**A PROJECT REPORT**

**ON**

**“RealTime Chatting Application”**

**(A WEB APPLICATION)**

**SUBMITTED**

**TO**

****

**GURU NANAK DEV UNIVERSITY, AMRITSAR**

**IN THE PARTIAL FULFILLMENT OF**

**BACHELOR OF COMPUTER APPLICATIONS**

**SEM- VI**

**ACADEMIC SESSION 2021-2024**

**SUBMITTED BY: PROJECT GUIDE:**

**ARUN BHANDARI PROF. ANSHUMAN SHARMA**

**CERTIFICATE**

**This is to certify that Arun Bhandari having roll no:1072222504 are student of Hindu College studying in BCA Sem- VI have completed the Project report entitled “Rental Service” for the partial fulfillment of requirement of degree of Bachelor of Computer Applications submitted to Hindu College, Amritsar.**

**They have carried out the work satisfactory.**

**Place: Amritsar**

**Date:**

**Prof. Anshuman Sharma**

**(Project Guide) External Examiner**

**DECLARATION**

**To**

**The Project Incharge**

**Hindu College**

**Respected Sir**

**We undersigned hereby declare that the project report entitled “Rental Service” written and submitted is our original work. The design documentation, testing documentation, coding and other findings in this project are based on the data collected by ourselves while preparing this project. We have not copied from any other project report.**

**We understand that, any such copying is liable to be punished in a way the University authorities may deem fit.**

**Place: Amritsar**

**Date:**

**Arun Bhandari (10722122504)**

**LETTER PHOTOSTATE COPY**

**ACKNOWLEDGEMENT**

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. We would like to share our sincere gratitude to all those who helped us in completion of this project and in final compilation of our idea to a shaped sculpture.

We would like to thanks Dr. Rama Sharma, Head of Department Computer Science for her guidance and support. We would also like to share our sincere gratitude Mr. Anshuman Sharma sir for his supervision and monitoring during the project work.

We would like to acknowledge the co-operation of our parents and friends for encouragement which help us in completion of this project.

Our thanks and appreciations also go to our college in developing the project and people who have willingly helped us out with their abilities.

**Thank you All.**

**Arun Bhandari**

**ABSTRACT**

Our project is related to a new way of chatting with people. Chatting and communicating with people through internet is becoming common to people and is connecting people all over the world. Mainly, chatting apps in today’s world mainly focus on connecting people, providing users with more features like GIFs, stickers etc. But this app, is different from them. This chatting application includes chatting through internet using IP address. It mainly focuses on chatting and connects people all around the world. Mostly, chatting applications like WhatsApp requires mobile no. of the person and then we can chat and connect with the person. But here, the person only has to login with the system, and then he can connect with the people which he wants with.

The Discuss Chat app is an open - source chatting app. It means people all over the world can join the chat between people easily. We can check and see the people joining and leaving the chat group. For using the app, firstly we have to register our name in the application. After registration, the person will be given a particular IP address, which is only used by that person, so that people will same name can be differentiated easily. The IP address can only be seen by the person which is registered under that name. Once, the registration of the person is done, he can join the chat room.

The chatting between 2 people can be easily converted into group, as the people chatting easily know if there is another person, who wants to join the chat between them. If we have to chat with a specific person, then we just have to know the name of the person and its IP address. It’s different from the present chatting applications, as it includes the personal information of the person, which gets accessed by the person which is following him or is friends with the person. This can save the person from sharing his personal stuff to strangers, without his consent. One of the features of this application is that, if a new person connects with them while chatting in person or in group, then the app shows the people the person is connecting or joining them while being in the group. Not only group chatting, but personal chatting between people also takes place.

**TABLE OF CONTENTS**

**OBJECTIVES**

♦ **Easy login** – in this app, we only need to add our name and then the system will provide us an IP address, which will be used only by the user under whose name it is registered.

♦ **Open source** – people which are interested to talk about a topic which they like and people which are interested in it, can join and communicate with others easily, as the chat is open sourced; which means people can join and easily leave the chat room.

♦ **Connects people globally** – as the application runs through help of internet, it helps in connecting people easily. Not only from a particular country, but it can connect people all over the world.

♦ **Different from normal chatting apps** – Discuss chat app being bit different from normal chatting applications, can attract people easily. Not only because it takes less logging time, but also because it has some features which are not available in current chatting applications.

♦**Support Multiple Platforms** – Chat applications often support multiple platforms, including web browsers, desktop computers, and mobile devices (iOS, Android). This ensures that users can access the app from their preferred device and operating system**.**

♦**Enable Real-Time Updates:** Chat applications aim to provide real-time updates and notifications to keep users informed of new messages, mentions, or activities within their conversations.

