, it has many benefits for the IT project. One major advantage is faster development cycles. Because full-stack developers can manage both front-end and back-end, there is reduced dependency on different expert teams which leads to faster integration and delivery of functionalities (Lemonaki, 2021).

Another big benefit is better communication and consistency through the app. Full-stack developers know everything about the app from front-end to back-end. The same developer or a working closely team deal with the whole system to make a perfect flow.

As a consequence, misunderstandings are reduced, debugging becomes faster, and the final product is more cohesive.

Cost-effectiveness is also a notable advantage. Using full-stack developers, the staffing cost is lower and everyone can still remain highly technical on different layers of the system. (IBM, 2023). It is also cost-effective. Full-stack developers tend to have a more entire perspective, making it easier to pinpoint potential issues early on (MongoDB, n.d.-b).

Last but not the least easier to maintain and scale. Since full-stack developers know the complete architecture so they can easily update, debug, and scale the system as user need increases. This flexibility is especially helpful in situations like IT Support work, where the system requirements keep changing regularly to meet new business needs.

The use of full-stack development has made this project faster, cheaper, more consistent, and more flexible over time.

3.5 Challenges in full-stack development

Full-stack development comes with multiple benefits but it also has some challenges that one must overcome to make a success. Mastering both front-end and back-end technologies is one of the significant challenges. It is expected of full-stack developers to have a wide range of knowledge. However, it’s easy to suffer from an information overload with tools, frameworks, and best practices that keep evolving, which may lead to (Lemonaki, 2021).

Workload management is another significant challenge. Full-stack developers usually do the job for the whole app, which often leads to more workload and even burnout if not handled well. Strong project management skills are needed to get work done without hampering quality(GeeksforGeeks, 2023).

It is hard to maintain security in full stack. Full-stack developers work on both client-side and server-side code. Thus, they ought to be alert about employing security best practices throughout the full stack to prevent cross-site scripting (XSS), SQL injection, data breach, and other vulnerabilities. (IBM, 2023)

Another challenge lies in system scalability. Making an architecture which grows with number of users is not easy and needs a high expertise in database optimization as well as

server-side load management. Full-stack applications with poor architecture may choke when users increase.

Although full-stack development can be a challenging approach to master, it can be very fruitful when carried out with the right strategies, team support, and continuous learning.

4 Methodology & system design

In this section, the research methodology and the development approach employed in the thesis is discussed. It contains also the tools and technologies used, the conceptual design of the system via diagrams, and ethical related issues in question of development process.

4.1 Research methodology

This study was conducted from an operating perspective for developing a software system that works. This project does not involve surveying or analysing any datasets. It is a web-based IT support ticketing system. This computer program will be developed and tested to ensure it will prove practically useful to some organization and its IT department.

The methods used throughout this project were inspired by prototyping and agile approaches. The manufacturing process was repeatedly invoked where each feature was designed, coded, tested and improved. The incorporation of a prototype made sure feedback was continuous throughout the development process resulting in a final system that the user wants.

The full-stack JavaScript technologies were adopted for the whole development process for better workflow efficiency. The user interface is developed using React.js which is a JavaScript library. Node.js and Express.js were utilized to build the back end of the system which allows the server-side logic to be scalable and efficient. Since MongoDB doesn’t have any schemas and is a NoSQL database, it is useful for having non-flexible ticket structures.

Also, GitHub was used for version controlling. The IDE (Integrated Development Environment) used was Visual Studio Code. the system used Node Package Manager (npm) for managing all dependencies. Through the use of actual users that simulated reality, the system function was developed locally as to test what was being produced.

This method will help us tackle both the technical issue and the researcher issue. The technical issue involves the development of a scalable application. The researcher issue

concerns the actual implementation of the application and how that improves the IT support efficiency and user satisfaction in the workplace.

4.2 Tools and technologies used

Table 2.Summary of Tools and Their Roles in the Project

Category	Technology/Tool	Purpose
Front-End	React.js	Build dynamic and responsive UI
Back-End	Node.js, Express.js	Handle server logic and API endpoints
Database	MongoDB	Store and manage ticket and user data
Authentication	JSON Web Token (JWT)	Secure login and protect private routes
Version Control	Git, GitHub	Source code tracking and collaboration
IDE	Visual Studio Code	Writing and debugging code
Package Manager	npm (Node Package Manager)	Managing project dependencies
Testing (Manual)	Browser/Console	Manually testing API and UI workflows

4.3 System design

The application design of a ticketing system determines the user experience and flow of information through the ticketing system. The platform has a modular architecture to ensure scalability, efficient performance, and easy manageability of the Platform. The user interface, business logic and database were separated. The next sections illustrate different cases where the system gets used.

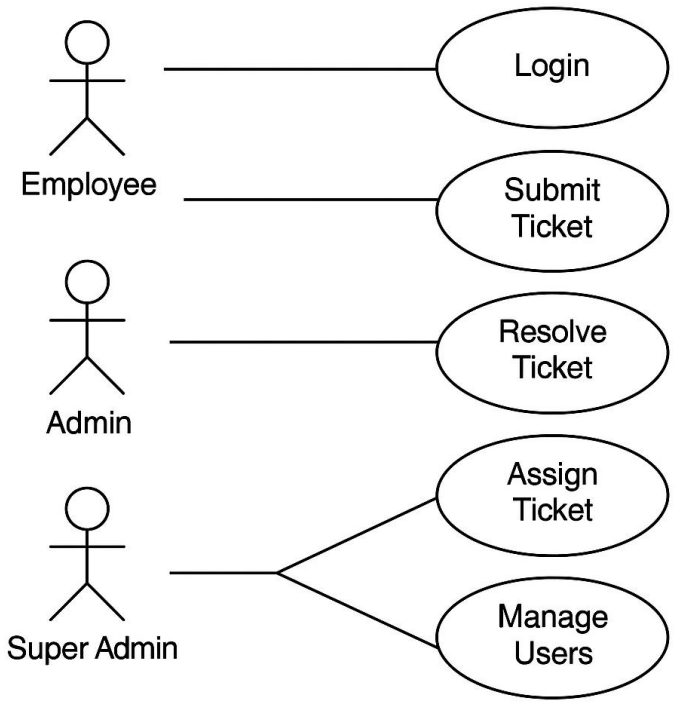
4.3.1 Use case diagram

There are three types of users who will use this system. Super Admin, Admin and Employee, Each user type has specific actions they can perform. Figure 2 shows the use case diagram that defines how different user roles interact with the system.

- Employee: Can log in, submit tickets, view ticket status.
- Admin: update ticket statuses and respond to issues.

- Super Admin: Has all administrative privileges, including managing users, assign tickets to admin and system settings.

Figure 2 Use Case Diagram for Ticketing System



4.3.2 System architecture

The IT support ticketing system is built utilizing a full-stack three-layer architecture. This architecture separates the application into three distinct parts. They are the client side (frontend), server side (backend) and database. This design prevents bugs from appearing, helps add features quickly, and allows for the release of updated application modules that won't impact the entire system.

