A logo for a university

Description automatically generated

(Chat Verse Web application)

Synopsis Submitted to

D Y Patil International University, Akurdi, Pune

in partial fulfillment of the degree of BCA.

BCA (Bachelor of Computer Applications)

(Cloud and Information Security (CIS))

Submitted By:

1) Anurag Mahada (20210801070)

2) Umesh Patil (20210801109)

3) Vikram Jadhav (20210801133)

Under the Guidance of

MS. Shobhana Patil Signature:

School of Computer Science Engineering & Applications

D Y Patil International University, Akurdi, Pune, INDIA, 411044

[Batch 2021]

**CONTENTS**

**CHAPTER PAGE NO.**

1. INTRODUCTION
2. REVIEW OF LITERATURE
3. PROPOSED METHODOLOGY AND FRAMEWORK DESIGN
4. PREPARING PROJECT PLAN
   * PERT/GANTT Chart
5. REFERENCES

**Chapter 1:INTRODUCTION**

Chatting is a method of using technology to bring people and ideas together despite of the geographical barriers. The technology has been available for years but the acceptance was quite recent. Our project is an example of a chat server. It is made up of two applications - the client application, which runs on the user’s web browser and server application, runs on any hosting servers on the network. To start chatting client should get connected to server where they can do private and group chat. Security measures were taken during the last one.

**BACKGROUND**

A chat application is a software program or platform that enables users to communicate with each other in real-time over the internet. It will work on web applications. Chat applications facilitate text-based conversations, but many also support additional features such as voice and video calls, file sharing, and group chats.

The old manual system was suffering from a series of drawbacks. Since whole of the system was to be maintained with hands the process of keeping, maintaining and retrieving the information was very tedious and lengthy. The records were never used to be in a systematic order. there used to be lots of difficulties in associating any particular transaction with a particular context. If any information was to be found it was required to go through the different registers, documents there would never exist anything like report generation. There would always be unnecessary consumption of time while entering records and retrieving records. One more problem was that it was very difficult to find errors while entering the records. Once the records were entered it was very difficult to update these records.

Here are some key components and features commonly found in chat applications:

* Authentication: user authentication is implemented for secure access.
* Real-time chat: Chat functionality is implemented using Socket.io for real-time messaging.
* Real-time notifications: Notifications are sent in real-time to users.
* Typing indicators: Indicates when a user is typing message.
* Guest login: Allow users to log in as quests without creating an account.
* Group chat: Supports group chat functionality.
* One-on-one chat: Supports one-on-one private messaging.
* Responsive: The application is designed to be responsive and work well on different devices.

**PROBLEMS & GAP IDENTIFICATION**

Problem Identification:

1. Lack of Features:
   * Many existing chat applications lack essential features such as file sharing, voice/video calls, and group chat management.
   * Users often require a comprehensive set of features to meet their diverse communication needs.
2. Complexity:
   * Some chat applications are overly complex, with cluttered interfaces and confusing navigation.
   * Users may find it challenging to navigate through features or understand how to use advanced functionalities.
3. Security Concerns:
   * Security breaches and privacy concerns are prevalent in chat applications, especially those with weak encryption protocols or inadequate data protection measures.
   * Users are increasingly concerned about the safety of their personal information and conversations.
4. Platform Dependence:
   * Many chat applications are platform-dependent, meaning they may not work seamlessly across different operating systems or devices.
   * Users may encounter compatibility issues when trying to communicate with contacts using different platforms.
5. Performance Issues:
   * Some chat applications suffer from performance issues, such as slow loading times, frequent crashes, or message delivery delays.
   * These performance issues can disrupt the user experience and hinder effective communication.

Gap Identification:

1. User-Friendly Interface:
   * There is a gap in the market for a chat application with a clean and intuitive user interface that prioritizes ease of use and simplicity.
2. Comprehensive Feature Set:
   * There is a need for a chat application that offers a wide range of features, including messaging, file sharing, voice/video calls, group chat management, and robust security features.
3. Enhanced Security Measures:
   * There is a gap in the market for a chat application that prioritizes user privacy and data security by implementing strong encryption protocols, secure authentication mechanisms, and robust data protection measures.
4. Cross-Platform Compatibility:
   * There is a demand for a chat application that works seamlessly across different platforms and devices, allowing users to communicate with anyone regardless of their operating system or device.
5. Reliable Performance:
   * There is a gap in the market for a chat application that delivers reliable performance, with fast loading times, stable connectivity, and efficient message delivery.

