OULU UNIVERSITY OF
APPLIED SCIENCES

Kenneth Mutai

DEVELOPING AN APPLICATION WITH METEOR FRAMEWORK

DEVELOPING AN APPLICATION WITH METEOR FRAMEWORK

Kenneth Mutai
Bachelor's Thesis
Spring 2017
Business Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Business Information Technology

Author(s): Kenneth Mutai
Title of Bachelor´s thesis: Developing an Application with Meteor Framework
Supervisor(s): Juoni Juntunen
Term and year of completion: Spring 2017 Number of pages: 40

The development of web and mobile application using software framework is common nowadays due to the rise of web services. The use of framework enable developers to facilitate the development process by making it easy and quick. The framework provides the developer with readymade tools such as packages and environment for implementation. Some of these frameworks can be used to develop application and prototypes.

The objective of the thesis is to demonstrate Meteor.JS framework by developing a responsive web application. The development process also enables the exploration of Meteor.JS extensively. The prototype could also be used as support material for other students during lectures or self-study. The responsiveness enables of the application to be used on different platforms such as mobile devices. Meteor.JS was the main tool used for the application also, MongoDB and Bootstrap supported the development of the application.

The result of this thesis was a web application with several functionalities. These functionalities include the use of camera, geolocation, instant messaging and authentication. The files where hosted with Cloud9 and tested on both mobile devices and web.

Keywords: Framework, Meteor, Web application.

CONTENTS

1 INTRODUCTION

Software framework is a conceptual development platform that provides a fundamental support to software development process. The platform simplify the development environment where pre-defined code structure with generic functionality can be customized by the developers. The code structure forms modules of an application which consist of structures and templates that are reusable in application programming interface (API) during the development stage. Also, a software framework has inversion control where the global flow of control exists within framework.

The development of hybrid applications using software framework enables developers to implement various features of an application by combining the existing packages in an application. Currently, there are several open source frameworks for mobile and web development available that are suitable for developing rapid prototype for demonstration purposes and software for production usage, for example Meteor, Firebase, Ionic and Cordova just to name but a few.

The full stack frameworks provide a platform that enable developers to fully implement the functionalities of an application both front end and back end which includes application server, web browser and mobile devices. These frameworks consist of technologies that create the connection between client and database; and, it includes packages that are required to implement the functionality of the application. There are several tools that support the implementation of the application namely, compilers, libraries and application programming interfaces (APIs), these tool sets enable the developer to combine different components of the system.

The packages consist of pre-existing libraries that allows developers to primarily focus on implementation of the application and thus avoiding the repetition of writing new code hence facilitate the development process and reducing the general cost of development. Moreover, taking advantage of using the existing libraries implies that; less coding is required during development phase hence less bugs hence high quality, secure and finally reliable product.

2 PROJECT OBJECTIVE

The objective of this thesis is to explore and demonstrate Meteor.JS framework. This is accomplished by developing Give and Get web application demo using the framework. The application demonstrates the implementation of several functional features such as instant messaging, capturing images and the use of Google map for geolocation. This application was developed during Devlab studies, which are part of the optional studies at Oamk.

The main purpose for developing the Give and Get demo was to study and learn Meteor.JS framework. In addition, other students can use this demo to study independently or it can also be used as support material during lectures. The application demonstrates both the structure and the content of Meteor application as well as the implementation process of the Meteor application.

2.1 Background

Give and Get application was started its development process during the project related to Devlab studies. Devlab is of the three departments at Oulu University of Applied Science LABs research (Oamk Labs). Oamk labs is a research center comprising of both local and international students from different study disciplines with the aim of exercising their knowledge and experience by working on projects and providing solution to challenges from organization within the region through interdisciplinary, entrepreneurship, innovation and pedagogy (Oamk 2017).

The theme of Devlab 2016 was promoting sustainability, green care, and natural resources. Working as a group was beneficial because the working environment support the team motivation to concentrate on the project, moreover, within the group, members from different fields of study facilitated the evaluation process through various stages mainly because of strong knowledge each individual member in the group had through their respective studies. Tutors and coaches also provided a lot of guidance and mentorship especially during the early phase when ideating and validating the concept solution.

2.2 Give and Get

At the early stages of this project my group was given a task from a local waste management company to solve the challenge they are facing; reducing the amount of household waste that eventually ends up at the landfill. The goal of the project was to create a demo solution that is both sustainable, and able to solve the challenge they were facing at the time. Through a long process of evaluating and validating the available solutions we finally decided to create a hybrid mobile application.

The purpose for developing Give and Get web application was to promote re-use. Re-use is among the three R's in waste management hierarchy. The application provides a platform that facilitates the exchange of items that are still functional but the owner does not need it anymore, thus avoiding disposal which ends up in the landfill. The web application connects persons who intend to give away or dispose an item without requiring any cost from it with anyone who might need.

First, to provide a channel of communication between the giver and the receiver instant messaging proofed to be the commonly used channel and because the application will run on a mobile device. This provides a chat platform for both to negotiate more regarding the item and if the receiver has questions to ask the donor and to plan on the time to pick the item.

Secondly, Geolocation feature provides the receiver to recognize the distance in between the donor to evaluate the means of reaching out to the donor. The means of transportation especially large items such as furniture are cumbersome if one neither has means of transport nor own any transport equipment, therefore Google map provides a good location system and ability to plan accurately the cost and time it takes.

Other important feature that could enable donor to post the item to the platform is the camera; the image and brief description of the item provide a clear identification of the status of the item. With this clarity, the receiver can evaluate and determined whether the items meet the expectation. The application also includes basic functionality of an application such as authentication which enables the user with the privileges of controlling their account.

2.3 Project requirements

The application consists of several features that enable the functionality. The application will also be responsive hence can run on both mobile and web platforms. Therefore, the requirements that are implemented are important when developing the application. The application requirements include UI, navigation, data persistence, account management, camera, instant messaging and a map.

The User Interface (UI) enables the user to interact with the application. Therefore, the user can perform commands and execute tasks such as adding text and images. There are two separate packages that are combined in the application to create the UI i.e. Bootstrap and Blaze. Bootstrap is a front-end framework and it is used to implement the UI because it is responsive hence suitable for both mobile and web development. Blaze library also is used for rendering purposes because it is easy to use. Meteor also supports other rendering libraries such as React and Angular, these two technologies requires previous knowledge which takes time to study them.

Give and Get contains several pages therefore it requires routing to enable the user to navigate through different pages of the application. Navigation guides the user and allows opening of different pages within the application to accomplish task. In addition, navigation buttons help the user to perform commands tasks such as capturing the image and editing the content of the object. Therefore, the application needs a router because the current hyperlinks cannot support the Meteor format of navigation. The navigation of the application is implemented with flow router, although there are other routers such as Iron router but flow router was selected for this project because it is easy to use and recommended by Meteor.

The application contains data such as images, account information and text. MongoDB provides storage for these data layers. The data in the database can be access in the server and client side, MongoDB runs on the server and mini-Mongo runs on the client. The data remains in sync therefore same data is available on both sides, and when the data is change is made it takes place first on the client side then transferred to the server through latency compensation.

The account management provides the user with the control of the account and settings. Through correct authentication the user is authorize the access of the application with certain privileges. The user must create an account with the application to have privileges or manage the account.

The user must always login into the application to have the access to the privileges. When a user is creating a new account a name and email address are required, the username is optional.

The camera package in the application allows the user to capture the images with their device. The package gets the image as a base64-encoded data which is a uniform resource identifier (URI) in JPEG format. URI consist of a string of characters that identifies an abstract or physical resource (Berners-Lee 2005, 6). The image is attached it to text which describe the image and the title name, this forms an item that is save in the MongoDB as an object. Furthermore, it has one function to call and other optional argument such as customizing the height, width and quality.

Instant messaging provides a communication channel between the users via text. The owner of the item receives a notification informing that someone may be interested in taking the item, they can both continue with their conversation. The text conversation is majorly based on the agreement and collection time of the item. The text conversation only exists between two users per item i.e. the owner and recipient. Therefore, if the user has several items there will be other conversations related to that individual item. The text conversation remains saved in the database until when the transaction is over and deleted.

Google map package is also required for the application for geolocation feature to be implemented. Geolocation provides identification of an object on a real world geographic location. The user will add the item by posting to the application which will then appear on the map as icons representing each category i.e. furniture, electronics, clothes and toys. The Icons on the map will be location on the same location where the user added the item. This is possible by capturing the longitude and the latitude of the user from the device. Therefore, it helps the user to know the exact distance from the item and the route.

For the project, Meteor will be installed on windows although it also supports other development platforms such as OS X and Linux. Atom editor is used for editing the codes. Finally, the application will be hosted on cloud9 which provides the opportunity to run and test the application. Test such as browser compatibility and functionality of the features through live preview will be carried out. Cloud9 is an open source integrated development environment supporting several programming languages including JavaScript with Node.js.

The documentation for the requirements of the application is obtained from official website for Meteor, Blaze, Flow-router. Other sources include athmosphere.js and online materials. The documentation on the implementation may vary depending on the application functionality and can selected from a wide range of sources depending on the features required.

This thesis report will not focus in depth MongoDB and cloud 9. They are both important to the application but they have a wide scope and therefore may require a topic of its own. They will be mention in brief on how they will be used together with Meteor application.

3 METEOR FRAMEWORK

Meteor JS is an open-source full stack JavaScript-based framework used for developing web and mobile applications. It consists of existing technologies that are combine to creates the building block of an application. The components may vary depending on the application requirements. The role of these components is described and how they are implemented in the development of Meteor application.

3.1 Meteor.JS

Meteor was founded 2011 by Meteor Development Group (MDG) with an aim of providing open source platform for web application development. Several months after the released MDG got financial boost from major companies in the software industry. Meteor growth was quick because it provided support especially by basic development infrastructure such as data synchronization and compiling the code. Therefore, the developers could only concentrate on the business functionality of the application. With more support from the investors, Meteor focuses to target larger enterprises and adding more development tools to developers. The current version of Meteor.JS is 1.4 and it supports both Windows and Linux operating systems.

Meteor main objective was initially to create basic infrastructure such data synchronization that deploy an application on web or mobile devices. MDG also focused on the use of software libraries that could be customized from the existing Node.js modules to implement the functionality. Another area was to promote the use of standardized protocols such as Distributed Data Protocol (DDP) and the ability to provide service for the developer. (Hochhaus & Schoebel 2015a, 5).

Meteor development platform consist technologies that contribute to the development process of the application. The technologies are build tools, a collection of software packages, web sockets and MongoDB (Hochhaus & Schoebel 2015a, 5). Meteor.JS development process uses JavaScript in all environments; application server, web browser and mobile device which runs in both frontend and backend and uses same application programming interface (API). The combined technologies enable the developer to builds reactive application.

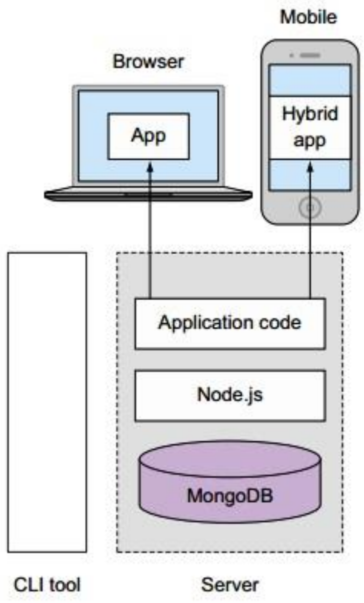


FIGURE 1. Meteor Stack Components (Hochhaus & Schoebel 2015, 6).

