Privacy Network: Design and Implementation of a Security-Conscious, Location-Sharing Geosocial Network

Thesis submitted by

Manas Pratim Biswas (002011001025)
Anumoy Nandy (002011001121)
Kunal Pramanick (302111001005)

under the guidance of

Dr. Munmun Bhattacharya

in partial fulfilment of the requirements
for the award of the degree of

Bachelor of Engineering in Information Technology

Department Of Information Technology
Faculty of Engineering & Technology
Jadavpur University

2020 - 2024

This page has been intentionally left blank

BONDAFIDE CERTIFICATE

This is to certify that this project titled ***"Privacy Network: Design and Implementation of a Security-Conscious, Location-Sharing Geosocial Network"***, submitted to the **Department of Information Technology, Jadavpur University, Salt Lake Campus, Kolkata**, for the award of the degree of **Bachelor of Engineering**, is a bonafide record of work done by **Manas Pratim Biswas (Regn. No.: 153760 of 2020-2021)**, **Anumoy Nandy (Regn. No.: 153823 of 2020-2021)** and **Kunal Pramanick (Regn. No.: 159991 of 2021-2022)**, under my supervision from 03/07/2023 to 15/05/2024.

Countersigned By:

Dr. Munmun Bhattacharya
Assistant Professor
Information Technology

Dr. Bibhas Chandra Dhara
HOD, Information Technology
Jadavpur University

Acknowledgements

We would like to express our sincere gratitude to Dr. Munmun Bhattacharya for allowing us to pursue our final-year project under her supervision. In addition to the meticulous research guidance, her encouraging support for our ideas as well as the constructive criticisms during our experimental and initial developmental stages of the project would not have been possible.

We would also like to thank the entire Department of Information Technology including all the professors, lab assistants and non-teaching staff for being extremely cooperative with us throughout the developmental stages of the product - *Privacy Network*.

Above all, we thank our parents for their support and encouragement in our academic pursuits.

Department of Information Technology
Jadavpur University
Salt Lake Campus, Kolkata

Manas Pratim Biswas Anumoy Nandy Kunal Pramanick

JADAVPUR UNIVERSITY

Dept. of Information Technology

Vision:

To provide young undergraduate and postgraduate students a responsive research environment and quality education in Information Technology to contribute in education, industry and society at large.

Mission:

M1: To nurture and strengthen the professional potential of undergraduate and postgraduate students to the highest level.

M2: To provide international standard infrastructure for quality teaching, research and development in Information Technology.

M3: To undertake research challenges to explore new vistas of Information and Communication Technology for sustainable development in a value-based society.

M4: To encourage teamwork for undertaking real life and global challenges.

Program Educational Objectives (PEOs):

Graduates should be able to:

PEO1: Demonstrate recognizable expertise to solve problems in the analysis, design, implementation and evaluation of smart, distributed, and secured software systems.

PEO2: Engage in the engineering profession globally, by contributing to the ethical, competent, and creative practice of theoretical and practical aspects of intelligent data engineering.

PEO3: Exhibit sustained learning capability and ability to adapt to a constantly changing field of Information Technology through professional development, and self-learning.

PEO4: Show leadership qualities and initiative to ethically advance professional and organizational goals through collaboration with others of diverse interdisciplinary backgrounds.

Mission - PEO matrix:

Ms/ PEOs	M1	M2	M3	M4
PEO1	3	2	2	1
PEO2	2	3	2	1
PEO3	2	2	3	1
PEO4	1	2	2	3

(3 – Strong, 2 – Moderate and 1 – Weak)

Program Specific Outcomes (PSOs):

At the end of the program a student will be able to:

PSO1: Apply the principles of theoretical and practical aspects of ever evolving Programming & Software Technology in solving real life problems efficiently.

PSO2: Develop secure software systems considering constantly changing paradigms of communication and computation of web enabled distributed Systems.

PSO3: Design ethical solutions of global challenges by applying intelligent data science & management techniques on suitable modern computational platforms through interdisciplinary collaboration.

Abstract

The recent advancements in mobile and internet technology, coupled with cheap internet data, have witnessed an exponential increase in the usage of internet and people connecting via social media networks. However, the traditional social networking sites such as *Facebook*, *Instagram* or *Twitter* have had witnessed multiple allegations of capturing and selling the user data for their corporate and business purposes. Most importantly, a breach in the user's location data can expose sensitive information regarding that user's frequent places of visit, acquaintances they meet with, food habits and political preferences.

Our project explores the applicability and implementation of a Privacy Network as a scalable Geosocial Networking website. A user-centric design for a secure location sharing is implemented into our project. The final product, *Privacy Network*, is a web application that allows the users to register into our website with their details, add or remove friends and most importantly, decide and set their visibility and other privacy parameters very precisely to ensure a secure and robust social media usage. Even more, the users can query their friends based on parameters including but not limited to *age*, *gender*, *college* and *distance*.

The user auth and data is handled by a MongoDB backend, and the user location attributes are handled by a Postgres backend. PostGIS along with PostgreSQL is utilized to create custom SQL queries to fetch the user location details on-demand. Real-time location broadcasting and multicasting is achieved by upgrading the *HTTP* protocol to *WebSocket* protocol and utilizing raw *WebSockets* natively available in the client's browser. A separate WebSocket backend server handles all the WebSocket connections from the client side. Services such as *Vercel* and *Render* are utilised to host the frontend and backend servers. A complete documentation of all the backend API end-points, in compliance with the standardized OpenAPI specifications, is available as a Swagger documentation.

However, scaling WebSockets is challenging. Therefore, a *distributed Redis Pub-Sub* based architecture along with a *HAProxy load balancer* is proposed. Moreover, a *change-data-capture* mechanism using *Apache Kafka* is proposed to keep the MongoDB and PostgreSQL databases consistent and in synchronization with each other.

Keywords: typescript, websockets, mern, postgresql, postgis, redis, haproxy, kafka, social-network-analysis

Contents

List of Tables

List of Figures

This page has been intentionally left blank

Chapter 1

Introduction

This chapter gives a basic introduction about the background of Privacy Network, its necessity and inception. The later subsections give the consequent flow of the project report.

1.1 Geosocial Networking

Geosocial networking[1] is a type of social networking that integrates geographic services and capabilities such as geolocation to enable additional social dynamics. This technology allows users to connect and interact based on their physical locations, enhancing the social networking experience by facilitating real-time, location-based interactions. Popular applications of geosocial networking include location-based services like check-ins, geotagging, and finding nearby friends or events. These functionalities are supported by technologies such as GPS, Wi-Fi positioning, and cellular triangulation. By leveraging these technologies, geosocial networking can provide users with personalized content and recommendations based on their current location, thereby enriching the user experience. However, it also raises significant privacy concerns, as the collection and sharing of location data can lead to potential risks such as unwanted tracking and profiling. Consequently, there is a growing emphasis on developing secure geosocial networking applications that prioritize user privacy and data protection, incorporating advanced encryption and user consent mechanisms to safeguard sensitive location information.

The past decade has seen unprecedented advancements in mobile and internet technologies, resulting in an exponential increase in internet usage and social media connectivity. Affordable internet data has made the online world more accessible than ever before, fundamentally transforming how people interact and communicate. Social media platforms such as Facebook, Instagram, and Twitter have become integral to daily life, providing users with the means to connect, share, and engage with others on a global scale. These platforms, however, have also come under intense scrutiny for their handling of user data, with numerous allegations of data misuse for corporate gain.

1.1.1 Security Vulnerabilities in Geosocial Networking

The convenience and connectivity offered by social media come at a significant cost to user privacy. Traditional social networking sites have been accused of capturing and monetizing user data, leveraging personal information for targeted advertising and other business

purposes. This practice has raised serious concerns about data security and user consent, highlighting a critical need for more transparent and ethical data management practices.

Among the various types of data collected, location data is particularly sensitive. A breach in location data can reveal detailed insights into a user's daily routines, including frequent places of visit, social circles, dietary preferences, and even political affiliations. For instance, if a social media platform tracks a user's location, it can infer whether the user visits specific types of restaurants, attends political rallies, or frequents particular neighborhoods. Such information, if exposed, can lead to a myriad of privacy invasions and security risks, including unwanted surveillance, targeted harassment, and identity theft.

The exploitation of user data by social media giants has prompted a growing public outcry and a demand for greater privacy protections. High-profile incidents, such as the Cambridge Analytica scandal[2], have underscored the potential for misuse of personal data and the far-reaching consequences of such breaches. These events have sparked a broader conversation about the ethical responsibilities of technology companies and the need for robust regulatory frameworks to safeguard user privacy.

