Expertise
and insight
for the future

Sagar Aryal and Ishwor Kunwor

Rusher: Using Internet and Geolocation to Promote Tourism

Metropolia University of Applied Sciences

Bachelor of Engineering

Name of the Degree Programme

Bachelor's Thesis

20 May 2019

Author Title Number of Pages Date	Sagar Aryal and Ishwor Kunwor Rusher: Using Internet and Geolocation to Promote Tourism 42 pages + 0 appendices 20 May 2019
Degree	Bachelor of Engineering
Degree Programme	Degree Programme in Information Technology
Professional Major	Software Engineering
Instructors	Janne Salonen, Head of Department

This paper analyses many Geolocation based social media application and the recommendation models described in those applications. Furthermore, it studies the socio-economic impact of those applications to propose a new application called Rusher. The application described in this paper is built on JavaScript stack and uses MongoDB backend. Likewise, Google Maps Platform is used to serve location sensitive content to the user. The application allows the users to add custom locations, organize events and review the locations and events. Finally, this paper discusses the design and implementation of Rusher and provides basic usage instructions along with the scope for further development.

Keywords	Geolocation, tourism, JavaScript, Node.js, Express.js, MongoDB

Contents

List of Abbreviations

List of Abbreviations

DBMS	Database Management System
API	Application Programming Interface
GPS	Global Positioning System
LDA	Latent Dirichlet Allocation
ECMA	European Computer Manufacturers Association
ES6	ECMA Script 6
CSS	Cascading Style Sheets
DOM	Document Object Model
W3C	World Wide Web Consortium
HTML	Hyper Text Markup Language
WHATWG	Web Hypertext Application Technology Working Group
MVC	Model View Controller
HTTP	Hyper Text Transfer protocol
HTTPS	Hypertext Transfer Protocol Secure
ODM	Object Data Modeling
BSON	Binary JavaScript Object Notation
SQL	Standardized Query Language

ADE	API Development Environment
JSON	JavaScript Object Notation
UI	User Interface
UX	User Experience

1 Introduction

Geolocation has evolved to become a staple feature of modern applications ranging from location-based services like weather forecasting and travel routes to general purpose implementations like social media. A large number of people use applications based on geolocation every day to find nearby businesses and other places of interest. Evolution of these technologies has also led to a large library of places worldwide, which has been used by businesses and developers to promote their businesses and to provide travel recommendation services.

Google Maps platform is the most popular geolocation and mapping service used to develop modern location-based applications. It is a mapping and geolocation service developed by Google that serves geolocation, navigation and information requests over HTTP. It consists of three different APIs, namely Maps, Places and Routes, that work together to provide a complete and robust solutions to third-party application developers.[1] Therefore, large number of developers use Google Maps Platform as a de facto industry standard to develop thousands of applications and online services. However, the use of place information from Google Maps Platform alone to create recommendation services is not adequate. The platform has vast collection of data but the human tendency to care more about the opinions of the people they know, and trust often renders diminishes the meaning of the data. Furthermore, different implementations of geolocation APIs such as live location sharing have raised concerns regarding privacy and effectiveness of those services.

Therefore, this document studies various existing practices used in geolocating and location-based applications and analyses their efficiency and acceptability. In addition to that, the potential usage of those technologies in tourism industry and its consequences are also analyzed to propose a new application to enhance tourism using internet and geolocation technologies.

The application named Rusher described in this document collects user generated data about various locations, activities and specialties in a given area and populates them over google generated map. The user generated data can be reviewed and promoted. Furthermore, the locations can host events intended to draw larger audiences. The application is developed with the intention to discover and promote relatively unknown places of interest, their culture and traditions to a larger audience.

2 Background

Google Maps Platform was launched as Google Maps in 2005 after Google acquired Where 2 Technologies and its mapping software. In 2018, the endpoints of Google Maps API were organized into three different services, namely, Maps, Routes and Places and re branded as Google Maps Platform. [2] As Google's mapping services got better over time, the increase in availability of smartphones and that of mobile internet speeds made geolocation services more accessible to end users. This has led to an emergence of newer, precise and faster geoinformatics services and brought about a flurry of new implementations.

Google Maps Platform is not only a tool for everyday solutions like directions but also a great tool for discovery and advertisement. Applications built on the platform can exploit Google's robust mapping service to create targeted services. These services in turn can be used by end users to discover businesses and place of interest and by businesses to promote themselves to the audience. Furthermore, recommendation models based on people and their location preferences is a potential arena for the usage of Google Maps Platform.

Muir B.M [3] studied the development of reliable and credible applications that produce results that can be trusted by general population. He used the established models of trust between two humans and extended those models to analyze the trust ability of human-machine relationship and the factors affecting those relationships. Muir discovered human distrust of machines originates from their willingness to pay more attention to their peers' emotion and their subjective analyses and recommendations as opposed to objective analyses performed by algorithms. He recommended developers to design systems by addressing and calibrating algorithms to account for human fondness to subjective recommendations. [3] Most of the existing recommendation models and tourism applications, however, focus on gathering larger sets of data as opposed to providing their users with subjective information that may have more influence on the user and their choices.

Likewise, Flanagin A. and Metzger M. [4] analyzed the validity of crowd-sourced geographical data. They discovered crowdsourcing of geographical data made the systems more precise and helped in the pace of the evolution of geoinformatics systems. According to their analysis, the involvement of volunteered information directly contributed to the development of accurate maps and navigation systems. [4] Their analysis concludes that local users are accurate source of geological data. Furthermore, their conclusion validates Muir's discovery opens potential application areas for crowd-sourced and subjective geological data. Rusher's development team has taken these recommendations into account while developing recommendation models focused more on subjective user content.

Babu S.R and Subramoniam S [5] analyse the potential effects of internet and IoT technologies on different aspects of toursim industry supply chain. They suggest the tourism industry to adapt its structure and culture to keep pace with evolution of technology and reconsider their business practices to integrate more technology to it. Babu and Subramoniam describe the evolution and usage of social media applications and content sharing platforms and their impact on tourism industry.Furthermore, they account for the necessity to incorporate big data into the operations of tourism industry and the need to create newer analysis mechanisms to cope with higher volume of data. [5]Their overview of a tourism industry supply chain is displayed in the figure below.

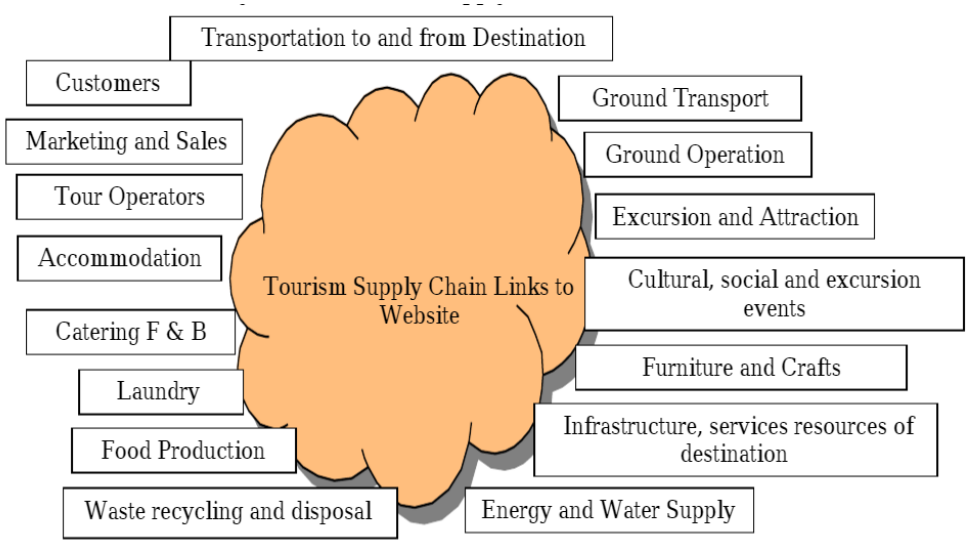

Figure 1. Babu and Subramoniam's overview of tourism supply chain [5]

The figure above describes the overview of a tourism supply chain and the interaction of various aspects of the supply chain. Babu and Subaramoniyam study the trends of tourism created by the development of social media and data analysis and recommend a new model for business information systems. The image below provides the overview of the BIS proposed by them.