♦**Support Multimedia Communication:** In addition to text messages, chat applications often support multimedia communication, including image sharing, file attachments, voice messages, and video calls.

**Project Category** **:-**

In this project, software which have been used for frontend and backend are as follows: Used (Front End): Visual Studio Code [Index.html] (Version: 1.57.1 (user setup):

- Visual Studio Code is a source-code editor made by Microsoft for Windows, Linux and macOS.

- Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality.

-Microsoft has released most of Visual Studio Code's source code on the Microsoft/VScode repository of GitHub using the "Code – OSS" name, under the permissive MIT License, while the releases by Microsoft are proprietary freeware.

Visual Studio Code was ranked the most popular developer environment tool, with 50.7% of 87,317 respondents reporting that they use it.

Visual Studio Code was first announced on April 29, 2015, by Microsoft at the 2015 Build conference. A Preview build was released shortly thereafter.

On November 18, 2015, Visual Studio Code was released under the MIT License, having its source code available on GitHub. Extension support was also announced

On April 14, 2016, Visual Studio Code graduated from the public preview stage and was released to the Web.

In this project, “iDiscuss Chat App” uses Visual Studio as front end and as back end.

Xampp:- is used for database with MYSQL.

**Software Development Lifecycle:-**

The software development lifecycle also referred to as the SDLC, is a process that companies follow in order to design, implement, and manage software systems. When it comes to producing and maintaining software, developers have a guide in the **Software Development Life Cycle (SDLC)** to follow. It is possible for businesses to guarantee the success of their software development projects by adhering to the software development life cycle (SDLC). The software development life cycle is comprised of a number of different phases, each of which serves its own unique purpose.

**The following is a list of the phases that comprise the SDLC:**

1. Research ,Preparation and analysis of options
2. The Analyze Step
3. The Design Step
4. The Implementation Step
5. The Testing Step
6. The Deployment Step
7. Maintenance

The planning and feasibility stage of the software development life cycle (SDLC) comes first. At this point, companies examine the viability of a software development project and decide whether or not it is worthwhile to move forward with it.



**Phase 1: Research**

**Preparation and analysis of options:-** Prior to getting started with the application development, we did some research on the current messaging platforms out there. We were looking forward to building a unique experience, rather than an exact clone of an existing chat platform. We already knew of the existence of several messaging applications, and a few chat applications that suited developers. However, never before had we done an indepth analysis of their tools to find out whether they were good enough for developers. Soon, we realized that none of the sites were heading in our direction. Some of them were missing features which we considered crucial and others had opportunities for further enhancements. Contrary to what many people think, having a few platforms around is not a necessarily a bad thing. We were able to get ideas of what to build and how and determine which technologies and strategies to use based on their experience. Often, this was as simple as checking their blogs. Companies like Slack regularly post development updates (such as performance reviews, technology comparisons, and scalability posts). Other times, we had to dig into the web to find out the different options we had and pick out the one which we considered to be the most appropriate.

During the competitive analysis, we investigated the following platforms:

• Flowdock - <https://www.flowdock.com>

• Gitter - <https://gitter.im>

• Hangouts - <https://hangouts.google.com>

• Matrix - http://matrix.org

• Messenger - <https://messenger.com>

• Rocket.chat - <https://rocket.chat>

• Skype - <https://web.skype.com>

• Slack - <https://slack.com>

• Telegram - <https://web.telegram.org>

• What App - <https://web.whatsapp.com>

**Communication Protocols :-** For most web applications, communication protocols are not a subject of discussion. AJAX through HTTP is the way to go since it is reliable and widely supported.

• However, that is not our case. We need, albeit not in every single situation, an extremely fast communication method to send/receive messages in real time.

• For messaging:- there are a few communication protocols available for the web. The most popular ones are AJAX, WebSockets, and WebRTC .

• AJAX :-is a slow approach. Not only because of the headers that have to be sent in every request, but also, and more important, because there is no way to get notified of new messages in a chat room. By using AJAX, we would have to request/pull new messages from the server every few seconds, which would result in new messages to take up to a few seconds to appear on the screen, not to say the numerous redundant requests that this would generate.

• WebSockets :- are a better approach. WebSockets connections can take up to few seconds to establish, but thanks to the full-duplex communication channel, messages can be exchanged swiftly (averaging few milliseconds delay per message). Also, both client and server can get notified of new requests through the same communication channel, which means that unlike AJAX, the client does not have to send the server a petition to retrieve new messages but rather wait for the server to send them.

• WebRTC :- is the new communication protocol available for the most modern browsers (Chrome, Firefox, and Opera). It is designed for high-performance, high-quality 2.3. COMMUNICATION PROTOCOLS 11 communication of video, audio, and arbitrary data[1].