1.1.2 Addressing Privacy Tradeoffs

In response to these concerns, there has been a surge in the development of privacy-centric social networking alternatives. These platforms prioritize user data protection and transparency, offering users greater control over their personal information. By employing advanced encryption techniques and decentralized data storage solutions, these new networks aim to mitigate the risks associated with data breaches and unauthorized data exploitation.

In our current approach, we utilize two security parameters that the user can fine tune to filter his/her location. The user can set their own visibility to either *on/off* and can also *limit their visibility* till a certain range of distance. Thereby, The user can control the location shared to the rest of the users in the social network. We utilize a variable called ***isVisible*** which the user can update to control his/her visibility. *isVisible true* implies that the user wishes to share his/her locations, and *isVisible false* denotes that the user does not wish to share his/her location. The idea is that a particular pair of user can have four possibilities for the privacy parameters, as shown below:

A **Marker** shows the exact location of the user's friend, since the user's friend as allowed his/her location to be shared.

A **Blue Patch** or a Medium visibility means that the exact location will not be shared to the user's friend, rather a Blue Patch would be shown which denotes the user to be present

Table 1.1: Privacy Filtration

Connection	*isVisible*	*VisibilityLevel*
Friend	True	High (Marker)
Friend	False	Medium (Blue Patch)
Non-Friend	True	Low (Red Patch)
Non-Friend	False	None

within 10 *kms* radius.

A **Red Patch** or a Low visibility means that the exact location of the stranger will not be shared to the user, rather a Red Patch would be shown which denotes the user to be present within 20 *kms* radius.

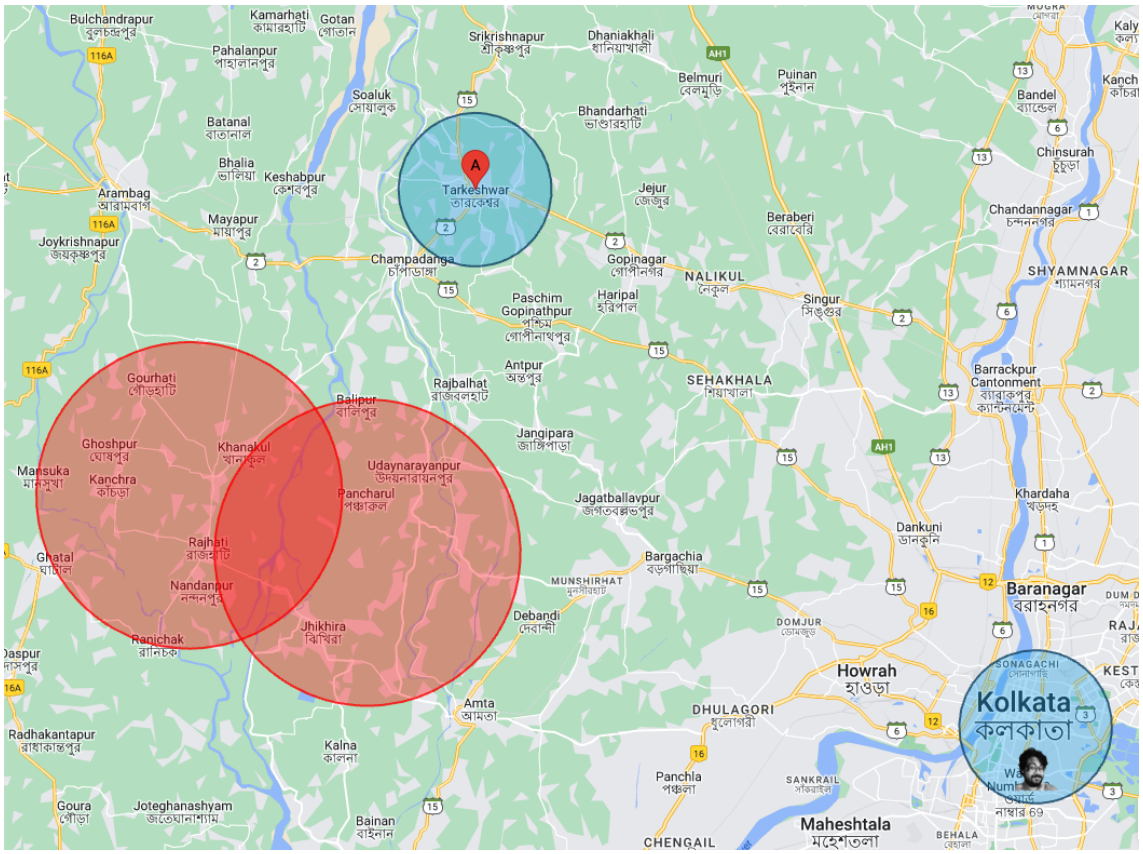

Figure 1.1: Privacy Filtration

1.2 Report Outline: The Privacy Network Approach

The report is organized as follows:

Chapter 2 presents the existing literature on privacy preserving schemes and techniques of location sharing for online mobile online social networks. Further, we summarize the features and technicalities of a similar web application based social networking site called *Loopt* that aimed to solve the similar issue.

Chapter 3 explores the architecture and the implementation details of Privacy Network. We briefly discuss the history and relevance of the MERN stack followed by a thorough scrutiny of the proposed architecture and the consequent coding implementation, CI/CD pipelines and deployment of the web application.

Chapter 4 and Chapter 5 provides a summary of the implementation outputs through a conclusion and discusses various potential future features and proposes advanced architectural to attain a distributed and scalable system.

Chapter 2

Literature Review

This chapter starts by briefly covering the relevant works in designing privacy preserving and efficient location sharing schemes for mobile online social networks. We discuss the case studies and vulnerabilities found through previous works. This is followed by the introduction of a product - Loopt, now discontinued, that tried to solve a very similar problem as that of ours.

2.1 Related Works

In the domain of mobile Online Social Networks (mOSNs), privacy and security have garnered significant research interest. Various privacy-preserving schemes have been proposed, each with distinct advantages and limitations. Early research primarily focused on information privacy, user anonymity, and location privacy. Techniques for maintaining location anonymity often involve encrypting the current location before transmission. K-anonymity methods obfuscate user locations, while dummy locations are used to mask real ones. Location encryption and pseudonym methods, such as mix zones and m-unobservability, are other prominent approaches.

K-anonymity for location privacy involves obfuscating a user's actual location, as demonstrated by researchers like Gruteser and Grunwald[3]. The use of dummy locations, where fake locations are sent along with the real one, has been explored by Kido et al.[4]. Location encryption methods, such as those by Khoshgozaran et al.[5], offer another effective means of protecting location privacy. Pseudonym-based methods, mix zones, and m-unobservability, as explored by Beresford and Stajano[6], have also been developed.

Rahman et al.[7] proposed privacy context obfuscation based on various location parameters to achieve location obscurity. The concept of location sharing with privacy protection in online social networks was initially addressed by SmokeScreen, which facilitated location sharing between friends and strangers. This approach was later enhanced by Wei et al.[8] with MobiShare, which separated social and location information into Social Network Services (SNS) and Location-Based Services (LBS), respectively. However, MobiShare faced the issue of LBS potentially revealing social network topologies during queries.

Li et al.[9] advanced this concept with MobiShare+, introducing dummy queries and private set intersection to prevent LBS from accessing users' social information. BMobiShare

further improved transmission efficiency by using a Bloom Filter instead of the private set intersection method, though it incurred high computation costs.

To counter insider attacks, Li et al.[10] proposed a multiserver location-sharing system in 2015, enhancing security at the cost of increased resource demands and inefficiency. These systems, relying on third-party location servers, are susceptible to collusion between LBS and SNS, leading to potential social information exposure and high transmission and storage costs.

Recently, Xiao et al.[11] introduced CenLocShare, which combines SNS and LBS into a single server, reducing communication and storage costs while enhancing user privacy protection. This amalgamation addresses the inefficiencies and security concerns of previous multiserver approaches, offering a more streamlined and secure solution for privacy-preserving location sharing in mOSNs.

