BILKENT UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

CS 319 - Object Oriented Software Engineering

S2T9-undefined

Design Goals, High Level arch-D4

19/11/2023

Group T9

Yassin Younis - 22101310

Muhammed Shayan Usman - 22101343

Zahaab Khawaja - 22101038

Table of Contents

1. Design Goals

The design objectives for our application are outlined based on crucial non-functional requirements tailored to meet the expectations of our user base. These goals are identified through comprehensive background research and are strategically planned for seamless implementation.

1.1. Usability (UI/UX)

The UX should be straightforward, facilitating an intuitive understanding of how to navigate through the application. Key information should be accessible within one or two clicks from the main page, avoiding deep navigation trees that can lead to user frustration. While the interface should be comprehensive, it must also avoid cognitive overload; critical data should be displayed prominently, with secondary information accessible through expandable menus or tabs as needed. The aesthetic design should favour a clean, modern look with a neutral colour palette that is easy on the eyes, avoiding any elements that could distract or detract from the user experience, such as flashing graphics or overly vibrant colours.

Interactivity is a cornerstone of a good UX. Buttons and interactive elements should be of a size that is easy to interact with across devices, avoiding elements that are too small to be tapped accurately on touchscreens or too large as to appear unwieldy.

Moreover, the application should employ a consistent visual language throughout. Icons, buttons, and interactive elements should maintain a consistent design to prevent user confusion. Typography should be legible and accessible, with adequate contrast against the background. Text size and spacing should be optimised for readability across different devices.

Finally, the design should be future-proof, allowing for easy updates and scalability without major overhauls to the user interface, thus preserving the user's familiarity with the application over time.

1.2. Responsive UI

The UI's scalability across various devices is crucial for ensuring a seamless user experience. This multi-device compatibility can be achieved through responsive design principles that include fluid grid layouts, flexible images, and media queries that adjust to the user's behaviour and environment based on screen size, platform, and orientation.

For desktop views, the UI should leverage the wider screen to display more comprehensive views and possibly include additional navigational aids like side panels. This can provide users with quick access to various sections of the application without overwhelming them with information.

On mobile devices, where screen space is limited, the UI should prioritise content over navigation. Menus can be made collapsible into a hamburger icon or a similar pattern, which expands when tapped. This conserves space and reduces clutter, allowing the core content to remain front and centre. Interactive elements like buttons and links should be sized and spaced to accommodate touch interactions, with enough padding to prevent accidental taps.

Adopting a mobile-first approach can often streamline the development process, ensuring that the UI is fundamentally designed to handle the most restrictive constraints first, then enhanced for larger screens. This also aligns with the progressive enhancement strategy, where core functionality and content are accessible on the most basic devices, with additional features and styling layered on for more advanced browsers and larger displays.

1.3. Speed

The performance of the web app is paramount to user retention and satisfaction. To achieve optimal speed, the application should be designed to load the most critical content quickly. The app should also be tested across different devices and connection speeds to ensure performance consistency. The goal is to craft a fluid experience where the performance is so seamless that the user remains focused on their tasks, not on waiting for pages to load.

1.4. Security

For the security aspect, restricting access to Bilkent University members by utilising Bilkent email addresses for registration and authentication can establish a trustable user base and significantly reduce the risk of fraudulent activities. Implementing this measure requires a robust verification process during account creation, ensuring that only email addresses with the Bilkent domain can be used. Additionally, the platform should enforce strong password policies to further secure user accounts. Regular security audits and adherence to best practices in data protection, such as using HTTPS for data transmission and secure handling of user information, will be essential. This security measure not only protects users but also strengthens the platform's credibility and distinguishes it from competitors by creating a safer transaction environment.