The figure 1 above illustrates a summary of the core components needed to run a Meteor application. Regarding the server side, it consists MongoDB, Node.js and application code. MongoDB provides storage for the data and Node.js is a server side JavaScript environment, it is highly scalable runtime environment and an event driven. It has the similar functions as an Apache web server in LAMP (Linux, Apache, MySQL and PHP). Finally, the application code comprises of packages that provide and enforce the functionality of the application. The example of these functionalities includes OAuth logins, request routing and reactive UI. (Hochhaus & Schoebel 2015b, 6.)

Build tool such as CLI assist in creating and managing the infrastructure of the development environment by quickening the process setting up the workflow which involves a lot installing or configuring using these tools. In Meteor development environment community packages such as npm and atmosphere provide several tools, adding authentication through Gmail OAuth or Facebook OAuth can be implemented quickly by Meteor user accounts to the packages list.

The browser and mobile platform deploy the application and interact with the user. Therefore, through designing and testing Meteor stack core packages are integrated to work well without any conflict within the packages. Despite the using specific core packages with Meteor, it is also possible to use other technologies supported by Meteor.JS such as using Angular.JS on the client side instead of Blaze.JS.

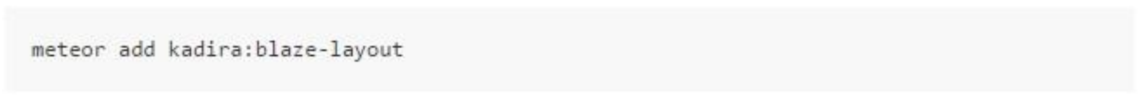
3.2 **Packages**

Package system facilitates the development of an application and it is easy to customize i.e. adding and removing the functionality (Strack 2015a). These packages are available in the atmosphere.js and package stars and flags. The atmosphere contains many packages of which majority supports Meteor and is used in developing Meteor application (Meteor 2017a). They are community based packages developed by individuals or organization and used by any interested person for free.

According to Strack, within a period of 4 years MDG had developed over 140 packages. These packages not only provide features and functionalities but they are customizable and easy implement on the project (Strack 2015b).

Meteor consist of core packages that are used to implement the functionality of the application such as Blaze package which is used to implement a reactive user interface (UI) library and User-Accounts package that manage the user accounts libraries, just to name but a few. The packages can be customized by extending the existing packages that had been already developed or it can be installed from node.js modules through node package manager (npm). (Hochhaus & Schoebel 2015a, 5)

The installation of the package to Meteor application is done by adding the name of the package from the atmosphere.js through the command prompt. A package can also be installed in the application by typing in the name of the package into the package folder in Meteor application. The line of Code in Figure 2 below show an example of installing Blaze.js. These command is run in the command prompt and the installation the blaze package to Meteor application will run automatically.



```
meteor add kadira:blaze-layout
```

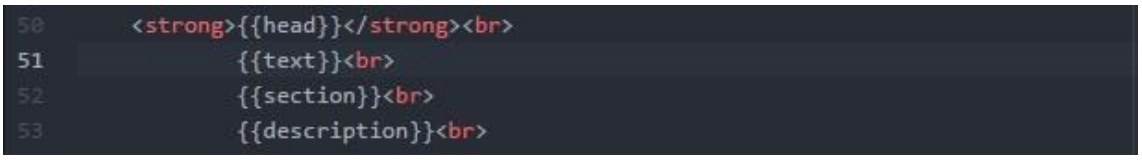
Figure 2 Blaze.JS Package.

3.3 Blaze.JS

Blaze is a rendering library build in meteor that is used to create user interface (UI), with blaze it is possible to write reactive HTML templates. Reactive template can listen and update automatically to changes incurred by the JavaScript code in the mini-Mongo database from the directives that is integrated with tracker library and reflect the result on Document Object Model (DOM). Therefore, Blaze reacts to the JavaScript code that listens to changes in the mini-mongo.

Blaze contains a compiler that runs against Blaze runtime library which supports template syntax and compiles files from the template files into JavaScript UI components. The UI templates contain Spacebars that render the reactive and dynamic data context. (Blaze 2017a.) Spacebars is templating language used by Meteor inspired by Handlebars (Strack 2015c).

Block tags are JavaScript helpers that are either build-in or custom that passes a template when initiated by a directive or helper. There are different template tags that are used for different purposes for example control structures such as conditions and loops. Block helpers such as {{#if}}, {{#Unless}}, {{#with}}, {{#each}}, {{#let}} are build-in tags and the difference with custom block helpers is that the template must be declared then invoked in the double curly braces.

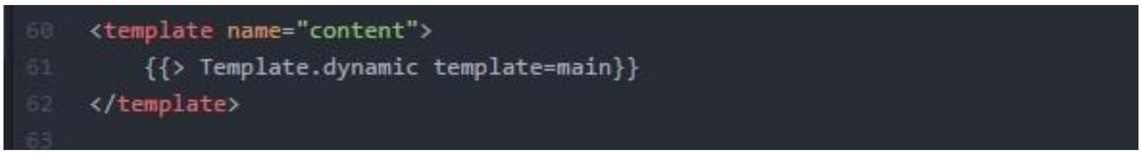


```
    <strong>{{head}}</strong><br>
            {{text}}<br>
            {{section}}<br>
            {{description}}<br>
```

Figure 3. Spacebars.

The figure 3 above show an example of spacebars in a template where the data context such as head, text, section or description is expected to be rendered, the names are objects with key.

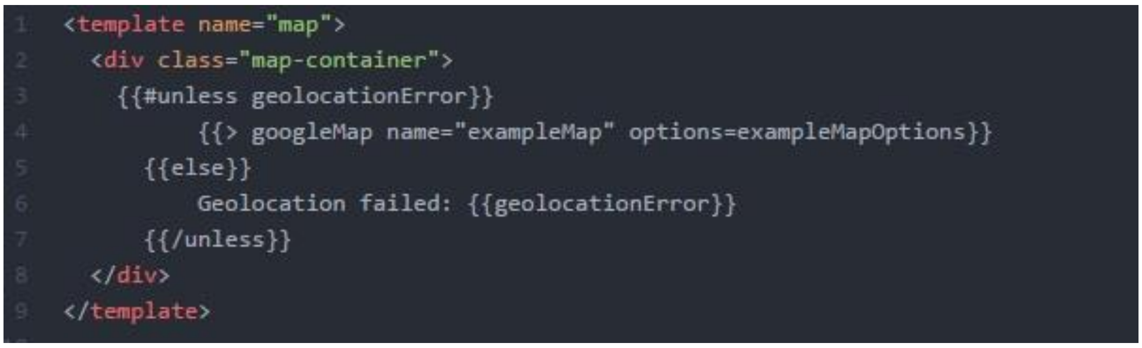
The template layout on figure 4 below is used to render template into to the main area using spacebars.



```
<template name="content">
    {{> Template.dynamic template=main}}
</template>
```

Figure 4. Spacebars rendering main template.

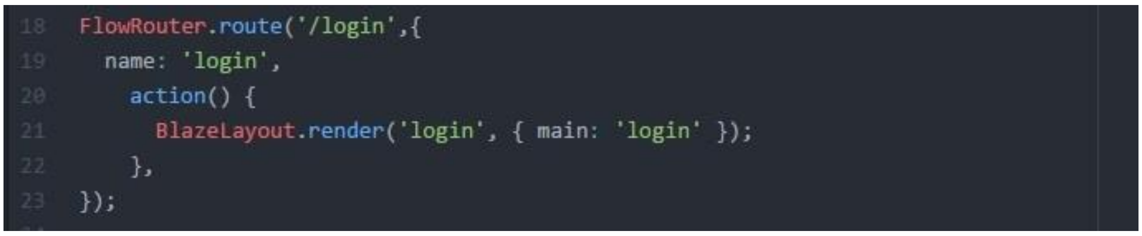
Another dynamic template block can also be added to the template which can be conditionally rendered or switched between multiple layouts.



```
<template name="map">
  <div class="map-container">
    {{#unless geolocationError}}
          {{> googleMap name="exampleMap" options=exampleMapOptions}}
      {{else}}
          Geolocation failed: {{geolocationError}}
      {{/unless}}
  </div>
</template>
```

Figure 5. map template

The main variable from the figure 5 above connects to the name of the template that is rendered with Template. Dynamic by passing it through BlazeLayout.render(). Therefore, according to the figure 6 below the template map will be rendered.



```
FlowRouter.route('/login',{
  name: 'login',
    action() {
      BlazeLayout.render('login', { main: 'login' });
    },
});
```

Figure 6. Blaze Layout Rendering.

3.4 Reactivity

Meteor uses web sockets to allow communication in both ways between the client and server, these enables structured data to be fetched from server to client through Distributed Data Protocol (DDP) and receives live updates when the data is changed.

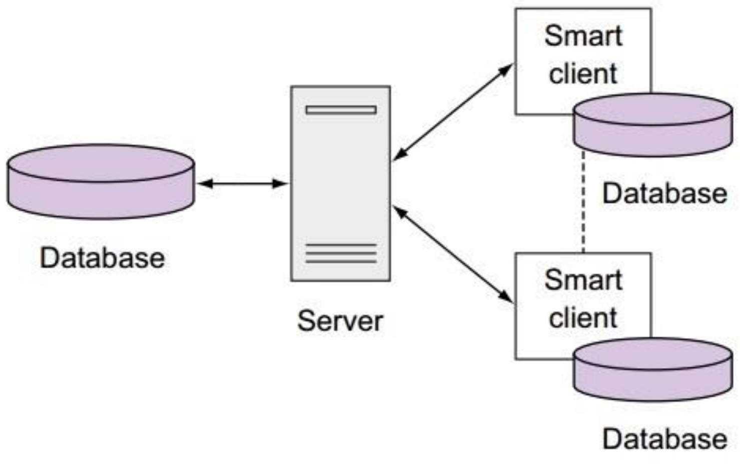


FIGURE 7. DDP (Hochhaus & Schoebel 2015, 10).

As shown in the figure 7 above the processing of data from a single server and distributing to multiple clients empower the browser to act as a smart client because the server pushes updated content to the connected clients. The use of DDP results in quick response time because less data is transferred from server to client and the processing of data avoids the long running request. (Hochhaus & Schoebel 2015c, 10).

The communication within the applications is done over both HTTP and Web Sockets. During page request, static resources are transferred over HTTP. While the exchange of data between client and server rely on DDP, SockJS provides the infrastructure. The server returns the data as JavaScript Object Notation (JSON) objects.

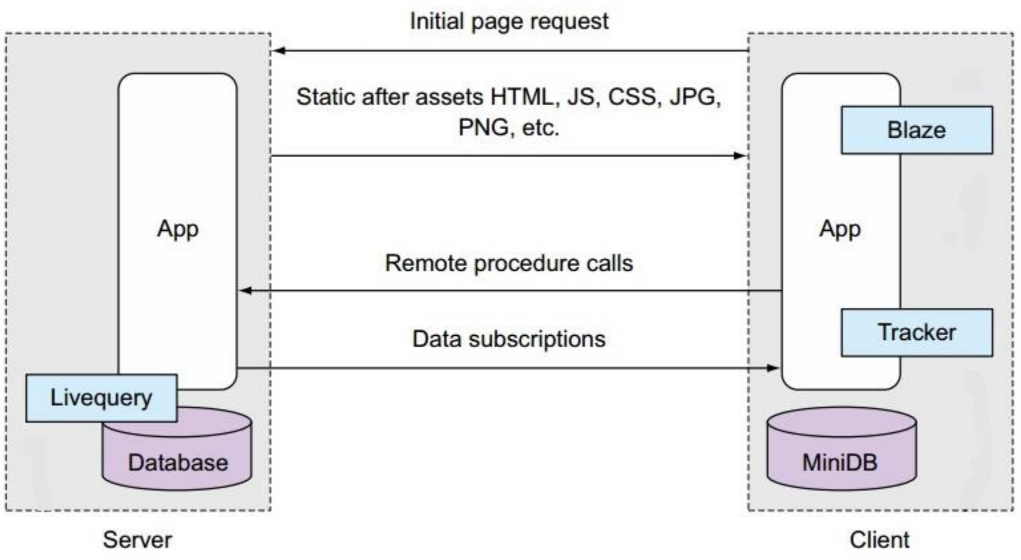


FIGURE 8. Communication between the server and the client (Hochhaus & Schoebel 2015, 14).