React.js was used to design the UI on the front end. It is a popular library in JavaScript that keeps web apps dynamic. Using React, developers have the ability to make reusable pieces and render pages more efficiently using Virtual DOM. The job of the frontend within the system is to handle user login and authentication interfaces, ticket submission forms, personalized dashboards for employees, admins, and super admins, and real-time ticket

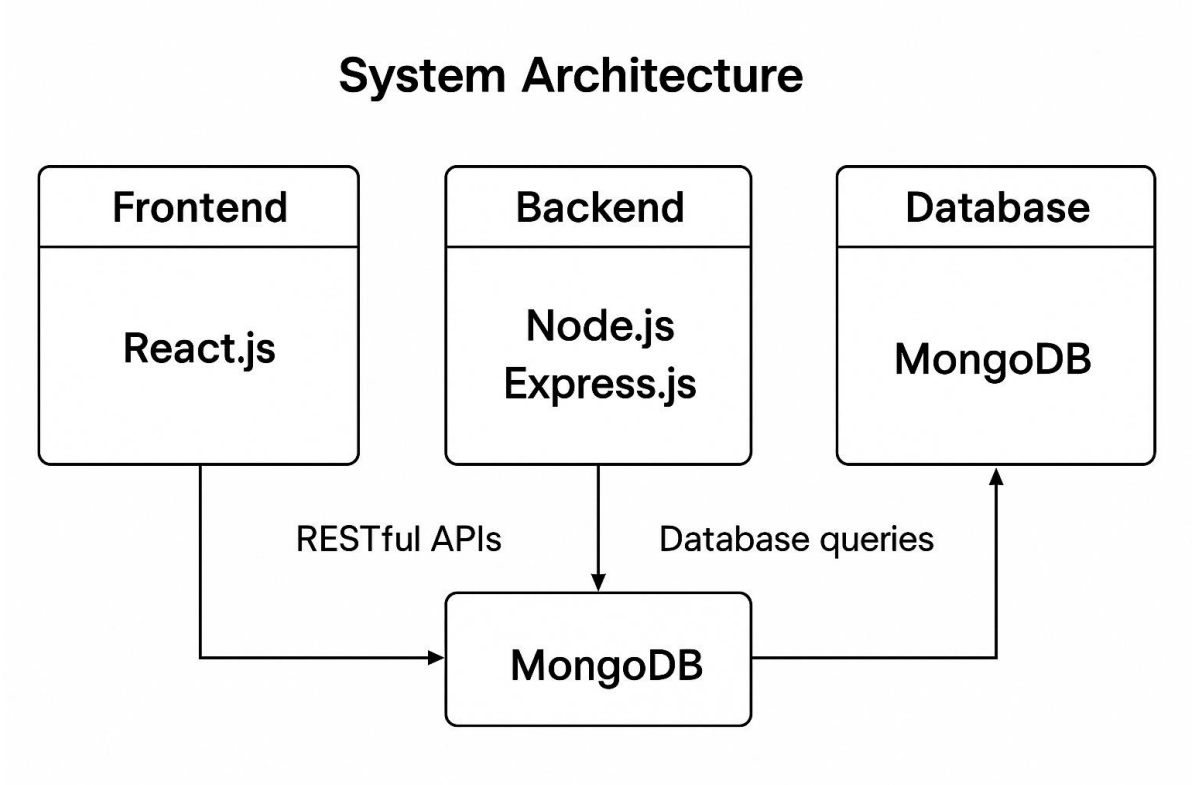
status updates. The front end and back ends communicate via REST APIs that make asynchronous calls - i.e. sending/receiving data without requiring the page to reload, thus improving UX.

Node.js and Express.js help build the backend layer. Node.js is used to run the JavaScript code on the server-side and Express.js is for server-side routing and middleware. The backend is responsible for two main areas. One is authenticating the user. The second is validating the assigned role of the user. The user could be an Employee, Admin, or Super Admin. It also creates a ticket and assigns it. Further, it updates the ticket status which is designed to store and retrieve data. The data would be stored in MongoDB. Mongoose ODM or Object Data modeling is then used to connect to the database. It provides various RESTful API endpoints like /api/tickets, /api/users and /api/login for two way communication between the system's user interface and database server.

MongoDB is a NoSQL document oriented database that does not impose a rigid schema on the data they are handling. MongoDB stores data in various collections; employee is the collection that contains details of user accounts and the role assigned to them; tickets is the collection that stores the issue submitted by the user and keeps track of their status; priorities is the collection that defines the urgency of issues such as Low, Medium, and High; usertypes is the collection that defines users as Employee, Admin, and Super Admin. MongoDB uses ObjectId references to establish relationships between these collections, which keeps the data organized, accessible, and scalable as the number of users and tickets grows. This design helps the system easily manage complex data and perform well as the application grows.

The three layers work together to ensure that the IT support ticketing system is modular and scalable, secure in terms of performance and data integrity and responsive to users across all spectrums while being easy to manage. The system's modular architecture is depicted in Figure 3.

Figure 3 System Architecture of the Web-Based IT Support Ticketing System



4.3.3 Entity relationship (ER) diagram

The database of the IT support ticketing system is made with the help of MongoDB (document-oriented NoSQL database) that uses collections to store complex data structures. Though MongoDB does not depend on a relational schema, the relationship between collections is organized logically. To visualize these relationships, we use an Entity Relationship Diagram (ERD) showing how the important entities in the database relate to each other.

The principal collections of the database are employees, user types, tickets, and priorities. Store users' name, email, password and a reference to the role of the user in the employees collection. Each employee document contains a user_type_id field referencing an entry in the usertypes collection. People from the system can create a ticket or be assigned to handle a ticket. This establishes a many-to-one relationship between ticket and employee.

The usertypes collection defines the different user types in the application, which include Employee, Admin, and Super Admin. Employees will reference their user type by using the user_type_id field, allowing for how access and permissions would work in the system.

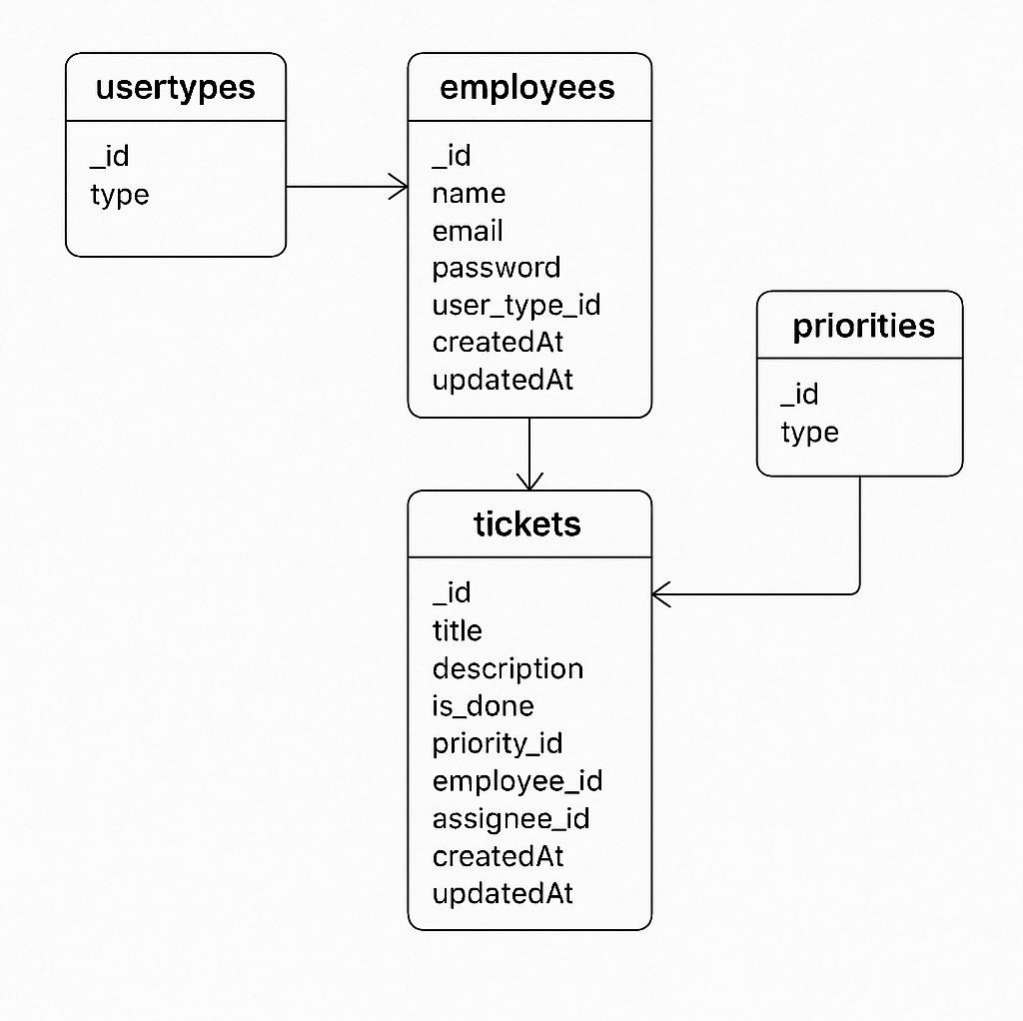
The heart of the ticketing system is the tickets collection, which keeps records of all support requests submitted. Each ticket document records the title of the issue, a description, and fields that connect it to an employee that submitted the ticket (employee_id), an admin that has been assigned to resolve it (assignee_id), and assigned priority (priority_id).