• WebRTC :- does not require any server as a proxy to exchange data, other than the signaling server that is needed to share the network and media metadata (often done through WebSockets). The fact that stream data can be exchanged between clients directly often means faster messaging and less server-side workload.

WebRTC :- can run over TCP and UDP, but it often runs with UDP by default. Although UDP can lead to packet loss it does give a better performance which can lead to a more fluid voice or video call, and we can afford to lose a few frames when video calling.Given the advantages and disadvantages of the three technologies, we decided to use WebSockets for real-time messaging, which guarantees us packets delivery (unlike frames on a video call, we do not want to miss out any text message), as well as having a good compatibility and being a popular and documented choice.

When it comes to random requests, such as authentication, room creation or listing, AJAX is a good option. It does not require a permanent connection to the server, which results in less power usage for both the client and the server and the requests response times are be decent. However, none of these kinds of requests require an extremely rapid response.

• AJAX Requests :- are so popular that the latest browser’s versions themselves already offer high-level APIs as well as the low-level legacy libraries, which makes it trivial for any programmer to fetch JSON from any remote host without using any specific library.

**Phase 2: Planning, Prototypes, and Technology :-**

**2.1 Project Methodology** :-

• Agile is a set of techniques to manage software development projects. It consists

• Being able to respond to changes and new requirements quickly.

• Teamwork, even with the client.

• Building operating software over extensive documentation.

• Individuals and their interaction over tools. We believed it was a perfect fit for our project since we did not know most requirements beforehand.

•By using the Agile, we were able to focus only on the features which had the most priority at the time.

**Scrum :-** Scrum is one of the most popular Agile software development frameworks. It is iterative and incremental.

• Scrum’s objective is to create working versions of the product in a short amount of time. The final product will get more and more complete in every iteration. One of Scrum’s main features is that it takes for granted that requirements can vary at any given point and that few requirement changes should not be a hassle to have the product completed within a limited time frame

• **Advantage and disadvantages of Scrum**

•Scrum is a popular Agile framework characterized by short, iterative development cycles known as sprints.

•The development team breaks down the project into a series of user stories or tasks, each representing a specific feature or functionality of the chat application.

•During each sprint, the team focuses on implementing and delivering a set of user stories, typically within a time frame of 1-4 weeks.

•Daily stand-up meetings are held to review progress, discuss any obstacles, and plan the day's work.

•At the end of each sprint, a working increment of the chat application is demonstrated to stakeholders, allowing for feedback and adjustments to be made

**Use Cases and Scenarios : -**

User stories are one of the primary development artifacts when working with Agile methodology. A user story is a very high-level definition of a requirement, containing just enough information so that the developers can produce a reasonable estimate of the effort to implement it[3].

Gathered from stakeholders (people, groups or organizations who are interested in the project), they show us what we have to work in.

Since we were working with Agile, this list did not have to be complete before we started working on the project, but it was desirable to have at least a few items to start with so that we could establish proper feature priorities.

The value was quite subjective. We gave the highest priority to features which we believed they were essential to the platform (such as instant text messages) or were very related to the chat’s topic — coding. We gave them a score from 1-10.

Time cost was an estimation of how much we thought an individual story was going to take to implement. The measurement was done in days, considering each working day to be as long as 4 hours. We then translated this value as follows:

• 1-2 days: 1

• 3-4 days: 2

• 5-6 days: 3

• 7-9 days: 4

• 10+ days: 5

To sort both the product backlog and sprint backlog lists, we relied on a third number, the priority, which was simply the result of the value minus the time cost.

Nonetheless, in some cases, we had to make exceptions due to user stories dependencies. For example, sign in and sign up features had to be implemented the first, since we needed user information to properly identify the room owner or the message sender.