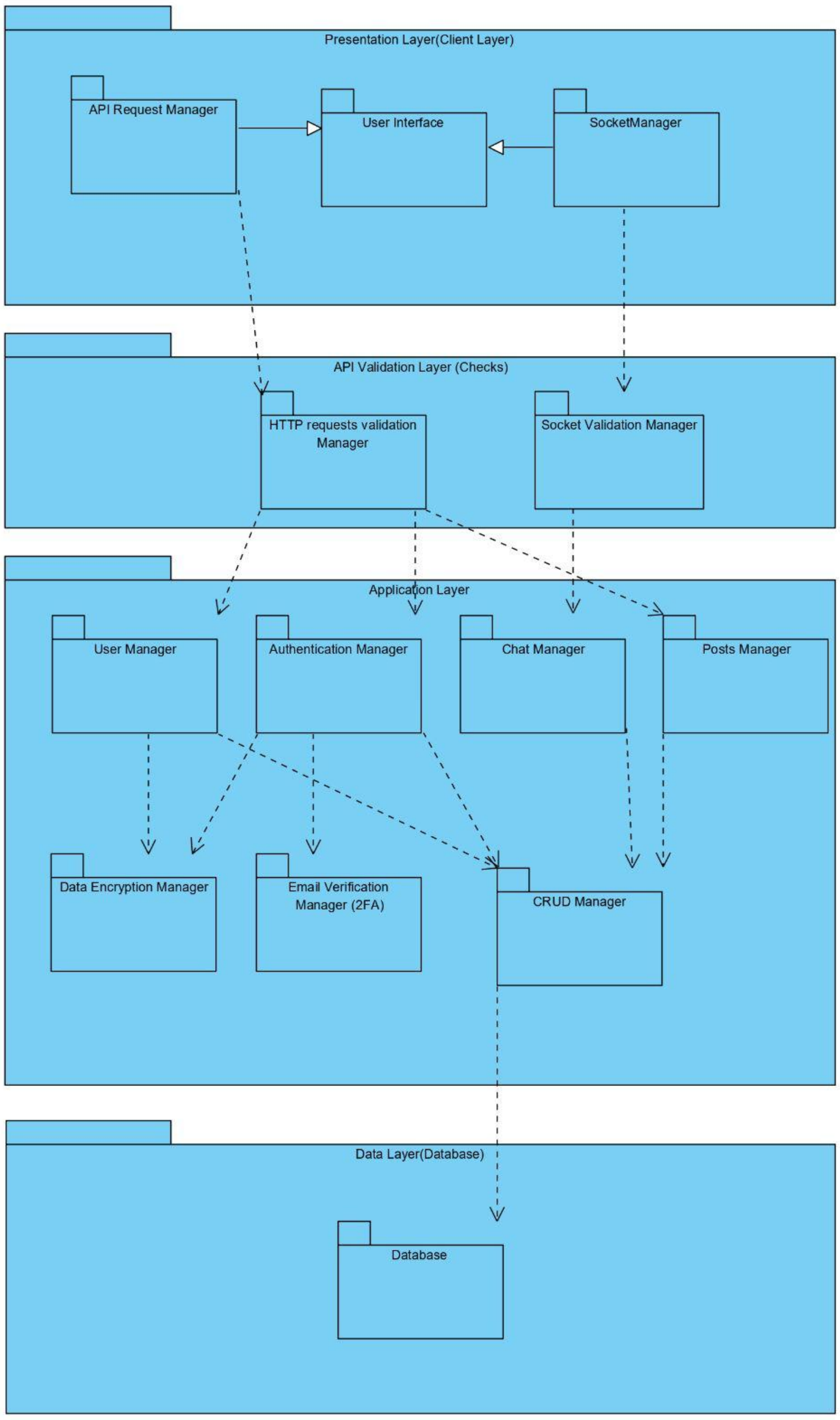
2. High-Level Software Architecture

2.1. Overview

Our campus connect system operates on a client-server architecture, where components are organised around the interaction between clients and servers. The subsystem decomposition section details the specific components within the client and server domains, highlighting their roles in the overall architecture. The deployment diagram visually represents the distribution of these components across hardware and software environments. Hardware/software mapping illustrates the correlation between software elements and the supporting hardware infrastructure. Persistent data management focuses on how data is stored and accessed over time in the client-server context. Access control and security measures are discussed to ensure the integrity of interactions between clients and servers. Boundary conditions, covering initialization, termination, and failure scenarios, outline critical behaviours and responses within the client-server framework.