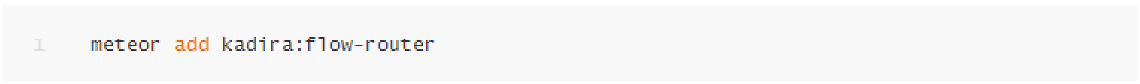
According to figure 8 the role of Live query assists in detecting the changes or update in the database and pushing the changes to the client through subscription. It updates the clients automatically without refreshing the page for the changes to be reflected on the client side. This is because of the distributed environment meteor uses. (Hochhaus & Schoebel 2015d, 15).

Tracker library enables transparent reactivity; it automatically listens to each template helper, if there is any value change in the mini-Mongo it evaluates to a new value. Tracker helps in developing a transparent reactive programming by providing crucial functional reactive programming (FRP). The FRP has dependencies that are connected to data and function that detect the changes if incurred by invalidating the current context and execute the new data. (Hochhaus & Schoebel 2015e, 14-15). The tracker helps developers to avoid defining or declaring explicit data dependencies of each template helper because of the ability to re-render. Template syntax such as handlebars is mostly in Blaze because of the clarity and readability.

3.5 Flow Router

Flow Router is a front-end router for an application. Routing is the change of UI by calling an action on the Uniform Resource Identifier (URL). When an application loads in Meteor all the data available on the server is also reflected on the client-side, this is due to DDT which subscribe to data, therefore the URL will represent the current state of the user and does not render. (Meteor 2017b.)

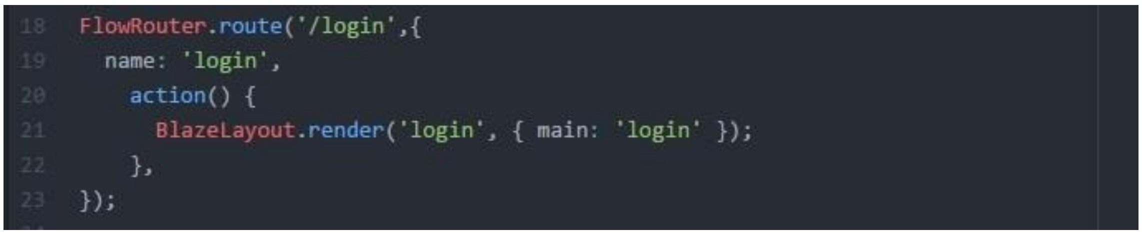
Figure 9 shows how to install flow router package to meteor application by adding through the command line.



```
meteor add kadira:flow-router
```

Figure 9. Flow router

The route matches the URLs pattern on the browser which runs on the client side to perform certain action. There are two type of routes; single route and grouped route. Figure 10 shows the definition of a single route.



```
FlowRouter.route('/login',{
  name: 'login',
    action() {
      BlazeLayout.render('login', { main: 'login' });
    },
});
```

Figure 10. Defining a single route

3.6 MongoDB

MongoDB is the database used by meteor to store data layer as objects. The data in MongoDB is accessed through a collection, which is a set of related data. A collection can be created on several places on Meteor framework such as server-side collection, client-side collection or local collection. (Meteor 2017c.) Therefore, since there will is no directed connection to MongoDB database the client-side collection will be cache to the database this is possible because of mini-mongo library. Mini-mongo consist of JavaScript code that helps in data transfer through subscription and publication. Meteor applications are based on DDT, this allows data to be transferred in both server and client unlike the Http-based application that requires request from the client and respond from the server.

MongoDB is a NoSQL database hence a non-relational database and therefore it uses a nested document structure to facilitate relationship. MongoDB provides the developer with unique query operations. The query enables the developer to access and manage data such as creating, reading, updating and destroying. These operations are not possible with plain JavaScript. (Robinson J. et al, 2016).

4 IMPLEMENTATION

4.1 Overview

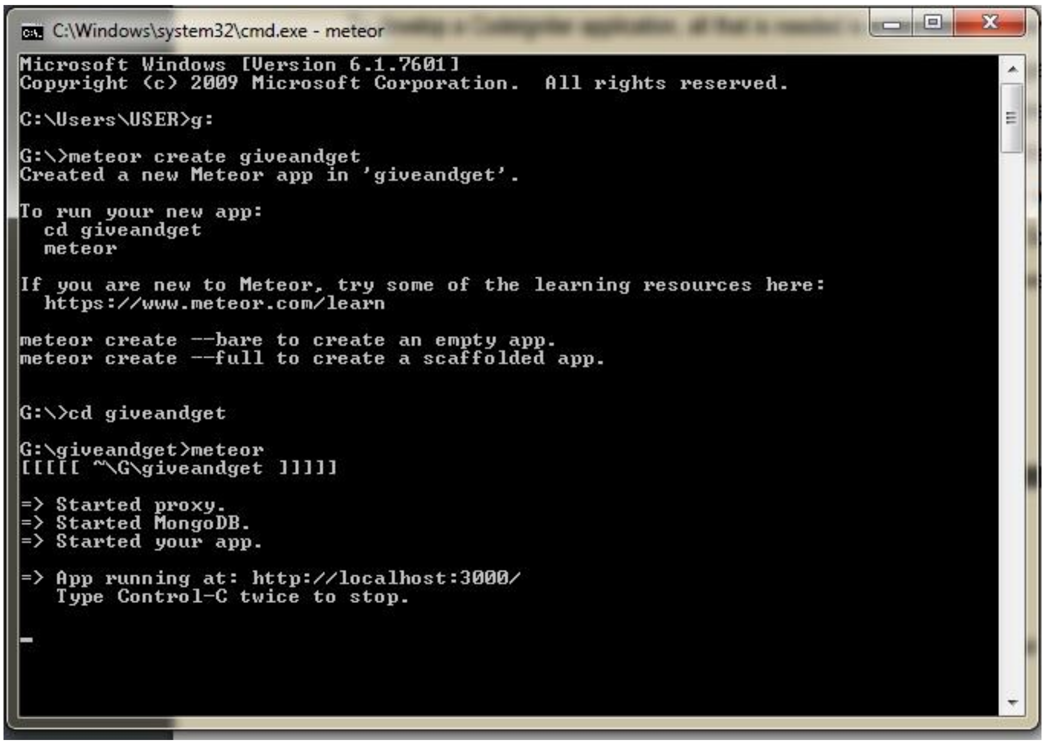
The application enables the user to use a camera from the device to capture and store the images in the database. A map displays icons if the item is available, the display is on the location where the item was uploaded to the application. Instant messaging feature facilitate the communication between two users. The user received a text notification if someone is interested in acquiring the item.

Meteor application will be developed to demonstrate Meteor features and development process. First, setting up the development environment by installing necessary tools. Then the required packages will be added individually as the development progresses. Finally, it will be hosted in cloud 9 for testing purposes. The addition of packages to the application does not require a specific order.

4.2 Meteor installation and setup

The development process begins by installing meteor framework and atom editor. To install Meteor on windows Meteor installer is downloaded from the documentation and it is executed by running an install command. Similarly, the atom editor can also be install from Atom official website. Meteor documentation is important during the whole process. It provides necessary information and guidelines for setting up meteor application.

The packages required are installed independently as the development progress. In case a mobile application had to be developed Cordova package is required therefore it is installed separately. Once Meteor is installed the process of creating the application begins. The figure 11 shows the initial stage of creating and running the meteor application from the terminal.



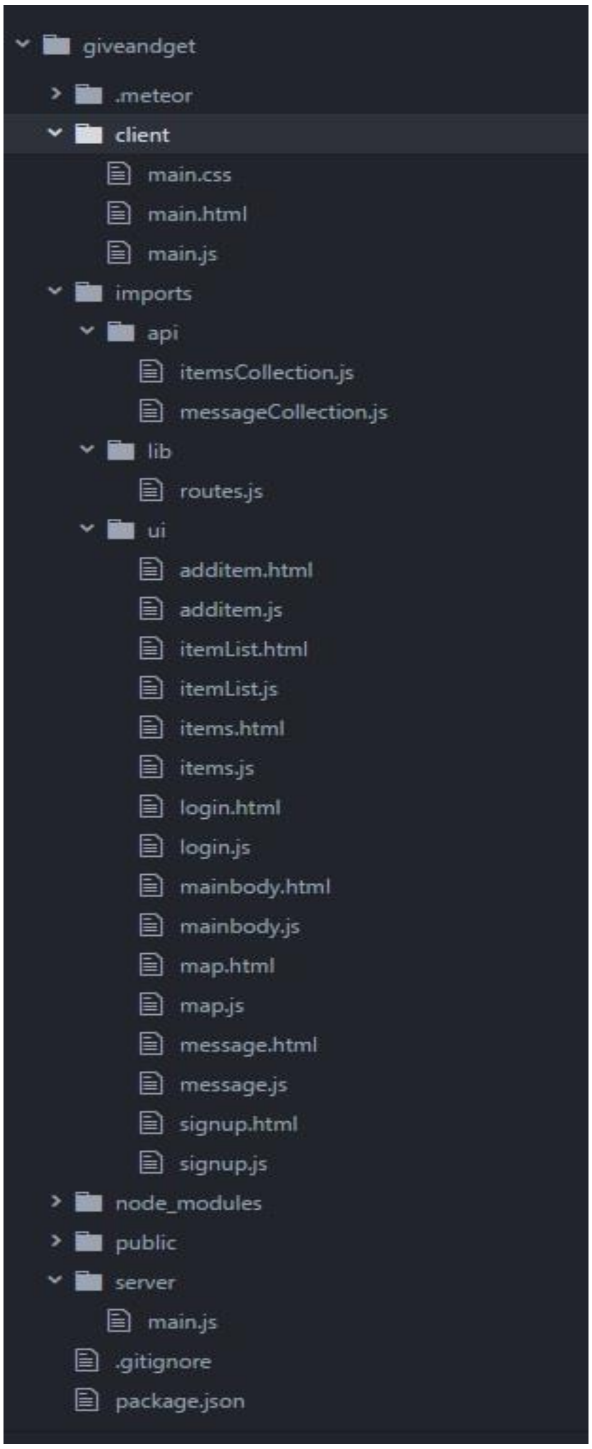
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\USER>g:

G:\>meteor create giveandget
Created a new Meteor app in 'giveandget'.

To run your new app:
  cd giveandget
  meteor

If you are new to Meteor, try some of the learning resources here:
  https://www.meteor.com/learn

meteor create --bare to create an empty app.
meteor create --full to create a scaffolded app.


G:\>cd giveandget

G:\giveandget>meteor
[[[[[ ~\G\giveandget ]]]]]

=> Started proxy.
=> Started MongoDB.
=> Started your app.

=> App running at: http://localhost:3000/
   Type Control-C twice to stop.

