Introduction

Introducing the Hyperion Tech Service Request System, a ticketing system designed to streamline processes for a creative digital agency dealing with web design and development. The system aims to help the company manage multiple projects by creating a ticketing system that tracks client requirements, assigns tasks to the appropriate team members, logs work hours, and monitors progress. The Hyperion Tech Service Request System is designed to support user management, ticket subsystem, time and expenses tracking, and AI analysis. The system also includes architectural and technology specifications, such as security, scalability, performance, and maintainability requirements, and technology stack suggestions. Overall, the Hyperion Tech Service Request System is an innovative solution to the challenges faced by digital agencies, and it promises to improve project management and client satisfaction.

Objectives

Objective Evaluation of the Hyperion Tech Service Request System:

The Hyperion Tech Service Request System, as described, has the potential to significantly impact a creative digital agency positively. Its main strengths and potential impacts are as follows:

Streamlined processes: By implementing a ticketing system, the agency can better manage client requirements and assign tasks to appropriate team members, leading to improved organisation and efficiency.

Enhanced collaboration: The system enables seamless communication between the functional, technical, and management teams, allowing for real-time updates on ticket status, inquiries, and testing results.

Time and expense tracking: The inclusion of a time and expense tracking subsystem will help the agency monitor work hours and expenses, leading to better resource management and potentially reduced costs.

AI-assisted ticket assignment: The AI subsystem can help identify the most suitable employees for each ticket based on their skills and efficiency, leading to optimised resource allocation and potentially improved project outcomes.

Improved security and user management: The system's security features and user management capabilities ensure that only authorised individuals have access to sensitive information, maintaining confidentiality and compliance with data protection regulations.

Scalability and performance: The system is designed to accommodate growth in the number of teams, projects, and data without affecting performance, allowing the agency to expand its operations smoothly.

User Characteristics

Management Team:

- Responsible for overseeing the entire operation of the digital agency
- Involved in high-level decision-making and project approvals
- Have the authority to create user accounts for team members
- Can view all tickets and their progress, regardless of group assignment
- Monitor and manage time and expenses for each user and project
- Confirm and mark tickets as complete after testing and approval

Functional Team:

- Responsible for gathering client requirements and creating project specifications
- Communicate with clients to address their needs and concerns
- Collaborate with the technical team to ensure accurate implementation of project requirements
- Create and assign tickets for each project, including priority and deadline
- Test developed solutions to identify errors and ensure functionality
- Upload test evidence documents and communicate with management for final confirmation