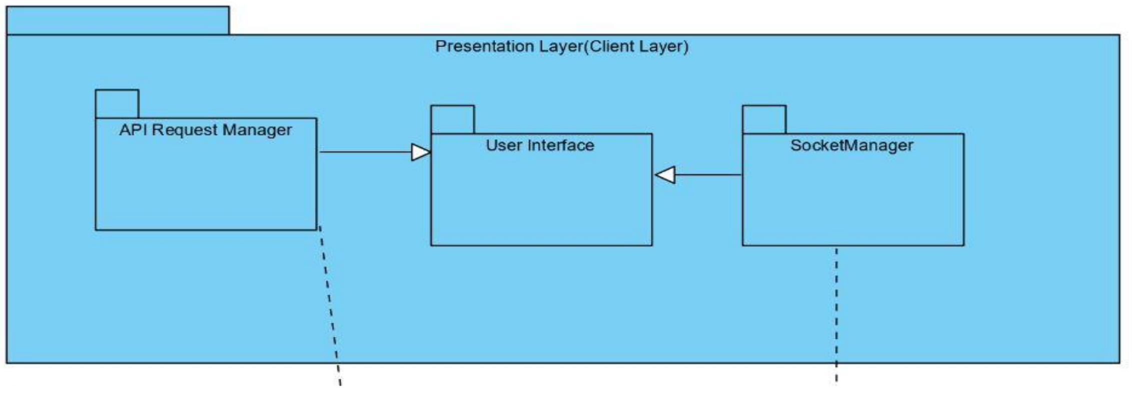
2.2. Subsystem Decomposition

In our system, we've implemented a 4-layered architecture—Presentation Layer, Validation Layer, Application Layer, and Data Layer. This structured approach enhances security, functionality, maintainability, and usability. The Presentation Layer manages client interfaces, the Validation Layer ensures the integrity of HTTP requests, the Application Layer handles core logic, and the Data Layer focuses on efficient database storage and retrieval. This layered model streamlines our system for easy maintenance and scalability, aligning with our goal of creating a secure, functional, and user-friendly environment.


Presentation Layer(Client Layer)
API Request Manager
User Interface
SocketManager
API Validation Layer (Checks)
HTTP requests validation Manager
Socket Validation Manager
Application Layer
User Manager
Authentication Manager
Chat Manager
Posts Manager
Data Encryption Manager
Email Verification Manager (2FA)
CRUD Manager
Data Layer(Database)
Database

2.2.1 Presentation Layer

In our system, the Presentation Layer serves as the interface through which users interact with the program. Users can input information through prompted interfaces, and this data is subsequently transmitted to the Validation Layer.



Socket Manager:

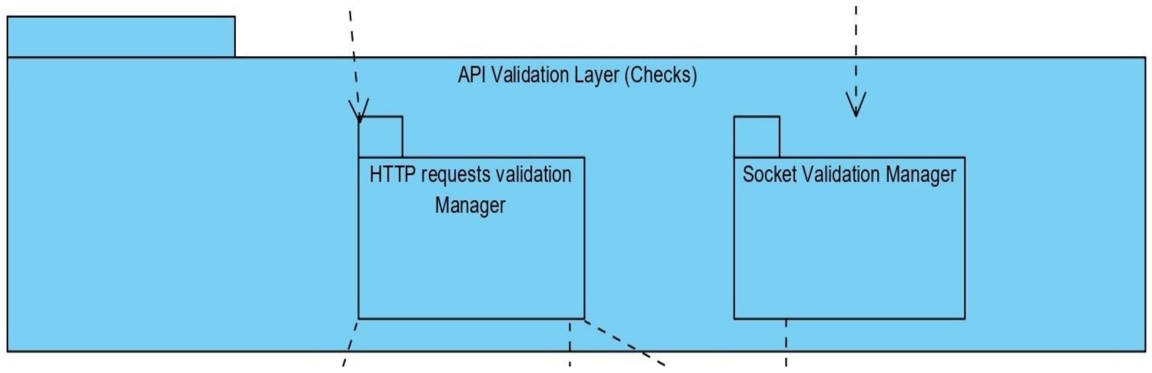
- Manages socket events by receiving events from the application layer and transmitting events to the Validation Layer.