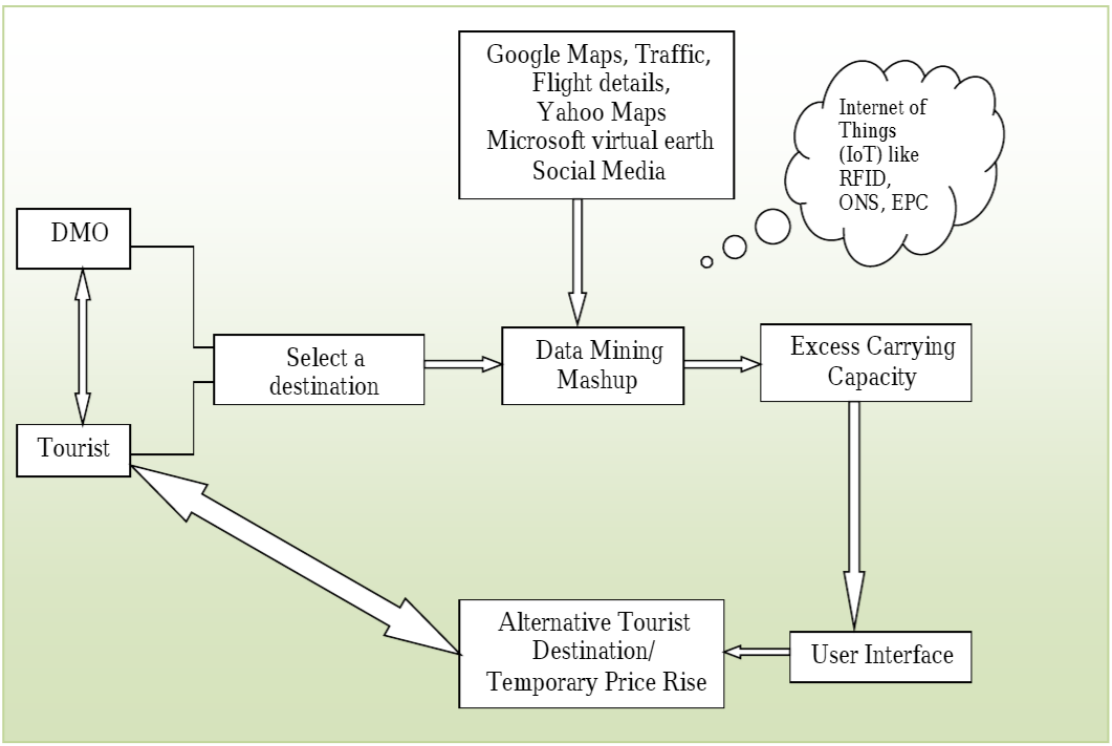

Figure 2. Babu and Subramoniam's BIS model [5]

In the proposed BIS model above, Babu S.R and Subramoniam propose solutions such as alternative destinations and pricing.They recommend tourism industry to incorporate IoT technologies in their daily operations such as ticketing. Furthermore, they advocate the development of comprehensive and personalized recommendation algorithms based on location history and preferences of the user and input from local stakeholders. Likewise, they have developed algorithms to create real time ranking of tourism spots based on potential social risks and pollution factors.[5]

Cacho A et. al. [6] detail the development and implementation of a mobile tourist guide called Find Natal. The application was developed with the co-operation with city of Natal, the capital and the largest city of Rio Grande do Norte state of Brazil. The application was developed to help increase the number of tourists visiting Natal and help the growing number of tourists navigate the city. Find Natal is a mobile tourist guide delivered as a smartphone application designed to make up for the shortage of skilled tourist guides originating from the poverty of the region. To enhance tourism and increase profitability, the city studies social trends and challenges and summarized them in the picture below.

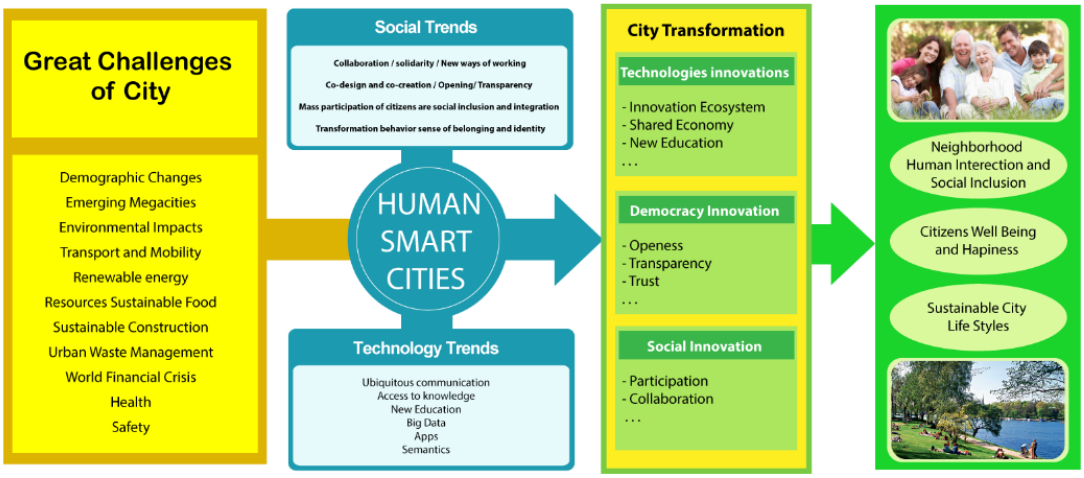

Figure 3. Study of social trends and challenges of Natal, Brazil

In the image above the social trends and challenges of the city of Natal are presented. To counter those issues, and help more people find jobs and improve their lifestyle via tourism, the city developed Find Natal Find Natal application was designed to be used by both the city administration and the visiting tourists. The city administration could use the application to monitor and validate crowd sourced data about tourism attractions in and near the city. Furthermore, they were allowed to add detailed information on the history and current state of the attractions including pictures, videos and related media content. In addition to that, the city officials could use Find Natal to monitor the number of tourists visiting a site and formulate policies based on total number of tourists. The tourists, on the other hand, used Find Natal as a user-friendly and responsive tourist guide eliminating the possible language barriers and guidance costs. The tourists could find information about the ways to get to places of their interest and learn more in relation

to the history, architecture and other vital information about the location. Find Natal application also had a sizable impact on the socioeconomic situation of the area. The application led to creation of more jobs and reduction in the rate of violent crimes in the area. The trial period also had positive impacts on public's participation in the design and implementation of city's policies.

Bao J. et.al [7] surveyed the recommendations used in location based social media applications to define a recommendation model based on user preference and location history. Their model aggregates the users' current and historic location to define user preference and suggests possible locations of interest based on reviews and recommendations collected from local experts and travelers. Their application intends to encourage users to travel larger distances from their home cities and visit newer locations. [7] The picture below presents an overview of Bao et al.'s concept of location based social media.

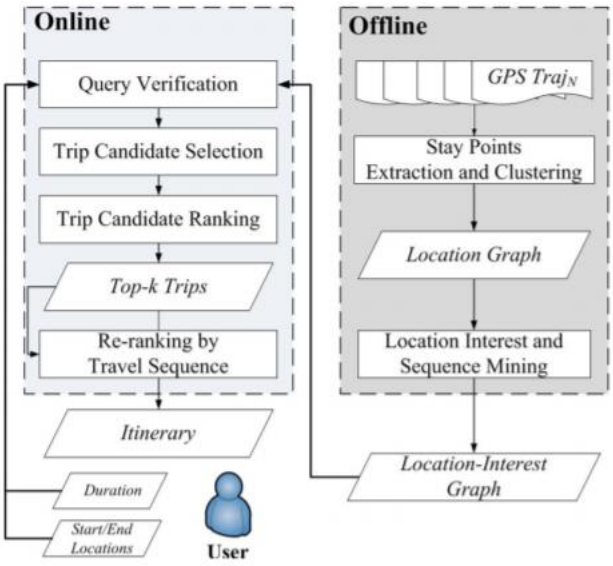

Figure 4. Bao et. al.'s concept of differentiation of recommendation services

The recommendations rely on the machine analysis of user's location graph and the clustering of their location points to create a location-interest graph. The application then creates a ranking of possible locations based on the graph and proposes a travel itinerary to the user. The recommendations made by this application are based on the objective analysis of subjective data gathered from local stakeholders. Therefore, this approach of

recommendation models somewhat considers Muir BM's [4] recommendation to increase human machine trust in modern applications and services.

Scallato et. al. [8] studied user activity of large social media applications with millions of active daily users. The authors used Double Pareto law to establish the relationship between the distribution and number of places they check in to. Likewise, they also studied the patterns of social media use and the number and distribution of friends of an average social media user. Likewise, the authors studied the accumulation of check-ins and other user activity over time in relation to the growth in the number of social media connections. Their report concludes the disparity in number and distribution of social media user's connection in those platforms comes from Geo-political divides, socioeconomic isolation or topographical constraints. The aforementioned constraints also impact the number of check-ins and distribution of new locations visited by the user. According to the author, the social media users tend to make online friends from different locations in an effort to escape their geo-political restrictions at no financial cost and gather information about the places they are interested in.

Chow et. al [9] describe the development a set of location-based services called GeoSocialDB. Their application offered three location based social media services. First of those services was location-based news- feed which queried the user to enter a distance and display social media posts from within entered distance. Second was location-based news ranking, which ranked the news feed based on user's location, their preferences and the entered distance. Finally, the third service was location-based recommendation, which used user's geographical preferences and their interests along with the entered distance to recommend places to the user. The basic overview of Chow et.al's proposed system is described in the image below.