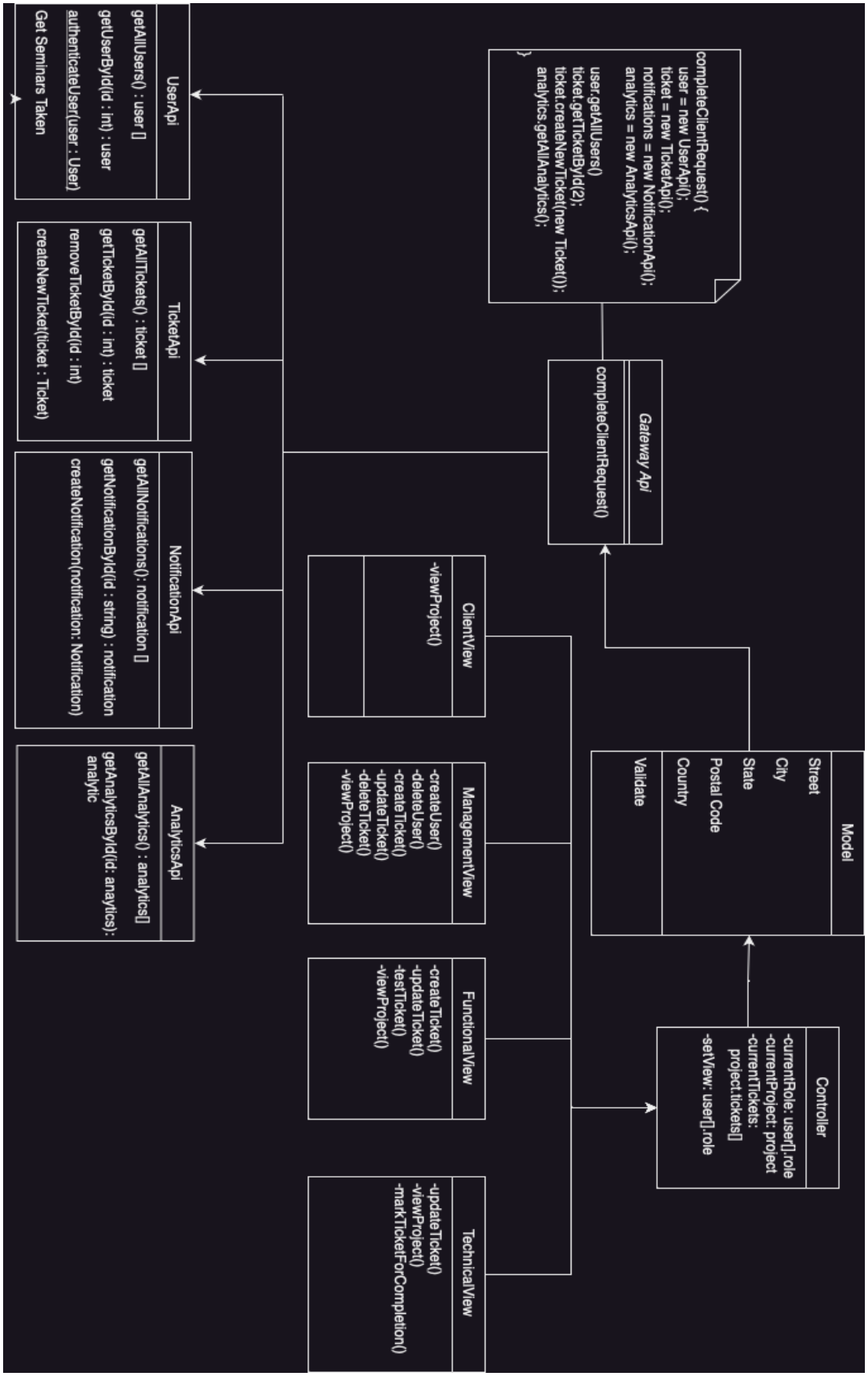
Technical Team:

- Responsible for implementing project requirements based on functional specifications
- Work on assigned tickets, providing updates on progress and status
- Collaborate with the functional team to address inquiries and ensure correct implementation
- Fix errors identified during testing and update tickets accordingly
- Log work hours for each project, ensuring accurate billing and resource allocation

Client:

- Communicates with Functional Team about the requirements
- Change requests regarding the ticket

Class Diagram



Development Tools

Angular

- We will be using Angular for the front-end. We will be using Angular because we have had experience with this programming language with many of our previous modules. We will also make use of open source libraries such as material UI.

NodeJS

- NodeJS provides a professional and efficient way to build scalable, real-time web applications and it is built on Javascript.
- Therefore, NodeJS will be used for our back-end. This programming language integrates well with Angular as it uses Typescript, which is a superset of Javascript.
- This means that front-end and back-end will be written in compatible languages and in turn minimises risks such as front-end and back-end incompatibility, which will provide for a better experience.