**Our initial product backlog list was the following :-**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | User story | Value | Time | Priority |
| 1 | As a user I want to be able to sign up with a unique username (that serves as an identifier throughout the platform). | 1 | 4 | 10\* |
| 2 | As a user I want to be able to sign back in. | 1 | 4 | 10\* |
| 3 | As a user I want to be able to log out when I will no longer use the platform on a given device. | 1 | 1 | 10\* |
| 4 | As a project manager I want to be able to create a new room for a project of mine | 5 | 2 | 10\* |
| 5 | As a user I want to be able to join rooms, previously created by the project manager. | 7 | 1 | 10\* |
| 6 | As a user I want to be able to create discussion chats. | 6 | 2 | 10\* |
| 7 | As a user I want to be able to enter the chats in the room I’m in. | 6 | 1 | 10\* |
| 8 | As a user I want to be able to read real time text messages in a chat. | 10 | 4 | 8 |
| 9 | As a user I want to be able to write text messages, which will be displayed to other room members in real time | 10 | 2 | 8 |
| 10 | As a user I want to be able to format messages (bold, italics, links, ...). | 8 | 2 | 7 |
| 11 | As a user I want to be able to share images when chatting. | 8 | 2 | 7 |
| 12 | As a user I want to be able to share code snippets | 9 | 3 | 7 |
| 13 | As a user I want to be able to stick a chat message on top of the chat. | 7 | 2 | 6 |
| 14 | As a user I want to be able to edit a chat. | 7 | 2 | 6 |
| 15 | As a user I want to be able to highlight some parts of a snippet | 7 | 2 | 6 |
| 16 | As a user I want to be able to link a chat with a GitHub repository, so that chat members can watch its updates in real-time (pull requests, issues, commits, comments, ...). | 9 | 6 | 6 |
| 17 | As a user I want to be able to fork a chat based on an activity message. | 8 | 4 | 6 |
| 18 | As a user I want chat information updates to be displayed in real time. | 6 | 1 | 5 |
| 19 | As a user I want to be able to merge a fork after the forked chat discussion has ended. | 7 | 4 | 5 |
| 20 | As a user I want to be able to read URLs’ description to know what the page is about before entering. | 6 | 2 | 5 |
| 21 | As a user I want to be able to post emoji’s on a chat. | 6 | 1 | 5 |
| 22 | As a user I want to be able to send files of any kind. | 7 | 4 | 5 |
| 23 | As a project manager I want to be able to kick members of a room. | 6 | 2 | 5 |
| 24 | As a user I want to be able to post real time polls on a chat. | 6 | 4 | 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 25 | As a developer I want to have access to a public API, so as to be able to build an application based on this one (such as bots integration). | 9 | 30 | 4 |
| 26 | As a project manager I want to be able to remove a room. | 4 | 2 | 3 |
| 27 | As a user I want to receive desktop notification when someone has sent a message, and I was not on the chat page. | 3 | 1 | 2 |
| 28 | As a user I want to be able to change my avatar. | 2 | 2 | 1 |
| 29 | As a user I want to be able to change my status (online, offline, away, busy, ...). | 2 | 2 | 1 |
| 30 | As a user I want to be able to find contacts by connecting to social networks. | 2 | 2 | 1 |
| 31 | As a user I want to be able to do group calls | 6 | 20 | 1 |
| 32 | As a user I want to be part of a reputation system (based on my chat and GitHub activity). | 6 | 15 | 1 |
|  |  |  |  |  |

**Use Case Title:** Testing Chat Message Delivery

**Actors** :-

* **Tester** : -Responsible for executing the test case.
* **Chat Application Users**:- Users who send and receive chat messages within the application.

**Preconditions**:

• The chat application is installed and accessible to all users.

• Users have logged into their respective accounts and are connected to the internet.

**\* Inflated priority due to dependencies.**

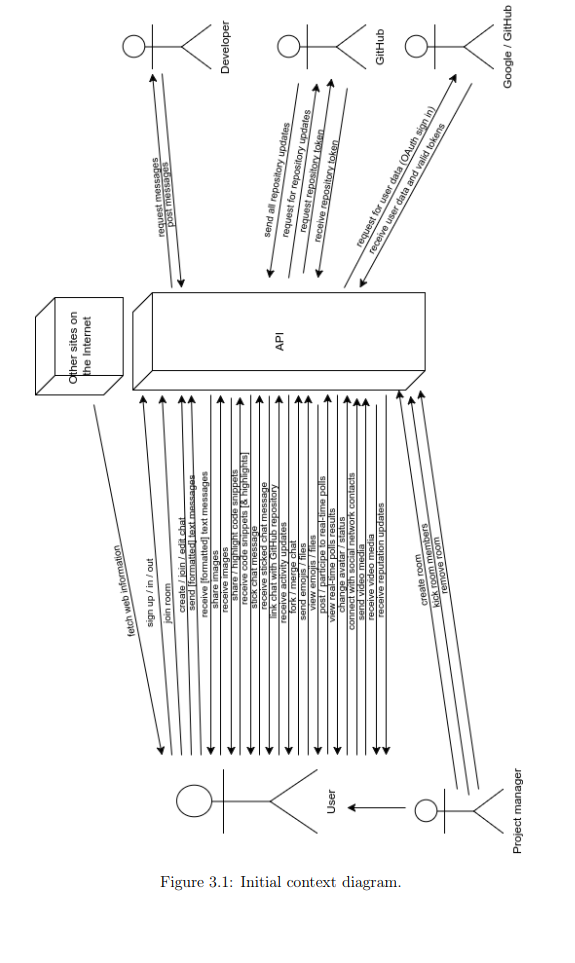
Later on, when we were implementing the project, we found out a few requirements that were missing and we noted them down.