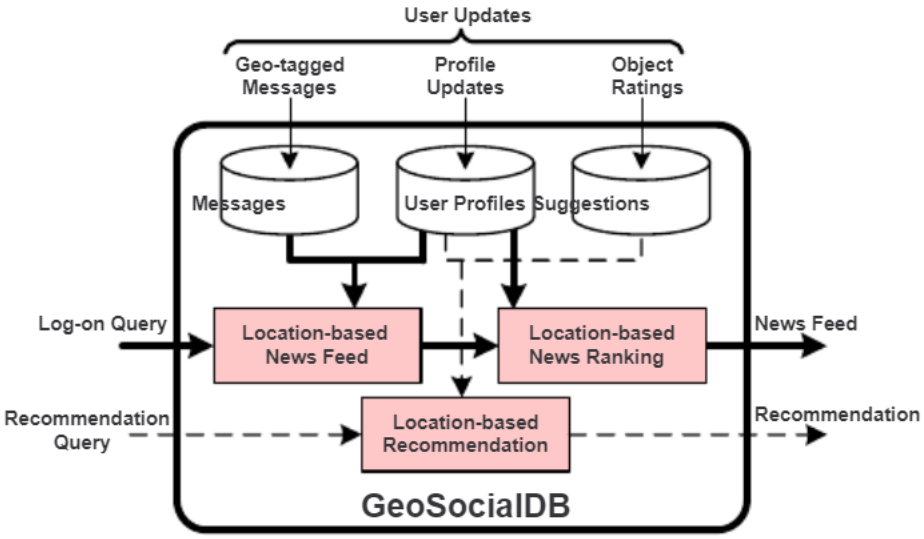

Figure 5. Chow et. al.'s model of GeoCocial DB [9]

The figure above described the basic operation of Chow et. al.'s geosocial database. It described the interconnection of recommendation and ranking queries and their implementation. Their implementation accepts three types of queries, namely, user-updates, log-on queries and recommendation queries. The user update queries are automatically tagged with the user's location and are used to store and access user's messages. Similarly, log-on queries are used to log in the user and store the user's current location and distance preferences. Finally, recommendation queries are used to generate recommendation based on the user's location, friend circle and common affinity among the friends. Chow.et.al also discuss the possible challenges in developing location-based services in their paper. The primary challenges discussed by them include continuous processing of user queries and the privacy concerns regarding user-based recommendations. [9]

Based on the review of relevant literature, we learnt about the effects of social media in the tourism supply chain and realized the need for the tourism supply chain to maximize the impact of social media to increase profitability. However, the usage of social media as a possible tool to escape geo-political constraints and the preference for subjective opinions of peers over objective analysis performed by recommendation algorithms makes the task of developing recommendation systems tougher. Likewise, the impact of tourism and a well-developed tourist guide application as trialed by Natal leads us to

conclude that tourism can have highly positive impact on socio-economic status of a location. Furthermore, the study of existing recommendation systems and their operations suggest that user's location history and opinions from peers and local experts were key to creating effective recommendation systems. Therefore, based on the study, we conclude that well made recommendation systems require the recommendations to be based on subjective reviews of peers and account for user's general preferences and history. Finally, based on studies in Natal, a location-based social networking and recommendation system focused on local communities and culture could have a positive impact on those communities and help increase the living standards there.

3 Application Objective and Target Audience

It is evident from the background section that many geolocations based social media and location-based recommendation algorithms have been planned. The applications such as Find Natal have been successful in improving tourism and local community at once because of the involvement of all related parties in different aspects of the applications implementation. From those observations, a conclusion can be drawn that tourism applications aided by internet technologies and human involvement can be a success both fiscally as well as socially. Therefore, based on those observations, an application called Rusher, based on those learnings is described in this document.

The primary objective of Rusher is to promote relatively unpopular places, events and traditions to an audience of local or distant tourists. The data about the location is generated from the user. This allows the users to promote their local places of interest, extreme traditions, different cuisines and festivals to a larger audience. The tourists on the other hand, can avoid the crowds in popular cities or events and experience much cozier and comfortable experience. They can also challenge themselves in local events or festivities. The location can benefit from the world of mouth advertisement from the visitors resulting in local businesses and an overall increase in quality of life.

The target audience of the application are travelers and tourism promoters in general. This application is developed as an alternative tourism app which focuses more on smaller communities and responsible tourism. Likewise, local government bodies such as cities and communities can also benefit from the usage of this application.

4 Core Technologies

This section provides insights into the techniques and technologies used during the development of Rusher.

4.1 Programming Languages

Project Rusher is built primarily as a web application. Therefore, high volume of its code is written in JavaScript. JavaScript is the most popular language for scripting web applications. It is a high level, weakly-typed and multi-paradigm programming language. It was designed by Brenden Eich and developed by the collaboration of Netscape, Mozilla and ECMA international. Project Rusher is written in ES6 (ES2015) standard. Project Rusher uses JavaScript to build a Node.js backend as well as in the frontend to serve dynamic and responsive web pages.

Likewise, CSS is the primary stylesheet language used in Project Rusher. Stylesheet languages are used to describe the presentation DOM elements and define how those elements are rendered in the browser window. CSS was developed by W3C. Project Rusher uses CSS to style and scale the web pages to deliver a responsive web application.

In addition to that, HTML is the primary markup language used as it is the de-facto standard for web development. Markup languages are used to define the logical structure and format of a web page. HTML was developed by W3C and WHATWG. Project Rusher uses HTML5 specifications to describe the web pages.

4.2 Libraries and Frameworks

This section provides brief description about different libraries and frameworks used during the development of Rusher.

4.2.1 Node.js

Node.js is an event-driven and asynchronous JavaScript runtime environment built on top of Chrome V8 engine. Node.js compiles JavaScript to machine code and therefore allows the developer to write back-end code in JavaScript. It is open-source and runs on all major platforms. The primary Node.js runs single-threaded, non-blocking I/O i.e. Node.js puts a request thread to sleep till the response is served and moves on to the next request. When the request is completed, the thread wakes up and delivers the response to the original process. This makes Node.js more memory efficient thus resulting is faster and easily scalable applications. [10]

In addition to the benefits mentioned above, using Node.js allows the developer to use JavaScript end-to-end across the application. Using single programming language across the application makes the code easier to read and maintain. In addition to that, it allows for tighter integration of web-server and server-side scripts. Likewise, Node.js is easier to set up and comes with various extensions that can be installed via npm or similar package managers. [10]

Rusher uses Node.js as the web-server because of the relative ease of installation and development. In addition to that, the availability of numerous modules to fulfil various requirements across the development aided in using Node.js technology in the backend.

4.2.2 Express.js

Express.js is a web application framework for Node.js. It is a free and open source framework written in JavaScript. Express.js is based on connect and http components of the Node.js environment, also called middleware. It is the de-facto standard to develop Node.js applications. Express provides a barebone MVC like structure to the application

and allows the developer to choose required libraries and middleware to suit their development needs. [11]

Express.js usually has an entry-file in which all the third-party dependencies, applications, database connections and route settings are defined. When the application is running, express constantly listens to the requests and directs them to appropriate routes as described in the route definitions. Likewise, it also performs various parsing and processing tasks with the aid of middleware used in the application. [11]

Rusher is built on top of scaffolding generated by express-generator. The directory structure of the application was also generated by the express-generator. In addition to that, Express.js is used to define the routes of the web-server.

4.2.3 Bootstrap

Bootstrap is a responsive, mobile-first front end component library. It consists of a responsive grid system, many prebuilt components and supports various plugins built in jQuery. Bootstrap is the world's most popular front-end component library. Rusher uses bootstrap as the primary front-end component library.

4.3 Node modules

This section provides the overview of Node.js modules and middleware used in the application.

1. https: The Node.js https module is used to serve the application over HTTPS as opposed to traditional HTTP. HTTPS was preferred to HTTP as HTTP traffic is not encrypted and thus vulnerable to different types of wiretapping attacks.

2. fs: fs is used to perform file operations. It is used to read SSL-keys while serving HTTPS application.

3. mongoose: mongoose is an ODM which provides schema-based solution to model application data. It includes built-in validation, type casting and query-building. In addition to that, mongoose supports middleware and schema-hierarchy.

4. http-errors: http-errors is a Node.js module that creates http-errors for failed requests.

5. path: path is a Node.js module that provides tools to work with file and directory paths.

6. cookie-parser: cookie-parser is a middleware that is used to populate request.cookies object with cookies. HTTP cookies are used to store user preferences and to perform user authentication.

7. body-parser: body-parser is a Node.js middleware used to parse request bodies before performing actions such as validation and processing.