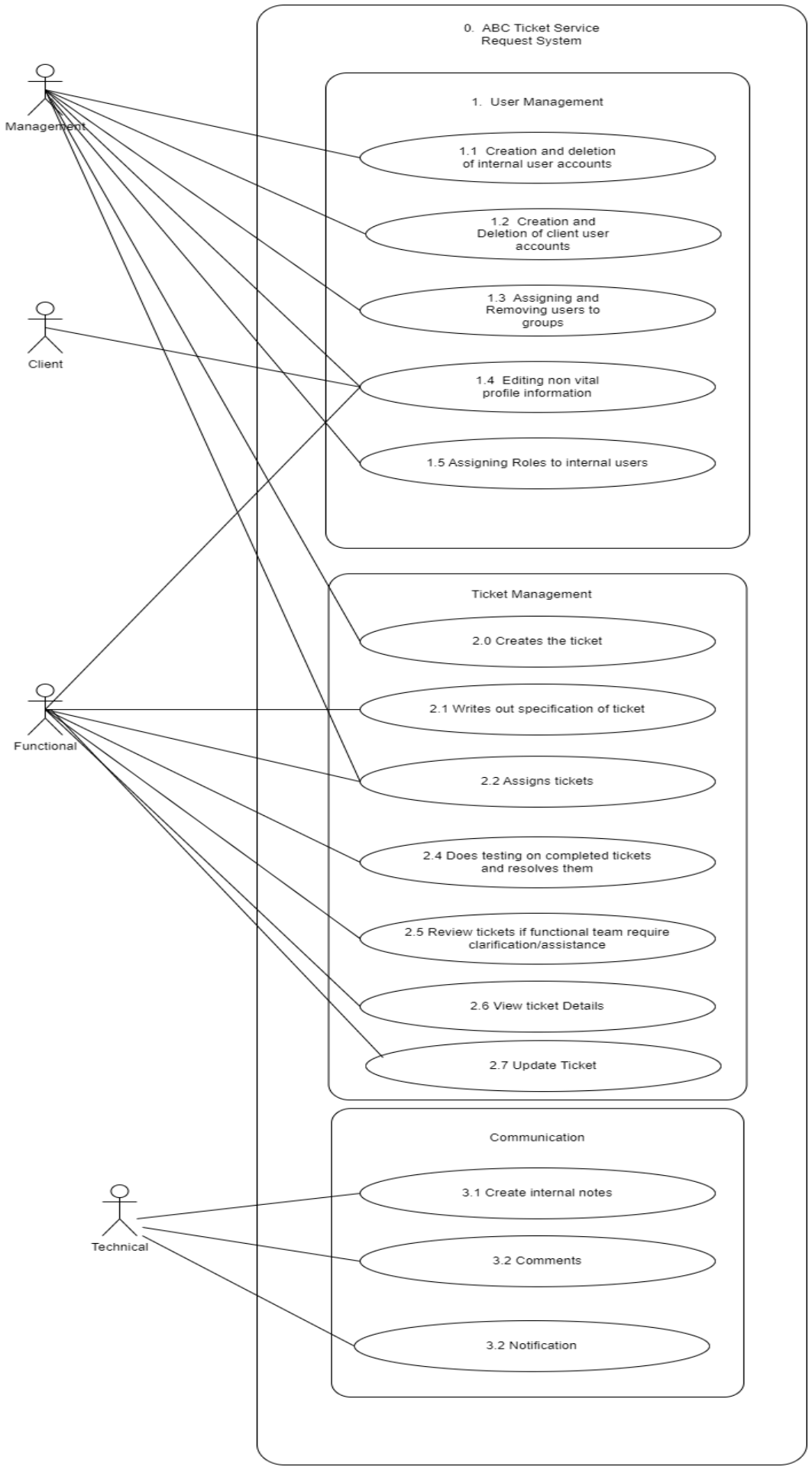
MongoDB

- MongoDB is a NoSQL Database that will allow for photos and other media to be stored together in a single repository. It is a high-performance database which helps with query performance and reduces response times.
- MongoDB is also designed to be highly scalable which fits with the specifications of this project. It is also a database that has an excellent integration with Angular and NodeJS, which are the frameworks we will be using in this project.

TensorFlow and Keras

- We will be using a combination of TensorFlow and Keras for this project. TensorFlow and Keras offer a wide range of tools and resources to implement the AI part of the specifications. This makes building and training of these models much simpler. Although TensorFlow and Keras are more complex, this combination can provide a flexible, high performance and scalable framework. TensorFlow and Keras work hand-in-hand so there will be no issues on compatibility.

Use Case Diagram



User Interaction

Account Creation and Login:

1. The management team creates user accounts for team members.
2. Users receive an email with an authorization code to verify their identity.
3. Users log in to the system using their email and password.

Ticket Creation and Assignment:

1. The functional team gathers client requirements and creates a ticket for the project.
2. The ticket is assigned to a group and then to an individual within the technical team.
3. The AI subsystem can recommend the most suitable technical team members based on skills and efficiency.

Ticket Workflow:

1. The assigned technical team member works on the ticket, changing its status to "Active."
2. If the technical team member has any inquiries, they write an internal note on the ticket and assign it back to the functional team member.
3. The functional team member addresses the inquiry, writes a note, and reassigns the ticket back to the technical team member.

Testing and Approval:

1. Once development is complete, the ticket is assigned to a functional team member for testing.
2. If errors are identified, the ticket is returned to the technical team member for corrections.
3. After successful testing, a test evidence document is uploaded, and the ticket is assigned to a management team member for final confirmation.
4. The management team member marks the ticket as “Done”.