Existing solutions however, typically follow two main approaches: using separate servers for location-based and social network information, which incurs high storage and communication overhead, or integrating both into a single server, which may lead to server bottlenecks and vulnerabilities to security attacks. Bhattacharya et al.[12]. proposed a novel privacy-preserving, secure, and efficient location-sharing scheme for mOSNs that addresses these issues. The proposed scheme ensures efficient and flexible location updates, sharing, and queries among social friends and strangers. Security validation is performed through a random oracle-based formal security proof, Burrows-Abadi-Needham (BAN) logic-based authentication proof, and informal security analysis. Additionally, the scheme's security is verified using ProVerif 1.93. Experimental implementation and evaluation demonstrate the scheme's efficiency and practicality.

2.2 Loopt

Loopt, Inc. was an American company headquartered in Mountain View, California. It provided a service for smartphone users to selectively share their location with others. They created an interoperable social-mapping service that allowed individuals to use their location to discover the real world around them – enabling them to find and enjoy the people, places, and events that mean the most right here just by using their mobile phones[13].

Founded in 2005, Loopt received initial funding from Y Combinator[14]. It secured Series A and B financing led by Sequoia Capital and New Enterprise Associates. Thereafter, Loopt partnered with Boost Mobile, Sprint, and Verizon to expand its reach. The iPhone app, Loopt Mix[15], enabled users to find and meet new people nearby. In 2012, Loopt was acquired by Green Dot Corporation for $43.4 million.

Figure 2.1: Loopt Inc

Features:

- **Location Sharing**: Loopt allowed users to share their real-time location with friends and family.
- **Privacy Controls**: Users could customize who could see their location, ensuring privacy.
- **Cross-Platform Support**: The service worked across major mobile operating systems.
- **Integration with Social Networks**: Users could link Loopt with Facebook and Twitter.
- **Proximity-Based Messaging**: Users could send messages and photos based on their location1

Figure 2.2: Loopt: The Geo Social Network

It is noteworthy to mention that Loopt was among the pioneers of location-based social mapping services. Its innovative approach influenced subsequent location-sharing apps and services.

Chapter 3

Architecture

In this chapter, we go through a brief overview and history of the MERN stack and then have a detailed discussion about the architecture of Privacy Network - the frontend, backend, deployment and scaling.

3.1 MERN Stack with TypeScript

MERN stands for *MongoDB*, *Express.js*, *React.js* and *Node.js*. With MongoDB as the Database, Express.js acts a web server framework (integrated with Node.js), React.js is the web client library, and Node.js is the server-side JavaScript runtime. It helps developers to develop Web apps based solely on full-stack JavaScript. Due to the same JavaScript platform in both the frontend and the backend, uniformity in the codebase is maintained. Naturally, the stack is supported by a vast number of open-source packages and a dedicated community for programmers to increase scalability and maintain software products.

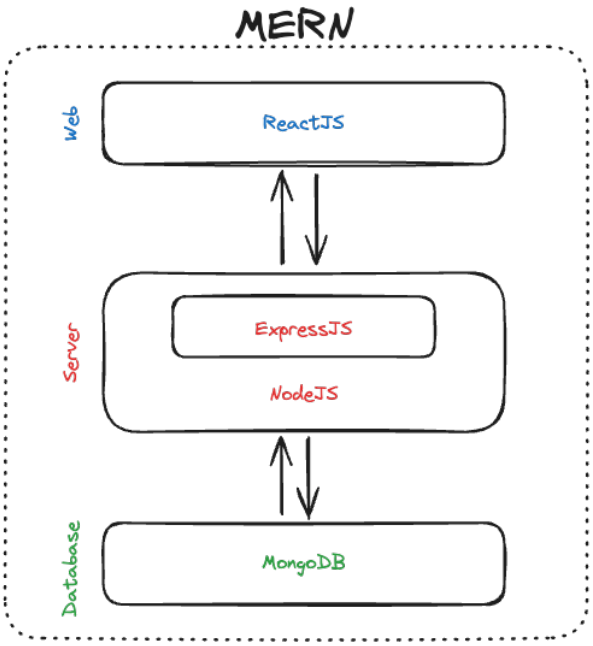

Figure 3.1: MERN stack architecture

3.1.1 Node

We are all familiar with the Google Chrome web browser. A typical web browser like Google Chrome runs HTML, CSS and JavaScript to render a website. The JavaScript code's execution actually happens by the virtue of an engine located inside the Chrome browser called ***V8***. Written in C++, V8 has proven to be an extremely powerful engine that can interpret and execute JavaScript very quickly.

Recognizing the robust capabilities of the V8 engine, the founders envisioned decoupling V8 from Chrome to develop a platform capable of executing JavaScript code on the server side. This vision materialized in 2009 when **Ryan Dahl** created ***Node.js***[16].

Node.js serves a role analogous to that of the *Java Virtual Machine* (JVM), functioning as a platform and runtime environment for applications. While the JVM executes *bytecode*, Node.js directly interprets and executes JavaScript code. In the JVM ecosystem, developers typically write source code in high-level languages such as Java, Scala, Groovy, or Kotlin, which is then compiled into bytecode. Conversely, Node.js natively understands and executes JavaScript, enabling developers to write application code directly in JavaScript. Additionally, languages that compile into JavaScript, such as TypeScript, can be utilized within the Node.js environment.

Non-Blocking programming in Node.js - The Node.js platform is versatile, perhaps not because of the V8 engine or the ability to support the JavaScript language, but in the Non-Blocking style of programming. Operations related to Non-Blocking in Node.js are mostly related to IO, such as reading and writing data to disk or calling Web APIs, for example. Furthermore, the use of Non-Blocking operations makes applications written on a Node.js platform capable of using computational resources (CPU) most efficiently.

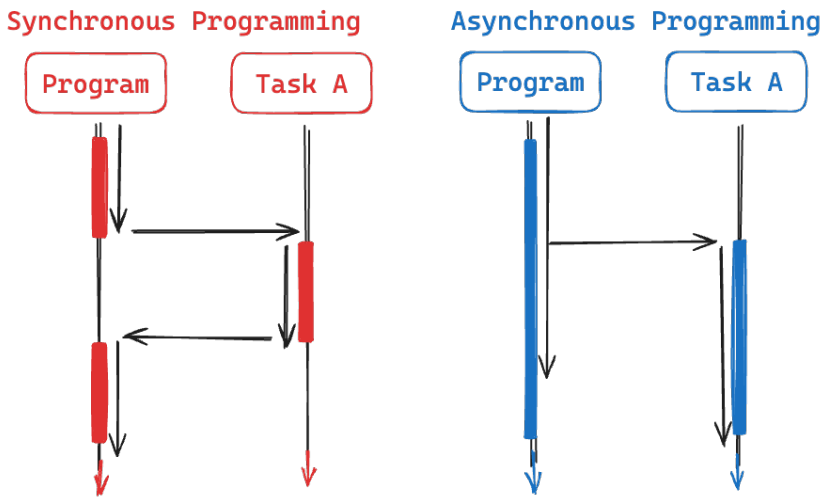

Figure 3.2: Synchronous vs Asynchronous Programming

In the creation of the Non-Blocking mechanism, Node.js applications operate according to the Event Loop design pattern. The illustration below explains this design in more detail:

- **Event Queue**: The Event Queue functions as a repository for storing events, which are specific processes within the program. Each event includes metadata that identifies the event type and its associated data. As a queue structure, the Event Queue operates on a First In, First Out (FIFO) principle. This ensures that events are processed in the order they were enqueued, maintaining the sequence of operations.
- **Main Thread**: The Main Thread is the primary thread responsible for the execution and termination of the program. It handles the computational processing of events retrieved from the Event Queue. This thread is the sole thread within the Node.js application that developers directly control, underscoring the single-threaded nature of Node.js applications. Consequently, developers are relieved from managing concurrency issues that are prevalent in multithreaded environments such as Java.
- **Node API**: The Node API is responsible for handling input/output (IO) operations through a multithreaded mechanism. Each completed IO operation generates a result that is encapsulated as an event and subsequently placed into the Event Queue for processing.