8. morgan: morgan is a HTTP logger middleware. It is used to log HTTP requests and responses and primarily used as a debugging tool to check if HTTP requests are performing optimally.

9. multer: multer is a Node.js middleware used to process multipart/form data, which is used to upload files as body-parser alone can not handle multipart data. multer is used in Rusher primarily to upload files i.e. images, videos etc.

10. dotenv: dotenv is used to store environment variables in a separate .env file.

11. bcrypt: bcrypt is a Node.js module to perform bcrypt based password hashing. bcrypt is an adaptive cryptographic algorithm based in Blowfish cipher which slows down with iterations thus making it resistant to rainbow-table and brute force attacks even with powerful computers.

12. apidoc: apidoc is Node.js module used to document RESTful APIs. It allows the usage of pseudo-annotations inside the comment block of the API design and generates a static webpage containing API documentation based on those annotations.

4.4 Database

Rusher used MongoDB to store user and application data. MongoDB is a document-based database which stores data in BSON format. MongoDB is a NoSQL database that stores data in a document as opposed to SQL databases which store data in tables. SQL databases are more rigid compared to NoSQL databases as they can only store data as defined in the table schema whereas NoSQL databases are more flexible. Data in SQL databases are stored in rows and bound to the schema whereas NoSQL databases store data in key-value pairs and allow multiple types as well as arrays as data value.

MongoDB was chosen as the database for Rusher because of its flexibility. In addition to that MongoDB supports JavaScript like queries, therefore making it easier to integrate with a JavaScript based application. The database for the application is hosted in mlabs, a free to use MongoDB server.

4.5 Tools and version control

Rusher was built primarily in Visual Studio Code. It is a free text editor from Microsoft that supports code-highlighting and completion for multiple JavaScript libraries with the use of plugins, thus reducing errors and development time. In addition to that, it supports eslint, the primary linting tool used in the application to ensure code quality. Furthermore, it provides a built-in terminal that can be used to run, monitor and debug the application.

Likewise, the API of Rusher was developed using Postman. It is an ADE that reduces time required to develop and test APIs. Postman provides a comprehensive set of tools to interact with the API. It allows the developer to make secure requests to the API consisting authorization packets and a request body and monitor the response to verify the proper functioning of API endpoints.

Git was used for Version control in Rusher. It is the primary version control used in software development because of its ease to use. In addition to that, it's speed and integrity were also the factors behind the use of git. The git repository was hosted as a private repository in github and accessed by the team as required.

4.6 Google Maps Platform

Google Maps platform is a collection of mapping and geolocation APIs developed by Google. Rusher uses different APIs from the Google Maps platform and performs HTTPS request to the APIs to serve location-sensitive experience. The major APIs used are described in this section.

4.6.1 Maps JavaScript API

Maps JavaScript API is a part of Google Maps Platform that allows the developer to customize the map and the content within it and display the map in a browser or an app. It allows the modification of map layers to add custom markers and define events and services on top of the map layer. Rusher allows the users to create their own markers and add places of interest to the map.

4.6.2 Places API

Places library serves the information about establishments and places of interest that are already added to the Google database. This allows the developer to perform HTTP requests to the Places API service and serve the information about those places including details and pictures to the end user.

4.6.3 Direction API

Direction API calculates the directions from one location to another and serves it using a HTTP request. Furthermore, it includes multiple modes of transportation and considers possible transits and traffic.

4.6.4 Geocoding API

Geocoding is the process of converting addresses to geological coordinates i.e. latitude and longitude values. Rusher uses Geocoding API to convert the places added by users to latitude and longitude values and create a custom marker layer on top of the Map used in the application.

4.6.5 Geolocation API

Geolocation is the process of locating an end user or a device. Geolocation can be performed by using A-GPS in mobile devices or Google's location services database. Geolocation API is used by Rusher to center the Map and calculate directions.

5 Design and Implementation

This section provides the details about the development of the application including API and UI design and its implementation.

5.1 Data Modeling

The data used by Rusher is modeled with mongoose schema. This section provides insight into the data used by the application and the relationships between them.

5.1.1 Users

Users is a collection of documents based on User schema that consists the information about the users of the application. This collection stores vital user information such as personal information, password and history. Furthermore, data essential for a user to use the application such as their 'connections' and user generated content such as events and location reviews are also linked with the Users collection. The schema used for the users is presented below.

```
{
      username: {
          type:String,
          unique : true
      },
      firstName : String,
      lastName : String,
      email : String,
      password : String,
      address :{
          country : String,
          city : String,
          zip : String,
      },
      lastLogin : Date,
      connections : [
          {
              type : Schema.Types.ObjectId,
              ref : 'User'
          }
      ],
      events : [
          {
              type : Schema.Types.ObjectId,
              ref : 'Event'
          }
```

```
    ],
    rereviews : [
        {
            type : Schema.Types.ObjectId,
            ref : 'Location'
        }
    ],
};
```

Listing 1. User schema

The schema above describes an individual document of the Users collection. The keys in the JSON schema above describe the variable name and the values define the type of data accepted. The keys connections, events and reviews hold an array that stores references to Users, Events and Locations Schemas respectively.

5.1.2 Locations

Locations collection consists the data about user defined locations. It stores information about user-added locations and data associated with it. The schema below describes the model for a location document.

```
{
    name : String,
    details : String,
    coordinates : {
        lat : Number,
        lng : Number,
    },
    cover : String,
    media : {
        photos : [
            {
                type : String
            }
        ],
        videos : [
            {
                type : String
            }
        ],
        audios : [
            {
                type : String
            }
        ]
    },
reviews : [
    {
        type : Schema.Types.ObjectId,
        ref : 'Review',
    }
],
totalReviews : {
```

```
    type:Number,
    default : 0
},
stars : Number,
added : {
        type : Schema.Types.ObjectId,
        ref : 'User',
    }
,
}
```

Listing 2. Location Schema

The above schema defines the model for location data. The field review holds the array of the IDs of the reviews associated with the provided location. This can be used to perform further queries and get all associated reviews and reviewers as required. Likewise, totalReviews is increased each time a location is reviewed, and stars stores the aggregate stars provided by the users to the given location. Similarly, the media field holds the path to different media files, i.e. photos, videos and audio associated with the location and uploaded by the user.

5.1.3 Events

Events collection consists of the records of events organized by users. The events are associated with a location and a user. The schema for the Events model is presented in the code listing below.

```
{
    location: {
        type : Schema.Types.ObjectId,
        ref : 'Location'
    },
    organizer :{
        type : Schema.Types.ObjectId,
        ref : 'User'
    },
    name : String,
    description : String,
    date : Date,
    duration : String,
    invited : {
        going : [
            {
                type : Schema.Types.ObjectId,
                ref : 'User'
            }
        ],
        notGoing : [
            {
                type : Schema.Types.ObjectId,
```

```
                ref : 'User'
            }
        ],
        noRSVP : [
            {
                type : Schema.Types.ObjectId,
                ref : 'User'
            }
        ],
    }

}
```

Listing 3. Events Schema

In the schema presented above, location holds a reference to the event's location and field organizer of the application. The invited field consists of three arrays; going, not-Going and noRSVP which hold the users invited by the organizer based on their RSVP to that event invite.

5.1.4 Reviews

Reviews collection consists of reviews of locations by the application users. The schema for a review is presented in the listing below.

```
{
    location : {
       type : Schema.Types.ObjectId,
        ref : 'Location'
    },
    date : {
        type: Date,
        default : Date.now()
    },
    reviewer : {
        type : Schema.Types.ObjectId,
        ref : 'User'
    },
    title : String,
    review : String,
    media : {
        photos : [
            {
                type : String
            }
        ],
        videos : [
            {
                type : String
            }
        ],
        audios : [
            {
```

```
                type : String
            }
        ],
        stars: Number
    },
}
```

Listing 4. Reviews schema

The schema presented above describes a review. Each document in the Reviews collection consists of a reference to the reviewed location and the reviewer. Furthermore, the date of the review, ratings in terms of stars and path to related media files if any are stored in this document.

5.2 API Design

The API for Rusher is designed using Express.js routing and tested using Postman ADE. This section provides brief descriptions of the different endpoints, the requests they support and their expected responses. The detailed version of the API documentation can be found at **https://sagarary.github.io/apidoc/index.html**.

5.2.1 API root (/)

The API root (/) endpoint serves as the entry point to the application. This endpoint simply renders the homepage of the application if the user's session is active and redirects to the /login endpoint if the user is not logged in.