Change Requests:

1. If a client requests changes on a completed ticket, a new ticket is created referencing the original ticket.

Notifications:

1. Users receive notifications via the app whenever any updates or internal notes are made to existing tickets.
2. Users receive notifications via the app whenever they are assigned a new ticket.
3. Users can view new notifications upon logging in to the system.

Time and Expense Tracking:

1. Users log their work hours for each project, ensuring accurate billing and resource allocation.
2. The management team can view each user's hours and projects they are involved in.

User Settings and Profile Management:

1. Users can update their non-vital information, such as name, surname and profile picture.
2. Users can choose their preferences for their notifications.

Functional Requirements

- Ticket Creation: The ability of a user to create a ticket.
- Ticket Specification: The ability of a user to specify the details of the ticket.
- Ticket Assignment: The ability of a user to assign tickets to a specific user or team based on their relevant expertise, workload, and availability.
- Ticket Prioritization: The system should allow users to set the priority of their tickets, to help support staff prioritize their work and resolve critical issues quickly.
- Ticket Viewing: The ability of any user to view the details of the ticket.
- Ticket Updating: The ability of a user to update the specifications and details of the ticket when the need arises, and when these changes are made the users assigned to the ticket are notified.
- Ticket Tracking: The ability of any user to track the status of each ticket.
- Ticket Resolution: The ability of a user to resolve tickets and mark them as complete, active or pending .
- Ticket Collaboration: The ability for users to collaborate on tickets with one another.
- Ticket History: The system should maintain a history and audit trail of all ticket-related activities, to ensure accountability, compliance, and auditability.
- User Management: The ability of management users to create, manage and delete user accounts, set roles, and permissions. The ability to create, manage and delete client accounts.
- User Authentication: The system requires users to authenticate themselves.
- User Group Assignment: The ability of users to assign and remove users from their respective groups.
- User Profile Editing: The ability to edit certain details of a users' profile.
- User Role Assignment: The ability of management users to assign users' a certain role.
- Notifications: The ability to notify users when they have been assigned a new ticket or when a ticket's status has been updated.
- Clients should be able to make requests without having to directly contacting their consultant.
- Clients should be able to keep track of their request's status

- Clients should have an in-app means of contacting a consultant

Quality Requirements

- Performance: The system should be able to handle a high volume of concurrent users and requests without having to experience issues like system crashes or slow response times. The system also keeps track of Time to First response, the appropriate time is 2 hours from when the request has been made.
- Reliability: the system should be available 24/7 with minimal down time. Clients are able to make requests and create new projects even if they are not able to contact a consultant immediately.
- Scalability: the system should be able to handle a growing number of users and requests on the platform.
- Security: The system should be secure so that the confidentiality of ticket data, user accounts, and communications is not compromised. User's are required to have an account created for them internally, and then they are sent a verification email where they have to change their password.
- Usability: the system should be simple and easy to use, it should have an intuitive interface so that the time spent on training or support can be minimised. In the event a user is unable to figure something out there is a user's manual to help them. Even those who are computer illiterate can use the system.
- Maintainability: the system should be easy to maintain, with clear documentation, modular design, and proper coding practices to support any updates or enhancements made in the future. We have made a coding standards document to support this.
- Interoperability: the system should be able to interoperate with other systems such as email or video/voice chat.
- Compliance: the system should comply with relevant industry standards and practices. i.e SLA's

Design Patterns

- Facade is used in our microservices, providing a simplified interface to a larger body of code. This makes our system easier to use, test, and manage. The façade is the gateway API in our system.
- The State pattern, implemented in our MVC structure, allows an object to alter its behaviour when its internal state changes, leading to cleaner code and more flexible logic.

Architectural Strategies

N-Tiered Architecture

- Divides the application into multiple layers with specific responsibilities.
- Common tiers include the presentation/UI layer, business logic layer, and data access layer.

MVC (Model-View-Controller)

Separates the application logic into three interconnected components:

- Model: Represents data and business logic.
- View: Handles presentation layer and user interface.
- Controller: Manages data flow and interactions between the model and view.
- Enhances modularity, maintainability, and testability.

Microservices

- Builds the application as a collection of small, independent services.
- Each service focuses on a specific business capability.
- Services can be deployed and scaled independently.
- Communication between services occurs via APIs.
- Provides flexibility, scalability, and fault isolation.