API Request Manager:

- Handles API requests by sending them to the Validation Layer for processing.

2.2.2 Validation Layer

In the Validation Layer, the focus is on validating both HTTP requests and socket events to ensure data integrity and security. Two key components, the HTTP Request Validation Manager and the Socket Validation Manager, play crucial roles in managing and validating these interactions and sending the validated requests to the Application Layer.


API Validation Layer (Checks)
HTTP requests validation Manager
Socket Validation Manager

HTTPRequestValidationManager

- Responsible for validating incoming HTTP requests, ensuring that the data adheres to predefined DTO objects.

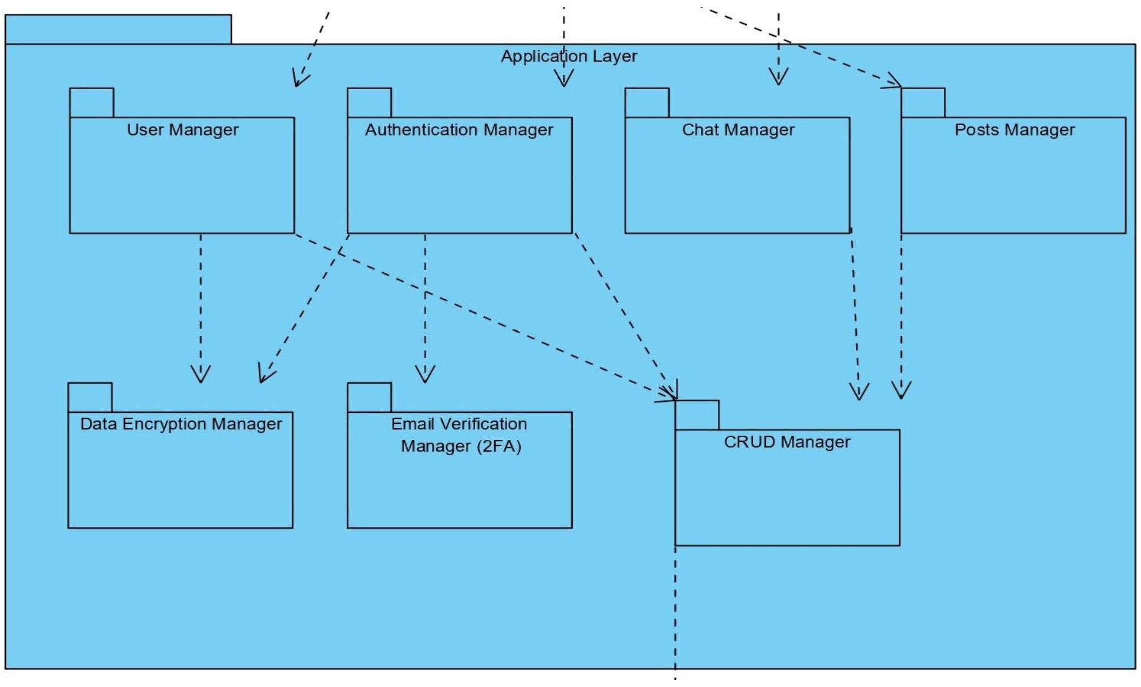
SocketValidationManager

- Manages the validation of socket events, ensuring they adhere to the predefined DTO objects before processing them within the system.