_
```

Figure 11. Installing and running Meteor

On the final stages the application is uploaded to Cloud9 environment. To run the application on Cloud9 Meteor.JS should be install on an empty workspace created on Cloud9. Thereafter, Meteor.JS files from the local workstation folder is uploaded to the workspace. Node packages required by Meteor application are also installed. Finally running the application using port option because Cloud9 does not support automatic ports.

4.3 Application Structure

After the installation Meteor application, it contains a default structure. The structure contains HTML, JavaScript and CSS files. Apart from the imports and public folder the other folders are default and they are created once Meteor.JS is install. The public folder contains images and logos that are used in the application both on the client and on the server. The import folder contains subfolders such as api, lib and ui. The api contains the collections, the collection is exported in and out of the import folder to any folder that requires the collection. The lib folder contains the routes that assists in connecting different folders. The ui consist of HTML and JavaScript files. The CSS files are in the client folder. Figure 12 show the overview of the application structure.



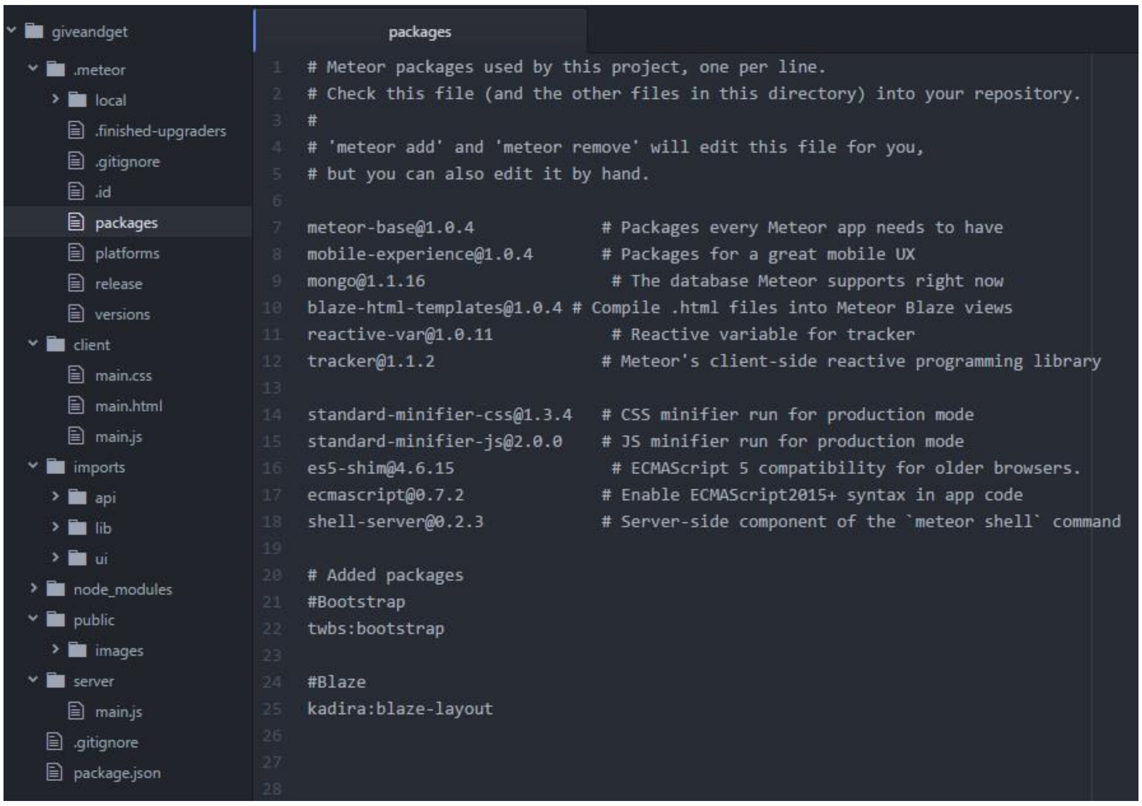
```
giveandget
  .meteor
  client
    main.css
    main.html
    main.js
  imports
    api
      itemsCollection.js
      messageCollection.js
    lib
      routes.js
    ui
      additem.html
      additem.js
      itemList.html
      itemList.js
      items.html
      items.js
      login.html
      login.js
      mainbody.html
      mainbody.js
      map.html
      map.js
      message.html
      message.js
      signup.html
      signup.js
  node_modules
  public
  server
    main.js
  .gitignore
  package.json
```

Figure 12. Application Structure.

Meteor.JS recommends that all the application code should be placed in the import directory. These will enable Meteor build system to access single import directory while looking for a file that is being referenced from either the server or the client. The main.js files in client and server helps in defining the entry points to the client and server when the file are loading.

4.4 UI

After the installation Meteor application, it contains a default structure. The structure contains HTML, JavaScript and CSS files. Therefore, the first step is to examine the existing codes and make necessary adjustment. The whole process is carried out in the atom editor but when adding the package, the terminal is used. As mentioned earlier Bootstrap and Blaze packages are installed first. This provides the base where all other pages will be connected to and from.



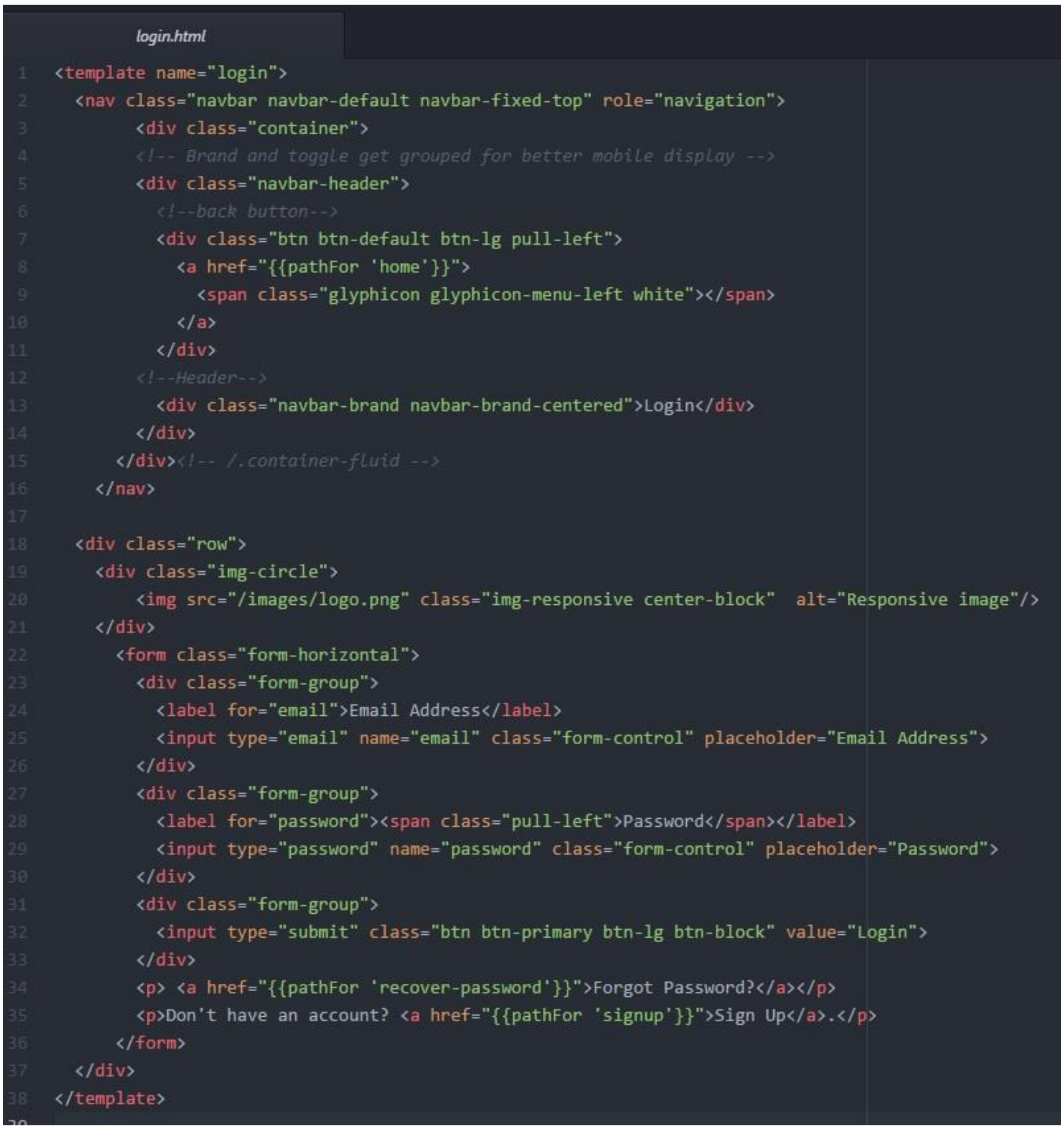
```
# Meteor packages used by this project, one per line.
# Check this file (and the other files in this directory) into your repository.
#
# 'meteor add' and 'meteor remove' will edit this file for you,
# but you can also edit it by hand.

meteor-base@1.0.4             # Packages every Meteor app needs to have
mobile-experience@1.0.4       # Packages for a great mobile UX
mongo@1.1.16                   # The database Meteor supports right now
blaze-html-templates@1.0.4 # Compile .html files into Meteor Blaze views
reactive-var@1.0.11            # Reactive variable for tracker
tracker@1.1.2                 # Meteor's client-side reactive programming library

standard-minifier-css@1.3.4   # CSS minifier run for production mode
standard-minifier-js@2.0.0    # JS minifier run for production mode
es5-shim@4.6.15                # ECMAScript 5 compatibility for older browsers.
ecmascript@0.7.2              # Enable ECMAScript2015+ syntax in app code
shell-server@0.2.3            # Server-side component of the `meteor shell` command

# Added packages
#Bootstrap
twbs:bootstrap

#Blaze
kadira:blaze-layout
```

Figure 13. Meteor application packages and folders.

The figure 13. Shows default packages after adding Bootstrap and Blaze.js. and folders that contains files of JavaScript, HTML, and CSS codes. Import and Public folder were added to the default settings. The current structure consists of the folders required for the Give and Get application. Meteor file structure is flexible therefore the structure can vary depending on the developer. From the above structure, each section can be customized separately. The UI folder contains pages of the application with JavaScript, HTML and CSS as the main components of the application code.

Bootstrap enables styling the user interface using HTML and CSS files. These helps in arranging the components and colors of a page. Other important use of bootstrap is that it enables the creation of responsive template. As a result, each page can be open on any device no matter the size of the display.

On the other hand, Blaze enables the HTML template to react hence updating the content through spacebars. Therefore, since the UI contains Spacebars that react to data from mini-mongo such as creating icons on the map, this enables the page to display the change content without requiring the user to refresh the page.



login.html

```html
<template name="login">
  <nav class="navbar navbar-default navbar-fixed-top" role="navigation">
        <div class="container">
        <!-- Brand and toggle get grouped for better mobile display -->
        <div class="navbar-header">
          <!--back button-->
          <div class="btn btn-default btn-lg pull-left">
            <a href="{{pathFor 'home'}}">
              <span class="glyphicon glyphicon-menu-left white"></span>
            </a>
          </div>
        <!--Header-->
          <div class="navbar-brand navbar-brand-centered">Login</div>
        </div>
      </div><!-- /.container-fluid -->
    </nav>