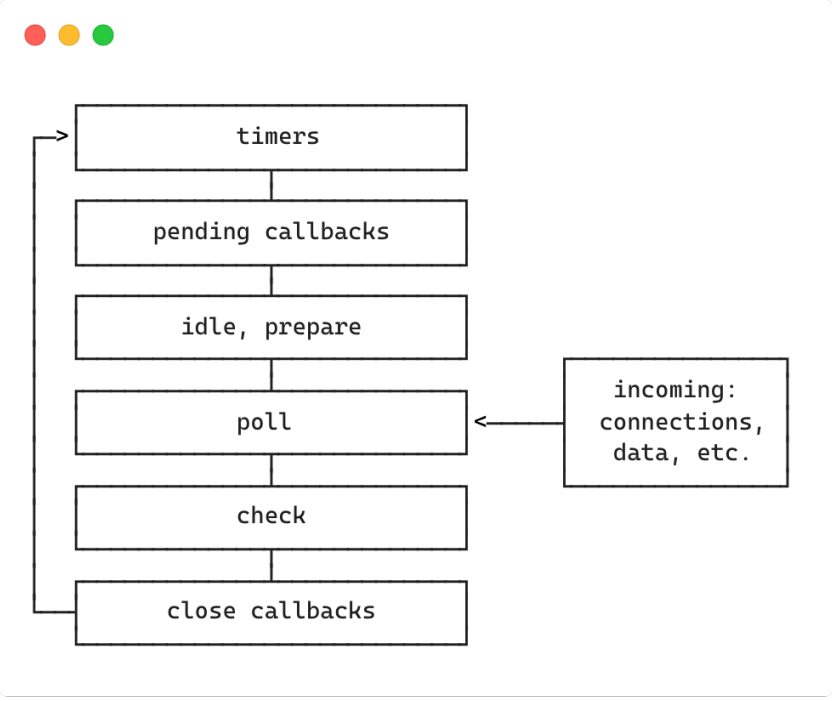

Figure 3.3: JavaScript Event Loop
[17]

With the above three components, the behavior of a Node.js application can be described as follows:

- The Main Thread executes the computational instructions as defined in the source code. When encountering IO operations, the Main Thread initiates a non-blocking call to the Node API and proceeds with executing subsequent commands without waiting for the IO operation to complete.
- Upon receiving a request from the Main Thread, the Node API handles the IO operations using multiple threads. Once an IO operation is completed, the Node API packages the result as an event and enqueues it into the Event Queue.
- The Main Thread processes events from the Event Queue sequentially. In the Main Thread, the code designed to handle these events is typically implemented as callbacks.

This process is iterative, creating a continuous cycle of event handling within the application. Consequently, the programming paradigm is oriented towards event-driven architecture rather than traditional sequential execution.

Node Package Manager - *npm, Yarn, and pnpm* are some of the dependency managers commonly utilized to support the development ecosystem of Node.js. With the vast majority of libraries available on NPM, integrating these libraries is straightforward via the npm install command, facilitating efficient package management.

NPM operates primarily in two significant roles:

- **Software Registry**: NPM functions as an extensive software registry used widely for publishing open-source Node.js projects. It serves as an online platform where users can publish and share various tools and libraries written in JavaScript. As of April 2020, the NPM registry hosts over 1.3 million code packages, making it a crucial resource for developers.
- **Command-Line Interface (CLI)**: NPM also provides a robust command-line interface that facilitates interaction with online platforms such as servers and web browsers. This CLI utility assists in package installation, uninstallation, version management, and server-driven management. Additionally, NPM manages the dependencies required to run Node.js projects efficiently.

Architectures of npm, Yarn, and pnpm:

- **NPM**: The architecture of NPM is designed around a centralized registry where all packages are stored and accessed. It follows a simple, straightforward approach where packages are installed in a *node_ modules* directory within the project structure. NPM supports both global and local package installations and includes a p*ackage-lock* file to manage dependency versions precisely.
- **Yarn**: Yarn offers an alternative architecture with a focus on speed, reliability, and security. It introduces a *lockfile* (*yarn.lock*) to ensure consistent dependency installations across different environments. Yarn uses a deterministic algorithm to install

packages, reducing installation times and preventing mismatched versions. It also supports offline caching, allowing packages to be installed without an active internet connection after the first download.

- **pnpm**: pnpm's architecture is designed to optimize disk space and improve performance. Unlike npm and Yarn, pnpm creates a single, centralized store for all packages on the system, and hard links are used to reference packages in individual projects. This approach avoids duplication of dependencies and speeds up installations. pnpm also ensures that the same package version is not installed multiple times, saving storage space and reducing redundancy.

3.1.2 Express

Express.js is a widely-adopted, open-source web application framework for Node.js, utilized extensively in commercial applications. Its robust community support and comprehensive documentation make it a reliable choice for developers, enabling the creation of projects ranging from small-scale applications to large enterprise solutions.

Express.js enhances the capabilities of Node.js by providing a wealth of support packages and additional features, which streamline the development process without compromising performance. Many renowned applications built on the Node.js platform leverage Express.js as a core component, highlighting its efficiency and effectiveness.

History and Development

Express.js was initially created by TJ Holowaychuk and first released on May 22, 2010, with version 0.12. In June 2014, StrongLoop took over the project management rights[18]. Subsequently, IBM acquired StrongLoop in September 2015[19], and by January 2016, Express.js management transitioned to the Node.js Foundation, ensuring its continued development and integration with the Node.js ecosystem.

Routers and Middleware

In web development, routers and middleware play crucial roles in managing HTTP requests and enhancing application functionality.

- **Routers**: A router is responsible for defining and handling the routes (endpoints) in a web application. It maps incoming HTTP requests to specific handlers based on the URL and HTTP method (*GET, POST, PUT, DELETE*, etc.). Routers help organize application logic and enable modular development by separating route definitions and their associated handlers.

- **Middleware**: Middleware functions are a series of functions that execute sequentially during the request-response cycle. They have access to the request object (*req*), the response object (*res*), and the next middleware function in the application's request-response cycle. Middleware can perform various tasks, such as request logging, authentication, parsing request bodies, handling errors, and more. By using middleware,

developers can extend the functionality of their applications in a modular and reusable manner.

Express.js as a Router

Express.js acts as a powerful router, enabling developers to define routes using a simple and intuitive syntax. Here is how Express.js handles routing:

- **Route Definition**: Developers can define routes using methods corresponding to HTTP verbs (*app.get(), app.post(), app.put(), app.delete()*, etc.). Each route can have a handler function that processes incoming requests and sends appropriate responses.
- **Route Parameters**: Express.js supports dynamic route parameters, allowing developers to capture values from the URL and use them within route handlers.
- **Modular Routing**: Express.js enables modular routing by allowing the creation of multiple router instances. Each instance can define a subset of routes, which can be mounted to specific path prefixes, improving code organization and maintainability.

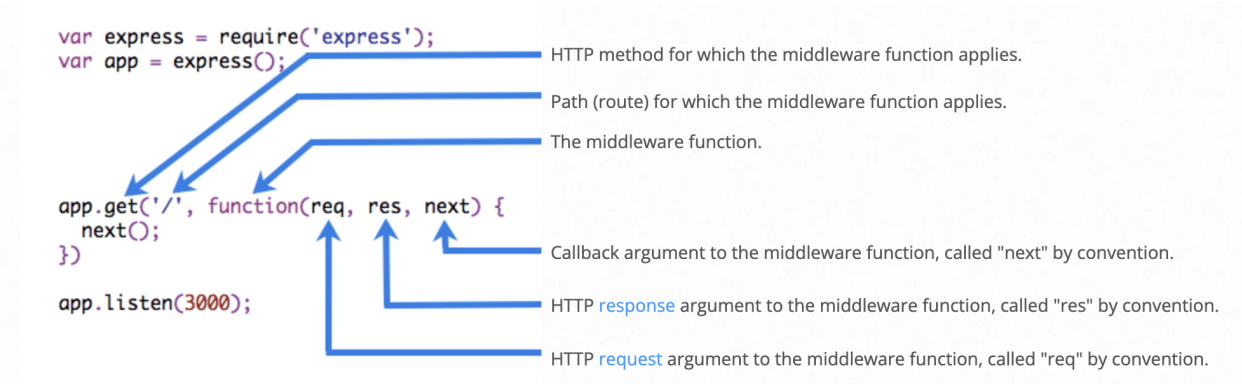

Figure 3.4: Anatomy of an Express Middleware
[20]

Express.js and Middleware

Express.js excels in middleware support, offering a flexible and powerful way to handle the request-response cycle:

- **Built-in Middleware**: Express.js includes built-in middleware for common tasks such as serving static files (express.static), parsing JSON and URL-encoded request bodies (*express.json, express.urlencoded*), and more.
- **Third-party Middleware**: Developers can leverage a vast ecosystem of third-party middleware to add functionality like session management, authentication, input validation, logging, and error handling.
- **Custom Middleware**: Express.js allows developers to create custom middleware functions tailored to their specific needs. These functions can perform a variety of tasks, from modifying the request and response objects to terminating the *request-response* cycle or passing control to the next middleware function.