This Validation Layer is integral to maintaining the reliability and security of the data flow within the system. The HTTP Request Validation Manager and Socket Validation Manager work collaboratively to validate incoming requests and events, contributing to the overall robustness of the system.

2.2.3 Application Layer

In the Application Layer, various managers handle distinct functionalities, contributing to the overall operation and security of the system.



UserManager:

- Manages user-related functionalities such as registration, profile editing, and retrieval/deletion of user data.

AuthenticationManager:

- Handles the security aspects of user login, including password verification and session management.

ChatManager:

- Oversees the messaging or chat features of the application, managing conversations and message history.

PostsManager:

- Manages the creation, retrieval, updating, and deletion of posts within the application.

DataEncryptionManager:

- Ensures that sensitive data is encrypted before storage or transmission, adding a crucial layer of security. This layer also generates authentication tokens for the client to store in their local storage.

EmailVerificationManager (2FA):

- Manages the process of verifying user emails and implements Two-Factor Authentication (2FA) for enhanced security. This layer emails the user a link to verify their email address to enable them to login to their account.

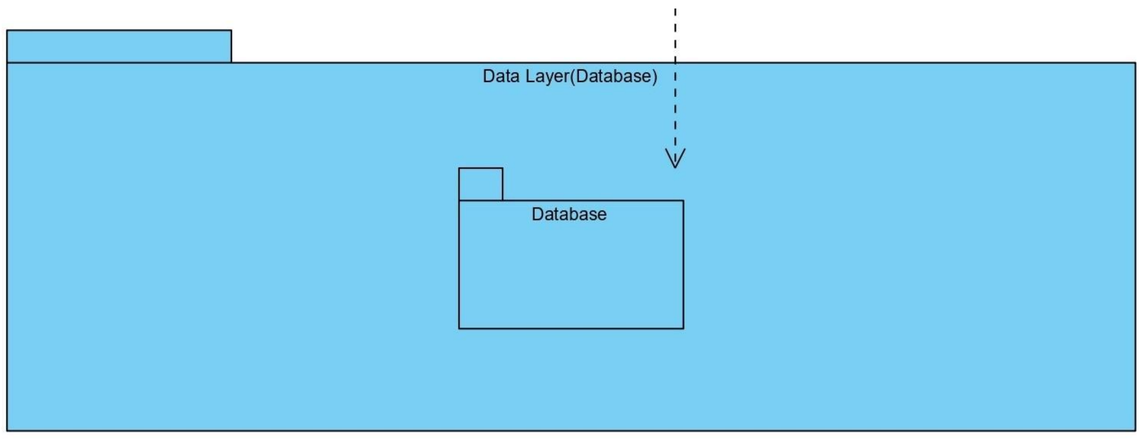
CRUDManager:

- Manages fundamental data operations (Create, Read, Update, Delete) linking the Application Layer with the Data Layer.

This Application Layer encapsulates essential functionalities, from user management to data encryption, ensuring the system's security, functionality, and user experience. The different managers collaborate to provide a comprehensive and efficient application framework.

2.2.4 Data Layer

In the Data Layer, the primary focus is on managing and interacting with the database to ensure efficient storage and retrieval of information.