5.2.2 Login Endpoint (/login)

Login endpoint is responsible for checking user credentials and performing login. This endpoint accepts HTTP POST requests and checks the validity of the credentials provided in the request body before allowing access to the user of redirecting back to the login page. The login endpoint is described in the listing below.

```
router.post('/', (req, res) => {
  User.findOne({
    'username' : req.body.username
  }).then(function(user, err) {
```

```
  if (!user){
    res.send("no user found")
  } else{
    const allow = bcrypt.compareSync(req.body.password, user.password)
    allow ? res.redirect('/') : res.redirect('/login')
  }
})
})
```

Listing 5. Express router for handling login requests

The listing above is an express router to handle login requests made to the /login endpoint. When a HTTP POST request is sent to the endpoint, it first checks if the username exists in the User collection of the MongoDB database used by the application. If the username does not exist within the database, a response message is sent back to the end user. If the username exists, the entered password is compared to the bcrypt hash in the database and the user is allowed to move forward to the application in case of a match. However, if the hash does not match the provided password, the user is redirected back to the login page with an error message.

5.2.3 Users Endpoint (/users)

The users endpoint (/users) serves requests related to end users. The queries to this endpoint serve the results of database queries to the Users collection. The possible queries for this endpoint and the expected responses are listed below:

1. /users

   /users accepts HTTP POST requests. Queries made to this endpoint create a new User document in the Users collection with values corresponding to those sent in the request body. The generation of bcrypt hash for the password takes place at this endpoint before the database is queried to add a new user document.

2. /users/userid/:id

   This endpoint accepts HTTP POST, GET and DELETE requests. The parameters and expected responses for each request is described below.

   HTTP POST: HTTP POST requests made to /users/userid/:id accepts the users' details as parameters. This query results in a findByIdAndUpdate() query to the MongoDB da-tabase. This request updates the user information of the user

represented by the id in the API parameter. For example, a HTTP POST query to /users/userid/1 updates the user with user id 1 with changes coming from request body.

HTTP GET: HTTP GET requests made to /users/userid/:id returns the details of user with the provided id. This request results in a findOne() query to the MongoDB database. For example, GET request sent to /users/userid/1 responds with the data of user with user id 1. This request is used during login to find a user and check their credentials.

HTTP DELETE: HTTP DELETE requests made to this /users/userid/:id results in dele-tion of the user with given id. This request results in findOneAndDelete() query to the MongoDB database which deletes the records for given user. This request is made when the user wants to delete their account.

3. /users/username/:name

   The endpoint /users/username/:name accepts HTTP GET requests. The GET requests made to this endpoint trigger a find() query to the MongoDB database which responds with all users with matching names. The corresponding query is described in the listing below.

```
router.get('/user/username/:name', (req, res) => {
   const reg = new RegExp(req.params.name, 'i');
   User.find({
       '$or': [{
                'username': reg
            },
            {
                firstName: reg
            },
            {
                lastName: reg
            }
       ]
   }).then((err, users) => {
       err ? res.send(err) : res.send(users);
   })
```

Listing 6. Express router to find users by name

The Express router above queries the MongoDB database to find the user with the query string in either their firstname, lastname or the username. The API then responds with the output of the find() query.

5.2.4 Location Endpoint (/locations)

The location endpoint serves requests related to user defined locations stored in Rusher database. The requests to this endpoint serve responses by querying Locations collection of the database. The accepted requests and expected results are listed below:

1. /locations
   This endpoint accepts HTTP GET and HTTP POSTS requests. The corresponding queries and responses of each of those requests are mentioned below.

   HTTP GET: HTTP GET requests made to /locations responds with all locations added by all the users. This request performs find() query to the Locations collection of the MongoDB database. This request is used to generate map markers and populate the front page of the application with markers and location details.

   HTTP POST: HTTP requests made to /location adds a new location with details from the request body. This request performs create() query to the Locations collection of the MongoDB database. This request is used when an end user tries to add a new location or place of interest to the Rusher application.

2. /locations/locationid/:id
   The /locations/location/:id accepts HTTP GET, POST and DELETE requests with a parameter called id. The possible requests, their corresponding database queries and expected responses are listed below.

HTTP GET: HTTP GET requests to /locations/locationid/:id responds with the details of location with provided id. This request corresponds to findById() query to Locations collection that responds with single document. This request is used to search among locations.

HTTP POST: HTTP POST requests to /locations/location/:id updates the document corresponding to the provided id based on request body. This request corresponds to findOneAndUpdate() query made to the Locations collections with the request body. This request is used to update the details of user defined locations featured in the Rusher application.

HTTP DELETE: HTTP DELETE requests to /locations/locations/:id delete the location with given id. This request triggers findOneAndDelete() query to the MongoDB collection Locations. This request is made to delete a location from the Rusher application. Only the user who originally added the location is allowed to make this request and delete the location.

3. /locations/locationname/:name
   The locations/locationname/:name accepts HTTP GET requests with a name parameter. This request returns the list of locations whose name contains the query parameter. GET request to this endpoint triggers find() query to Locations collection of the MongoDB database. This request is used to search for locations.

5.2.5 Event Endpoint (/events)

The event endpoint (/events) serves HTTP requests related to user added events. The requests to this endpoint trigger queries to Events collection of the database. The possible requests, their corresponding queries and responses are listed below.

1. /events
   The /events endpoint accepts HTTP POST requests. HTTP POST requests to /events endpoint create new events with data acquired from the body of the POST request. The request triggers a create() query to the Events collection of

the MongoDB database. This request is performed when a logged in user creates a new event in a selected location.

2. /events/eventid/:id
   The events/eventid/:id endpoint responds to HTTP GET, HTTP POST and HTTP DELETE requests and accepts a parameter called id. These requests are described below with their corresponding queries and expected responses.

   HTTP GET: HTTP GET requests to /events/eventid/:id responds with a event with id given in the request parameter. This request initiates a findOne() query to the Events collection which responds with a event document with the provided id.

   HTTP POST: HTTP POST requests to /events/eventid/:id endpoint updates the event with the given id. This request triggers a findOneAndUpdate() query on the Events collection, which finds and updates the document with given id based on the information provided in the request body. This request is used to update an event. However, it should be noted that this request can only be performed by the user who created the event or has admin rights to the event.

   HTTP DELETE: HTTP DELETE requests to /events/eventid/:id endpoint cause the deletion of event that has the id passed in the request parameter. This request queries the Events collection with findOneAndDelete() query. This request is used to delete events; however, the events can only be deleted by the user who created it.

3. /events/organizer/:id
   The /events/organizer/:id responds to HTTP GET requests and accepts a parameter named id. This request returns the list of public events organized by the user with the id which matches the request parameter. It triggers find() query with the passed username. The sample query is described in the listing below.

```
Event.find({
   'organiser': req.params.id
 }, (err, events) => {
   err ? res.send(err) : res.send(events);
})
```

Listing 7. Query to find an event by organizer

The query above responds with the list of all the events organized by the user with requested id. This request is used to search for events and to display events in the users' event pages.

4. /events/location/:id

   The /events/location/:id endpoint responds to HTTP GET requests and serves the list of events in location with id provided in the request parameter. This request triggers a find() query with location id from the request parameter. This request is used to populate location page with the events and to search events in a selected location.

5. /events/name/:name

   The /events/name/:name endpoint responds to HTTP GET requests. This request triggers find() query on the Events collection and responds with the Events that contain the provided string parameter in the event name. This request is used to search events by name.

5.2.6 Reviews endpoint (/reviews)

The request endpoint (/request) serves HTTP requests related to user reviews to different locations. The requests to this endpoint trigger queries to Reviews collection of the database. The possible requests, their corresponding queries and responses are listed below.

1. /reviews

   The /reviews endpoint serves HTTP POST requests. This request results in two different queries to Locations and Reviews collections. The queries performed by this request are described in the listing below.

```
Location.findByIdAndUpdate({
   _id: req.body.location
      }, {
           $inc: {
           totalReviews: +1
       }
      }).then(() => {
   Review.create(req.body).then((err, review) => {
       res.redirect('back');
   })
})
```

Listing 8. Queries triggered by HTTP POST request to /reviews

The first query is made to Locations collection to find the selected location and increase its review count by one. After the completion of the query, another query is made to the Reviews collection to create a review. This HTTP request is made when a user reviews a location.

2. /reviews/reviewid/:id

   The /reviews/reviewid/:id endpoint responds to HTTP GET, HTTP POST and HTTP DELETE requests. These requests are described below with their corresponding queries and expected responses.

   HTTP GET: HTTP GET requests to this endpoint trigger a find() query to the Reviews collection. The request serves a review with the id passed in the request parameter. This query is used to focus on a single review for possible edit options.

   HTTP POST: HTTP POST requests to /requests/requested/:id endpoint initiates a findOneAndUpdate() query to the Review collection. This request is used to update a review and can only be performed by the user who originally wrote the review.

   HTTP DELETE: HTTP DELETE requests to this endpoint is used to delete a review. It triggers two different queries; first to Locations collection to decrease the number of reviews and then to Reviews collection to delete the review. This request can only be performed by the user who initially contributed the review.