Again, that was not a problem and it was in fact expected. After all, that is the main reason why are were using Agile development.

The new requirements table looks like the following:

|  |  |  |  |
| --- | --- | --- | --- |
| **User story** | **Value** | **Time** | **Priority** |
| As a user, it would be handy to have a landing page that points to existing chat rooms. | 8 | 2 | 7 |
| As a user, I want to have Mark Down support for messages, which includes at least bold, italics, tables, lists, and hyperlinks. | 8 | 2 | 7 |
| As a user, I want chat messages to auto scroll, other than having to manually scroll them down once I get a new message and the chat container is full. | 7 | 1 | 6 |
| As a user, I want activity messages to auto scroll, other than having to scroll them down manually once I get new updates and the messages container is full | 7 | 1 | 6 |
| As a user, I want the chat text input box to automatically resize itself if the text message is too large and doesn’t fit the default one. | 7 | 1 | 0 |
| As a user, I want to be automatically reconnected to the chat if my connection drops. | 5 | 3 | 3 |
| As a user, I would rather not have all conversations loaded at once, but rather have old ones retrieved whenever I need them | 4 | 3 | 2 |

**As you might have noticed already, product backlog, and consequently sprint back-**

****

**Estimated Timeline:-**

* Based on the initial user stories, we wrote the following timeline.
* We built it taking into consideration the product backlog priorities and each of the feature’s development time to make an approximation of what could be done within a semester.
* It did not mean to be our final timeline, and we were expecting some features to be developed sooner or later than anticipated, and to have few new user stories every sprint.

|  |  |
| --- | --- |
| **Week** | **Tasks** |
| 0-2 | **Planning and Requirement Analysis** (1-2 weeks):   * Gather requirements from stakeholders including students, faculty, and administration. * Define the project scope and objectives, considering features such as personal messaging, group chats, file sharing, and user authentication. * Identify key content requirements such as chat message formats, media file types supported, and user profile information. * Create wireframes or prototypes to visualize the app's layout and functionality. |
| 2-3 | **Design (2-3 weeks):**   * Design the user interface (UI) and user experience (UX) of the chat app, incorporating branding elements and ensuring intuitive navigation. * Develop a database schema to store user profiles, chat messages, and media files. * Define the system architecture, including frontend and backend components, and establish data flow diagrams. |
| 6-9 | **Development (6-8 weeks):**   * Implement frontend development using HTML, CSS, and JavaScript frameworks like React.js or Vue.js to create the user interface. * Develop backend logic using technologies such as Node.js with Express.js or Python with Django to handle user authentication, message routing, and data storage. * Integrate third-party APIs for features like user authentication via social media accounts or file storage services. * Implement content management features such as text formatting, emoji support, and media file uploads. |
| 2-4 | **Testing (2-4 weeks):**   * Conduct functional testing to ensure all features work as intended, including sending and receiving messages, joining group chats, and uploading files. * Perform user acceptance testing (UAT) with a sample group of users to gather feedback and identify any usability issues. * Test the scalability and performance of the app by simulating a large number of concurrent users and message exchanges. * Address any bugs or issues discovered during testing and conduct regression testing to ensure fixes are effective. |
| 1-2 | **Deployment and Launch (1-2 weeks):**   * Deploy the chat app to production servers, configuring hosting and domain settings as needed. * Perform final checks to ensure all components are functioning correctly in the live environment.   Create user documentation and help resources to assist users in navigating the app.   * Launch the chat app to the college community, promoting its availability and encouraging adoption among students and faculty. |

|  |  |
| --- | --- |
|  | **Post-launch Support and Maintenance (Ongoing):**   * **Week 20 onwards** :- Monitor app performance, gather user feedback, and address any issues or bugs reported. * **Ongoing** :- Implement updates and enhancements based on user feedback and emerging requirement**s.** |
|  |  |

**Sure! Here's a breakdown of the project phases with estimated time durations:**

**Planning and Requirement Analysis (2 weeks):**

**Week 1 :-**

* Gather requirements from stakeholders and define project scope.

**Week 2:-**

* Analyze requirements, create user stories, and develop wireframes or prototypes.

**Design and Coding (3 weeks):**

**Week 3-4:-**

* Design UI/UX, develop database schema, and define system architecture**.**

**Week 5: -**

* Finalize design elements and prepare design documentation**.**

**Week 6-9:-**

* Implement frontend development (UI/UX) and backend logic.

**Week 10-11 :-**

Integrate third-party APIs and implement content management features.

**Week 12 :-**

Conduct unit testing and address any issues identified.

**Testing (4 weeks):**

**Week 13-14 :-**

* Perform functional testing and user acceptance testing (UAT).