The combination of robust routing capabilities and extensive middleware support makes Express.js an extremely suitable tool for building scalable and maintainable web applications on the Node.js platform.

3.1.3 MongoDB

MongoDB, developed by MongoDB Inc., was initially created in October 2007 as part of a Platform as a Service (PaaS) product akin to Windows Azure and Google App Engine. It was subsequently released as an open-source project in 2009.

MongoDB is a document-oriented NoSQL database, distinguishing itself from traditional relational databases by utilizing a flexible, JSON-like document structure called BSON (Binary JSON). This schema-less design allows for a dynamic and adaptable data model, making MongoDB particularly well-suited for applications requiring the storage of complex and evolving data.

Advantages of MongoDB

- **Flexible Schema**: MongoDB's document-based structure, with data stored in JSON-like formats, allows each collection to have documents of varying sizes and structures. This flexibility is advantageous for applications that require frequent and diverse data modifications without the rigidity of predefined schemas.
- **High Performance for CRUD Operations**: Since MongoDB is not bound by relational constraints, such as foreign keys or join operations, CRUD (Create, Read, Update, Delete) operations are executed more efficiently. This results in lower latency and higher throughput, making it suitable for applications with high transaction rates.
- **Scalability**: MongoDB supports horizontal scaling through sharding, where data is distributed across multiple nodes in a cluster. This architecture allows for seamless addition of new nodes to the cluster, facilitating system expansion and accommodating large-scale data growth.
- **Indexing for High Performance**: MongoDB automatically indexes the unique identifier field ($\_id$) in each document, ensuring high performance for query operations. Additional indexes can be created on other fields to further optimize query performance.
- **Efficient Data Access**: MongoDB caches frequently accessed data in RAM, minimizing disk I/O and improving read and write speeds. This in-memory caching capability is critical for applications demanding fast data retrieval and real-time processing.

Suitability for Dynamic User Data and Authentication Backend

MongoDB's flexible schema and powerful querying capabilities make it an excellent choice for storing dynamic user data and attributes. User profiles in modern applications often include a wide array of data points that can change over time, such as preferences, activity logs,

and social connections. MongoDB's document-oriented model allows for easy modification and extension of user data structures without the need for costly schema migrations.

Additionally, MongoDB is highly suitable for use in authentication backend servers due to the following reasons:

- **Scalable User Management**: As the number of users grows, MongoDB's sharding capability ensures that the authentication system can scale horizontally to handle increased load, providing consistent performance.
- **Secure Data Storage**: MongoDB offers robust security features, including encryption at rest and in transit, role-based access control, and auditing capabilities, ensuring that sensitive user data is protected.
- **Efficient Session Management**: Authentication systems often require efficient session management and storage of session data. MongoDB's high read and write speeds, coupled with its ability to cache data in RAM, enable rapid access to session information, enhancing user experience.
- **Real-time Data Processing**: MongoDB's real-time data processing capabilities are critical for authentication systems that need to validate user credentials, track login attempts, and enforce security policies promptly.

By leveraging MongoDB, developers can build robust, scalable, and secure authentication systems that efficiently handle dynamic user data and meet the performance demands of modern applications.

3.1.4 React

React.js is a highly efficient JavaScript library developed by engineers at Facebook, widely adopted by major companies such as Yahoo, Airbnb, Facebook, and Instagram for their product development. It excels in creating large, scalable applications, making it a preferred choice to smaller projects.

Features of React.js

- **Component-Based Architecture**: React promotes the development of reusable components, enabling developers to break down complex problems into manageable, testable units. This modular approach simplifies the management and extension of the system. Unlike Angular, which requires an optimal structure and coding style, React offers flexibility in structuring components, making it adaptable to various project requirements.
- **Stateless Components**: React encourages maintaining components as stateless as possible, enhancing predictability and simplifying state management. Stateless components function similarly to static HTML pages, receiving inputs and rendering outputs based on those inputs. This characteristic explains their reusability and ease of testing.

Strengths of React.js

React is designed with performance optimization in mind, particularly in rendering views. Traditional MVVM (Model-View-View-Model) frameworks can struggle with efficiently displaying large datasets. React addresses this issue by re-rendering only the components that have changed, rather than the entire view.

One of React's core strengths lies in its use of the **Virtual DOM (Document Object Model)**. When a view update is required, React creates a virtual representation of the DOM. It then compares this virtual DOM with the actual DOM, identifies the differences, and applies only the necessary changes. This process, known as reconciliation, minimizes direct DOM manipulations, significantly enhancing performance.

For instance, in a list of 20 products, if the second product is updated, React will only re-render that specific product, leaving the other 19 products unchanged.

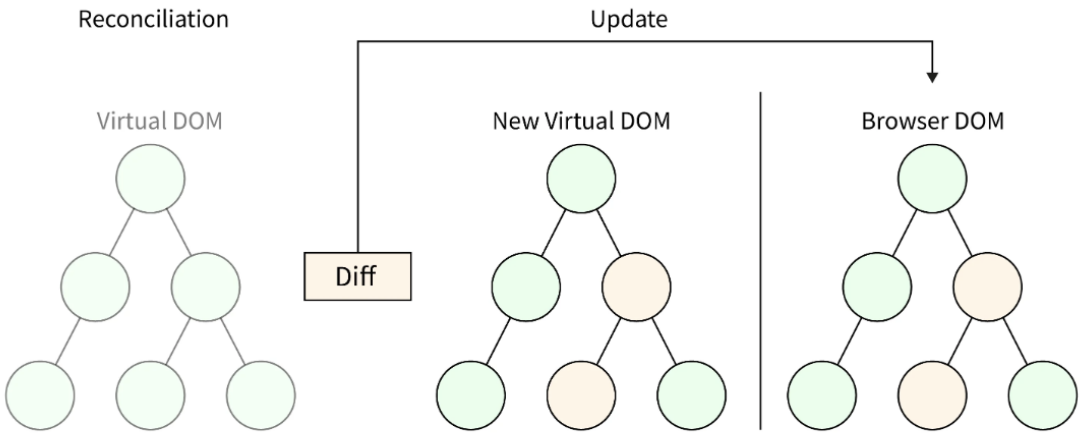

Figure 3.5: React Virtual DOM Diffing and Reconcilliation

Suitability for Dynamic UIs with Real-Time Data

React's architecture and capabilities make it exceptionally well-suited for building dynamic user interfaces, particularly in scenarios where location data changes frequently due to real-time updates from a WebSocket backend.

Moreover, React can be easily integrated with numerous UI component driven libraries such as Material UI, shadcn, ChakraUI, Shoelace resulting in a faster development process.

Why React for Privacy Network ?

- **Efficient Data Handling**: React's ability to manage and render data efficiently is crucial for applications that require frequent updates. The Virtual DOM ensures that only the necessary parts of the UI are re-rendered, providing a smooth and responsive user experience.
- **Seamless Integration with WebSockets**: React can seamlessly integrate with WebSocket-based backends to handle real-time data updates. Components can subscribe to WebSocket events, ensuring that any changes in location data are immediately reflected in the UI without the need for full-page reloads.
- **Declarative UI Updates**: React's declarative approach to UI development allows developers to specify how the UI should look based on the current state. When location data changes, React automatically updates the affected components, ensuring that the UI remains consistent and up-to-date.
- **Improved User Experience**: The efficient update mechanism provided by the Virtual DOM enhances the user experience by reducing the time and resources needed to reflect data changes in the UI. This is particularly important for applications that display real-time location data, where quick updates are essential.

Hence, React's component-based architecture, performance optimization through the Virtual DOM, and seamless integration with real-time data sources like WebSockets make it an ideal choice for developing dynamic UIs that handle frequently changing location data.

3.1.5 TypeScript

TypeScript is a statically typed superset of JavaScript that compiles to plain JavaScript. It offers several advantages for both frontend and backend development.

Transpilation and Runtime Execution

In TypeScript development, the TypeScript compiler (tsc) is used to transpile TypeScript code into plain JavaScript code. The transpilation process involves converting TypeScript source files (.ts) into JavaScript files (.js) that can be executed by JavaScript engines.

Once the TypeScript code is transpiled into JavaScript, it can be run in the same way as any other JavaScript code. In frontend development, the transpiled JavaScript files are included in HTML documents and executed by web browsers' JavaScript engines. Similarly, in backend development, the transpiled JavaScript files are executed by Node.js, a runtime environment for executing JavaScript code outside a web browser.