5.3 UI/UX Design

The UI/UX design of Rusher follows mobile first design principles to serve fluid and responsive user interface to the user. Bootstrap and jQuery are used in the application to achieve this design principle. Furthermore, the design is minimalistic and focuses on providing immersive map experience to further highlight its location sensitive capabilities. In addition to that, the application uses a map that is always centered on the users' current location to provide an overview into location and events nearby. Likewise, the markers are customized so that they do not congest the map window and obscure the location awareness of the application.

6 User Guide

This section provides brief guidelines on how to use the application to perform most basic features. It must be noted that geolocation and https traffic must be allowed to access the application.

6.1 Home page

The home page of Rusher is a simple web page with just the name of the application and the Login menu. The home page is shown in image 1 below.

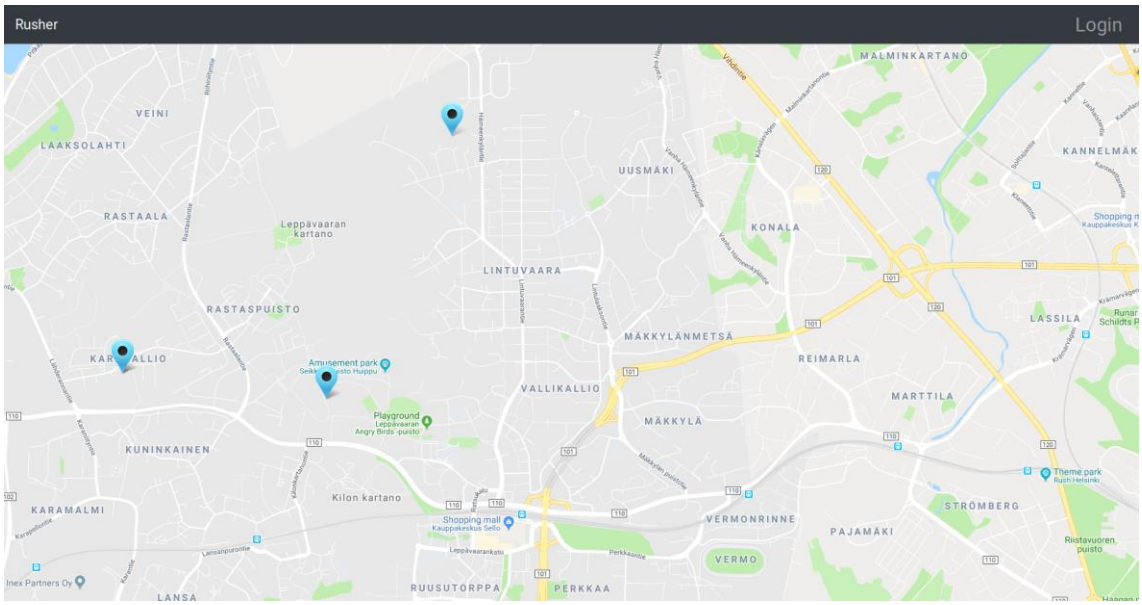

Image 1. The homepage of Rusher

The homepage is a minimal page, that provides an overview of nearby locations and the option to login. The user must click the Login menu to access the login and register pages to login or register to use the application.

6.2 Signup and Login

A user must authenticate or register themselves in order to use the application. This can be performed by navigating to the login page by clicking the login menu in the front page. That results in signup/login page as displayed in the image below.

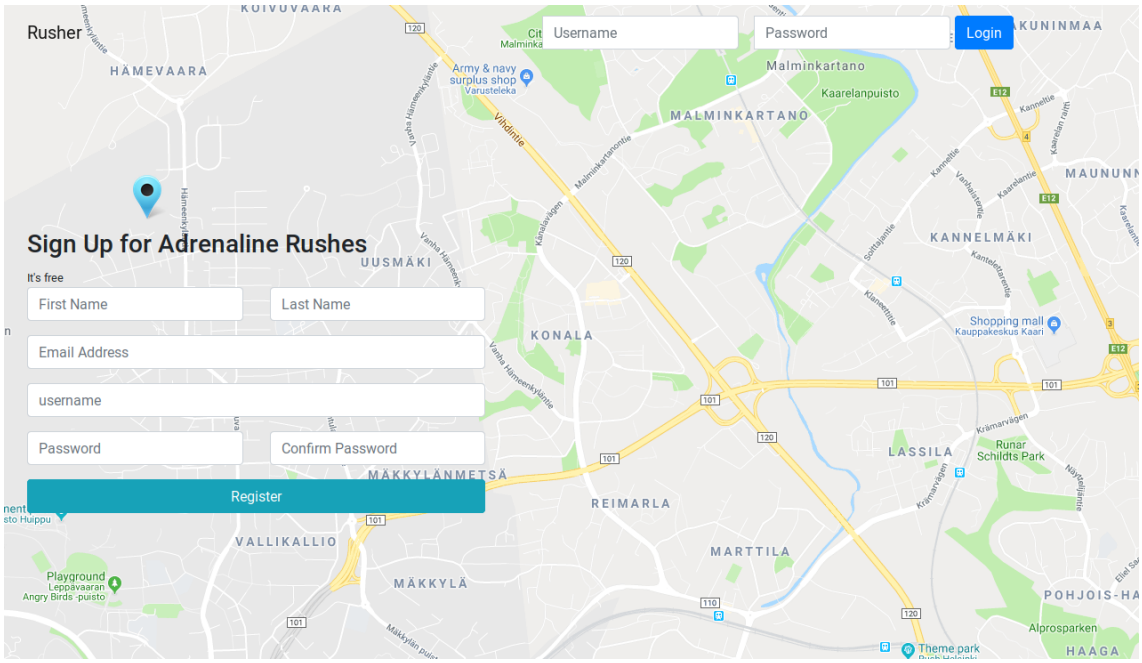

Image 2. Signup and Login page

The login page has forms for both sign in and login. If a user is registered and has the credentials required, the user can simply fill the login form and proceed to use the application. However, if the user is not registered and needs to create a new account, the user must fill up the signup form. The signup form collects necessary data such as user details and contacts. Once the user registers or signs in. the user is redirected to the main application page.

6.3 Main Page

The user can get to the main page by registering as a new user or logging in with their credentials. The resulting page is displayed in the image below.

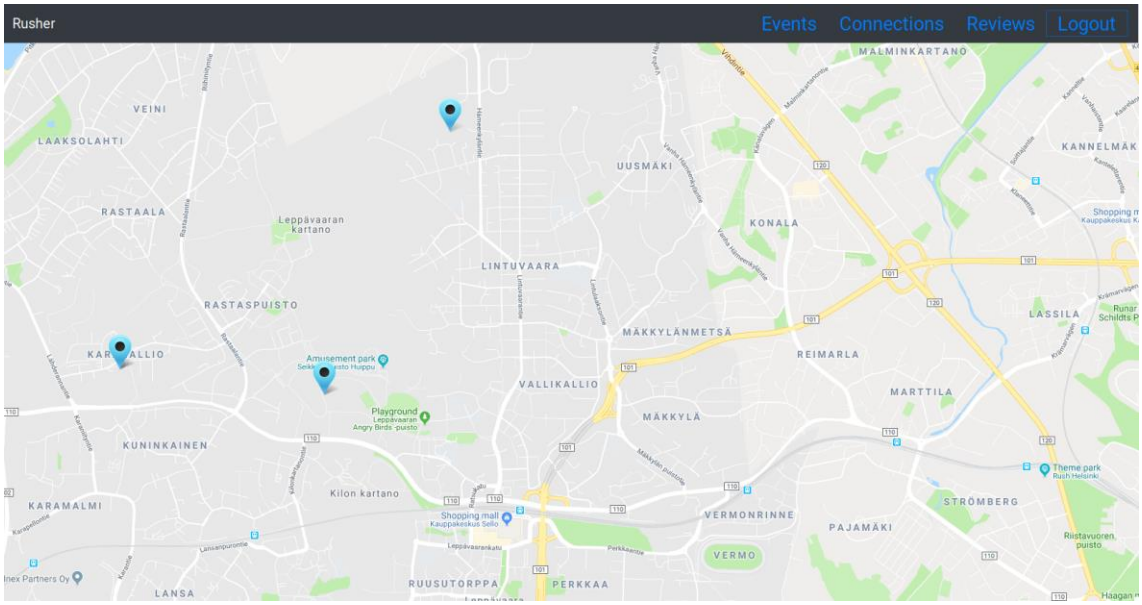

Image 3. Main page after logging in

The main page has a navigation bar that lets the users access the application features. The notifications for different factions of the application. i.e. events, connections and reviews are separated for clarity purposes. The events notifications include notifications for events created by the user and the events that are being attended by the user. Likewise, the notifications regarding connection requests and responses are grouped in connections. Furthermore, reviews groups the notifications for reviews of places added by the active user. Lastly, the logout menu is used to destroy the current session and log the user out of the application. This is the main page of the application and all the major features are accessed through this page.