The references allow tickets to be efficiently queried and organized according to user roles and ticket urgency.

The priorities collection outlines how urgent a particular request for support is, through defining it as being a type Low, Medium or High. Each ticket document references a priority record through the “priority_id,” enabling tickets to be filtered on priority easily.

These structured relationships between collections maintain logical data integrity within MongoDB while ensuring there is enough flexibility for the future expansion of the system. The following diagram illustrates the overall relational structure in which the employees, usertypes, tickets, and priority fields are connected such that information flows freely across the different aspects of the system. Figure 4 illustrates the relationships between collections such as employees, tickets, and priorities within the MongoDB schema.

Figure 4 Entity Relationship Diagram (ERD) of the IT Support Ticketing System



4.4 Ethical considerations in system design

Throughout the design of the IT support ticketing system, ethical design practices such as confidentiality, system transparency, granting the user choice and providing behavioural control were followed. The system scrambles passwords (salted using bcrypt). It showers users with a token (JWT), and that ensures credential theft is impossible. Only authorized users that are defined as per their roles (Employee, Admin, Super Admin) can access or change the required data on the platform.

The system guarantees that it does not request sensitive personal information apart from that which is required for user verification and ticket tracking. The system tracks which user

performed which action in the system. They can also check on the status and progress of the tickets at all times.

The system should fulfil the criteria of functionality and ultimately, trustworthiness. It should respect user rights and be non-discriminatory.

5 Implementation and features

In this chapter, the ticketing system's practical administration is presented. It decomposes the system architecture into front-end, back-end and database and emphasizes on fundamental features that were developed for the purpose of functionality, security and user friendliness.

5.1 System overview

The IT support ticketing system is a full-stack web-app developed for an organisation that allows issues to be reported and resolved. The method allows employees to submit tickets and track the status of issues and communicate with IT people according to the prescribed user role, therefore, managing the IT support through a system. The user-facing interface is generated by the front end React.js which is supported by Node.js/Express.js in the backend of the web app. MongoDB is a database layer that can be used to store flexible collections that are usually "schema-less". Besides collections can be referenced to each other through ObjectId.

There are three types of users, each with specific access and permissions:

- Super Admin: Has full control of the system. They can view all tickets, assign tickets to Admins, create new employee/admin/super admin accounts, and manage user roles but they can not delete submitted tickets of employees.
- Admin: Handles ticket assignments from the Super Admin. Admins can view, respond to, update, and mark tickets as complete.
- Employee: Employees can log in, create new support tickets, and track the status of their own submitted issues.

Key Features Implemented:

- Secure Login System with JWT authentication and role validation.
- Role-Based Dashboards to show user-specific functionality (Employees see their own tickets, Admins see assigned tickets, Super Admins manage users and assignments).

- Ticket Submission Form where employees describe the issue and select priority.
- Admin Assignment interface (used by Super Admin) to assign tickets to Admins.
- Ticket Status Updates, allowing Admins to update progress and Employees to view ticket states (Pending, In Progress, Completed).

The architecture of the scheme is modular in nature with a clear distinction between front end and back end logic. The frontend parts go in the src/components folder. Models, routes, and controllers are the backend's three components. You should place models in the backend/models and routes in the backend/routes. MongoDB uses Mongoose schemas to properly reference employees, tickets, usertypes and priorities collections.

5.2 Front-end implementation

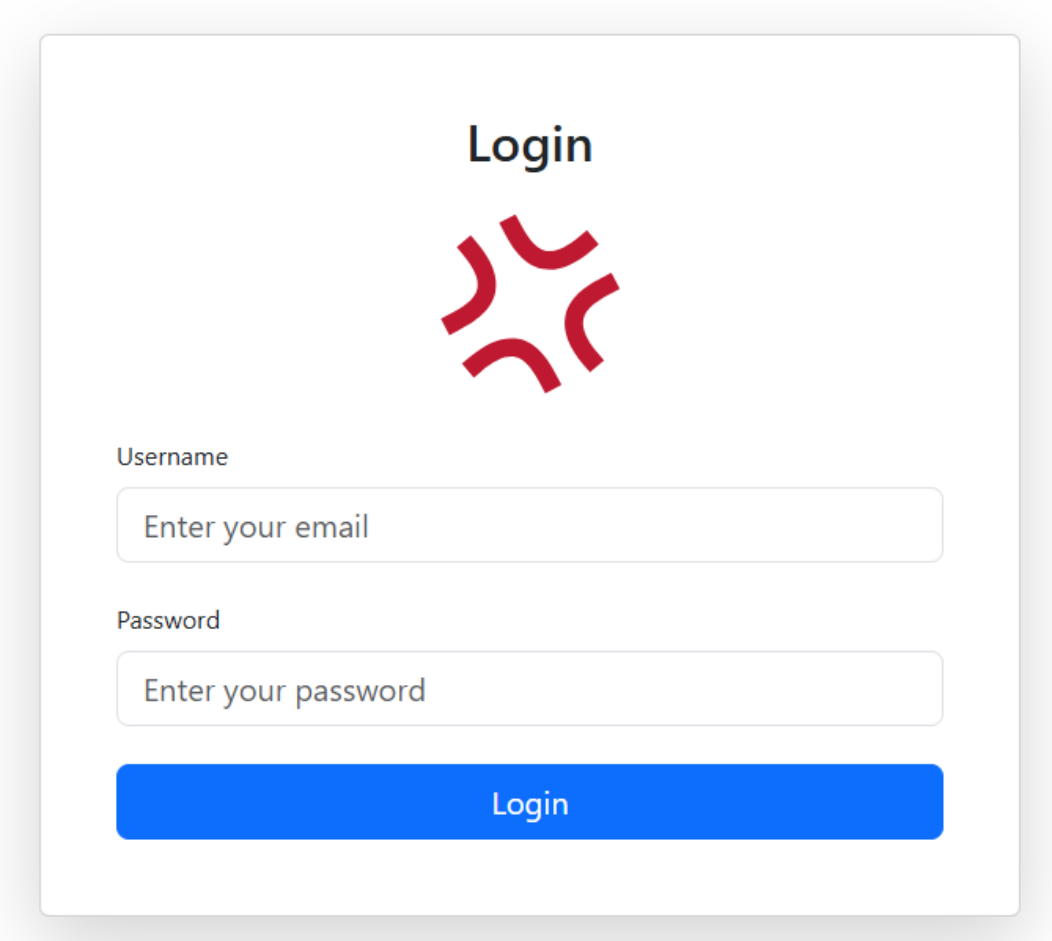
The IT support ticketing system's front-end was done in React.js. This is a JavaScript library that supports the development of reusable components. The application adapts dynamically based on the logged-in user's role allowing Employees, Admins and Super Admins with separate Interfaces and Permissions. When the AdminLayout.js component loads, it checks for the presence of a session by verifying the cookies. If there is no session present, it redirects the user to the login page; otherwise, it redirects the user to the intended dashboard. This is illustrated in Program Code 1.

