D2: Software Design Document

Group 1: InTheLoop

Kevin Mai, Austin Black, Rashaad Jones, Clarke Le, Joshua Good, Joseph Milton

Project Overview:

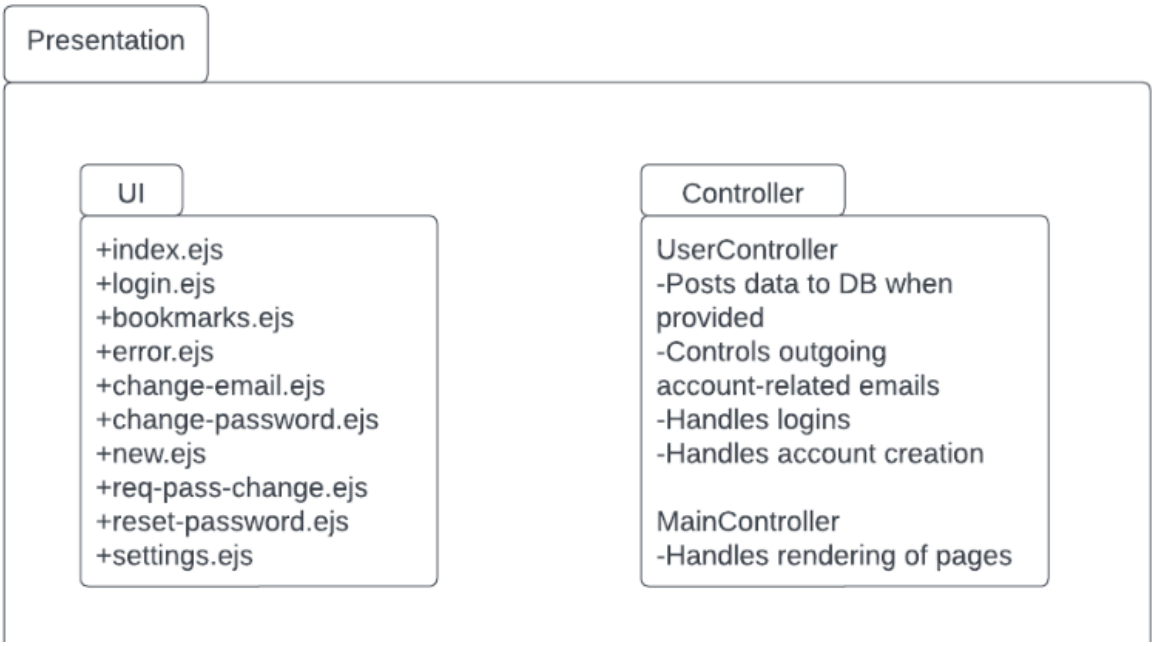
In The Loop, our program is built to find activities and events around the user or an otherwise defined area. This can be used by residents of the area and visitors who may be looking for things that align with the user's interests.

The project considers user data such as likes, dislikes, and locations to provide the most valuable and relevant information. Users can share discovered events with others, which allows for interaction between the users. This can also allow less technologically adept users to still participate by receiving events from others. However, the program is built to be simple for the user, so this should not be an issue in most cases.

Architectural Overview:

Our core architecture is built with ejs files, css, javascript, and integration with a MongoDB Atlas database to store user account information. We initially looked at using HTML, although we found ejs was a better alternative for our purposes, allowing for easier integration with Node.js and more dynamic HTML.

Subsystem Architecture:



Pages

Index
-Requests location data
-Displays location (city/state)
-Allows addition of likes/dislikes
-Performs searches