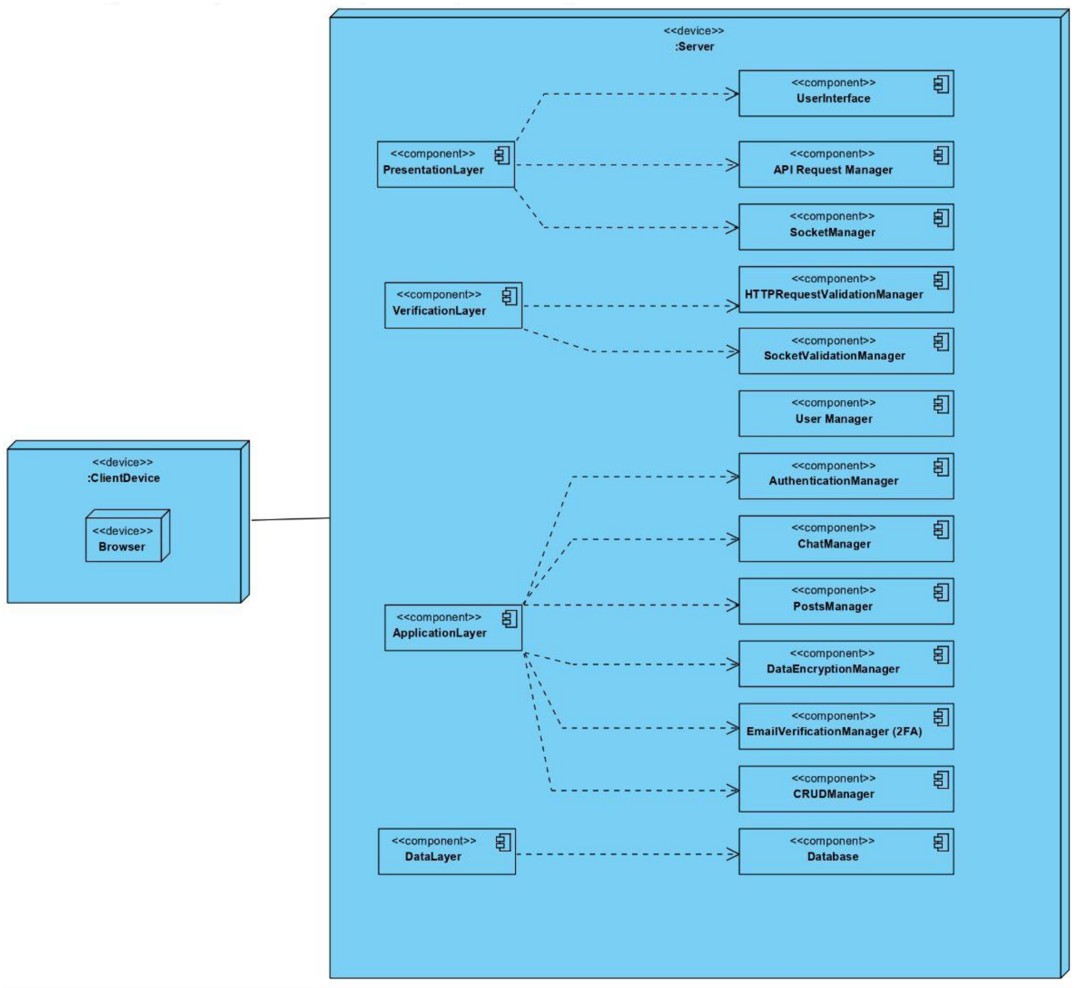
Database:

- Oversees operations such as data insertion, retrieval, updating, and deletion.

The Data Layer serves as the backbone for data-related operations, ensuring seamless and secure interactions with the database to support the functionalities of the Application Layer and the overall system.

2.3. Deployment Diagram

This diagram illustrates the dynamic interaction between the client and server. It organises classes into component groups. Artefacts provide a visual representation of the relationships and dependencies between the various components in the client-server architecture.



2.4. Hardware/Software Mapping

The application employs a technology stack that balances efficiency across the frontend, backend, and database. Node.js and Express power the backend, offering ease of use and streamlined deployment. React, known for its robust component system, drives the frontend, providing a dynamic user interface. MongoDB serves as our database, offering versatility in data management.

The deployment process benefits from Node.js's lightweight and event-driven architecture, facilitating quick updates and modifications with minimal downtime. Additionally, the application's resource efficiency is maintained through Node.js's lower hardware requirements, ensuring smooth scalability without imposing heavy demands on hardware.