**CHAPTER 2: REVIEW OF LITERATURE**

A review of literature on chat applications provides insight into the evolution, functionalities, usage patterns, and impacts of various chat platforms. Here's a structured overview:

**History Evolution:**

* Early chat applications like IRC (Internet Relay Chat) and ICQ paved the way for modern chat platforms.
* The emergence of instant messaging (IM) services such as AOL Instant Messenger (AIM) and MSN Messenger revolutionized online communication in the late 1990s and early 2000s.
* The shift towards mobile messaging apps like WhatsApp, WeChat, and Facebook Messenger in the smartphone era reshaped how people communicate globally.

**Functionalities:**

* Core features include text-based messaging, multimedia sharing (photos, videos, documents), voice and video calling, group chats, and presence indicators.
* Advanced functionalities such as end-to-end encryption (e.g., Signal), self-destructing messages (e.g., Snapchat), and chatbots (e.g., Slack) have become increasingly common.

**User behavior and social impact:**

* Chat applications have transformed social interactions, enabling real-time communication across geographical boundaries.
* They facilitate both personal communication among friends and family and professional collaboration in work environments.
* Chat platforms have also influenced consumer behaviour, with businesses leveraging messaging apps for customer support and e-commerce transactions.

**Privacy and security concerns:**

* Privacy issues, data breaches, and concerns over data ownership and surveillance have plagued some chat applications.
* End-to-end encryption has become a key feature in response to growing privacy concerns, ensuring that only the sender and intended recipient can access message content.

**Cultural and Global Perspectives:**

* Chat applications have different levels of adoption and popularity across regions, influenced by cultural preferences, technological infrastructure, and regulatory environments.
* WeChat dominates the Chinese market, while Line is popular in Japan and KakaoTalk in South Korea.

**Future trends and challenges:**

* The convergence of chat applications with artificial intelligence (AI) and machine learning is expected to enhance user experiences through features like smart replies and predictive typing.
* Challenges include addressing misinformation and fake news dissemination, ensuring inclusivity and accessibility, and navigating regulatory complexities, particularly regarding data privacy and security.

**C****HAPTER 3: PROPOSED METHODOLOGY AND FRAMEWORK DESIGN**

• Easy to use GUI (Graphical User Interface), hence any user with minimal knowledge of operating a system can use the software.

• Platform independence: The messenger operates on any system irrelevant of the underlying operating system.

• Unlimited clients: “N” number of users can be connected without any performance degradation of the server.

There is always a room for improvements in any apps. Right now, we are just dealing with text communication.

• File Transfer

• One-on-One chat

• Group chat

Designing the framework for a chat application involves structuring the software components and defining how they interact with each other. Here's a high-level framework design for a chat application:

* **Client-side Framework:**

**User Interface (UI) Layer:**

* **Chat Interface Components:**
  + Messaging area
  + Typing indicator
  + Input field
  + Contact or conversation list
  + User profile pictures
  + Timestamps
  + Message Status Indicator
  + Notification
  + Search Bar
  + Setting and Option Menu

**Communication Layer:**

* **Socket.io Client:**
  + Enables real-time bidirectional communication with the server.
  + Handles message sending and receiving in real-time.

**Data Management Layer:**

* **State Management:**
  + Manages application state, including user authentication, chat messages, and user presence.
  + Facilitates data flow between UI components and the communication layer.
* **Server-side Framework:**

**Application Layer:**

* **Socket.io Server:**
  + Listens for incoming connections from clients.
  + Handles WebSocket handshake and manages WebSocket connections.
  + Routes incoming messages to appropriate handlers.

**Business Logic:**

* **User Authentication:**
  + Handles user registration, login, and logout processes.
  + Manages user sessions and access control.
* **Message Handling:**
  + Processes incoming messages, including text, images, and other media.
  + Validates message content and sender permissions.