Change-Email
-Handles user request to change email info

Change-Password
-Handles user request to change password info

Login
-Handles login process for users

New
Allows for the creation of a new user

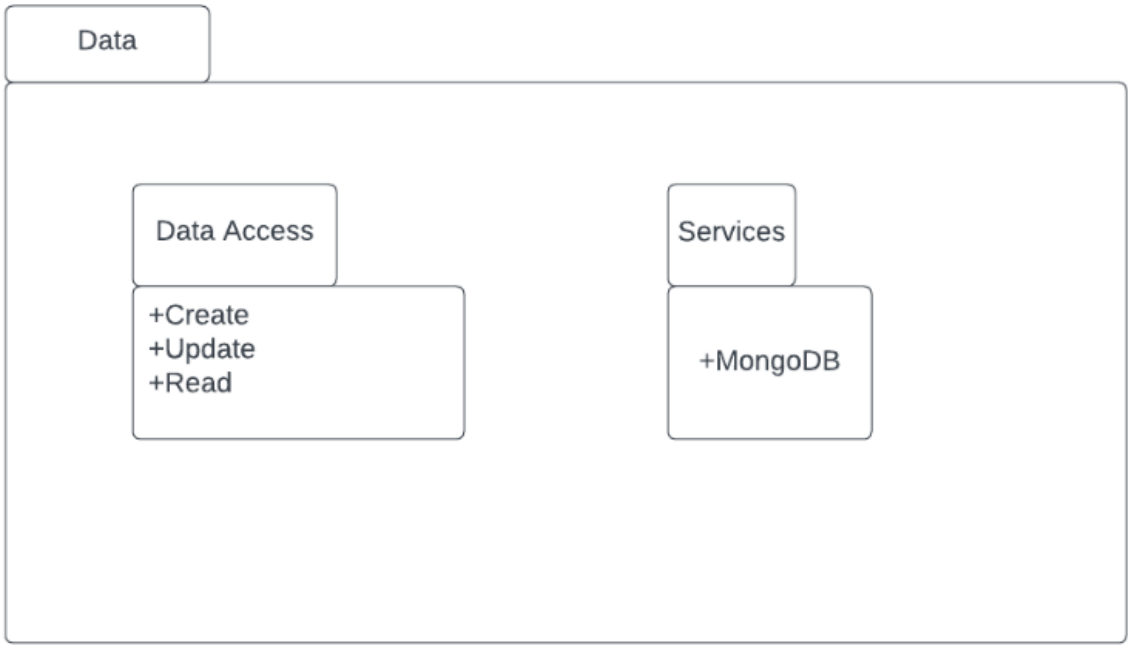
Req-Pass-Change
-Allows user to send email for a password reset

Reset-Password
-Allows user to enter a new password for their account

Bookmarks
-Maintains the bookmarked events for a user in one location

Error
-Redirect page for any unforeseen errors

Settings
-Lets users change their platform settings



Presentation:

UI:

The role of our UI is to provide a clean user interface that is easy for our users to use while also effectively accomplishing our desired tasks.

Controllers:

The controllers facilitate the behind-the-scenes work behind user information, such as posting new account data to our database and sending out emails to users to aid in resetting passwords and other possible troubles they may face.

Pages:

This outlines the basic functionality for all pages the user can visit on our website. Some may be more common than others, given user circumstances, and an error page exists in case of unforeseen events.

Data:

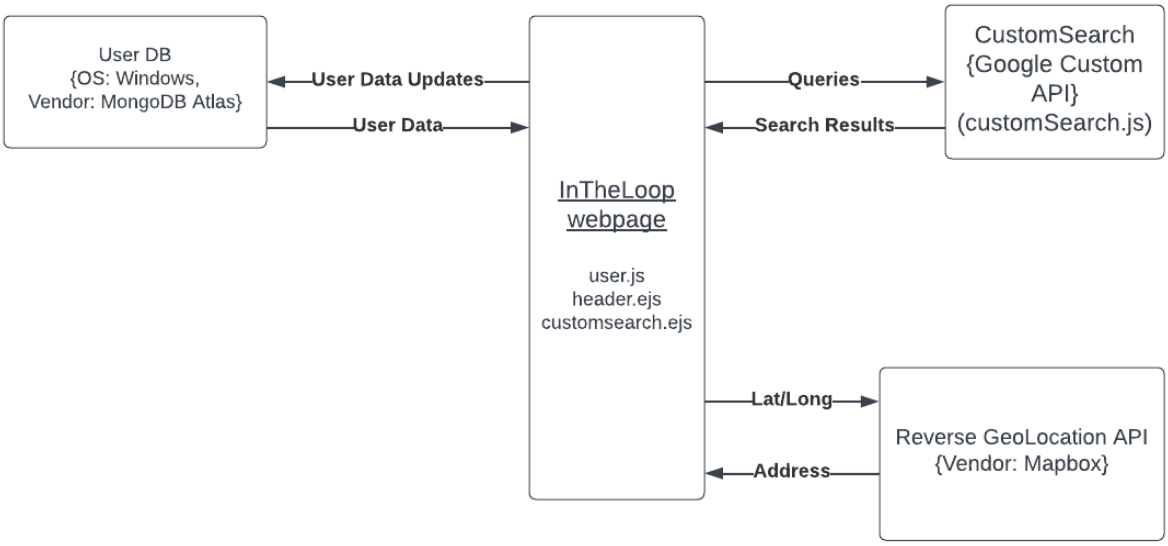
Data Access:

This section shows the alterations that can be made to data by our program. Currently, we allow for data creation, updates, and reading. We do not have any functionality for deleting existing accounts.