Program Code 1 Authentication check and role-based redirection logic (AdminLayout.js)

```
const sessionValid = Cookies.get('isAuthenticated') === 'true';
const currentRole = employee?.user_type_id?.type;
useEffect(() => {
  if (!sessionValid) {
    resetFormFields();
    redirectToLogin();
  }
}, [sessionValid, navigate]);
const resetFormFields = () => {
  setEmail('');
  setPassword('');
};
const redirectToLogin = () => {
  navigate('/');
};
```

After successful authentication, the user sees the login page where the credentials are submitted to the backend using Axios. The login form has inputs, form handling, and has a bootstrap style. This page is visualized in Figure 5.

Figure 5 Login page interface



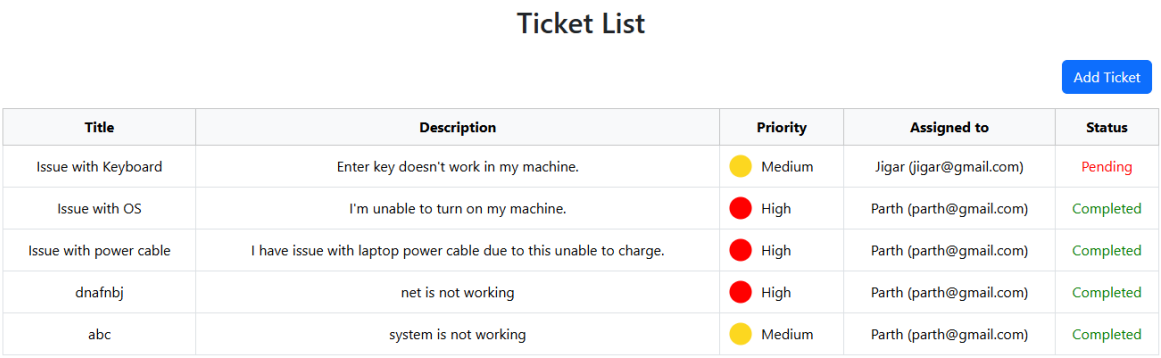
Upon login, employees get redirected to the dashboard where they can submit new ticket. TicketForm.js component does this functionality. The component sets up a local state for ticket fields like title, description, and priority level. The employee ID will be assigned from the cookie of the logged-in user. The state structure is shown in Program Code 2.

Program Code 2 Ticket object state initialization (TicketForm.js)

```
const [ticketData, setTicketData] = useState({
  subject: "",
  details: "",
  completed: false,
  priorityRef: null,
  reporterId: currentRole === "Employee" ? employee._id : null,
  handlerId: null,
});
```

The user role disables some of the fields of the ticket submitted form conditionally. For example, only super admins can assign tickets to admins and only admins can mark them as completed. This form is shown in Figure 6.

Figure 6 Ticket form interface for Employee



The ViewTickets.js component takes care of the dynamic rendering of ticket lists of each user. The table format varies according to whether the user is Employee or Admin or Super Admin. There is a function which returns different column headings based on the role, as seen in Program Code 3.

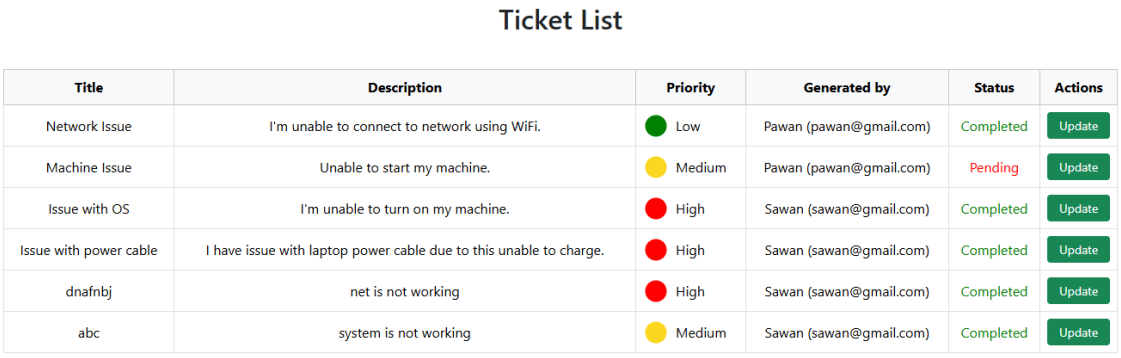
Program Code 3 Role-based ticket table column logic (ViewTickets.js)

```
const getTableHeaders = () => {
  switch (currentRole) {
```

```
    case "Super Admin":
      return ["Subject", "Details", "Urgency", "Submitted By", "Assigned Admin",
"Current Status", "Options"];
    case "Admin":
      return ["Subject", "Details", "Urgency", "Submitted By", "Current Status",
"Options"];
    default:
      return ["Subject", "Details", "Urgency", "Assigned Admin", "Current
Status"];
  }
};
```

The UI displays different tickets based on the logic of each column. Admins can see all tickets assigned to them. Employees can only see tickets assigned to themselves. The Admin's dashboard view is shown in Figure 7, while the Super Admin's dashboard, with full access to assignment and updates, is shown in Figure 8.

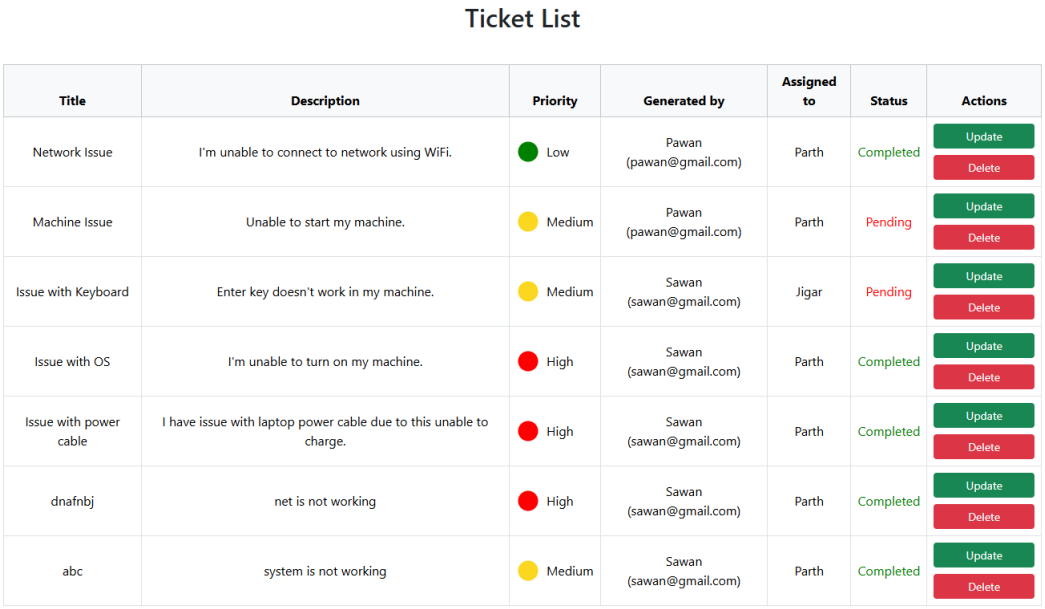
Figure 7 Ticket list for Admin



Ticket List

Title	Description	Priority	Generated by	Status	Actions
Network Issue	I'm unable to connect to network using WiFi.	Low	Pawan (pawan@gmail.com)	Completed	Update
Machine Issue	Unable to start my machine.	Medium	Pawan (pawan@gmail.com)	Pending	Update
Issue with OS	I'm unable to turn on my machine.	High	Sawan (sawan@gmail.com)	Completed	Update
Issue with power cable	I have issue with laptop power cable due to this unable to charge.	High	Sawan (sawan@gmail.com)	Completed	Update
dnafnbj	net is not working	High	Sawan (sawan@gmail.com)	Completed	Update
abc	system is not working	Medium	Sawan (sawan@gmail.com)	Completed	Update