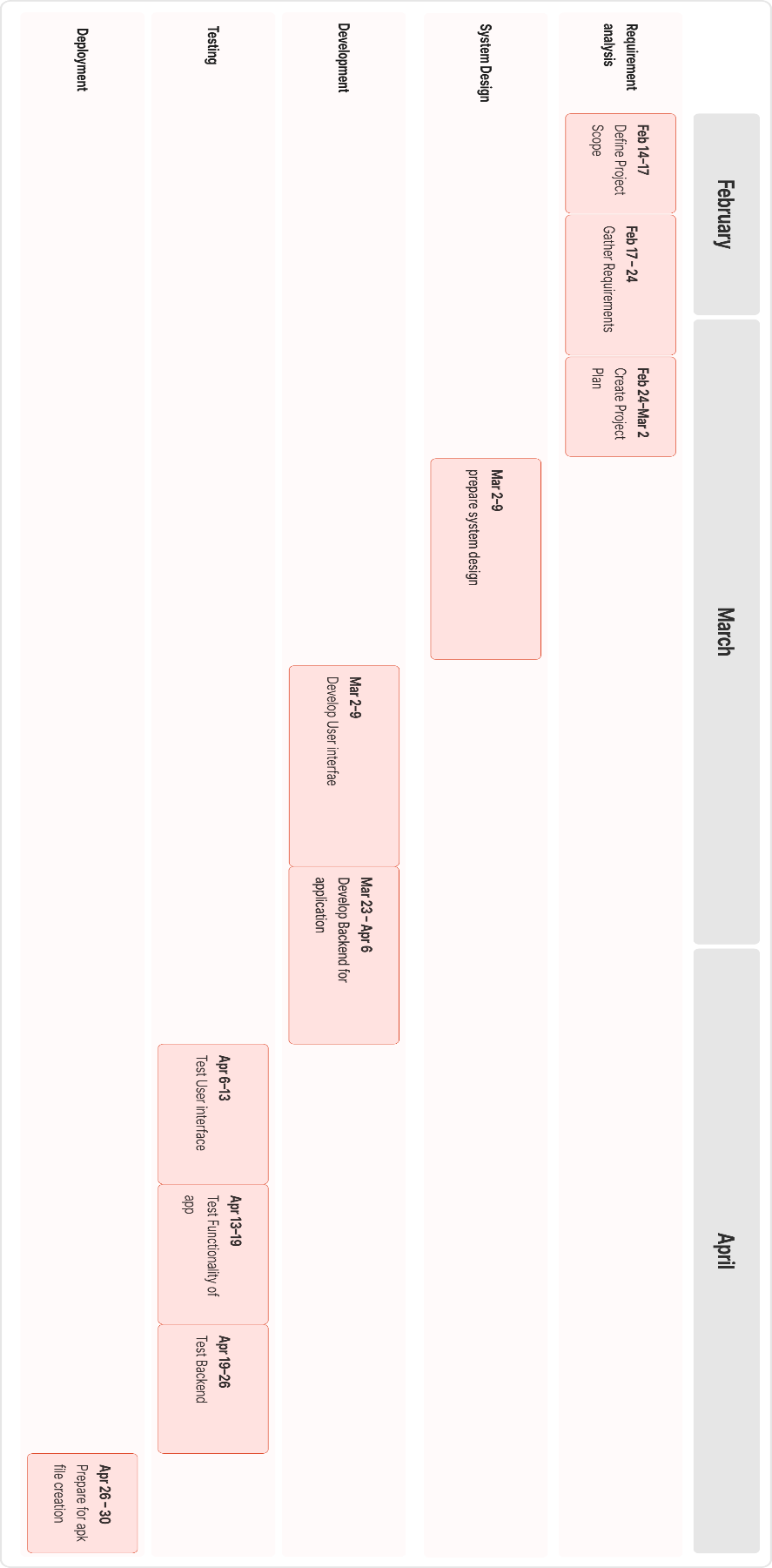
**Data Access Layer:**

* **Database (e.g., MongoDB,ExpressJS,NodeJS,React):**
  + Stores user profiles, chat messages, user relationships, etc.
  + Supports efficient querying and indexing for fast retrieval of data.
* **Application Components:**
* Authentication in a web chat application typically involves verifying the identity of users before allowing them access to chat features or other restricted functionality. Here's a basic overview of how authentication might work in a web chat application:
* User Registration: Users need to create accounts by providing necessary information such as username, email address, and password. Optionally, you might include additional fields or use alternative authentication methods like social media sign-in.
* Password Hashing: When users register or change their passwords, the passwords should be securely hashed and stored in the database. Hashing ensures that even if the database is compromised, passwords are not easily retrievable.
* Login: Users provide their credentials (usually username/email and password) to authenticate themselves. The server verifies the credentials against the stored information in the database.
* Session Management: Upon successful authentication, the server creates a session for the user. This session is often represented by a unique session ID stored as a cookie or in local storage on the user's device.
* Authorization: After authentication, the server determines what actions the user is allowed to perform. For example, in a chat application, users may be authorized to send messages, join specific chat rooms, or perform administrative tasks.
* Token-based Authentication (Optional): Instead of using sessions, you might implement token-based authentication using JSON Web Tokens (JWT). With JWT, a token containing user information and a signature is issued upon successful login. This token is then included in subsequent requests to authenticate the user.
* HTTPS: Ensure that all communication between the client and server is encrypted using HTTPS to prevent eavesdropping and man-in-the-middle attacks.
* Security Measures: Implement additional security measures such as rate limiting, CAPTCHA, and account lockout to prevent brute force attacks and other malicious activities.
* Logout: Provide users with the ability to log out, which invalidates their session or token and prevents unauthorized access to their account.