Services:

This section shows our service provider for data storage, which in our case, is MongoDB.

Deployment Architecture:



Our architecture consists of our core webpage and its interactions with an external database and two API calls from separate vendors.

The User DB is hosted by MongoDB Atlas on their servers and hosts the user data for our users. Updates can be posted, and data can be retrieved as needed.

The API calls come from two sources: the customSearch API comes from Google and returns search results when fed queries involving the user's location and data. The second API is for reverse geolocation. The original latitude and longitude are retrieved from a built-in call for the location provided by the user's permission in the browser. When fed into the API, this returns the basic address information, such as city, state, etc., used within the customSearch algorithm for local events.

Persistent Data Storage:

We use MongoDB Atlas as our database software to store user data. The data contains an email address, password, first and last name, bookmarks, interests, and dislikes. We also have cookies in place to store permission data, such as location information, to run our geolocation API. Our database storage is queried and modified within the program. While a user is logged in, they may update their information, and these changes will be updated in the database.

Global Control Flow:

The flow of our program is procedure-driven. The program will always begin by checking the user's location and populating this information on the screen. Next, the user

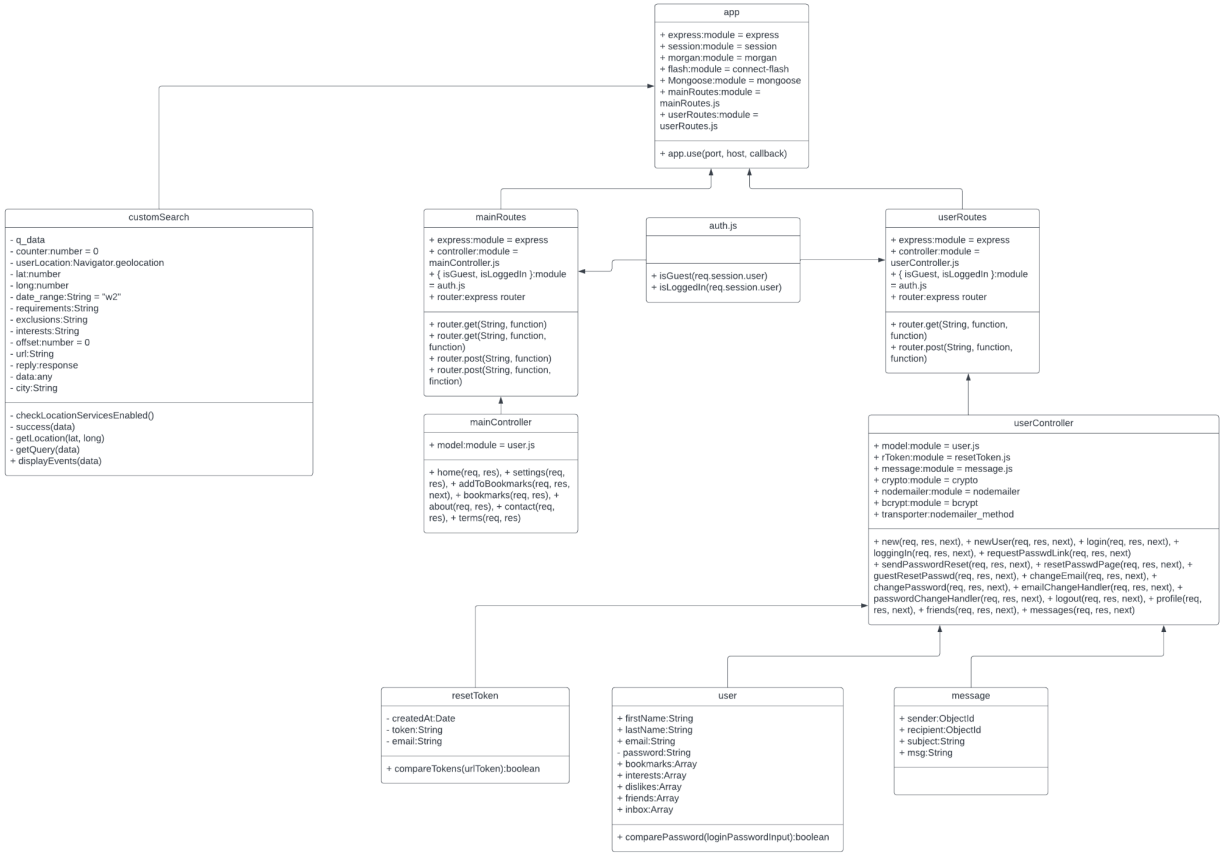
will search for events in their area, taking interests and dislikes into account as the target location. From here, the user will either bookmark an event or click the provided source links for more information directly from the source where the custom search API found this event. At any time, the user can adjust their data which will, in turn, influence the results received.