Figure 8 Ticket list for Super Admin



Ticket List

Title	Description	Priority	Generated by	Assigned to	Status	Actions
Network Issue	I'm unable to connect to network using WiFi.	Low	Pawan (pawan@gmail.com)	Parth	Completed	Update Delete
Machine Issue	Unable to start my machine.	Medium	Pawan (pawan@gmail.com)	Parth	Pending	Update Delete
Issue with Keyboard	Enter key doesn't work in my machine.	Medium	Sawan (sawan@gmail.com)	Jigar	Pending	Update Delete
Issue with OS	I'm unable to turn on my machine.	High	Sawan (sawan@gmail.com)	Parth	Completed	Update Delete
Issue with power cable	I have issue with laptop power cable due to this unable to charge.	High	Sawan (sawan@gmail.com)	Parth	Completed	Update Delete
dnafnbj	net is not working	High	Sawan (sawan@gmail.com)	Parth	Completed	Update Delete
abc	system is not working	Medium	Sawan (sawan@gmail.com)	Parth	Completed	Update Delete

The Super Admin can handle users via EmployeeForm.js. With this, the admin can add an employee or an admin. This form is having a dropdown for user type and input boxes for name, email and password. When sent, the data is posted to the backend using Axios. The code handling the submission logic is shown in Program Code 4.

Program Code 4 Super Admin user creation logic (EmployeeForm.js)

```
const submitForm = async (event) => {
  event.preventDefault();
  setIsLoading(true);

  try {
    const url = employeeId
      ? `http://localhost:3000/api/employees/${employeeId}`
      : "http://localhost:3000/api/employees";

    const method = employeeId ? axios.put : axios.post;
    await method(url, employeeData);

    navigate("/employees");
  } catch (err) {
    setError("Unable to process employee record.");
  } finally {
    setIsLoading(false);
```

```
  }
};
```

The application uses bootstrap for styling and Axios for communication with the APIs. Session management is maintained through js-cookie to store the session and user role. Role-based routing, conditional rendering, and form access logic collectively guarantee that users interact only with the components that apply to their roles. This front end structure makes the support system interface intuitive, role safe, and scalable.

5.3 Back-end implementation

The server-side of the IT support ticketing system was created using Express.js with Node.js which is a lightweight and scalable server-side framework. The backend handles user registration, ticket generation and control, access based on roles, and communication with the MongoDB database. Every API endpoint follows REST architectural style and offers functionality of GET POST PUT DELETE. The system authenticates with JWT for secure user sessions and role-based access. The user sessions are encoded and verified on server.

Backend project structure is carefully designed in order to make it clear. The /routes folder has all API endpoints to manage various resources, like users, tickets, and authentication. The /controllers folder stores all application business logic separate from routing logic. The data models that represent the MongoDB collections are in /models folder and custom middleware functions like authentication checks are in /middleware folder. The /config folder is where the MongoDB connection settings are configured. The local server uses port 3000 and a MongoDB database with mongoose.

Authentication in the system is based on JWT tokens. Users login by submitting their email and password. After we successfully verify your credentials, we generate a JWT token and store it in cookies. The token is used to maintain session security and identify user roles during the following request. Program Code 5 presents the login user logic and JWT token generation.

Program Code 5 User login and token generation

```
const jwt = require("jsonwebtoken");

const authenticateUser = async (req, res) => {
  const { email, password } = req.body;
  const userRecord = await Employee.findOne({ email });

  const credentialsInvalid = !userRecord || !(await bcrypt.compare(password,
userRecord.password));
  if (credentialsInvalid) {
    return res.status(401).json({ message: "Access denied: incorrect login." });
  }

  const sessionToken = jwt.sign({ id: userRecord._id },
process.env.JWT_SECRET, {
    expiresIn: "24h",
  });

  res.cookie("token", sessionToken, { httpOnly: true });
  res.status(200).json(userRecord);
};
```

Another critical backend function is ticket creation and management. Support tickets can be created by employee users. Furthermore, Admin or Super Admin can update tickets, change the status and assign to other staff. Program Code 6 illustrates the ticket creation process, where a new ticket document is generated and saved in the database.

Program Code 6 Ticket creation logic

```
const createNewTicket = async (req, res) => {
  const newTicket = new Ticket({
    subject: req.body.title,
    details: req.body.description,
    urgencyRef: req.body.priority_id,
    requesterId: req.body.employee_id,
  });

  await newTicket.save();
  res.status(201).json(newTicket);
};
```

The system utilizes role based access control to restrict functionalities based on the role. User roles are checked at various API endpoints to ensure that only authorized roles can

perform certain actions. You could for example have only a Super Admin create or edit user accounts, whereas Admins and Super Admins update ticket statuses. Program Code 7 shows an example of a role validation middleware that enforces these restrictions.

Program Code 7 Role validation middleware example

```
const restrictToRoles = (...allowedRoles) => {
  return (req, res, next) => {
    const hasPermission = allowedRoles.includes(req.user.role);
    if (!hasPermission) {
      return res.status(403).json({ message: "Permission denied: role not
authorized" });
    }
    next();
  };
};
```

The most important REST API endpoints built for this system include /api/login for login authentication, /api/tickets for ticket management, /api/employees for creating employees, and /api/user-types.

For instance, the /api/login route uses the POST method to authenticate users for JWT. The /api/tickets endpoint uses POST to create tickets, PUT to edit ticket details and DELETE to remove tickets (only possible by Super Admins). Through /api/employees Super Admin can create a employee and through /api/user-types User can get the available role types such as Employee, Admin, Super Admin.

Database works are handled by Mongoose models that correspond to the collections in MongoDB (employees, tickets, usertypes and priorities). Mongoose is responsible for data validation, reference linking, and enforcing schemas at the model level.

The IT support ticketing system's Node.js and Express.js backend is flexible, scalable, and resilient, as explained here. It has every API authority to manage authentication, tickets workflow, and user roles. All server-side aspects use the MVC architectural pattern which separates the route and business logic. I have tested all the functions of the API using Postman to ensure they would work.

5.4 Database schema and models

The database system for this web-app uses MongoDB to create the IT support ticketing system database. MongoDB is a popular database that stores data in documents. Mongoose was utilized as an ODM library to structure the interaction with the database. Schemas, validation rules, and references can all be managed with mongoose. The major collections in the system are employee, usertype, ticket, and priority. Each collection has a defined relationship and structure that supports the business logic of the application.

The Employee model describes who are the users of the system, i.e. Employees, Admin, and Super Admin. Every document of an employee has name, email, password and reference to user type. UserType model defines different types of users in the system. Employee, Admin and Super Admin are supported.

Role based access controls are enforced using the above user types. Mongoose's ObjectId reference is used to create an employee-user type association, as shown in Program Code 8.