6.4 Adding Locations

The locations can be added by an authenticated user. To add a new location, the user can simply click on the map and a new popup as displayed in image 4 will appear.

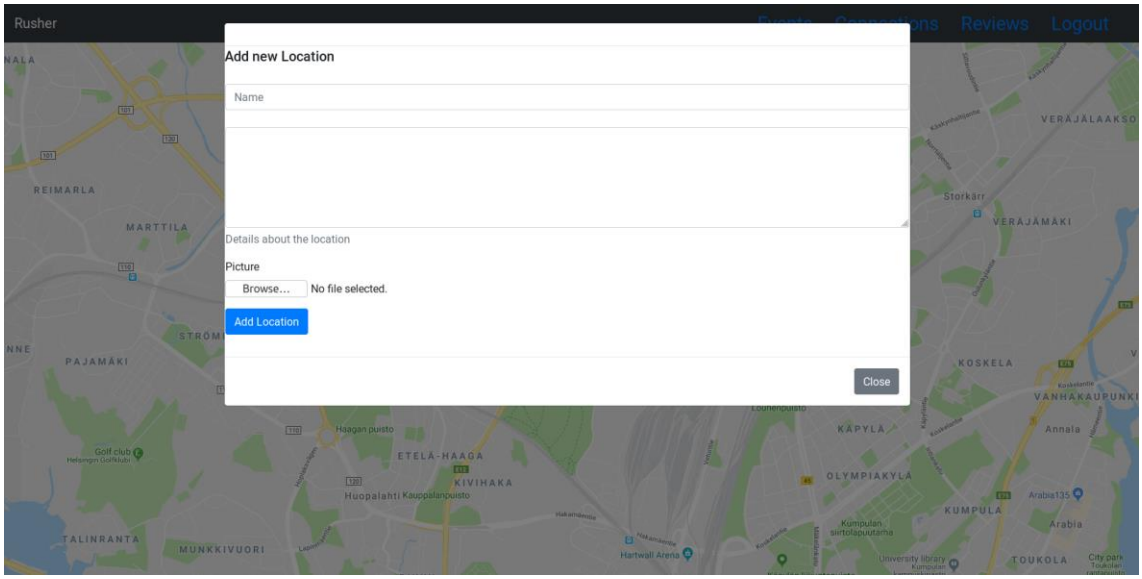

Image 4. Adding a new location

The image above displays the modal to add a new location. The user can fill the resulting form with the details and description of the location along with the related media files and submit the form using Add Location button to add the location. As soon as the location is added, a custom marker appears in the location, representing the addition of location to the MongoDB database.

6.5 Viewing Location Details

Location details can be viewed from the main application page. To view the details of a location, the user can click on the corresponding map marker, and a card popup displays the details of the location. The resulting interface is displayed in image 5.

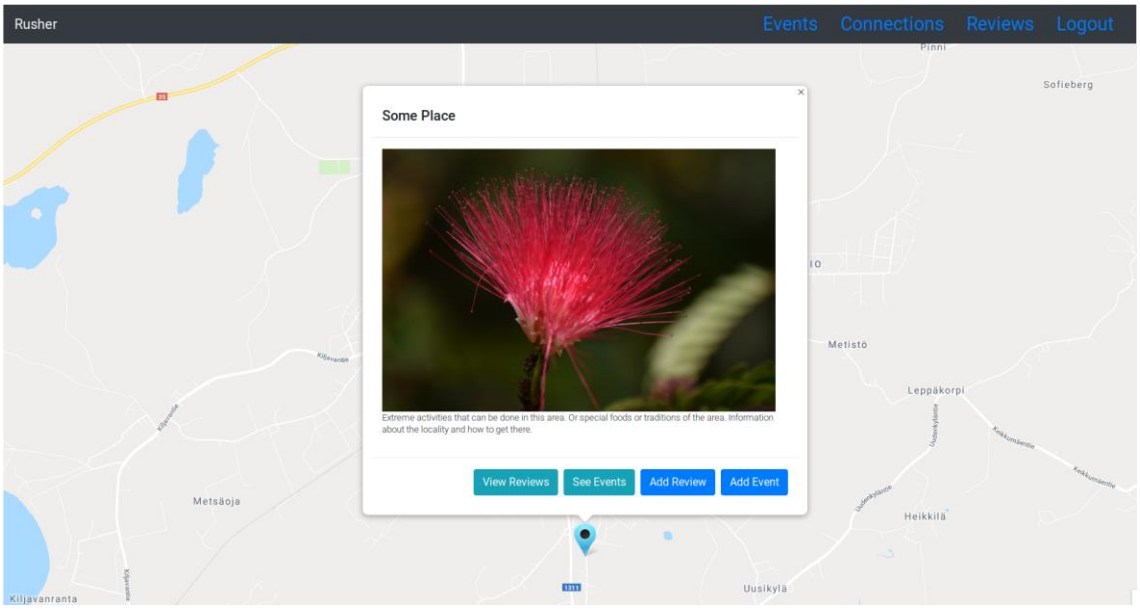

Image 5. Viewing a location

The image above displays the card created by clicking on desired map marker. The card reveals the name of the location, the picture and other media if any, a description added by the location creator and options to add and view reviews and events. If the user wants to view the reviews or the events in the location, they can click on respective buttons and access the service.

6.6 Add Events and Invites

Events are one of the main parts of the application. An authenticated user can create an event by clicking on the location marker and clicking on the add event button of the resulting card. A form as described in image 5 must be filled to create a new event and invite other users.

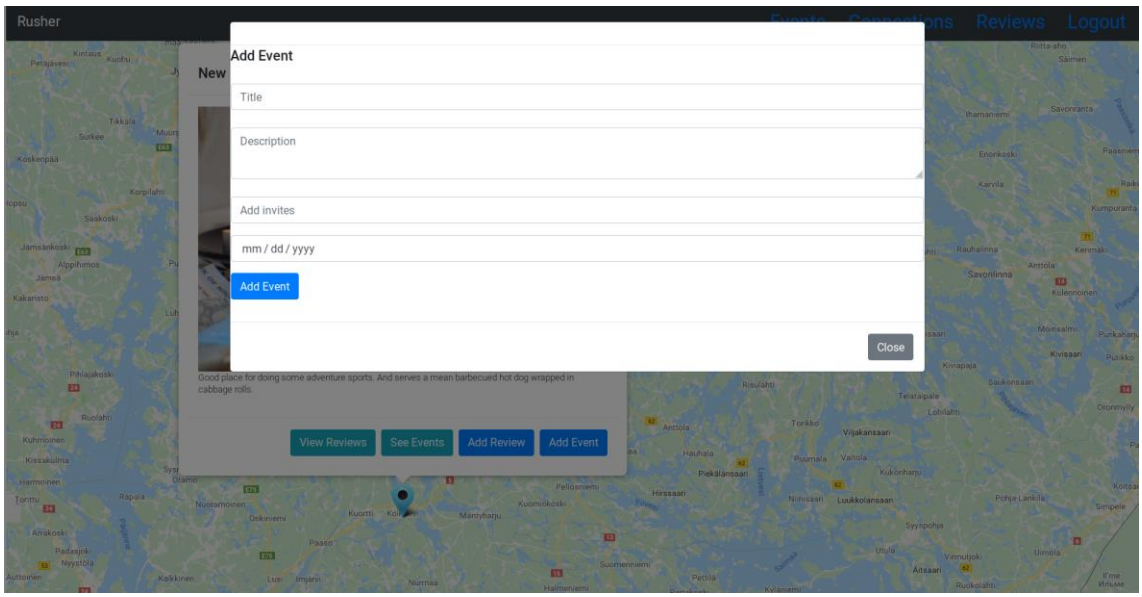

Image 6. Add events and invite

The image above shows the interface to add a new event. The user must fill up the displayed form with event details and select a date to create an event. Furthermore, in the Add invites field, the user can search by name or username to add invites to the users connected with the event creator. Once the event is created and saved to the database, an event notification is sent to the respective users and their RSVP is tallied to the database as well as notified to the event organizer.

6.7 Adding Reviews

Reviews are one of the most important aspects of location-based applications. The reviews in Rusher can be added by authenticated users. The interface for adding reviews is displayed below.