  <div class="row">
    <div class="img-circle">
        <img src="/images/logo.png" class="img-responsive center-block"  alt="Responsive image"/>
    </div>
      <form class="form-horizontal">
        <div class="form-group">
          <label for="email">Email Address</label>
          <input type="email" name="email" class="form-control" placeholder="Email Address">
        </div>
        <div class="form-group">
          <label for="password"><span class="pull-left">Password</span></label>
          <input type="password" name="password" class="form-control" placeholder="Password">
        </div>
        <div class="form-group">
          <input type="submit" class="btn btn-primary btn-lg btn-block" value="Login">
        </div>
        <p> <a href="{{pathFor 'recover-password'}}">Forgot Password?</a></p>
        <p>Don't have an account? <a href="{{pathFor 'signup'}}">Sign Up</a>.</p>
      </form>
  </div>
</template>
```

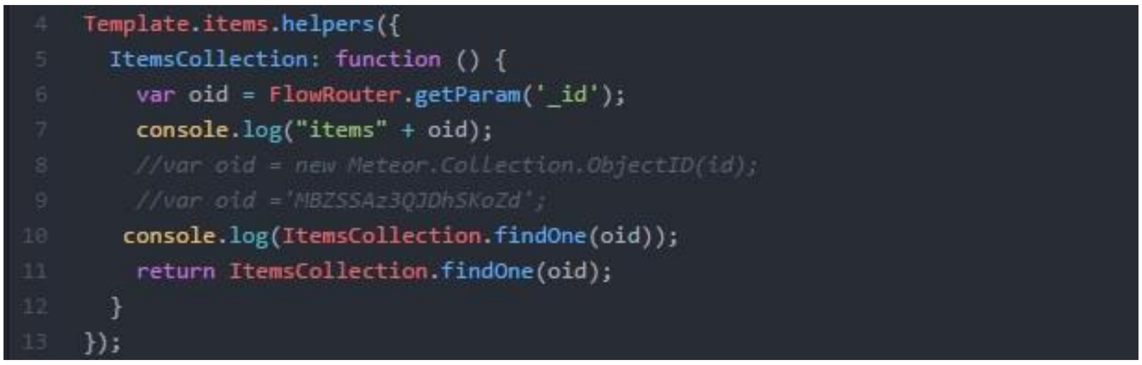
Figure 14. Login template.

The figure 14 above shows an example from the login page. The login template is rendered when the user is required to login to the application. These template is imported to login JavaScript folder where the validation of the username and password verified.

4.5 Routing

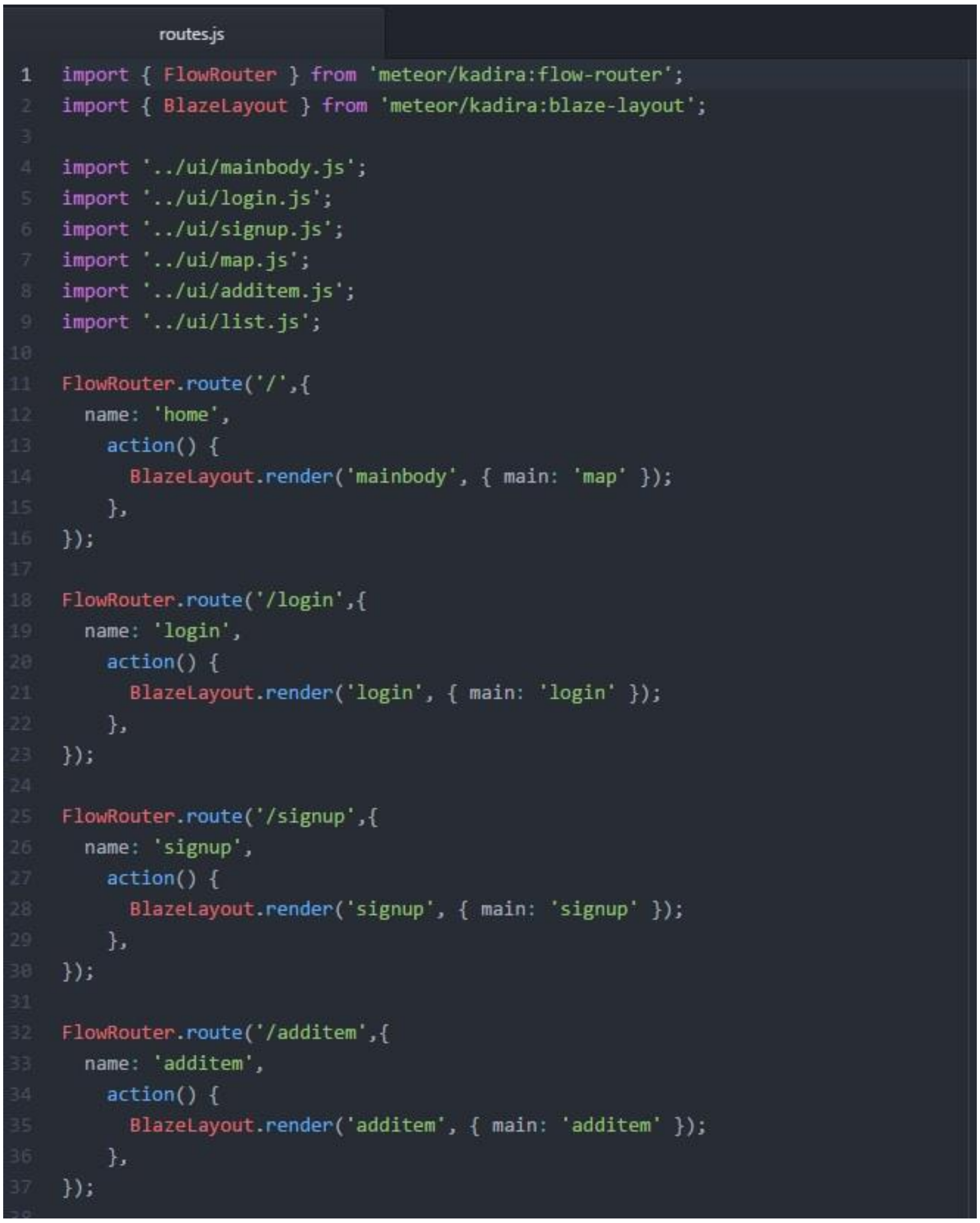
The flow router responsibility is to perform some action when the user visits a URL and blaze assist in rendering the template which contains the content of the page. According to the application structure on figure 15, the application lib folder containing routes.js consist of all routes that link to the pages of application.

Flow router also can use parameters to pass and argument. For example, as shown in figure 15 below, when the user clicked on the icon on the map, the parameter gets and passes the _id of that object and retrieve content of the object and displaying it to the user. The benefits of using parameters enables an individual or several arguments to be passed through.



```
Template.items.helpers({
  ItemsCollection: function () {
    var oid = FlowRouter.getParam('_id');
    console.log("items" + oid);
    //var oid = new Meteor.Collection.ObjectID(id);
    //var oid ='MBZSSAz3QJDhSKoZd';
   console.log(ItemsCollection.findOne(oid));
    return ItemsCollection.findOne(oid);
  }
});
```

Figure 15 Routes passing a parameter.



```
import { FlowRouter } from 'meteor/kadira:flow-router';
import { BlazeLayout } from 'meteor/kadira:blaze-layout';

import '../ui/mainbody.js';
import '../ui/login.js';
import '../ui/signup.js';
import '../ui/map.js';
import '../ui/additem.js';
import '../ui/list.js';

FlowRouter.route('/',{
  name: 'home',
    action() {
      BlazeLayout.render('mainbody', { main: 'map' });
    },
});

FlowRouter.route('/login',{
  name: 'login',
    action() {
      BlazeLayout.render('login', { main: 'login' });
    },
});

FlowRouter.route('/signup',{
  name: 'signup',
    action() {
      BlazeLayout.render('signup', { main: 'signup' });
    },
});

FlowRouter.route('/additem',{
  name: 'additem',
    action() {
      BlazeLayout.render('additem', { main: 'additem' });
    },
});
```

Figure 16 Routes folder.

As shown in the figure 16 above the management of the routes is easy and quick. All JavaScript files are imported to the routes folder. To render a template to a page Blaze action method is required. The action method passes two arguments; the name of the layout template and then the object containing regions that passes the template names to render. The name is to be rendered in the main section of the dynamic template.

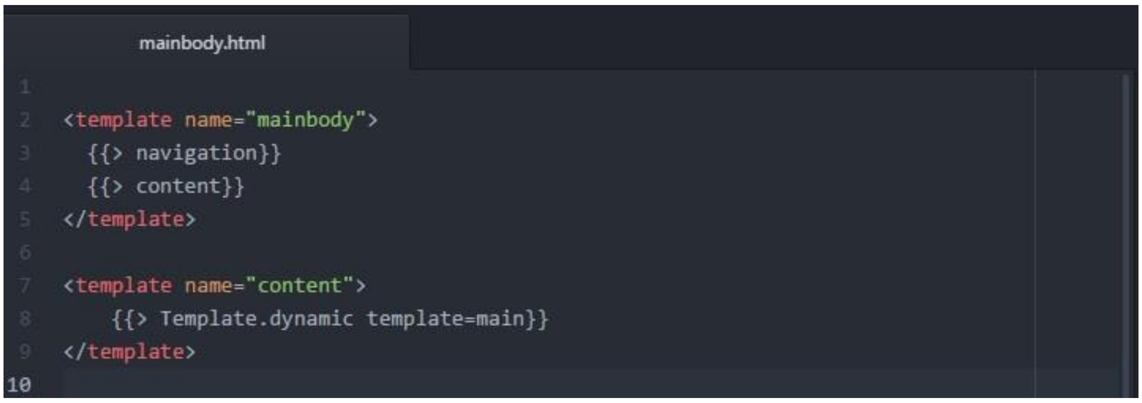
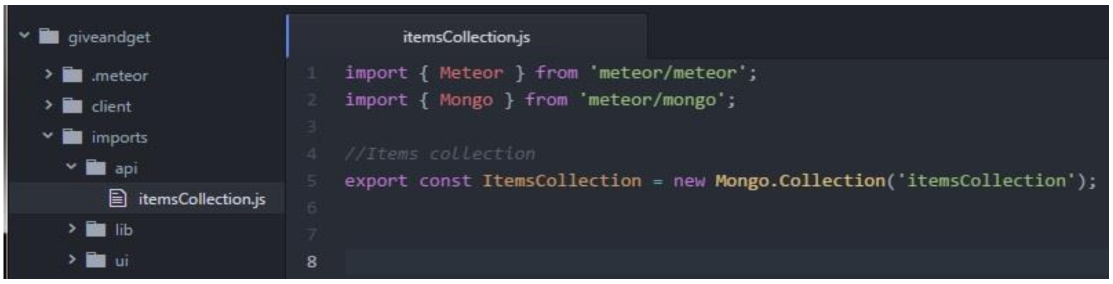

mainbody.html
```
<template name="mainbody">
  {{> navigation}}
  {{> content}}
</template>

<template name="content">
    {{> Template.dynamic template=main}}
</template>
```

Figure 17. Rendering dynamic template.

Figure 17 shows how the content template that contains the dynamic template and is passed to the main body. Meteor dynamic template acts as a placeholder where actual template is loaded by Blaze layout action method. Therefore, whatever is being passed by Blaze action method such as name or templates are taken and assigned to the dynamic template. This enables several templates to be into a single layout, but for the example above only one layout is required.

4.6 Database

Meteor stores data in a collection when using MongoDB. A collection can be created on either on the client or on the server. The collection for these application is created on the client and then imported to the server. The figure 18 show the location where the collection and how it is created on the client, it creates a cache that is connected to the server.

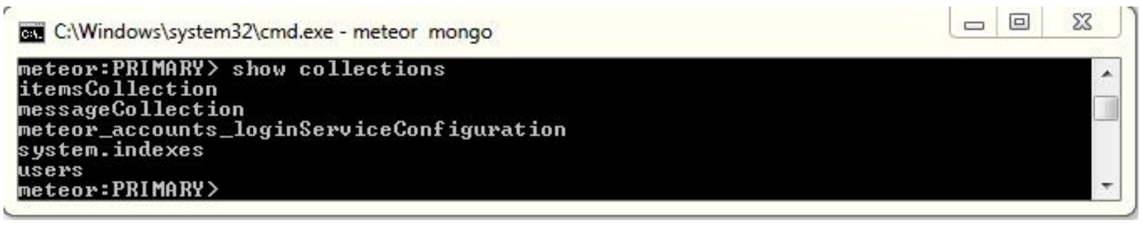


itemsCollection.js
```
import { Meteor } from 'meteor/meteor';
import { Mongo } from 'meteor/mongo';

//Items collection
export const ItemsCollection = new Mongo.Collection('itemsCollection');
```

Figure18 Creating a collection.

The collection can be exported to anywhere that is required within the application. That is why the it is created outside the server and the client because it can be accessed easily in both sections. The most important import of these collection is to the server.

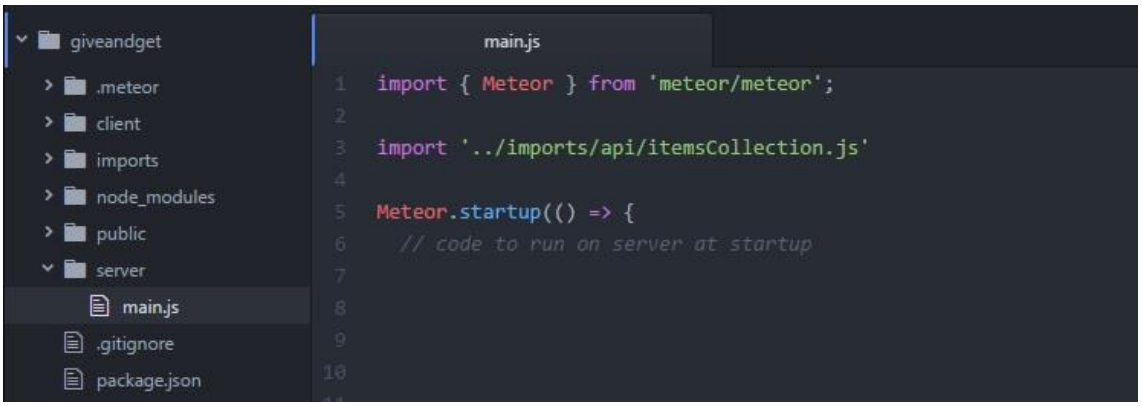
The database structure of the application consists of items, message, account configuration and user's collections. The figure 19 shows the structure of the collections in MongoDB.