Program Code 8 Mongoose schema for UserType

```
const mongoose = require("mongoose");

const roleSchema = new mongoose.Schema(
  {
    roleName: {
      type: String,
      required: true,
      enum: ["Employee", "Admin", "Super Admin"]
    }
  },
  { timestamps: true }
);

module.exports = mongoose.model("Role", roleSchema);
```

The core of the system revolves around the Ticket model. Each ticket document contains information on the issue raised by an employee. Key information in important fields includes the issue's title, a detailed description of the problem, priority, the name of the employee submitting the request, the assigned Admin (if any) and whether it has been

completed or not. The complete schema structure for tickets is detailed in Program Code 9, where the fields are defined and referenced accordingly.

Program Code 9 Mongoose schema for Ticket

```
const mongoose = require("mongoose");

const issueSchema = new mongoose.Schema(
  {
    subject: { type: String, required: true },
    details: { type: String },
    resolved: { type: Boolean, default: false },
    urgencyId: { type: mongoose.Schema.Types.ObjectId, ref: "Priority" },
    submittedBy: { type: mongoose.Schema.Types.ObjectId, ref: "Employee" },
    assignedTo: { type: mongoose.Schema.Types.ObjectId, ref: "Employee" }
  },
  { timestamps: true }
);

module.exports = mongoose.model("Issue", issueSchema);
```

The Priority model sets how urgent the submitted tickets are. Priority records contain a type field such as Low, Medium or High. The system can organize support requests in order of importance. The definition of the Priority schema is provided in Program Code 10.

Program Code 10 Mongoose schema for Priority

```
const mongoose = require("mongoose");

const urgencySchema = new mongoose.Schema(
  {
    level: { type: String, required: true }
  },
  { timestamps: true }
);

module.exports = mongoose.model("Urgency", urgencySchema);
```

All of the schemas create a system to work together to form a complete and interlinked ticketing system. By using Mongoose, each schema maintains a clear structure with referential integrity while using MongoDB’s flexible document model. When a citation is made to a particular document from another document, for example, the ticket is related to

the submitting employee or priority type etc. This allows you to query the data in a complex way and get it populated, but not at a heavy performance cost.

The design of the database will be flexible, scalable and easy to expand in future. For example, we can add new roles, new ticket statuses, or extend the ticket data model without too much upheaval. The backend uses the schema validation and reference population features of Mongoose to make sure user interactions and ticket workflows are managed consistently and securely.

In conclusion, the database structure is specially designed for the IT support ticketing system to be robust and scalable. Through the relationships between the collection defined in the database, whose effective supports user management, ticket management, role control, and priority control are achieved.

5.5 Security and authentication

The design and development of the IT support ticketing system utilize security and authentication protocols. The system prevents users from accessing other user information and limits users' access to sensitive resources. Password storage is secure and employs tokenization based on JSON Web Tokens (JWT). The implementation follows role-based permissions and session management based on cookies.

The login process is the first line of defense. When we try to log in, the entered credentials are validated against the stored values in the database. When it is written, the password is never stored as plain text. It gets stored in the database after passing through bcrypt. To ensure that no one can extract original passwords from the database in case of a breach, passwords are saved after hashing. The bcrypt implementation for hashing user passwords before saving to the database is shown in Program Code 11.

Program Code 11 Password hashing using bcrypt

```
const bcrypt = require("bcryptjs");

userSchema.pre("save", async function (next) {
  if (!this.isModified("password")) {
    return next();
```

```
  }

  const saltRounds = 12;
  this.password = await bcrypt.hash(this.password, saltRounds);
  next();
});
```

When a user logs in successfully, a JWT token is created that contains user ID. The browser saves this token as a secure HTTP-only cookie, preventing client-side scripts from accessing or modifying the token. This reduces the chances of cross-site scripting (XSS) attacks. JWT include expiry time, which de-authenticates the user session automatically after a certain period of time. As was shown in program code five in Section 5.3, the architecture for JWT token generation and storage.

Besides being securely authenticated, the system employs RBAC for managing user permissions. During account creation, various roles such as Employee, Admin, and Super Admin are assigned. Every API route verifies the role of the logged-in user and decides whether the resource should be allowed or denied to them. Middleware functions validate the JWT and extract user roles inside the request before accessing restricted endpoints in an Express Node.js app. The token verification and role validation logic is demonstrated in Program Code 12.

Program Code 12 JWT verification middleware

```
const jwt = require("jsonwebtoken");
const User = require("../models/Employee");

const verifyAuthToken = async (req, res, next) => {
  const tokenFromClient = req.cookies.token;

  if (!tokenFromClient) {
    return res.status(401).json({ message: "Token missing. Authentication
required." });
  }

  try {
    const payload = jwt.verify(tokenFromClient, process.env.JWT_SECRET);
    req.user = await User.findById(payload.id).select("-password");
    next();
  } catch (error) {
    res.status(400).json({ message: "Token validation failed." });
  }
```

```
};

module.exports = verifyAuthToken;
```

When we analyze the security model, we see that Employees can only create and view their tickets, Admins can update the status of the tickets assigned to them, and Super Admins can manage users and tickets. The enforcement of permissions prevents the use of unauthorised functions and helps maintain platform integrity.

Session cookies expiration and cookie security makes session management more effective and stronger. Cookies are going to be HTTP only and they can also have the Secure flag when deployed over the HTTPS. It ensures that cookies are sent only over secure channels.

The system uses standards from the industry such as bcrypt hash, JWT authentication, role-based access control, security cookie, and expiration session. Combining these practices guarantees a protected environment from user data breaches, made possible by not allowing unauthorized parties to access the data/uploads of users.

5.6 UI and User experience

IT support ticketing system UI is designed in a simple, clear, and user-friendly way so that the user easily understands. Also, React.js is used in the front end so that it can be a Responsive and Dynamic design for multiple users: Employees, Admins, and Super Admins. To ensure that every user may effectively use the system to perform intended actions, regardless of technical expertise, the overall UX is designed accordingly.

When the users visit the system, they first see a neat and professional-looking login page. The center of the screen displays the Login form comprising the email and password input fields. The form is simply validated so that when a user does not write anything in the field they get an immediate feedback. A corporate style logo is put above the form to enhance system identity. Once a user login on dashboard, the user is automatically redirected to his dashboard based on the account credentials saved in the cookies.

The dashboard layout changes depending on the logged-in user. For example, on a simple dashboard, employees can submit new tickets and check the status of their own requests. Super Admins, on the other hand, are given a wider dashboard view that allows them to manage employee accounts, view tickets all submitted, assigned tickets to Admins, operations etc. An Admin user dashboard is shown to update the tickets they are assigned and mark them as complete. The AdminLayout.js component centrally manages the routing based on role and rendering of the dashboard. It is an important part of the front-end.

The TicketForm.js Component makes it easy to create and edit tickets. To submit a ticket, employees must fill in a title, description and select its priority level from a drop-down menu (Low, Medium, High). All forms validate real-time so no wrong information is submitted. When the ticket form gets opened for editing by Super Admin or Admin user, they will get more additional ticket fields that allow them to change the assignment of ticket to another admin or change the completion status of the ticket.

The ViewTickets.js component is responsible for the ticket viewing experience. The component renders the tickets in a flexible table design. The table columns adapt based on user roles. Employees can view their ticket submissions, status updates, and designated admins. The Admins are able to check the tickets assigned to them. The Super Admins can see all the tickets. They can edit or delete a ticket as well. Improved use of colours: Now, users can easily see which tickets are priority tickets through the circular icons visible in the list view. Users will see red icons for high priority tickets, yellow for medium priorities and green for low priorities all of which enhance usability and lower cognitive load.