Detailed System Design:

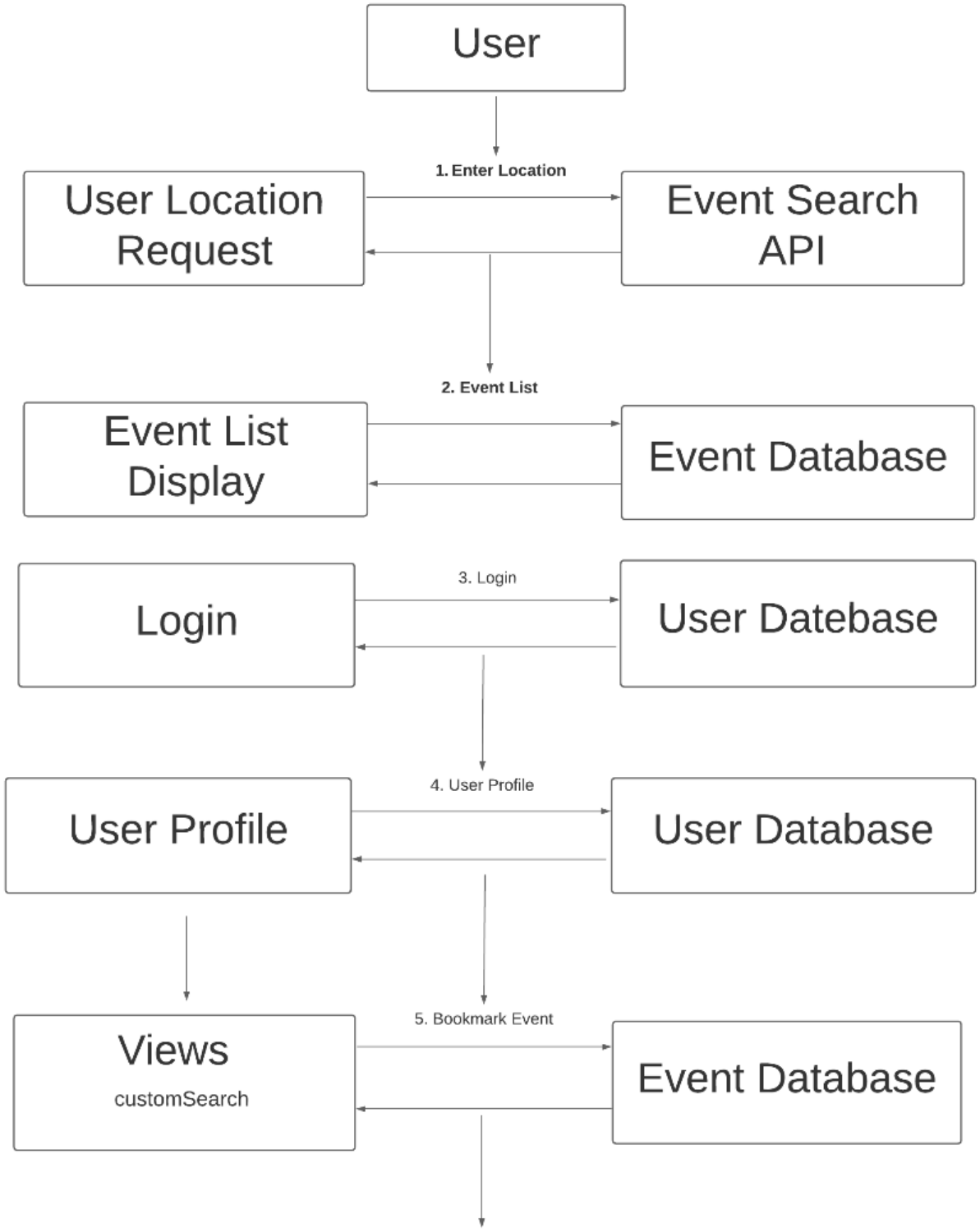
Static View:

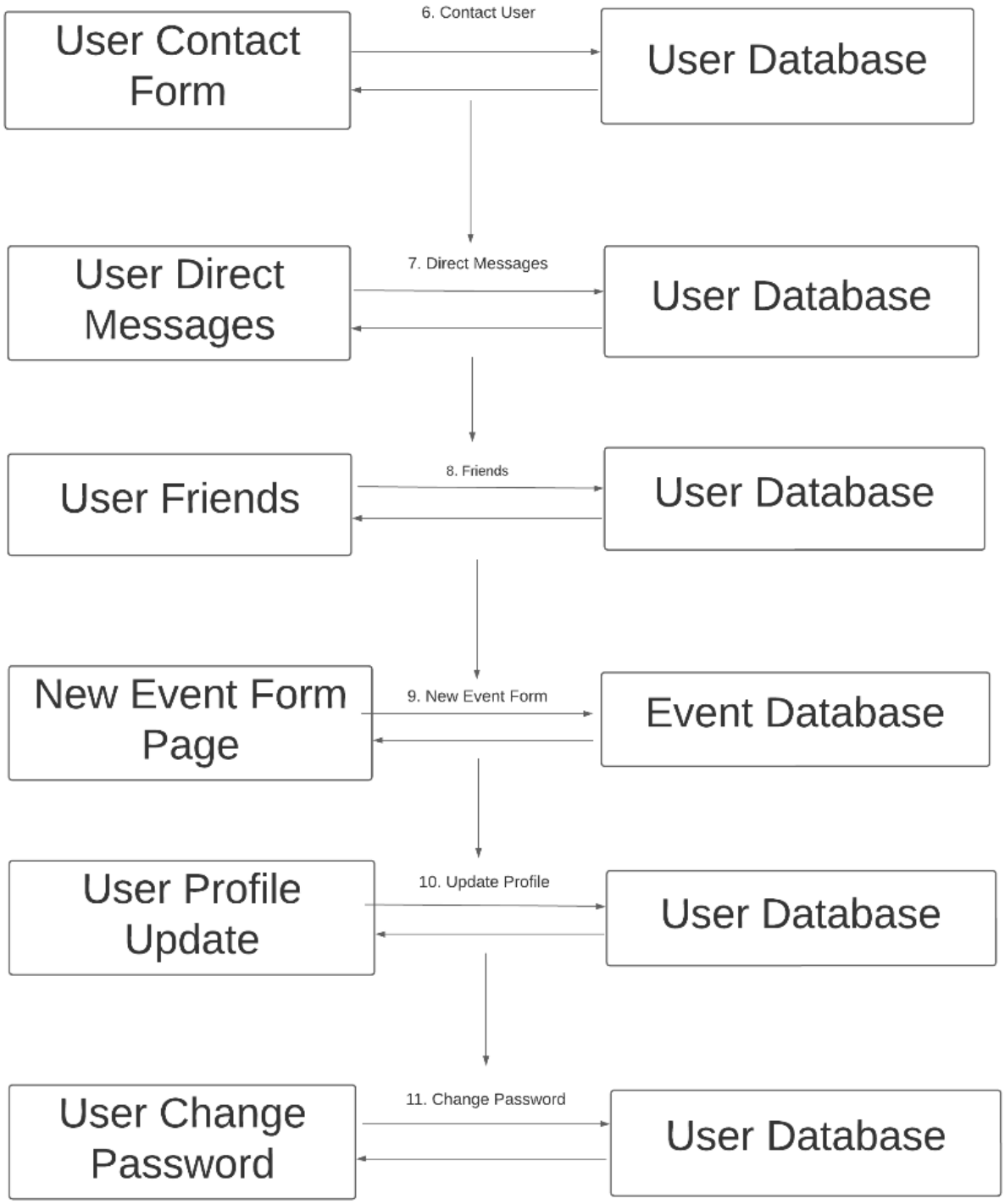
Our system design ***uses no JavaScript classes but*** JavaScript modules instead. As such, we have included the JavaScript modules in place of the classes that would be included in the UML class diagram. Additionally, as an alternative to displaying the multiplicity constraints, we have elected to show which modules utilize each other. Per our use of the Model-View-Controller (MVC) architecture for our web application, the following modules were created: userController, mainController, mainRoutes, userRoutes, auth, message, resetToken, user, and app.js. As the name suggests, the app.js file is responsible for listening to connections and implementing any modules the web application uses. The mainRoutes and userRoutes module implements express routing for the application.