Architectural Quality Requirements

- Performance: Ensure the system meets response time and throughput requirements.
- Reliability: Design for high availability, fault tolerance, and error handling.
- Scalability: Handle increasing workloads and accommodate growth.
- Security: Protect the system and its data from unauthorised access and breaches.
- Maintainability: Enable easy changes and enhancements over the system's life cycle.
- Usability: Provide intuitive user interfaces and accessibility across devices.
- Testability: Support effective and automated testing of the system.
- Interoperability: Enable seamless interaction and data exchange with other systems.

Architectural Constraints

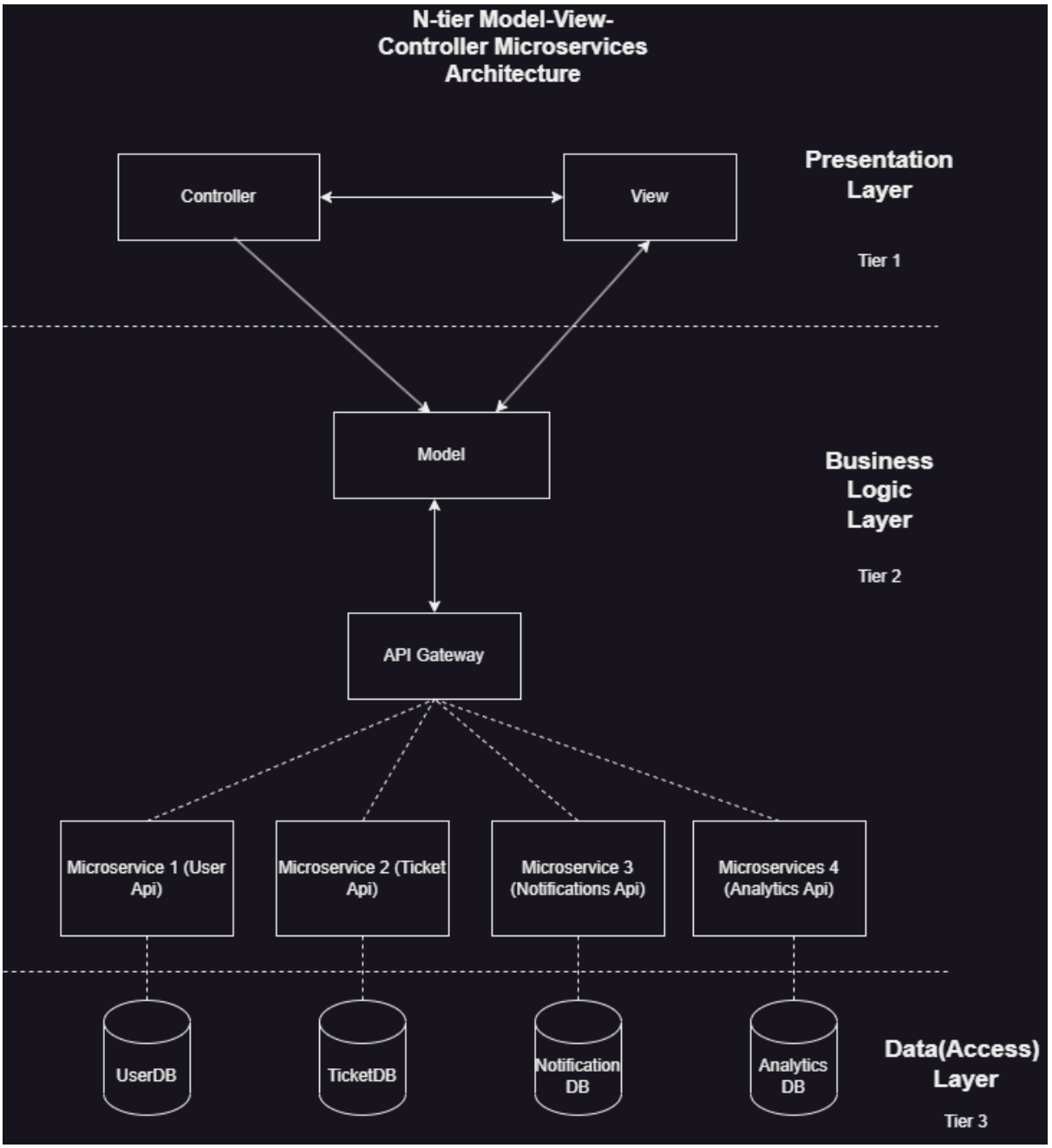
1. Client Requirements:
   - User interface constraints: The client may have specific user interface design guidelines or branding requirements that need to be followed.