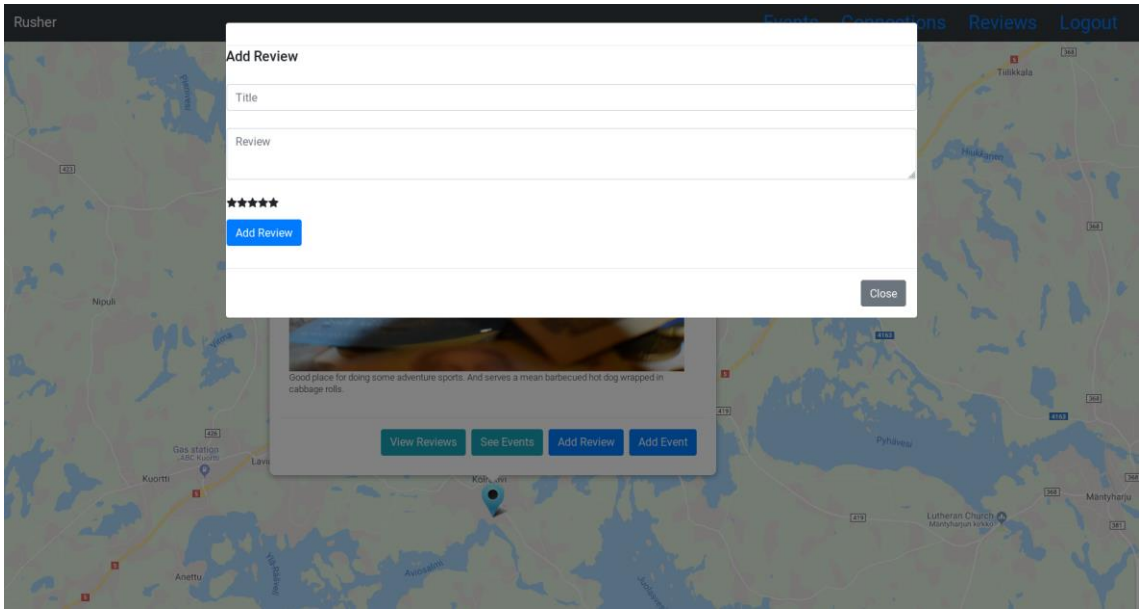

Image 7. Reviewing a location

The image above represents the interface to add a review. This interface can be accessed by clicking on the add review button in the location card. The user can then provide their review and their ratings to add a new review. The review is saved to Reviews collection of the database and a link to the location is also stored.

6.8 User Profile and Settings

The user profile and settings allow the user to change their preferences and settings. The interface to modify profile settings in described in the image below.

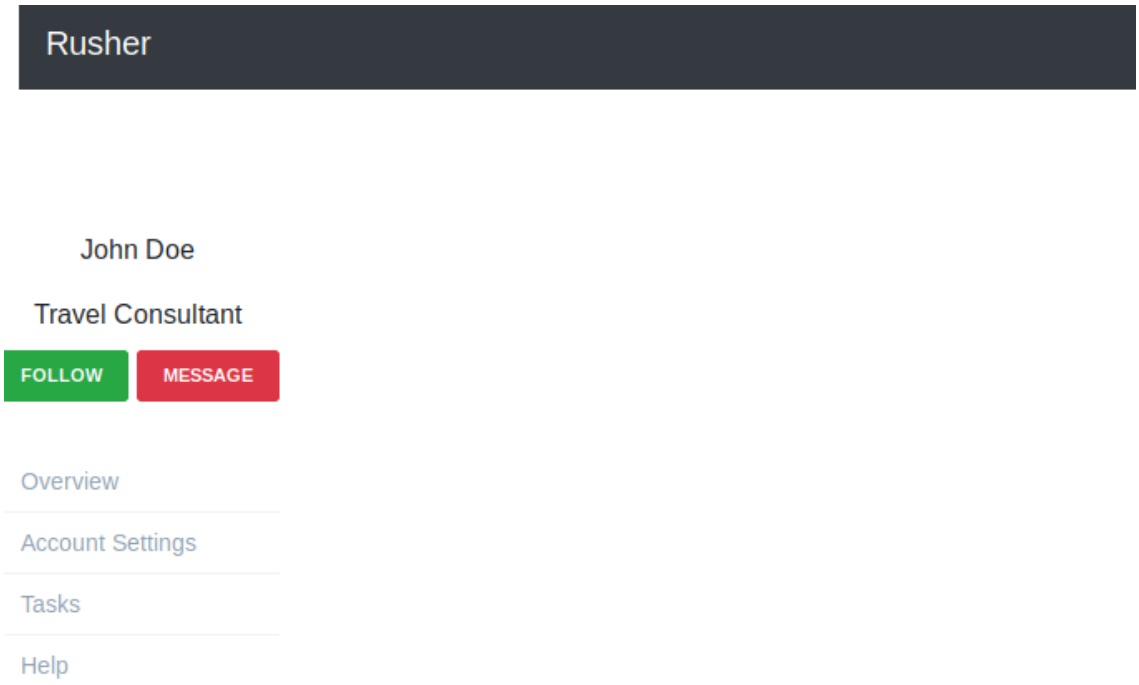

Image 8. User profile page

The above image shows the list of options available to the user to change their preferences and settings. The overview provides the overview of users' activity in the application, including alerts for locations and events they have added. Likewise, Account Settings allows the user to make changes to their account information, such as username and password. In addition to that, Tasks provide the overview of Events and provides a minimal event planning and task management interface. Furthermore, Help section offers a list of FAQs and their respective solutions.

7 Testing and Results

Rusher is currently in beta and is deployed in a development server and therefore not accessible to the general public. The API of the application is tested by using different HTTP requests via the Postman ADE. The application and its user interface however have been tested only by a limited number of developers and beta testers. The results from the beta testing and concept presentations have been encouraging so far. However, the concerns regarding privacy and data security were raised.

8 Proposed Developments

Rusher is still in beta and new features are being added to it along with improving the existing features of the application. The primary target for future developments is to improve the user experience by simplifying the user interface even further. Furthermore, more options to interact with the user and improvement of location data are planned for the next stage of development. Likewise, mobile application versions of the application are planned when the beta testing is completed.

In addition to that, there are plans to integrate AI into the application to create recommendation models. The recommendation models will be based on location and event history. This recommendation system will suggest user's new activity and/or location and will rely on the public location history of the user and their connections and average location ratings. Furthermore, the integrity and security of user data and content is another priority for future developments.

9 Conclusion

In conclusion, tourism is an ever-growing industry with the potential to bring positive changes in the quality of living of a community. However, with the rise of social media and increasing use of technology in day to day activities, tourism industry needs to study the changes in travel patterns as a direct result of social media influence and adapt to those changes. Furthermore, use of technology alone cannot create the most effective recommendation system as word of mouth is still the most efficient mean of advertisement. Hence, to create effective location based social media platform for tourism promotion human opinion should be integrated with behavioral analysis and study of travel to create personalized recommendation systems.

Therefore, based on the above learnings, a new application called Rusher was developed. The application utilized modern internet technologies to create a crowd sourced and location-based application to promote tourism in smaller communities and lesser known places. The application us targeted to local and foreign tourists searching for lesser crowded and less travelled destinations. The application lets authenticated users add locations and events and to post reviews of said locations. The user added locations are represented by custom markers on top of Google Maps. The primary objective of this application is to create a platform for travel and culture enthusiasts find new and uncommon locations in an effort to help local communities grow.

The application Rusher was developed by using Node.js, Express.JS and a number of Node.js middleware in the backend. The database was designed using Mongoose.js schemas and hosted in MongoDB database servers. Likewise, the front end of the application was designed using pug and styled with Bootstrap and jQuery.

This document provides the developers basic information about geolocation-based applications and the process involved in developing those applications. In addition to that, it also describes the potential applications of geolocation APIs and recommendation models.

References

1 Svennerberg G. Beginning Google Maps API 3. Berkeley, CA: Apress; 2010.

2 1. Google Maps Platform Official Documentation [Internet]. Google Developers. 2019 [cited 15 April 2019]. Available from: https://developers.google.com/maps/documentation/

3 Muir BM. Trust between humans and machines, and the design of decision aids. International Journal of Man-Machine Studies. 1987;27(5-6):527–39.

4 Flanagin AJ, Metzger MJ. The credibility of volunteered geographic information. GeoJournal. 2008;72(3-4):137–48.

5 Babu SR, Subramoniam S. Tourism Management in Internet of Things Era. Journal of Information Technology and Economic Development. 2016;(7-1):1-14

6 Cacho A et al. Mobile tourist guide supporting a smart city initiative: a Brazilian case study. International Journal of Tourism Cities; 2016;(2-2):164-183

7 Bao J. et. al. Recommendations in location-based social networks: a survey. GeoInformatica. 2015Jul;19(3):525–65.

8 Scellato S, Mascolo C. Measuring user activity on an online location-based social network. 2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). 2011.

9 Chow C, Bao J, Mokbel M. Towards location-based social networking services. Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks - LBSN '10. 2010.

10 Herron D. Node. Js Web Development. 4th edition. Birmingham: PACKT Publishing; 2018: 7-24

11 Mardanov A. Express.js guide. 1st edition. Vancouver: Lean Publishing; 2013: 2-4

12 Giamas A. Mastering MongoDB 3.x. 1st edition. Birmingham: Packt Publishing; 2017:11-20