Furthermore, those modules use the controller modules, adding functionality to the routing. This means taking care of any operations and rendering web pages. The routing modules also use the auth module to implement authentication to the web application; in our case, it provides an extra layer of security so that only registered users can access certain parts of the web app. Finally, the customSearch module adds functionality to the web application by using an API to grab the user's location for the web application and display data to the user.


app
+ express:module = express
+ session:module = session
+ morgan:module = morgan
+ flash:module = connect-flash
+ Mongoose:module = mongoose
+ mainRoutes:module = mainRoutes.js
+ userRoutes:module = userRoutes.js
+ app.use(port, host, callback)
customSearch
- q_data
- counter:number = 0
- userLocation:Navigator.geolocation
- lat:number
- long:number
- date_range:String = "w2"
- requirements:String
- exclusions:String
- interests:String
- offset:number = 0
- url:String
- reply:response
- data:any
- city:String
- checkLocationServicesEnabled()
- success(data)
- getLocation(lat, long)
- getQuery(data)
+ displayEvents(data)
mainRoutes
+ express:module = express
+ controller:module = mainController.js
+ { isGuest, isLoggedIn }:module = auth.js
+ router:express router
+ router.get(String, function)
+ router.get(String, function, function)
+ router.post(String, function)
+ router.post(String, function, finction)
auth.js
+ isGuest(req.session.user)
+ isLoggedIn(req.session.user)
userRoutes
+ express:module = express
+ controller:module = userController.js
+ { isGuest, isLoggedIn }:module = auth.js
+ router:express router
+ router.get(String, function, function)
+ router.post(String, function, function)
mainController
+ model:module = user.js
+ home(req, res), + settings(req, res), + addToBookmarks(req, res, next), + bookmarks(req, res), + about(req, res), + contact(req, res), + terms(req, res)
userController
+ model:module = user.js
+ rToken:module = resetToken.js
+ message:module = message.js
+ crypto:module = crypto
+ nodemailer:module = nodemailer
+ bcrypt:module = bcrypt
+ transporter:nodemailer_method
+ new(req, res, next), + newUser(req, res, next), + login(req, res, next), + loggingIn(req, res, next), + requestPasswdLink(req, res, next)
+ sendPasswordReset(req, res, next), + resetPasswdPage(req, res, next), + guestResetPasswd(req, res, next), + changeEmail(req, res, next), + changePassword(req, res, next), + emailChangeHandler(req, res, next), + passwordChangeHandler(req, res, next), + logout(req, res, next), + profile(req, res, next), + friends(req, res, next), + messages(req, res, next)
resetToken
- createdAt:Date
- token:String
- email:String
+ compareTokens(urlToken):boolean
user
+ firstName:String
+ lastName:String
+ email:String
- password:String
+ bookmarks:Array
+ interests:Array
+ dislikes:Array
+ friends:Array
+ inbox:Array
+ comparePassword(loginPasswordInput):boolean
message
+ sender:ObjectId
+ recipient:ObjectId
+ subject:String
+ msg:String

Dynamic View:





Upon arrival at the site for the first time, a user will be prompted for permission to use their location data. The website will display its location on the index page and events for the user. The user may log in at any time, but to proceed, it is recommended that they do so. Logging in will allow them to bookmark any events they are interested in. From here, users are taken to their user profile after creation so they may add their preferences and data. This allows for the customSearch API to be used to its full potential and take specific

likes and dislikes into account. Users may contact each other and share events between accounts. Users may also add one another as friends to retain their information and prevent the need to memorize usernames to send any shared events. Users can change their profile and password information at any time.