Under the hood, the TypeScript compiler parses TypeScript code, type-checks it based on type annotations and inferred types, and generates equivalent JavaScript code. This

JavaScript code retains the same functionality as the original TypeScript code but lacks type annotations and other TypeScript-specific features.

The transpilation process ensures that TypeScript code can be executed in any JavaScript environment, enabling developers to write TypeScript code for both frontend and backend development. Additionally, TypeScript's static typing system provides compile-time type checking to catch errors early in the development process, while the resulting JavaScript code runs efficiently in modern JavaScript engines.

Why is TypeScript a suitable choice ?

Frontend Development

- **Enhanced Code Quality**: TypeScript's static typing system allows developers to catch type-related errors during development, leading to higher code quality and fewer runtime errors in frontend applications.
- **Improved Developer Productivity**: With features like type inference and intelligent code completion, TypeScript enables developers to write cleaner and more maintainable code. This leads to increased productivity and faster development cycles.
- **Better Tooling Support**: TypeScript has excellent tooling support, including integrated development environments (IDEs) like Visual Studio Code and powerful debugging tools. These tools enhance the development experience and streamline the frontend development process.
- **Stronger Codebase**: By enforcing type safety, TypeScript helps build a stronger and more robust codebase for frontend applications. This reduces the likelihood of bugs and makes it easier to refactor and maintain the code over time.

Backend Development

- **Unified Development Experience**: Using TypeScript in both frontend and backend development creates a unified development experience across the entire stack. Developers can leverage their knowledge of TypeScript to work seamlessly on both frontend and backend codebases.
- **Shared Code and Interfaces**: TypeScript allows for the creation of shared code and interfaces between frontend and backend components. This promotes code reuse and ensures consistency across the application, leading to more maintainable and scalable software solutions.
- **Enhanced Error Detection**: Similar to frontend development, TypeScript's static typing system helps detect errors in backend code during development. This reduces the risk of runtime errors and enhances the overall reliability of backend services.
- **Improved Collaboration**: With TypeScript, teams can collaborate more effectively on backend development projects. The use of static types and clear interfaces makes it easier for developers to understand and contribute to the codebase, leading to better teamwork and faster project execution.

Leveraging TypeScript in both frontend and backend development offers numerous benefits, including improved code quality, developer productivity, and collaboration, as well as stronger codebases and enhanced error detection.

3.2 Privacy Frontend

3.2.1 A component based React + Material UI

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

3.2.2 Vite the build tool

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

3.2.3 React Custom Hooks

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

3.3 Privacy Backend and Databases

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

3.3.1 Database Schema, ORMs and ERDs

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

3.3.2 MongoDB and PostgreSQL

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

3.3.3 PostGIS for geolocation SQL queries

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

3.3.4 WebSocket Server for real-time location sharing

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

3.3.5 OpenAPI Specs and Swagger documentation

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in

voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

3.4 Cloud Hosting and Deployment

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

3.4.1 Vercel the cloud service provider for the frontend

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

3.4.2 Render the cloud service provider for the backend

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

Chapter 4

Conclusion

This thesis focuses on the implementation of a security-conscious Geo Social Networking web application, named *Privacy Network*. To address privacy concerns, *Privacy Network* prioritizes user control over location data. Users can define their visibility settings with granular control, ensuring a secure and customizable social media experience.

The front-end technologies used to develop the web application are:

- **User Interface**: React, a component-based UI library, is used for a smooth and interactive user experience
- **Styling**: Material UI, TailwindCSS, and raw CSS are
- **Build Tool**: Vite, a lightning-fast development server and build tool, is used for an overall smoother development experience.

The backend utilizes a distributed architecture for scalability, featuring:

- **MongoDB server**: Stores user authentication and profile data
- **Postgres server**: Handles user location attributes with PostGIS, enabling location-based queries.
- **WebSocket server**: Facilitates real-time communication through WebSockets, allowing for immediate location updates and interactions trough bidirectional fullduplex communication channel.
- **OpenAPI Specs**: A comprehensive documentation for all backend API endpoints, strictly adhering to the standardized OpenAPI specifications, is accessible in the well-known Swagger format.

Git is used for version controlling, and GitHub hosts the git repository of our application codebase. Platforms like Vercel and services like Render are utilized to deploy the web application's frontend and backends respectively, while GitHub Actions CI/CD pipelines ensuring smooth deployments.

By prioritizing user privacy and utilizing scalable architecture, *Privacy Network* lays the groundwork for a secure and dependable Geo Social Networking platform.

Privacy Network can be accessed at ***https://privacynetwork.sanam.live***

Chapter 5

Limitations and Future Work

In this chapter, we discuss a few of the bottlenecks, limitations and technical difficulties that exist in our current approach of building the *Privacy Network*. In particular, challenges in scaling the system, making it consistent, available and distributed are touched upon. We suggest some of the possible remediation for the same. In the later part of the chapter, we discuss the future scope of the application and propose some more challenging yet relevant features that can be incorporated into *Privacy Network*.

5.1 Scaling WebSocket Servers with Redis Pub-Sub and HAProxy

As the demand for real-time communication, location sharing and low-latency interactions between clients and the *Privacy Network* servers continues to escalate, the traditional paradigms of web communication face challenges in scaling effectively. WebSocket, with its stateful nature, presents a promising solution, yet its scalability remains a concern. This section explores a solution for scaling WebSocket servers to handle high volumes of concurrent connections and real-time communication demands.

Traditional HTTP servers excel in horizontal scaling due to their stateless nature. However, WebSockets, being stateful protocols, present challenges when scaling. A single WebSocket server maintains client state, and simply adding more servers doesn't inherently replicate this state across them. Additionally, a single server cannot efficiently broadcast messages to a large number of connected clients. Also note that, horizontal scaling alone is insufficient for replicating states across multiple websocket servers, while vertical scaling encounters hardware limitations.

The proposed architecture leverages Redis Pub-Sub for message broadcasting and keeping the connections stateful, while HAProxy is being used for load balancing client connections across multiple WebSocket server instances.