**Week 15-16 :-**

* Conduct performance testing and address scalability concerns**.**

**Week 17 :-**

* Finalize testing and prepare for deployment.
* Deployment and Launch (2 weeks):

**Week 18:**

* Deploy the chat app to production servers and configure hosting settings.

**Week 19 :-**

* Perform final checks and prepare for the official launch.
* Post-launch Support and Maintenance (Ongoing):

**Week 20 onwards:-**

* Monitor app performance, gather user feedback, and address any issues or bugs reported.
* Ongoing: Implement updates and enhancements based on user feedback and emerging requirements.

**By adhering to this estimated timeline** :-

* you can effectively manage the development process and ensure timely delivery of the chat app while allowing for flexibility to address unexpected challenges or changes in project requirements.

**Technology :-**

* The architecture of the application consists of the back end and the front end, both of them having their own set dependencies (libraries and frameworks).
* The front end is the presentation layer that the end user sees when they enter the site. The back end provides all the data and part of the logic and it is running behind the scenes.

**Back End :-**

* The "back end" refers to the logic and data layers running on the server side.
* In our case, the back end makes sure that the data introduced through the client application (the front end), is valid. Since the front end can be avoided or easily manipulated (the source code is available to the end user) we have to make sure that all the requests we receive are first verified by the server: the requested URI is supported, the user has the appropriate permissions, the parameters are valid, etc
* If the request data is valid, we do often proceed to execute some logic accompanied by one or more database accesses

**API (Application Programming Interface) :-**

Our application is all about I/O. We were looking forward a programming environment which was able to handle lots of requests per second, rather than one which was proficient at handling CPU-intensive tasks.

At the moment it seemed like the choice was between **PHP, Python, Java, Go or Node.js.** These languages have plenty of web development documentation available, and they have been widely tested by many already. The trendiest choice in 2016 was Node.js, which was exceptional for handling I/O requests through asynchronous processing in a single thread.

So we went for **Node.js** not only because of the performance but also because of how fast it was to implement stuff with it, contrary to other languages such as Java which are way more verbose. For web development, we would then **use Express**, which makes use of the powerfulness of Node.js to make web content even faster to implement.

A feasible alternative to Node.js would be Go, which is becoming popular nowadays due to **somewhat faster I/O than Node.js** with its Go subroutines, and unquestionably better performance when doing intensive calculations[4] (though we were not particularly looking for the last one).

We are writing Node.js with the latest **ECMAScript ES6** and ES2017 standard supported features. The development was started with ES6, but we also used a few features originally **from ES2017** as soon as Node.js turned to v7.

**ES6/ES2017** standards differ from the **classic Vanilla JavaScript** in that they have a few more language features and utilities out of the box which makes code easier to read, faster to write and reduce the need to make use of external libraries to do the most common operations. **For example**, Promises over callbacks or classes over functions, even though they are just syntactical sugar.

**A few remarkable frameworks/libraries we are using on the development of the application are :-**

**Express :-** A Node.js framework which makes web development fast. It abstracts most of the complexity behind the web server and acts as an HTTP route handler. It can also render views (a sort of HTML templates with variables) but are using the front end application for this instead.

By using Express, we are able to focus on the logic behind every request rather than on the request itself.

**Passport** :- An authentication library build specifically for Node.js. By using the different login modules (one module per provider), it hides all the complexity behind OAuth, OAuth2, and OpenID. Passport commits to notifying the developer in the same way regardless of the authentication method they have chosen.

**Socket.io :** -A JavaScript library which handles WebSocket connections. It abstracts most of the complexity behind WebSockets, and it also provides fallback methods which work without any special configuration. Socket.io takes care of the real time updates in our application, such as sending or receiving messages.

**Data storage :-**

We need to start our web server and create the directories and files we're going to use for our live support chat app. If you've installed XAMPP, follow the instructions below.

* Open **XAMPP Control Panel**
* Next to the Apache module, click **Start**
* Next to the MySQL module, click **Start**
* Navigate to XAMPP's installation directory (**C:\xampp**)
* Open the **htdocs** directory
* Create the following directories and files

**Front End :-**

Having separated the server-side from the client side, a SPA (Single-Page Application) was an outstanding choice. SPAs dynamically fetch data from the API as the user is browsing the site, avoiding to refresh the whole page whenever the user has filled in a form or navigated to another part of the site.

The UX boost a SPA can get over a traditional website is very significant. It is true that it often takes longer to load for the first time, due to having to download a bigger JavaScript file chunk, but once loaded the delay between operations is minimal which leads to a more fluid User experience, and less bandwidth use in most cases.

Implementing a scalable Single-Page Application by using Vanilla JavaScript only would take an enormous amount of time, since it has none of the high-level utilities that make it simple to develop one of this kind, such as a high-level HTML renderer that allows you to build elements on the fly, storage or router. Hence, it made sense to choose an actively maintained and documented framework/library to start with.