2. Deployment Constraints:
   - Hosting environment: The system may need to be deployed on a specific infrastructure, such as on-premises servers or a cloud platform

- Scalability requirements: The system may have to handle a high number of concurrent users or accommodate future growth.

3. Performance and Security:
   - Performance constraints: The system may have specific performance requirements, such as response time targets or throughput expectations. These constraints can influence architectural choices regarding caching, load balancing, or data storage optimizations.
   - Security requirements: The system has specific security requirements, such as encryption, access control.

Architectural Design:



The MEAN stack

1. MongoDB:
    - MongoDB is a NoSQL database that offers flexibility in data modelling and scalability. It stores data in a JSON-like format (BSON) and provides rich querying capabilities.
2. Express.js
    - Express.js is a web application framework for Node.js. It provides a minimalist and flexible approach to building web applications and APIs.
    - Express.js simplifies routing, middleware management, and request handling, making it well-suited for creating server-side components in our architecture.
3. Angular:
    - Angular is used for building dynamic and interactive web applications. It follows the MVC architectural pattern and provides a declarative approach to building user interfaces.
    - Angular simplifies data binding, dependency injection, and component-based development, enhancing code maintainability and reusability.
4. Node.js:
    - Node.js is a runtime environment that allows JavaScript to run on the server-side. It provides an event-driven, non-blocking I/O model, making it efficient for handling concurrent requests. Node.js is known for its scalability and high performance, making it a good fit for building the backend components of your application.

Conclusion:

The MEAN stack, which stands for MongoDB, Express.js, AngularJS, and Node.js, is a full-stack JavaScript solution that helps in developing fast, robust, and maintainable web applications. It's suitable for N-tier MVC (Model-View-Controller) microservice architecture due to a number of reasons:

1. Language Consistency: the entire stack is JavaScript-based, it maintains consistency across all layers of an application. This is beneficial for developers, as they can work seamlessly across different parts of the stack without needing to switch languages or understand different programming paradigms.
2. Separation of Concerns: The MEAN stack can be mapped neatly to the MVC architecture. MongoDB acts as the model, AngularJS as the view, and Express.js/Node.js as the controller. This clear separation aids in implementing N-tier architecture, where different responsibilities are divided among different tiers.
3. Microservices Ready: Node.js is ideal for developing lightweight, high-performing microservices. Node.js apps can be containerized using technologies like Docker and orchestrated with Kubernetes, allowing for easy scaling, load balancing, and other benefits of microservice architecture.
4. Scalability and Performance: Node.js is known for its event-driven, non-blocking I/O model, which makes it lightweight and efficient, perfect for data-intensive real-time applications. MongoDB is also horizontally scalable, allowing your database to grow as needed

5. Full Stack Development: With the MEAN stack, you can take care of everything from the front-end to the back-end, to database and even debugging. This can be extremely efficient and cost-effective for the development process.

Coding Standards:

File Structure:

- Each component, service, and model should have its own file.
- Group related components, services, and models in separate folders.
- Use meaningful and descriptive names for files.

Import Statements:

Import statements should be organised in the following order:

1. Angular core modules
2. Third-party modules
3. Custom modules and components
4. Services and other dependencies

Component Naming:

- Component file names should be in kebab-case, e.g., create-account.component.ts.
- Component class names should use PascalCase, e.g., CreateAccountComponent.
- Component selectors should use kebab-case, e.g., app-create-account.

Variable Declarations:

- Use const for variables that do not need reassignment.
- Use let for variables that need reassignment.
- Declare variables with meaningful and descriptive names.
- Avoid using single-letter variable names unless they are used in a limited scope (e.g., loop counters).

Constructor:

- Inject dependencies as parameters in the constructor.
- Declare class properties for injected dependencies.

Template and Styles:

- Use a separate template and styles file for each component.
- Name the template file with the component name, followed by .component.html.
- Name the styles file with the component name, followed by .component.scss.
- Use descriptive class names in styles.
- Avoid using inline styles unless necessary.

Methods:

- Use descriptive method names that reflect their purpose.
- Keep methods short and focused on a single task.
- Use comments to explain complex logic or provide additional context.