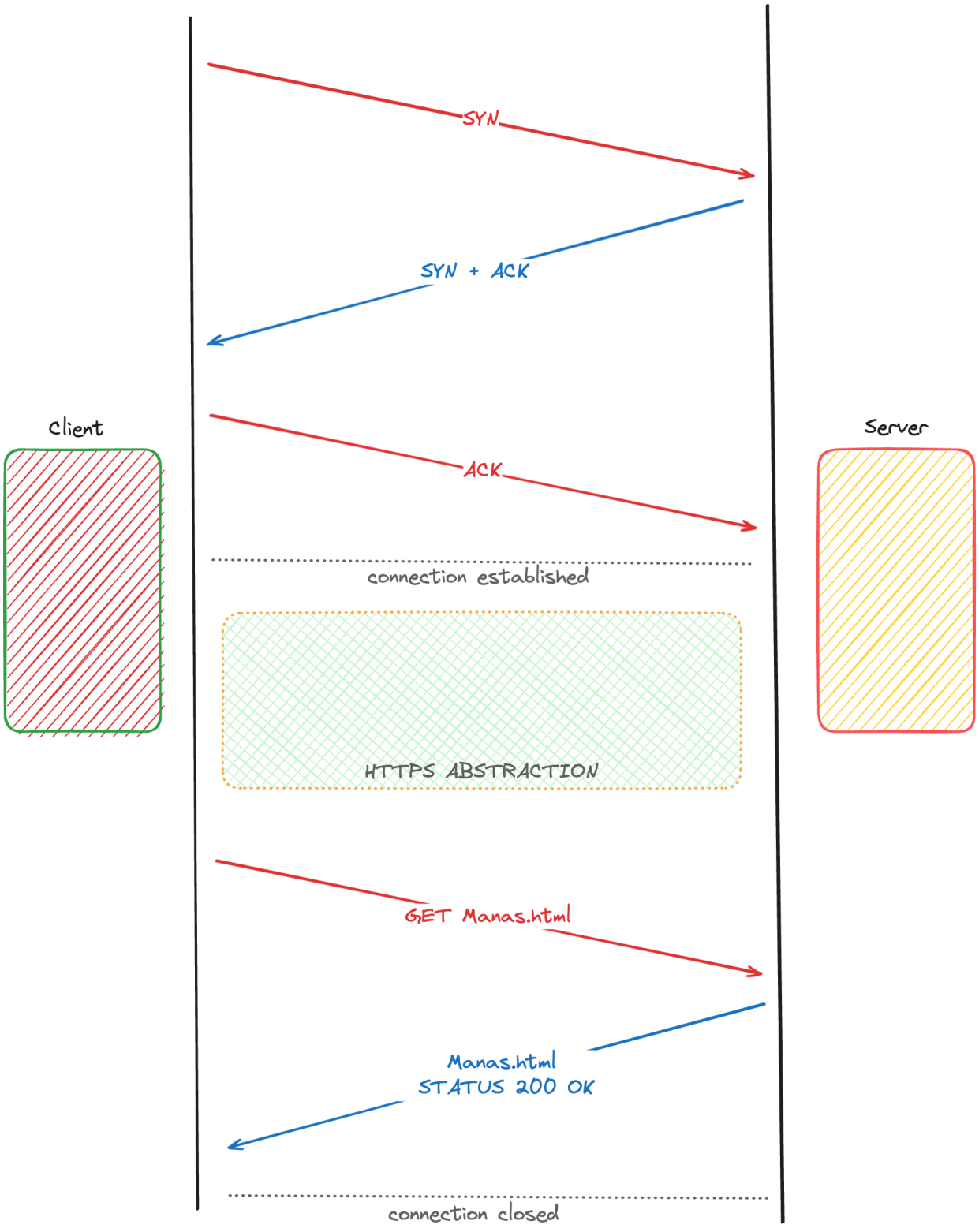

Figure 5.1: HTTP, is a stateless protocol. Millions of clients can be served independently across multiple HTTP servers that are load balanced by reverse proxy servers, making it horizontally scalable.

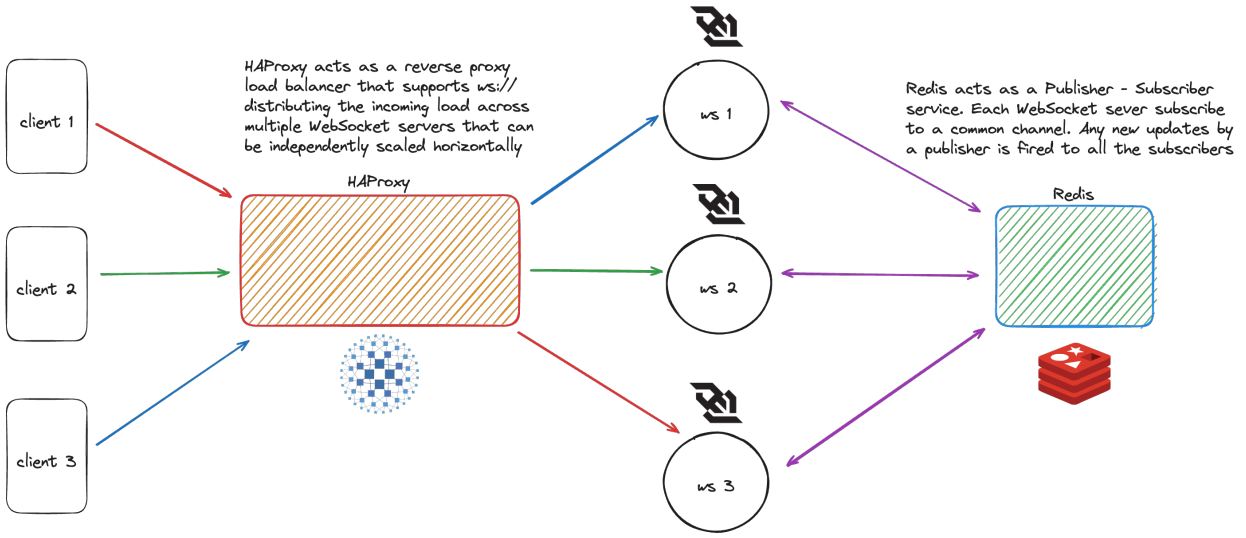

Figure 5.2: A scaled WebSocket architecture

- **Redis Pub-Sub**: Acts as a message broker, enabling real-time communication between WebSocket servers. Clients connect to individual WebSocket servers. When a server needs to broadcast a message, it publishes it to a dedicated Redis channel. All subscribed WebSocket servers receive the message and can then forward it to their connected clients. This approach decouples message broadcasting from individual WebSocket servers, improving scalability.
- **HAProxy**: Functions as a load balancer, distributing incoming client connections across the pool of WebSocket servers. This ensures even distribution of client load and prevents overloading any single server. Additionally, HAProxy can implement health checks to identify and remove failing WebSocket servers from the pool, maintaining overall system availability.

This architecture offers several advantages for scaling WebSocket applications:

- **Horizontal Scalability**: By adding more WebSocket servers to the pool behind HAProxy, the system can handle increasing client connections.
- **Real-time Communication**: Redis Pub-Sub facilitates efficient broadcasting of messages to all connected clients with low latency.
- **High Availability**: HAProxy ensures continuous service even if individual WebSocket servers fail.
- **Improved Performance**: Distributing client load across multiple servers enhances overall application responsiveness.

While this architecture provides a robust foundation for scaling WebSockets, some limitations require consideration

- **Redis Pub-Sub Single Point of Failure (SPOF)**: Although Redis offers high availability configurations, it can still be a SPOF. Exploring alternative pub-sub solutions with inherent high availability, like Apache Kafka, could enhance fault tolerance.
- **State Management Complexity**: Maintaining consistency across replicated state stores can be challenging. Implementing distributed locking mechanisms or optimistic locking could ensure data integrity during concurrent state updates.
- **Distributed Cache Invalidation**: The effectiveness of a distributed cache layer hinges on proper invalidation strategies. Implementing cache invalidation mechanisms like cache expiration or message bus invalidation could optimize cache utilization.

Future work directions can further improve the architecture:

- **Integration with Container Orchestration Platforms**: Investigating seamless integration with container orchestration platforms like Kubernetes could enable dynamic scaling of WebSocket servers based on real-time traffic demands.
- **Load Balancing Algorithm Optimization**: Exploring more advanced load balancing algorithms within HAProxy, such as weighted round-robin or least connections, could potentially improve client distribution across WebSocket servers.
- **WebSocket Protocol Extensions**: Researching the utilization of WebSocket protocol extensions for message fragmentation and compression could further optimize real-time communication efficiency.

5.2 Change Data Capture for synchronization of SQL-NoSQL Databases

This section discusses the limitations of the current approach for synchronizing data between the PostgreSQL database and the MongoDB database, and proposes a future work direction using Apache Kafka for Change Data Capture (CDC) to achieve real-time data consistency across these heterogeneous databases.

Current Approach and Limitations:

The current solution synchronizes common user data fields (college, age, gender, isVisible) from MongoDB to PostgreSQL upon updates in MongoDB. This approach offers several benefits:

- **Reduced Redundancy**: User data is not directly updated in PostgreSQL, minimizing redundancy.

- **Focus on MongoDB Updates**: Since CRUD operations primarily occur in MongoDB, this simplifies the synchronization logic.
- **ACID Compliance**: Updates are performed sequentially, ensuring ACID properties within each database.

However, a critical limitation exists:

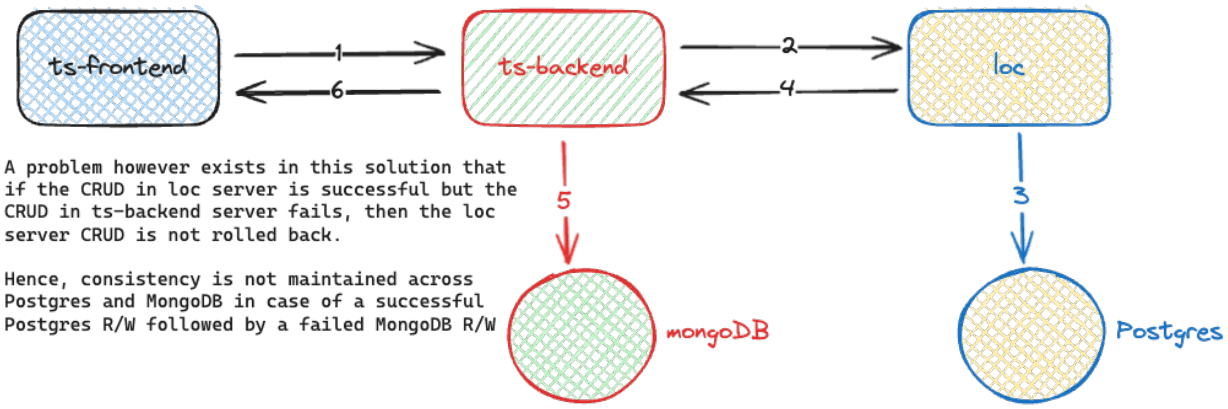

Figure 5.3: Database write-through approach

- **Eventual Consistency Issue**: If a successful update in PostgreSQL is followed by a failed update in MongoDB, data consistency is compromised. This scenario creates an orphaned update in PostgreSQL, as the corresponding MongoDB update fails to reflect the change.