**At the time, the decision was between Angular, React and Vue :-**

Both Angular and React were being maintained by powerful corporations, Google and Facebook respectively, so we had a brief look at their documentation and developers’ reviews before taking our final choice. Eventually, we chose React.

React is a very powerful library with an enormous ecosystem (you can find many utilities that were meant to be used with React). It is featured due to its fast performance and small memory consumption, which is especially useful when targeting mobile devices. Moreover, there is a plethora of documentation on its official site and around the Internet.

React itself, contrary to Angular, is just the V(iew) in the MVC architecture. Hence, we required of additional libraries to fulfill the missing parts, so as to focus only on our project content.

Fortunately, that was not a problem. React’s vast ecosystem got this covered. For example, there was react-router for route handling or Redux for storage. The most relevant libraries we are using on our client-side application are :-

**Babel :-** A few users coming to our site might be using old browser versions, which have little to no support to ES6/ES2017 features. To make sure all browsers can understand our code we make use of Babel, which transpiles our modern JavaScript code into JavaScript code that most browsers can understand.

**Redux :-** An in-memory storage for JavaScript. It saves application states, which in other terms are the different data that our application uses over the time.

A storage like Redux avoids having to transfer data up and down the React tree, since Redux stores it all in one place which can be accessed anytime. It is also modular which makes it ideal for our application since it helps towards scalability.

Redux was initially built for React, so it works hand to hand with it. The storage can be easily connected to React components, which will have access to any of the stored data and also be able to dispatch new actions to add/update the data in it .

**Express** :- Express on the client side? Yes, to provide server-side rendering.

SPA applications have no content in the HTML file that is sent to the client. All the content (and logic) resides inside the one or many JavaScript chunks that the site may have, which means that browsers have to download these file(s) and execute them prior being able to display anything useful into the user’s screen. That can take up to a few seconds depending on the sizes of the main JavaScript chunks.

**Command :-** npm install -s express

dependencies”: {

“express”: “.16.3”

}

var express = require(‘express’);

var app = express();

var server = app.listen(3000, () => {

console.log(‘server is running on port’, server.address( ).port);

});

**We can run the server.js using the command :-**

node ./server.js

**Role of Express.js in a Chat Application :-**

Express.js is a web application framework for Node.js that simplifies the development of server-side applications by providing a robust set of features and middleware. In the context of a chat application, Express.js plays several essential roles:

**Web Server Setup:-**

Express.js is used to set up and configure the web server that will host the chat application.

It creates an HTTP server to handle incoming requests from clients (web browsers or mobile apps).

**Routing:-**

Express.js facilitates routing by defining endpoints (URL paths) and corresponding request handlers.

Routes are used to handle different functionalities of the chat app, such as sending messages, fetching chat history, or managing user profiles.

**Middleware Integration :-**

Express.js middleware allows for preprocessing of incoming requests before they reach route handlers.

Middleware functions can handle tasks like authentication, request logging, data validation, and error handling.

**API Implementation :-**

Express.js is used to implement RESTFUL APIs for client-server communication in the chat application.

API endpoints are defined using Express's routing system to enable CRUD (Create, Read, Update, Delete) operations on chat messages, user profiles, and other resources.

**Static File Serving :-**

Express.js serves static files (HTML, CSS, client-side JavaScript) required for the chat application's frontend.

It simplifies the delivery of static assets and improves application performance by efficiently handling file requests.

**Socket.io Integration :-**

Express.js can be integrated with Socket.io, a library for real-time, bidirectional communication between clients and servers.

Socket.io can utilize Express's HTTP server to establish WebSocket connections, enabling instant messaging and live updates in the chat app.

**Database Interaction :-**

Express.js integrates with databases (e.g., MongoDB, PostgreSQL) to perform data operations such as storing and retrieving chat messages, user profiles, and session data.

Database queries are executed within Express route handlers, allowing seamless interaction with persistent data storage.

**Error Handling :-**

Express.js provides mechanisms for centralized error handling using middleware functions.

Custom error-handling middleware can catch and process errors, improving the reliability and robustness of the chat application.

**Node.js :-**

[Node.js](https://en.wikipedia.org/wiki/Node.js)  is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside the browser. The most important advantage of using Node is that we can use JavaScript as both a front-end and back-end language.

As we know, JavaScript was used primarily for client-side scripting, in which scripts were embedded in a webpage’s HTML and run client-side by a JavaScript engine in the user’s web browser.

Node.js lets developers use JavaScript to write Command Line tools and for server-side scripting — running scripts server-side to produce dynamic web page content before the page is sent to the user’s web browser.