Designed to run on any modern web browser, the application ensures compatibility with various devices, including computers and mobile phones. For computer access, standard peripherals are needed, while mobile devices with touchscreens require no additional accessories. The supported browsers encompass Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, and Opera, ensuring broad user reach and compatibility with modern web technologies.

This balanced technology stack and deployment approach contribute to a user-friendly, scalable, and efficiently deployable application.

2.5. Persistent Data Management

We opted for MongoDB as our database solution due to its inherent strengths. MongoDB's flexibility is a key factor, utilising documents that can accommodate sub-documents in intricate hierarchies, providing an adaptable structure. Its flexibility includes a native aggregation feature that allows for the direct extraction and transformation of data from the database, catering to a wide range of use cases, which may be beneficial in the future. The schema-less model of MongoDB empowers applications to interpret different properties within a collection's documents, allowing for dynamic adaptation to evolving data requirements. Additionally, MongoDB offers a flexible query model, allowing selective indexing and querying based on regular expressions, ranges, or attribute values.

These features collectively ensure ease of implementation, maintenance, and scalability, aligning well with our project's dynamic data needs.

To enhance user experience, we employ a strategy where user authentication tokens are stored in the browser's local storage. This approach ensures that users don't need to log in repeatedly, providing a seamless and convenient interaction with the application.

2.6. Access Control and Security

For Campus Connect, our web-based app, safeguarding user data is a top priority. We've taken a hands-on approach to security, developing our authentication system for the backend. Using Node.js and Express, we've created a robust authentication mechanism tailored to our specific needs, ensuring a secure environment for our users.

Our cloud infrastructure is powered by MongoDB Atlas, which has encryption for data both at rest and in transit, adding an extra layer of protection. Our authentication system and security practices remain key pillars in our overall approach allowing us to store information securely in our database.

The access matrix below highlights how actors can directly engage with control class methods.

	User	Database

APIRequestManager	get()
	post()
	patch()
	delete()	

SocketRequestManager	connect()
	disconnect()
	emit()	

UserManager	createUser()
	getUser()
		storeSpecifiedUser()
		retrieveSpecifiedUser()

	updateUser()
	deleteUser()
		deleteSpecifiedUser()
		updateSpecifiedUser()

AuthenticationManager	login()
	validateAuthToken()
	sendForgotPasswordEmail()
		checkAuthTokenForUser()

ChatManager	sendMessage()
	receiveMessage()
	getPreviousMessages()
		storeMessage()
		getPreviousMessages()

PostsManager	createPost()
	getPost()
	getPosts
	updatePost()
	deletePost()
		storePost()
		retrievePost()
		retrievePosts()
		updatePost()

DataEncryptionManager	encryptPassword()
	decryptPassword()	

EmailVerificationManage r	verifyEmail()	updateUserVerificationStat us()

2.7. Boundary Conditions

2.7.1 Initialization

Both the frontend and backend can be hosted on a machine using the predefined starting scripts. This will build and deploy the entire application. The application will then be hosted on the desired domain. No installation process will be required for users – the app operates on a remote server, accessible from any modern browser with an internet connection. Users can then login and use the application as desired.

2.7.2 Termination

In order to terminate the server, the admin will have to manually shut down the application. The use of Node.js and React allow for a simple and swift termination process. The user will have no control over this part of the application. When the user is done using Campus Connect, they can simply close their web browsers.

2.7.3 Failure

We plan to use a provider for server hosting which automatically restarts the backend in case of any failures. Since data is stored remotely, server failures won't compromise stored information, except for actively processed data. Users promptly receive notifications about errors and are encouraged to retry actions seamlessly. At every point in our frontend, the user is notified when their API request fails, letting them know they should retry their action. The inclusion of a validation layer in our application significantly reduces the possibility of any failures.