Navigation through the system is user-friendly and contains a side navigation bar (Sidebar component) which gives links access to the ticket lists, employee lists (for Super Admins), and logout button. By using similar icons and clear labels, navigation is easier.

For the styling of the application, we leverage Bootstrap classes for a consistent, mobile-friendly, and accessible design. Buttons, forms and tables are optimized for readability. Also, media queries ensure that pages render well on desktop and mobile devices.

The logout process is also a smooth experience. Clicking the logout button destroys session cookies and takes back the user to the login screen in a safe manner without compromising on security and the UX.

While developing the UI, the attention was paid to keep the design reach and user-friendly. The labels of the form inputs are correct, the error messages indicate logical mistakes, and the contrast is maintained between the text and the background.

The clean interface, role-based view, responsive layouts, visual clues and accessibility options of the IT support ticketing system creates a professional, effective and efficient experience for all users.

5.7 Notifications or ticket updates

The IT related support ticketing system has an easy but effective mechanism for users to get feedback and updates on their ticket actions.

Since there are no email alerts or push notification in place in the system, the user is always ensured the status of the actions they have performed through on-screen notifications and real-time feedback reflected on the UI of the system.

An employee fills in the TicketForm.js component to submit a new ticket. The new ticket will be validated and created instantly if successful. A success alert or simple page redirection indicates that the ticket has been successfully submitted. Right away, we can let users know the status of their request. Thus, this helps maintain customer confidence and assurance that the employee knows their request has been logged.

Just like this, even an Admin or Super Admin editing a Ticket Status or changing its assignment will result in automatic changes in the Ticket list shown in ViewTickets.js. The system refreshes the displayed tickets right after the completion of the update operation. As a result, the users do not have to refresh their page, log out and log in again to see them in action.

There is a visually distinctive treatment for tickets which marked as completed. As soon as the admin updates the ticket status in the ViewTickets.js table, the status field is changed

to Completed. The display of tickets is enhanced through visual styling: completed tickets may be shown in a specific colour or with a badge, so users can easily see which of their issues have been fixed. This visual update not only makes life easier for IT staff by preventing information overload while dealing with multiple tickets but also assures employees that their requests are addressed on a timely basis.

If there's a problem with the form or an invalid input, they'll get instant feedback. An example of a required field is the ticket title. The form will not allow submission until an entry is made and will show an error message close to the field as a reminder. This inline validation reduces annoyance and status errors.

Also, pop-ups are used for the critical things like ticket deletions. When a Super Admin tries deleting a ticket, a confirmation modal opens, requiring the user to explicitly confirm the delete. The design makes data loss less likely, and users have total control over what happens.

At present, the system does not send notifications via WebSocket or email yet users receive feedback immediately on screen, the UI updates in real-time, and a specific status colour indicates the feedback status beside the running visual of the machine that also turns red in case of a problem. Along with this, the information also receives a form validation similar to the mobile phone text field. This assures a very efficient user experience.

In future updates, we could set up email notification alerts for major incidents like a ticket assignment, ticket completion notification and change in user account. Teaming up with outside services like Twilio or Firebase could make alerts even more powerful to facilitate SMS and in-app push notifications. Nonetheless, even in its current state, it provides reliable, visible, and timely updates that meet the operational requirements of a typical IT support setup.

6 Testing and evaluation

In this chapter the testing process is described and its effectiveness in meeting the original research goals is examined. It contains the testing strategies, results of the testing and how the final implementation matches with the research questions stated in the introduction.

6.1 Testing methods

All the features of the IT support ticketing system were tested to ensure that it functioned correctly, securely and reliably for all types of users in their workflows. The purpose of testing was to make sure that the system was working according to its specification, user-friendly, and safe. To ensure that the system is working properly, manual functional testing and API testing were utilised.

Manual functional testing was performed with actual interaction with a user interface. Each each feature was tested for Employee, Admin, and Super Admin so that the permission and functionality behaved as it was supposed to. Employees were tested for their ability to log in, submit tickets, and track ticket status. The admins were tested with updating a ticket's status and replying to the assigned tickets, while super admins were tested with user management, ticket assigning, and overall control of the system.

All roles underwent positive testing (assessing standard processing) and negative testing (assessing rejection of invalid action). For example, the role-based access control mechanisms operated as intended when the Employee attempted to access any Admin level pages, preventing them from continuing with an error. The testing process also assures that the required fields, such as title, description and priority, are enforced at the point of ticket submission. The system could identify incorrect or missing entries and display useful error messages to the user efficiently.

We directly contacted the system's application programming interface (API) endpoints using Postman, a popular API client tool, to perform the system's testing. It made possible to test the workings of authentication flows, ticket creation and updating, user management, and roles assignment independent of the front-end. testing was conducted each endpoint for the handling of requests, data validation, response codes, and security

behaviours. Successful ticket creation returned a 201 Created response with the ticket object. Whereas, unsuccessful attempts for authentication returned a 401 Unauthorized status code. Thus, confirming the security protocols were intact.

A special focus was given to JWT-based authentication to check whether valid tokens were required to access protected routes and whether invalid or expired tokens correctly denied access. The behavior of token expiration was also manually simulated to ensure that users were logged out on session expiry.

Also, usability testing was done informally by allowing a handful of test users to use the app and provide feedback on it. The user's experience validated that the UI was intuitive, navigation was clear, and major usability software issues were not encountered for core workflows like logging in, submitting tickets, and managing tickets.

Blending the manual interface testing with backend API Testing gave it proof of overall stability, reliability, and security of the whole system. The complex testing process indicates that the IT support ticketing system can be deployed in an organizational environment. The employees, IT personnel, and administrators support operations on a day-to-day basis.

6.2 Evaluation against research questions

The major goal of this thesis was to create a web-based IT support ticketing system and verify whether the developed system can provide answers to the problem statement presented in Chapter 1. Based on the design, implementation as well as the testing of the system, the following evaluations can be made.

The first research question that investigated the primary concerns in IT support which decrease efficiency and productivity. In Chapter 2 of this thesis traditional IT support systems were analyzed to address these issues. It was observed that fragmented modes of communication, lack of a centralized issue reporting mechanism, slow response time and ineffective prioritization were serious blockers. The system solves these problems by providing a direct link for ticket submission, status checking, and ticket resolution to tackle all the issues centrally.

The second research question focused on how to improve IT support workflow and decrease issue resolution time with a web-based ticketing system. The system provides web-based communication between employees and IT staff by designing and implementing a web-based platform using React.js, Node.js, Express.js and MongoDB. The flow of work has improved significantly due to functionalities like real-time ticket updates, efficient dashboards for different roles, and dynamic tracking of ticket statuses. Through manual and API testing, it was verified that the system would deliver faster and more structured support than before.

The third research question wants to know what the ticketing system must have that could improve IT support efficiency. The system that was developed consisted of some features which were found necessary during research. The web application included secure login and authentication via JWT, access control differentiating Employee, Admin and Super Admin functionality, ticket priority, live ticket status monitoring, and user management (admin). Other aspects like error validation, visual indicators for ticket urgency, and session-based navigation further augmented the user experience and efficiency of the support process.

In summary, the IT support ticketing system created within the scope of this project answers all three research questions. It supplies one single platform that makes IT support operations more effective, easier to secure, and easier to use. The system consistently meets the goals established at the beginning of the thesis as evidence through extensive testing and evaluation.

7 Results

The developed web-based IT support ticketing system successfully achieved the technical objectives of the project. The system's API logic, authentication flow and role-based ticket management were tested using Postman and browser simulation. Author ensured that the Employee, Admin, and Super Admin in the system all worked fine. The results from this test confirmed that routing and session are handled correctly. Also, the ticket tracking stays the same throughout.