To address the eventual consistency issue and achieve real-time data synchronization across PostgreSQL and MongoDB, we propose implementing Kafka CDC. Here's how it would work:

- **Change Capture on Both Databases**: Kafka connectors would be deployed to capture changes (*inserts*, *updates*, *deletes*) from both PostgreSQL and MongoDB in real-time.
- **Publishing Changes to Kafka Topics**: Captured changes would be published as messages to dedicated Kafka topics for each database (e.g., "*postgres_changes*" and "*mongodb_changes*").
- **CDC Consumer Service**: A dedicated consumer service would subscribe to both Kafka topics. Upon receiving messages, it would perform the following actions:

  - **For PostgreSQL Changes**: If the message originates from the "*postgres_changes*" topic, the consumer service would attempt to update the corresponding data in MongoDB.
  - **For MongoDB Changes**: Similar to above, if the message originates from the "*mongodb_changes*" topic, the consumer service would update the corresponding data in PostgreSQL.

- **Error Handling and Transaction Management**: The consumer service would implement robust error handling mechanisms. In case of update failures in either database, the service would attempt retries or trigger rollback mechanisms to maintain data consistency across both databases.

The immediate benefits upon introducing the above architecture is as follows:

- **Real-Time Synchronization**: Updates are reflected in both databases with minimal latency, ensuring data consistency.
- **Bi-directional Synchronization**: The solution allows for updates originating from either database to be reflected in the other, providing flexibility.
- **Improved Scalability**: Kafka's distributed architecture facilitates horizontal scaling to handle large data volumes and high update rates.

Table 5.1: Comparison of Kafka and Redis

Parameter	**Kafka**	**Redis**
Message Size	Supports message size up to 1GB	Supports smaller message size
Message Delivery	Subscribers pull messages from queue	Servers pushes messages to subscribers
Message Retention	Retains message after retrieval	Does not retain message
Latency	Low latency. Slightly slower than Redis due to data replication	Ultra low latency when distributing smaller-sized messages
Parallelism	Supports parallelism, multiple consumers	Does not support parallelism
Throughput	High throughput - Asynchronous read/write	Lower throughput - Server waits for reply

5.3 Future Work: Integrating a Redis Distributed Caching Layer

While the proposed architecture utilizing Kafka CDC effectively synchronizes data across relational and non-relational databases, further optimization can be achieved by introducing a Redis distributed caching layer. This section explores the potential benefits and considerations for implementing a Redis cache.

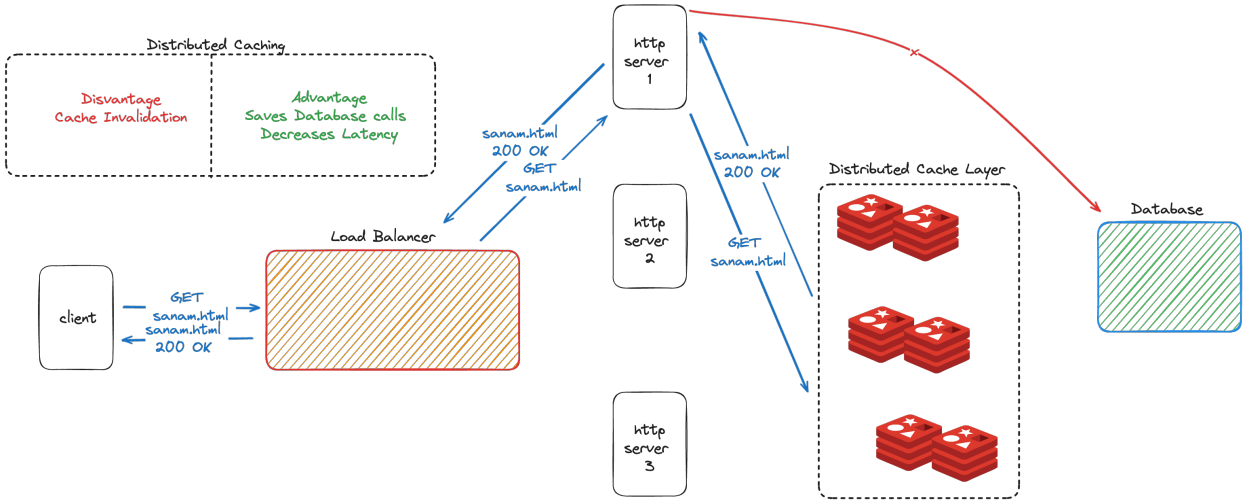

Figure 5.4: Distributed Redis Caching Layer

Potential Benefits

- **Reduced Network Traffic**: Frequently accessed data can be cached in memory by Redis, minimizing the number of database calls required. This can significantly improve application performance, especially for read-heavy workloads.
- **Faster Response Times**: By serving data from the in-memory cache, response times for data retrieval operations can be significantly reduced compared to traditional database access.
- **Improved Scalability**: Caching can effectively offload database load, allowing the system to handle increased traffic without overwhelming the databases.

We explored several promising avenues for future work that can enhance the robustness, scalability, and performance of the proposed architecture. By pursuing these future work directions, the *Privacy Network* architecture can be continuously refined to deliver a highly scalable, secure, and performant solution for protecting user data and enabling privacy-preserving location sharing.

Bibliography

[1] Wikipedia, "Geolocation networking," 2008-.

[2] BBC, "Meta settles cambridge analytica scandal case for $725m," 2022.

[3] M. Gruteser and D. Grunwald, "Anonymous usage of location-based services through spatial and temporal cloaking," in *Proc. 1st Int. Conf. Mobile Syst., Appl. Services (MobiSys)*, 2003.

[4] H. Kido, Y. Yanagisawa, and T. Satoh, "Protection of location privacy using dummies for location-based services," in *Proc. 21st Int. Conf. Data Eng. Workshops (ICDEW)*, 2005.

[5] A. Khoshgozaran and C. Shahabi, "Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy," in *Springer*, 2007.

[6] A. R. Beresford and F. Stajano, "Location privacy in pervasive computing," *IEEE Pervas. Comput*, 2003.

[7] M. Rahman, M. Mambo, A. Inomata, and E. Okamoto, "An anonymous on-demand position-based routing in mobile ad hoc networks," in *Proc. Int. Symp. Appl. Internet (SAINT)*, 2006.

[8] W. Wei, F. Xu, and Q. Li, "Mobishare: Flexible privacy-preserving location sharing in mobile online social networks," in *Proc. IEEE INFOCOM*, 2012.

[9] X. Chen, Z. Liu, and C. Jia, "Mobishare+: Security improved system for location sharing in mobile online social networks," *J. Internet Serv. Inf. Secur.*, 2014.

[10] J. Li, H. Yan, Z. Liu, X. Chen, X. Huang, and D. S. Wong, "Location-sharing systems with enhanced privacy in mobile online social networks," *IEEE Syst. J.*, 2017.

[11] X. Xiao, C. Chen, A. K. Sangaiah, G. Hu, R. Ye, and Y. Jiang, "Cenlocshare: A centralized privacy-preserving location sharing system for mobile online social networks," *Future Gener. Comput. Syst.*, 2018.

[12] M. Bhattacharya, S. Roy, K. Mistry, H. P. H. Shum, and S. Chattopadhyay, "A privacy-preserving efficient location-sharing scheme for mobile online social network applications," 2020.

[13] Y. Combinator, "Loopt," 2005.

[14] B. B. Nerd, "Y combinator's first batch (yc so5)," 2024.

[15] LoopTMix, "Loopt: The geo social network."

[16] WikiPedia, "Node.js."

[17] Nodejs, "What is the event loop?."

[18] T. F. Stack, "The unbelievable history of the express javascript framework," 2016.

[19] P. Newswire, "Ibm acquires strongloop to extend enterprise reach using ibm cloud," 2015.

[20] Expressjs, "Writing middleware for use in express apps."