Error Handling:

- Use try-catch blocks to handle synchronous errors.
- Use async/await or toPromise() for handling asynchronous operations.
- Handle errors appropriately and provide meaningful error messages.
- Log errors to the console for debugging purposes.

Testing:

- Use a separate file for tests related to a component or service.
- Name test files with the component or service name, followed by .spec.ts.
- Write meaningful test descriptions that reflect the test's purpose.
- Organise tests using describe blocks.
- Use beforeEach and afterEach hooks to set up and clean up test data.
- Use descriptive variable and function names in tests.
- Test both positive and negative scenarios.

API Routes:

- Use descriptive names for API routes.
- Write tests for each API route to ensure proper functionality.
- Handle different response scenarios (success, errors) and assertions accordingly.

Authentication and Authorization:

- Define guidelines for implementing authentication and authorization mechanisms.
- Ensure sensitive user information (e.g., passwords, tokens) is handled securely.
- Follow security best practices for token-based authentication (e.g., JWT).

Version Control and Collaboration:

- Define guidelines for using version control systems (e.g., Git) and branching strategies.
- Establish code review practices to ensure code quality and share knowledge among team members.
- Encourage the use of meaningful commit messages and adhere to commit message conventions.

Miscellaneous:

- Use consistent indentation and follow the chosen coding style guide (e.g., 2 spaces, 4 spaces, or tabs).
- Write code with readability and maintainability in mind.
- Use meaningful variable and function names to enhance code understanding.
- Follow best practices and guidelines provided by the Angular framework.

Trace-ability Matrix

Requirements	Description	Acceptance Criteria
Log in	Users log in to the application with their relevant username and passwords	Validations will have to be done with the database to see if the relevant user is a personnel in the database. Checks for malicious or wrong inputs should be done. If it meets the criteria, redirects to their own relevant “HomePages” of their accounts.
Create Account	A new account can be made for a personnel specifying their roles and their teams.	Admin accounts and management accounts of the relevant projects will be able to make new accounts for new members. They will need to specify the new account’s role and team. They will also make a username and generate a password for this account for the new member to use. This will then be added to the database.
Edit Account Information	Management and Administrators can alter roles and teams for the relevant users. They will also be able to change the usernames and passwords if requested to.	Management and Administrators will be able to change a user's role and team. This in turn changes what the user can view and use. These changes will reflect on the database but the statistics will still remain. If requested to by the user themselves, the management and

		administrators can also change the username and password and these will be changed in the database as well.
Delete Account	Management and Administrators can delete an account for an user.	The deleted account will also have all their relevant information deleted from the database. Their statistics will also be deleted as well.
Manage Teams	Management and Administrators can create new “groups” to work on a ticket. They can also modify these “groups” and change the “groups” members or delete these “groups” entirely.	All teams and members information will be shown and can be searched or filtered to find a specific team or member. New “groups” can be made for collaboration between different members of different teams. These “groups” have access to this ticket and their own teams tickets only and will not have access to all the pre-existing tickets from each others’ teams. This is for confidentiality. This new “group” will then be added to the database. Management and Administrators can also change the “groups” team members and delete the “group” entirely. All relevant changes will also reflect on the database.
Get Analytics	Management and Administrators are able to get and view the statistics of the teams.	Statistics are taken from the database of all the relevant teams. These statistics can be filtered to different teams so as to make it more relevant to the viewer. This section is to observe performance, which will also be used by the AI for recommendation.
Create Ticket	Ticket can be created for the system for technical team member/s that is assigned to work on	Tickets can be created by any other roles besides, Clients and Technical. During the ticket creation, priority and deadline needs to be assigned with the

		relevant information on the ticket. Through the AI which observes the statistics, we will gather all the technical teams or members that have the best performance and which are free to work on the ticket so that the ticket creator can assign it to them accordingly. Once the ticket is created, the ticket time creation will also be displayed on the UI. These will all be recorded on the database.Tickets are in the general summarised form with all the relevant information of the ticket. The full form of ticket can be viewed by clicking into the ticket.
View Ticket	Tickets can be viewed by everyone in the same team, management and admin. Clients will not be able to view the ticket.	Tickets for a specific team can only be viewed by that team, even if they were not the team member that is assigned. Management and Admin can view all the tickets, even from the other teams. When an existing ticket is clicked on, it will expand and display all of the expanded form of the ticket information. This data is taken from the database.