Thanks to MongoDB, too often changing structures of ticket data could be handled dynamically. Also, the use of React ensured real-time responsiveness.

The tests showed that it enhances transparency, speeds up response time, and centralizes support communication. A few benefits of getting an automated follow up will be regular priority, status update and dashboard by user and will enhance productivity overall.

There was no client for this thesis, but after user testing was done informally and received positive feedback on interface clarity and access control based on role. To validate the performance of systems on a larger scale, further testing needs to be conducted.

8 Conclusion and future work

The goal of this thesis was to design a secure and scalable web-based ticketing system for IT support. The system was able to address the research questions through the identification of the challenges posed by traditional IT processes, the leveraging of full-stack technology to engineer the system, the integration of ticket tracking, user roles, notifications, and more.

As shown by the study, old IT support systems have difficulty communicating, do not prioritize stuff, and delay fixing faults. The issues were solved through single ticket submission, dynamic workflows as per role, and instant updates.

All three research questions were answered.

- To answer the main research question concerning IT support limitations and resultant impacts, section 2.2 looks into traditional system woes.
- In the sections 2.3 and 6.2 respond to the second research question by exploring the aspects of the benefits of web-based ticketing systems and describing how software solutions assist workflow improvement as well as shorten the periods for issue resolution.
- Regarding the third research question, which relates to the most important features in order to advance IT support the fundamental technical aspects and their part in enhancing the performance of the system and the user are analysed in Sections 5.5 and 6.2.

In the future, it can improve on mobile responsiveness, email or Slack integration, multilanguage support, and analytics dashboards. It is also recommended to conduct further usability studies to real organizational IT teams.

This project shows how real-world operational problems can be tackled with full-stack development. The organization developed a solution to increase staff satisfaction and IT visibility whilst reducing downtime, by replacing point methods with structured support workflows.

References

CRM.org. (2025). *Freshdesk vs Zoho Desk Comparison 2025: Which Is Better?* Retrieved from https://crm.org/news/freshdesk-vs-zoho-desk

DeskDirector. (2024). *4 Challenges When Introducing an Automated Ticketing System.* Retrieved from https://www.deskdirector.com/dd-blog/4-challenges-with-an-automated-ticketing-system

Electric. (2024). *What is IT support?* Retrieved from https://www.electric.ai/blog/what-is-it-support

GeeksforGeeks. (2023). *What is Full Stack Development?* Retrieved from https://www.geeksforgeeks.org/what-is-full-stack-development/

GeeksforGeeks. (2023). *ReactJS - Introduction.* Retrieved from https://www.geeksforgeeks.org/reactjs-introduction/

GeeksforGeeks. (2023). *Express.js - Introduction.* Retrieved from https://www.geeksforgeeks.org/express-js/

GeeksforGeeks. (2024). *Most In-Demand Skills to Become a Full Stack Developer.* Retrieved from https://www.geeksforgeeks.org/most-in-demand-skills-to-become-a-full-stack-developer/

GeeksforGeeks. (2025). *MongoDB: An Introduction.* Retrieved from https://www.geeksforgeeks.org/mongodb-an-introduction/

IBM. (2023). *Full Stack Developer.* Retrieved from https://www.ibm.com/topics/full-stack-development

Lemonaki, D. (2021). *What is a Full Stack Developer? Full Stack Engineer Guide.* Retrieved from https://www.freecodecamp.org/news/what-is-a-full-stack-developer-full-stack-engineer-guide/

MongoDB. (n.d.). *What Is Full Stack Development? A Complete Guide.* Retrieved from https://www.mongodb.com/resources/basics/full-stack-development

MongoDB. (n.d.). *Introduction to MongoDB.* Retrieved from https://www.mongodb.com/docs/manual/introduction/

Muntakim. (n.d.). *Traditional IT support systems: A comprehensive analysis of drawbacks.* Retrieved from https://www.linkedin.com/pulse/traditional-systems-comprehensive-analysis-drawbacks-muntakim-pwvqc/

Node.js Official Documentation. (2023). *About Node.js.* Retrieved from https://nodejs.org/en/about/

OWASP. (n.d.). *OWASP Top 10 – Web Application Security Risks*. Retrieved from https://owasp.org/www-project-top-ten/

React Official Documentation. (2023). *Introduction to React.* Retrieved from https://react.dev/learn

SourceForge. (n.d.). *Freshdesk vs. Jira Service Management vs. Zoho Desk Comparison Chart.* Retrieved from https://sourceforge.net/software/compare/Freshdesk-vs-Jira-Service-Management-vs-Zoho-Desk/

Supportbench. (2024). *10 Challenges to Upgrading Your Customer Support and Ticketing Platform.* Retrieved from https://www.supportbench.com/10-challenges-to-upgrading-your-customer-support-and-ticketing-platform/

ThriveDesk. (2024). *Zoho Desk vs JIRA: A Comprehensive Comparison.* Retrieved from https://www.thrivedesk.com/zoho-desk-vs-jira/

Verizon Business. (n.d.). *7 Benefits of a Help Desk Ticketing System.* Retrieved from https://www.verizon.com/business/resources/articles/s/seven-benefits-of-a-help-desk-ticketing-system/

Appendix 1. Data management plan

The next section is an explanation of the thesis project data management plan, matching with the HAMK' criterions. A complete description of details about research data such as its collection and storage methods, ethical considerations, ownership arrangements, and future handling is discussed in this appendix.

Description of Thesis Research Data

The following are used to construct this thesis; system source code, data structures of databases, front end and back end implementation files and test results. These were produced at the design, development and evaluation stages of a web-based IT support ticketing system. No personal interview, survey or collection of human participant data methods were used. An auto created sample data using fictitious test accounts and auto generated sample data was used for all development and testing. These test scenarios were created in order to create interactions of Employees, Admins, and Super Admins to validate use cases.

Method of Data Collection

Data collection was working-oriented, and implemented. The author simulated and validated workflows through manual interaction with the system. All research data are structured in JavaScript and JSX files (for FE and BE), JS Onfiguration files and UI workflow screenshots. The data informed the building of features like role-based authentication, ticket submission, status tracking and ticket assignment workflows. No real users were tested and no personal or sensitive data was processed.

Data Privacy and Ethical Considerations

Because there was no involvement of any external data and no human participant was involved, therefore there was no need of Privacy Notice or any form of Informed Consent. Every test account used was 100% fictional and not connected to anywhere. As a consequence, the system did not collect, process or store any personally identifiable information. There was no threat of data leakage or unauthorized disclosure and, therefore, encryption as well as anonymization are unnecessary.

Data Storage and Security

All research related data, including source code and documentation was stored in the thesis author's password protected laptop. Regular backups have been taken to a local folder. A private GitHub repository with read-only access only to the author can be used as a backup with version control. There was no data placed on unsecured cloud platforms, nor was there any data with unauthorized parties.

Intellectual Property and Licensing

The data and code generated for this project are property of the thesis author. The project was developed independently and there are no co-development or external collaboration and no agreement exists about shared ownership or reuse rights. The libraries and frameworks employed (React.js, Node.js, MongoDB) are open source and if used in an academic capacity are regulated under applicable licenses.

Data Retention and Disposal

The data will be stored for one year, as described in HAMK's data verification policy, after thesis has been approved. During this period, it will continue to be held in secure local and GitHub repositories for purposes of audit. One year later all local files and obtrusive backups will be permanently removed. The complete dataset will not be presented publicly. Only relevant screenshots and code snippets are featured in the final submission of the thesis for purposes of illustration. The research data is not planned to be reused nor published in future projects.