* Remember Me Functionality (Optional): Offer users the option to stay logged in across sessions using long-lived tokens or cookies. This feature should be implemented securely to prevent unauthorized access.
* Password Reset: Allow users to reset their passwords in case they forget them. This typically involves sending a password reset link to the user's email address or using other verification methods.

**CHAPTER 4: PROJECT PLAN**

* Planning Phase:
  + Conduct market research analyse, existing chat applications, identified their strengths and weaknesses, and gather user feedback to understand user preferences and pain points.
  + Establish project timeline and milestones, break down the development process into manageable phases and set deadlines for each milestone.
* Requirements Gathering:
  + Identify key features and define the essential features of the chat application, including messaging, file sharing, group chat, user authentication, and security measures.
* Design Phase:
* Define the visual elements, color scheme, typography and branding elements to create a cohesive and visually appealing UI design.

* Development Phase:
  + Configure development tools, version control systems, and project management platforms for efficient collaboration among team members.
* Testing Phase:
  + Test the chat application's features and functionalities to ensure they meet the specified requirements and work as expected.
* Deployment Phase:
  + Deploy the chat application to the chosen hosting environment, ensuring proper configuration, scalability, and monitoring.
* Maintenance Phase:
  + Monitor server logs, error reports, and user feedback to identify and address performance issues, downtime, and scalability concerns.

**Gantt Chart**

**CHAPTER 5: REFERENCES**

1. Aaron, P., (2020) “Webrtc vs websockets.” http://stackoverflow.com/a/18825175, sep 2013. Accessed: 2017-04-19.

2. Ackley, B., (2020) “Webrtc samples.” https://webrtc.github.io/samples/, sep 2013. Accessed: 2017-04-19.

3. Abby, T., (2020) “User stories: An agile introduction.” http://www.agilemodeling.com/ artifacts/userStory.htm. Accessed: 2016-10-20.

4. Bairam, W., (2020) “Making the switch from making the switch from node.js to golang.” http://blog.digg.com/post/141552444676/ making-the-switch-from-nodejsto-golang, mar 2016. Accessed: 2016-10- 19.

5. Callum, S., (2020) “Sinon - best practices for spies, stubs and mocks.” https://semaphoreci.com/community/tutorials/best-practices-for-spies-stubsandmocks-in-sinon-js, 2016. Accessed: 2016-10-03.