```
meteor:PRIMARY> show collections
itemsCollection
messageCollection
meteor_accounts_loginServiceConfiguration
system.indexes
users
meteor:PRIMARY>
```

Figure 19. the collection structure.

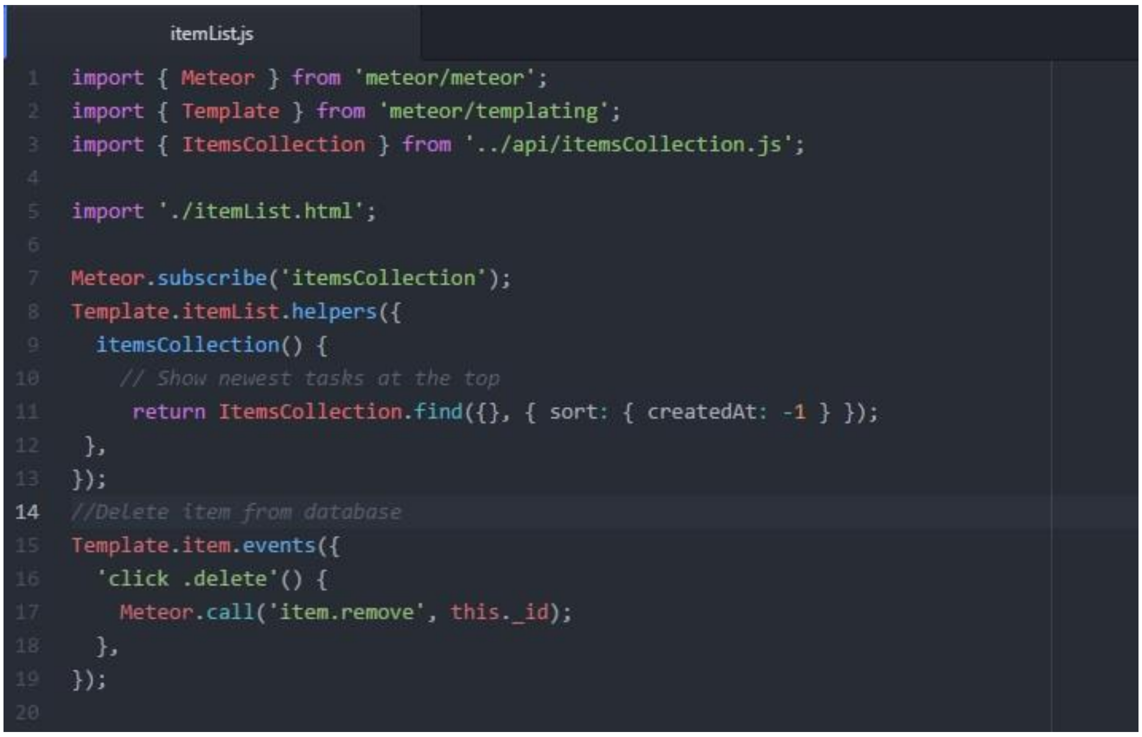
The items collection contains items added as object, message collection contains message conversation. These items and message collection are created but the account configuration and user's collection are created automatically when installing account package. The account configuration contains the authentication files while the user's collection contains the number of users that has the account. These data layers are stored in MongoDB. Through the client console data can be manipulated in the collection. The function such as can find (), fetch (), insert (), update () or remove () helps in manipulating data in a collection.



```
import { Meteor } from 'meteor/meteor';

import '../imports/api/itemsCollection.js'

Meteor.startup(() => {
  // code to run on server at startup
```

Figure 20 Importing a collection to server.

As shown on the figure 20 the server will be able to read the main collection from the API folder on the client. Therefore, if there are any changes to the client collection it will still be reflected on the imported collection on the server because they both share same collection. The syntax of importing a collection to the server and client are different.

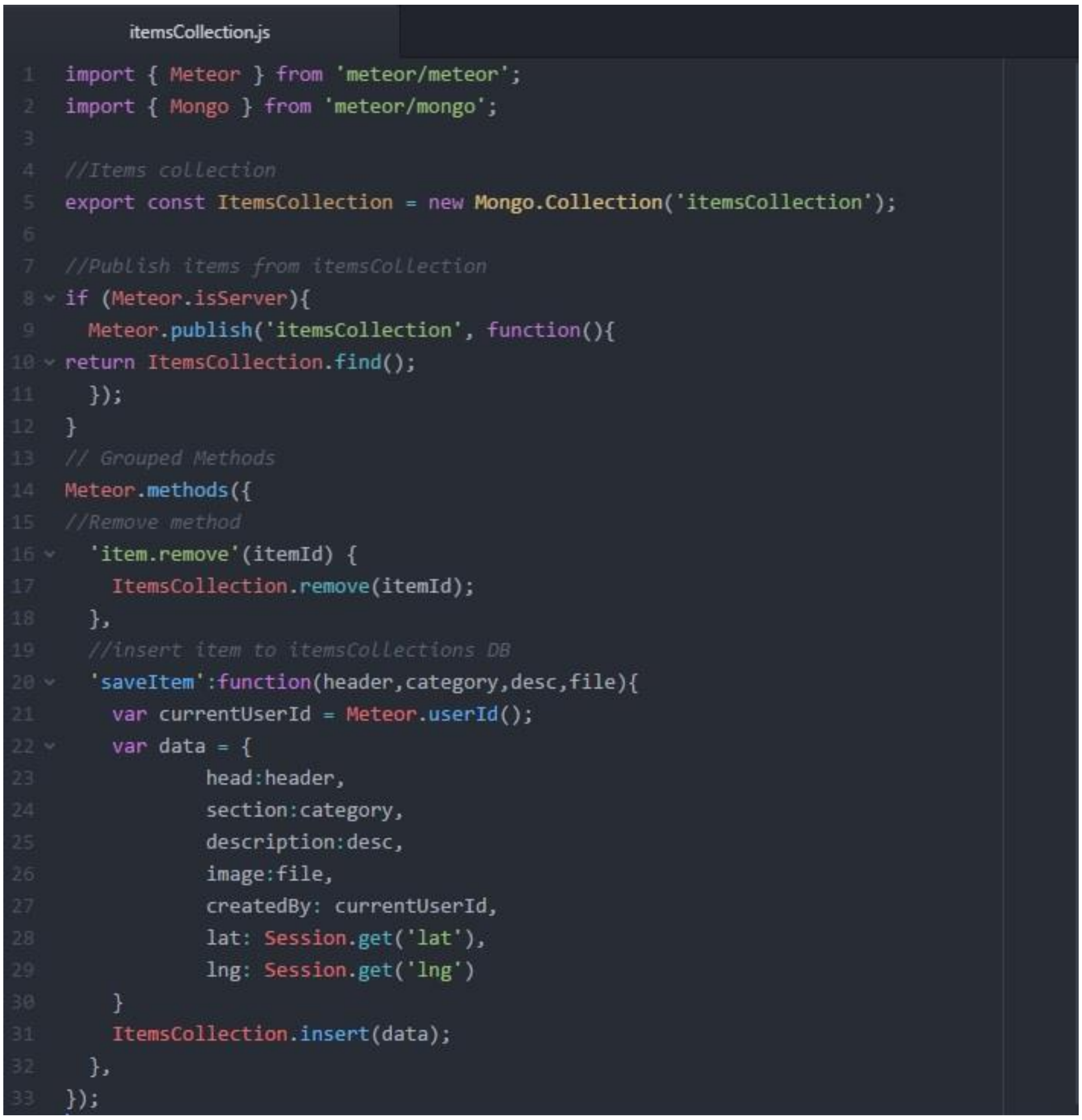


```
import { Meteor } from 'meteor/meteor';
import { Template } from 'meteor/templating';
import { ItemsCollection } from '../api/itemsCollection.js';

import './itemList.html';

Meteor.subscribe('itemsCollection');
Template.itemList.helpers({
  itemsCollection() {
    // Show newest tasks at the top
     return ItemsCollection.find({}, { sort: { createdAt: -1 } });
 },
});
//Delete item from database
Template.item.events({
  'click .delete'() {
    Meteor.call('item.remove', this._id);
  },
});

```

Figure 21 Importing a collection to the client.

There is situation where the collection is required on the client. The figure 21 show how to import a collection, once the template is accessible to a collection the data in the can added, updated, removed or subscribed. Meteor installation consist of default setting such as autopublish and insecure. These enables the user to access the database from the client, hence making the database insecure. Therefore, the safer way to access into the database is through a method. The two default packages i.e. autopublish and insecure are removed from the application.

From the figure 21 to read an object in the database a method find () and name of the collection is used, the user id can also be passed to get specific data created by same user. Similarly, to update or delete an object the event handler is used to call a method that passes a specific id.