To install node:

Ref :- [https://nodejs.org/en/download](https://nodejs.org/en/download/)/

Even though the node is single threaded it’s still faster to use asynchronous functions. For example, Node can process other things while a file is being read off disk, or while waiting for an HTTP request to complete. The asynchronous behaviour can be implemented using callbacks. Also the JavaScript works well with JSON and No-SQL databases.

**NPM Modules :-**

Node.js allows the modules of libraries to be included in the application. These modules can be user-defined or third party modules.

The third party modules can be installed using the following command:

**Command for Install** :- npm install module\_name

**For more packages :-**

or a package called **nodemon**, so that the changes made in the code will be automatically detected. We will download nodemon using the command .

npm install -g nodemon

**What are WebSockets and how do they work?**

WebSockets provide a bi-directional, full-duplex communication channel between the client and the server. This means that both the client and server can send and receive data at any time, without the need for additional HTTP requests or responses.

**WebSocket Protocol :-**

WebSocket is built on top of the[TCP protocol](https://www.fortinet.com/resources/cyberglossary/tcp-ip#:~:text=What%20is%20TCP%3F,data%20and%20messages%20over%20networks.), which provides a reliable, ordered, and error-checked delivery of data packets.

WebSocket provides an abstraction layer over TCP, which allows for a higher-level API for communication between the client and server.

**WebSocket Connection Lifecycle :-**

The client sends a WebSocket handshake request to the server, using the HTTP protocol.

The server responds with a WebSocket handshake response, which includes a protocol upgrade header, indicating that the connection will now use the WebSocket protocol instead of HTTP.

Once the connection is established, both the client and server can send and receive data over the WebSocket connection.

**Advantages of WebSockets :-**

Low latency: WebSocket provides a real-time communication channel, with low latency and reduced overhead compared to HTTP requests and responses.

Reduced server load: With WebSocket, the server can push data to the client without the need for additional HTTP requests, reducing the server load and improving scalability.

Bi-directional communication: WebSocket allows for full-duplex communication between the client and server, allowing both sides to send and receive data at any time.

### Socket.IO API :-

Socket.IO provides a simple API for sending and receiving messages, which makes it easy to build real-time applications.

The API provides events for connection, disconnection, message, and error handling, which can be used to implement real-time features.

Socket.IO is widely used for building real-time applications, including chat applications, online gaming, and collaboration tools. If you’re looking to build a real-time chat application, Socket.IO can simplify the process and make it easy to implement real-time features.

## Setting up the development environment :-

### Prerequisites

* Node.js and NPM ([Node Package Manager](https://en.wikipedia.org/wiki/Npm_(software))) should be installed on your system.
* A code editor such as Visual Studio Code or Sublime Text should be installed on your system.

### Creating a new Node.js project

* Open the command prompt or terminal and navigate to the directory where you want to create the new Node.js project.
* Run the command npm init and follow the prompts to create a new package.json file, which will hold the project’s metadata and dependencies.

### Installing dependencies

* Install the socket.io and socket.io-client packages using the following command:

*npm install socket.io socket.io-client –save*

* The –save flag adds the packages to the project’s dependencies in the package.json file.

### Creating the server-side code

* Create a new file server.js and add the following code to create a basic WebSocket server using Socket.IO:

*const io = require('socket.io')(3000);*

*io.on('connection', (socket) => {*

*console.log('New user connected');*

*socket.on('disconnect', () => {*

*console.log('User disconnected');*

*});*

*});*

* This code creates a WebSocket server that listens on port 3000 and logs a message when a new user connects or disconnects.

### Creating the client-side code

* Create a new file index.html and add the following code to create a basic HTML page that connects to the WebSocket server using Socket.IO:

*<!DOCTYPE html>*

*<html>*

*<head>*

*<title>Real-time chat application</title>*

*</head>*

*<body>*

*<h1>Real-time chat application</h1>*

*<script src="/socket.io/socket.io.js"></script>*

*<script>*

*const socket = io('http://localhost:3000');*

*socket.on('connect', () => {*

*console.log('Connected to server');*

*});*

*</script>*

*</body>*

*</html>*

* This code creates a WebSocket client that connects to the server on port 3000 and logs a message when the connection is established.

### Server-side code :-

* Modify the server.js file to handle messages sent by clients and broadcast them to all connected clients:

*const io = require('socket.io' , 3000);*

*io.on('connection' , socket) => {*

*console.log('New user connected') ;*

*socket.on('disconnect', () => {*

*console.log(' User disconnected ') ;*

*});*

*socket.on('chatMessage', (message) => {*

*console.log('Received message:' , message) ;*

*});*

*});*

* This code listens for the chatMessage event sent by clients and broadcasts the message to all connected clients using the io.emit method.