```
import { Meteor } from 'meteor/meteor';
import { Mongo } from 'meteor/mongo';

//Items collection
export const ItemsCollection = new Mongo.Collection('itemsCollection');

//Publish items from itemsCollection
if (Meteor.isServer){
  Meteor.publish('itemsCollection', function(){
return ItemsCollection.find();
  });
}
// Grouped Methods
Meteor.methods({
//Remove method
  'item.remove'(itemId) {
    ItemsCollection.remove(itemId);
  },
  //insert item to itemsCollections DB
  'saveItem':function(header,category,desc,file){
    var currentUserId = Meteor.userId();
    var data = {
            head:header,
            section:category,
            description:desc,
            image:file,
            createdBy: currentUserId,
            lat: Session.get('lat'),
            lng: Session.get('lng')
    }
    ItemsCollection.insert(data);
  },
});
```

Figure 22 Running Methods as server on the Client.

The method for finding and removing data in the collection should be in the server, but since the collection in on the client under the import folder, the Meteor.isServer condition is used as shown in figure 22. Therefore, even though the codes are only executed when they are in the server the condition enables the methods to be executed by the server.

4.7 Authentication

After installing Meteor account package, it consists of a basic account functionality which includes login and signup form. The account prevent usage until verification is complete. The UI of both pages had to match with the application as shown in the figure 23. The HTML files are imported to the JavaScript folder which handlers the event handlers.

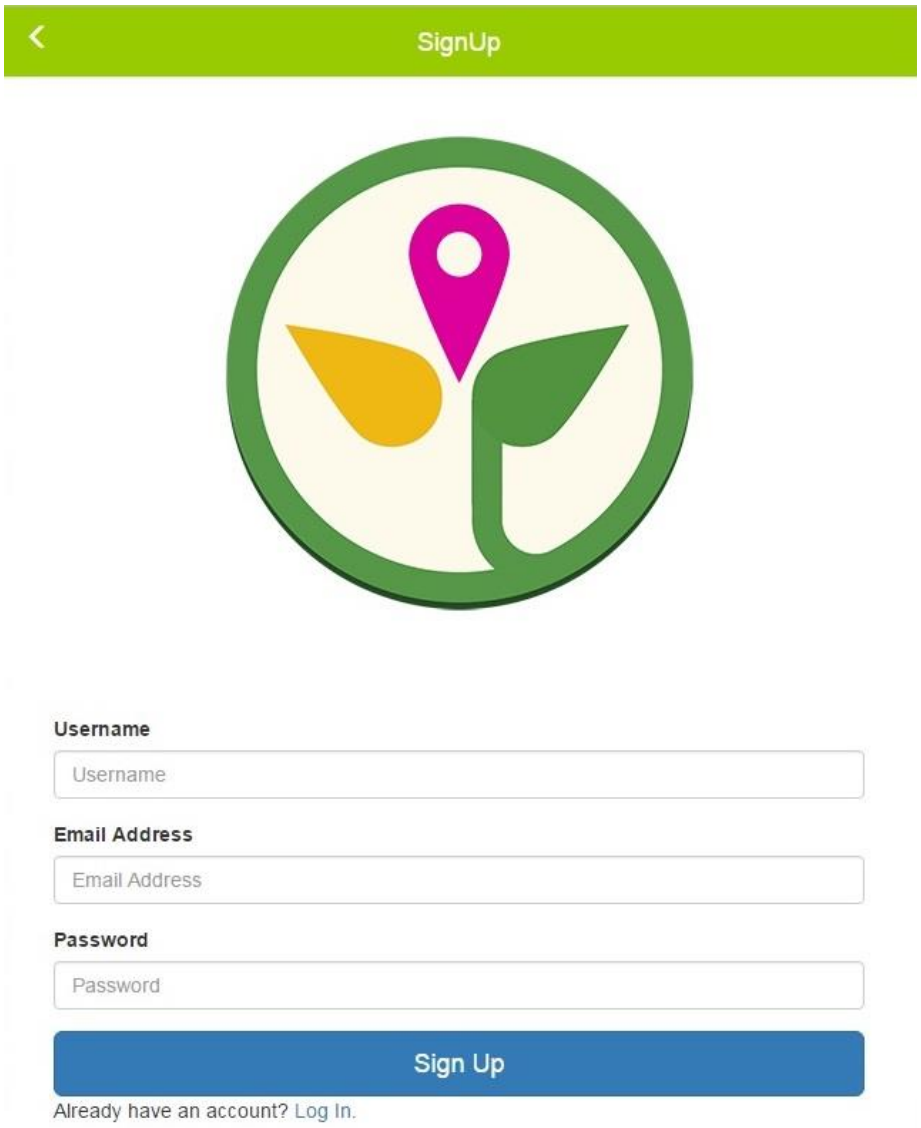
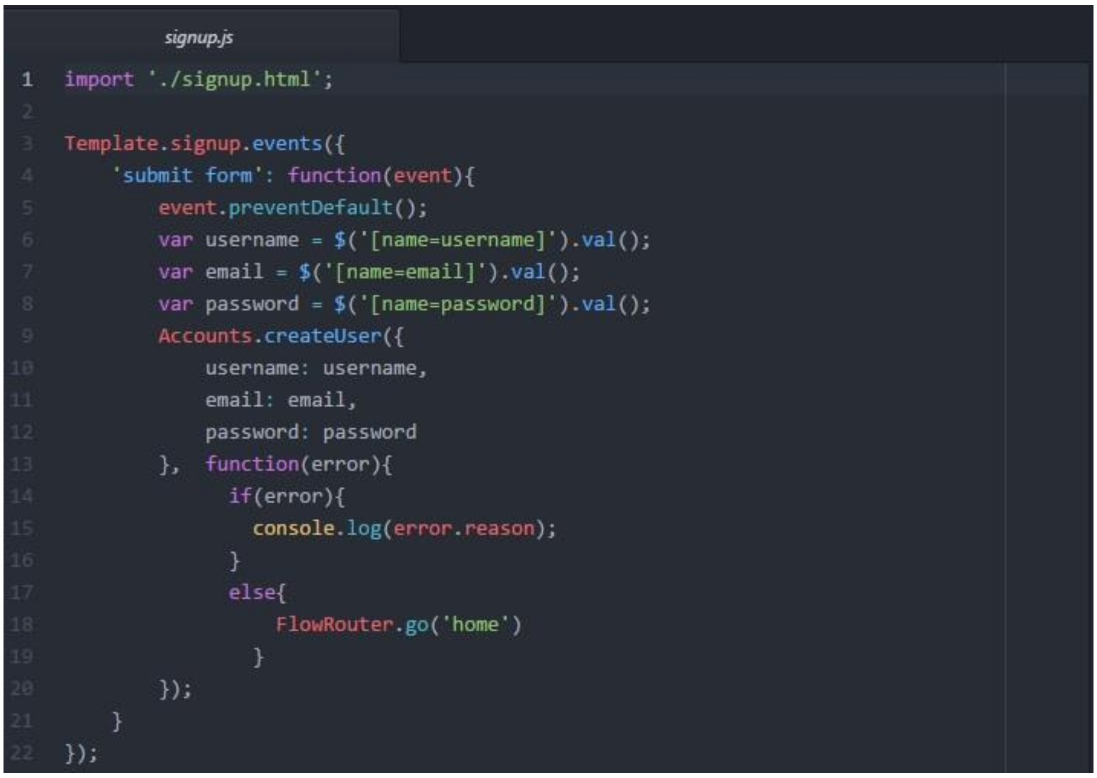


Figure 23 Signup page.

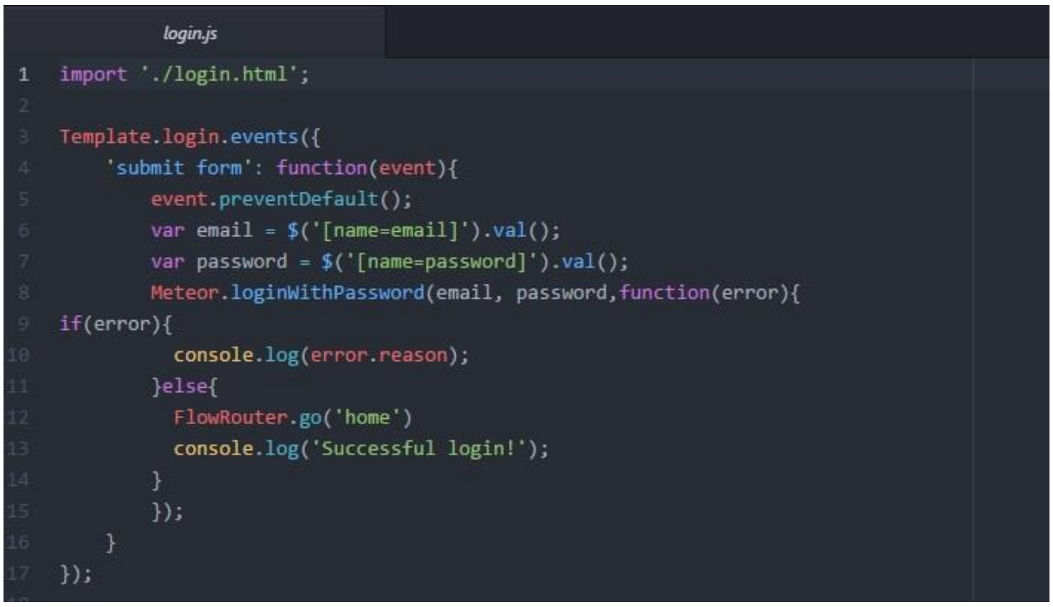
The signup form allows the user to create an account. Figure 24 shows the JavaScript code that allows the user to create an account. When the user submits the username, email and password the event handler passes the information to a method in the account package. Account.creatUser method is responsible for creating the user. If there are any errors the method is called back and the error message is displayed to the user and there is no error the user is guided to home page through with router.


signup.js

```
import './signup.html';

Template.signup.events({
    'submit form': function(event){
        event.preventDefault();
        var username = $('[name=username]').val();
        var email = $('[name=email]').val();
        var password = $('[name=password]').val();
        Accounts.createUser({
            username: username,
            email: email,
            password: password
        },  function(error){
              if(error){
                console.log(error.reason);
              }
              else{
                 FlowRouter.go('home')
                }
        });
    }
});
```

Figure 24 Creating user account.

During login, the user is required to submit the email and password. As shown in figure 25, the event handler passes the user details to be verified by Meteor.loginwithPassword method.


login.js

```
import './login.html';

Template.login.events({
    'submit form': function(event){
        event.preventDefault();
        var email = $('[name=email]').val();
        var password = $('[name=password]').val();
        Meteor.loginWithPassword(email, password,function(error){
if(error){
          console.log(error.reason);
        }else{
          FlowRouter.go('home')
          console.log('Successful login!');
        }
        });
    }
});
```

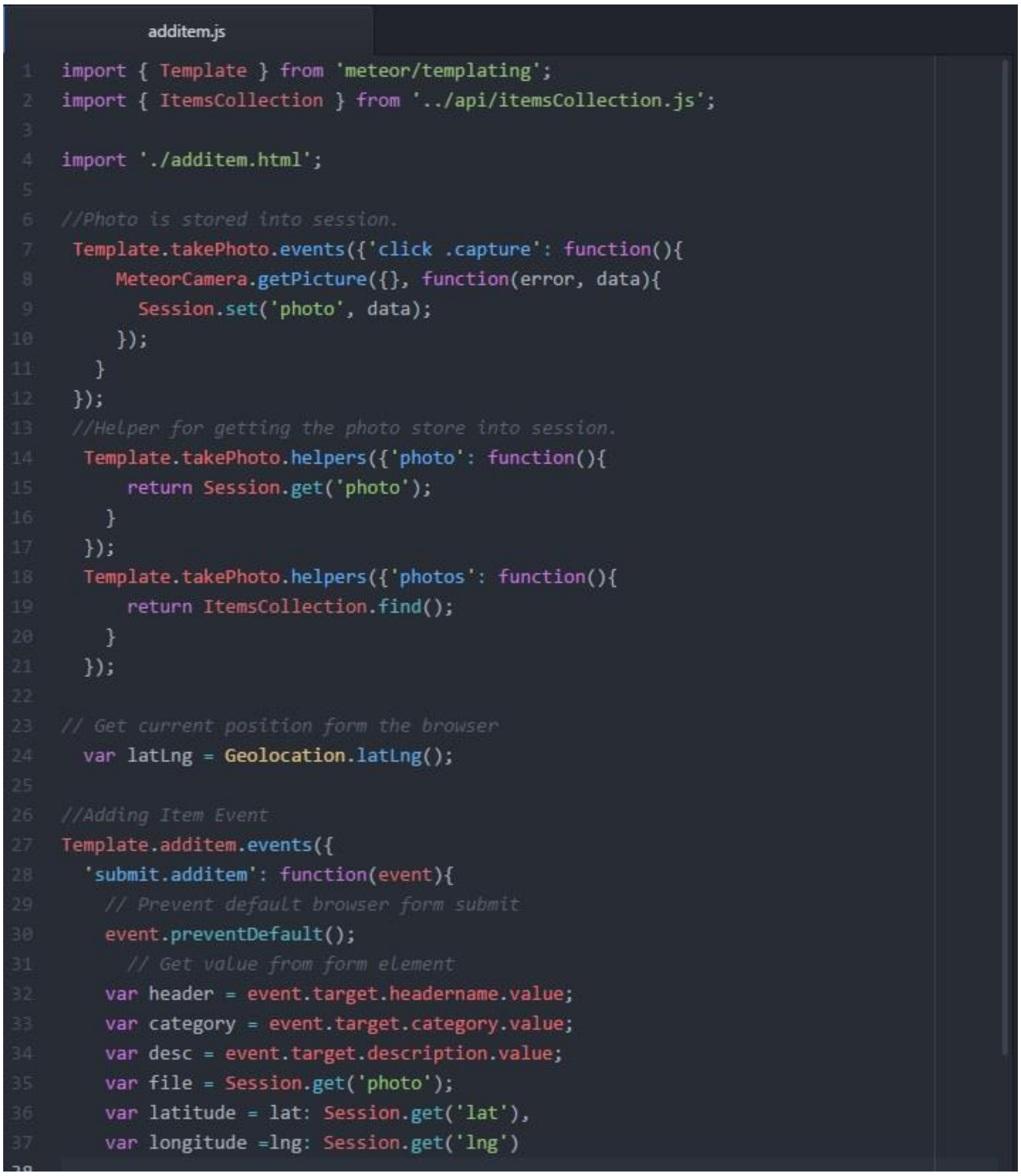
Figure 25 User Login.

The method checks if the user details are available. The user is directed to the home page if the verification is successful else error is sent to the user. The account also is used to authorize the user with the privileges of adding and deleting an item to the application and the ability to send a message.

4.8 Camera

The implementation of the camera functionality begins with the installation of mdg camera package from athmosphere.js. The installation enables setting the connection between the camera and the application. The figure 26 show the events listeners and setting photo into session

The event handler initiate the method which captures the photo and stores it in a session through a session. Set. The photo is stored in a session because there is other information that are still related to the image such as header, description and category. Therefore, once the all information is gathered they will be inserted to the collection and a document is created with primary key. The helper session. Get enables to retrieved the photo from the session before inserting into the database. The photo is cleared when the event handler submits and the item is inserted to the collection. Once all the information about the item is gathered, it is inserted to the item collection as a (Extended JavaScript Object Notation) EJSON object. The image is stored as based64 into the database.

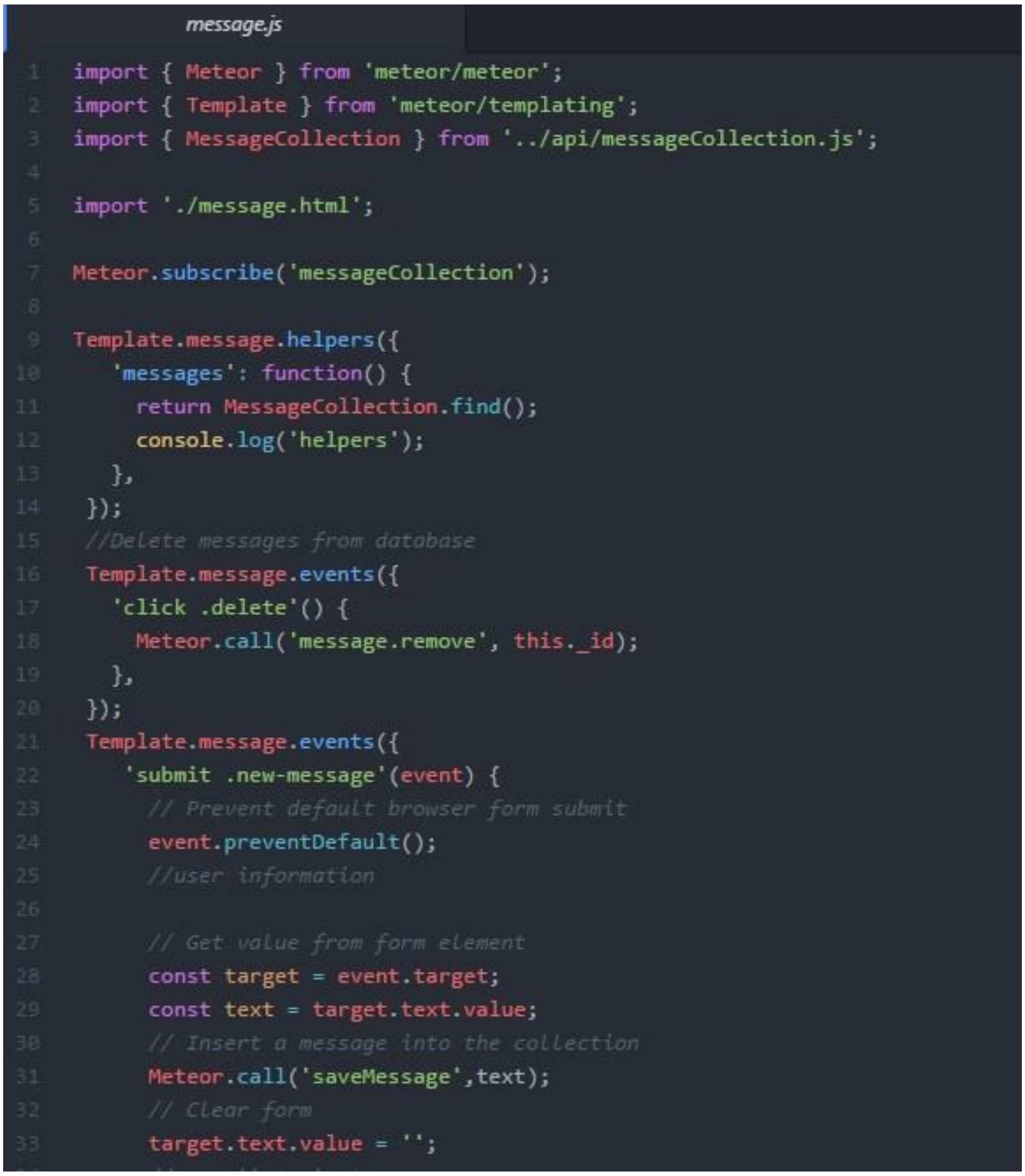


```
import { Template } from 'meteor/templating';
import { ItemsCollection } from '../api/itemsCollection.js';

import './additem.html';

//Photo is stored into session.
 Template.takePhoto.events({'click .capture': function(){
     MeteorCamera.getPicture({}, function(error, data){
       Session.set('photo', data);
     });
   }
 });
 //Helper for getting the photo store into session.
  Template.takePhoto.helpers({'photo': function(){
      return Session.get('photo');
    }
  });
  Template.takePhoto.helpers({'photos': function(){
      return ItemsCollection.find();
    }
  });

// Get current position form the browser
  var latLng = Geolocation.latLng();

//Adding Item Event
Template.additem.events({
  'submit.additem': function(event){
    // Prevent default browser form submit
    event.preventDefault();
      // Get value from form element
    var header = event.target.headername.value;
    var category = event.target.category.value;
    var desc = event.target.description.value;
    var file = Session.get('photo');
    var latitude = lat: Session.get('lat'),
    var longitude =lng: Session.get('lng')
```

Figure 26 Setting up the Camera.

4.9 Instant messaging

Instant messaging requires a collection to store, therefore creating a collection in the api folder would be the first thing to consider. When the user sends the text the template event reads the value in the text field. The value is then saved into the collection created earlier. The value is save through a method. The method is created on the server side. Accessing the message collection from the server is more secure than on the client. Once the value is save the field is Cleared.



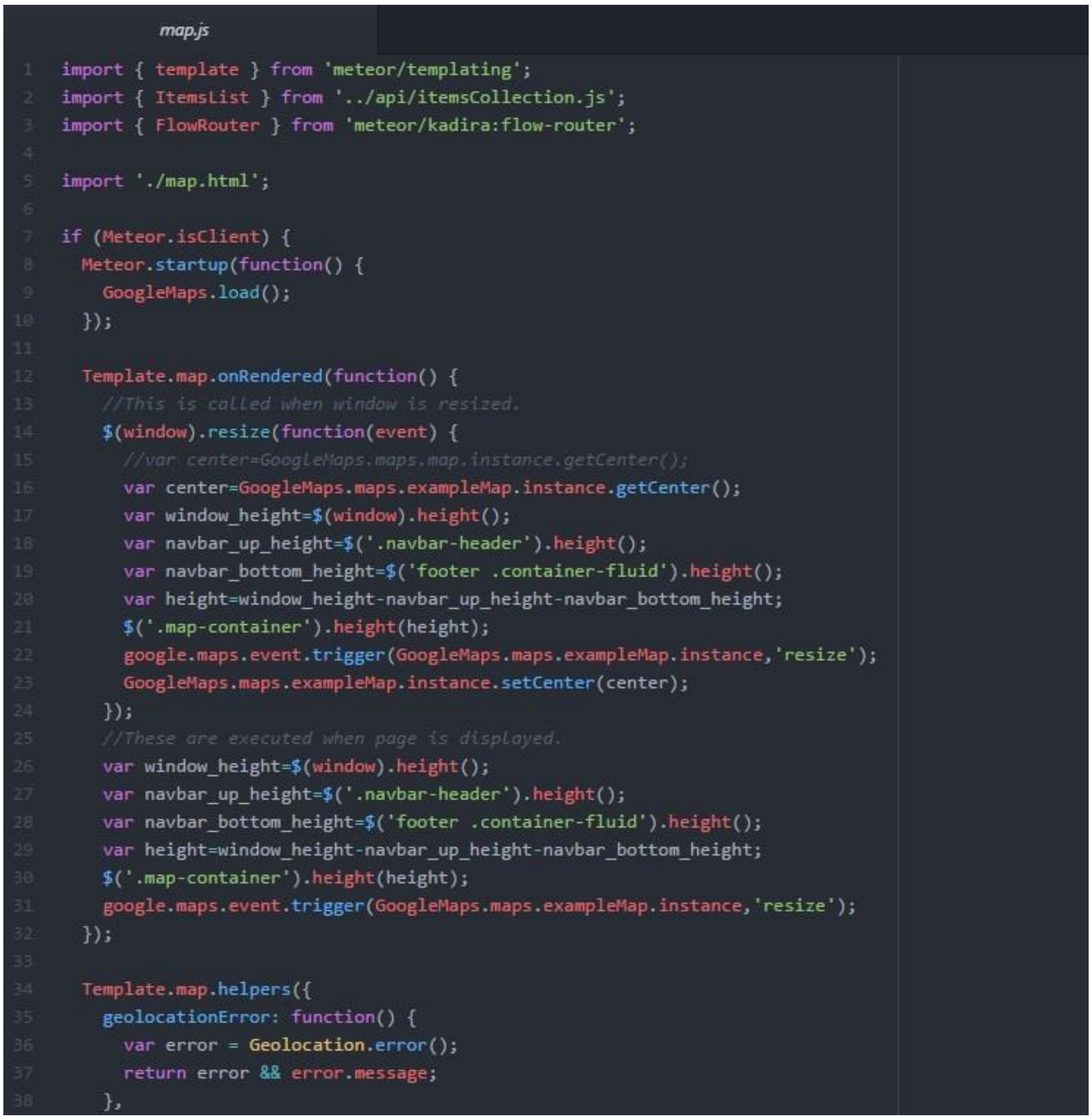
```
message.js
import { Meteor } from 'meteor/meteor';
import { Template } from 'meteor/templating';
import { MessageCollection } from '../api/messageCollection.js';

import './message.html';

Meteor.subscribe('messageCollection');

Template.message.helpers({
   'messages': function() {
     return MessageCollection.find();
     console.log('helpers');
   },
 });
 //Delete messages from database
 Template.message.events({
   'click .delete'() {
     Meteor.call('message.remove', this._id);
   },
 });
 Template.message.events({
    'submit .new-message'(event) {
      // Prevent default browser form submit
      event.preventDefault();
      //user information

      // Get value from form element
      const target = event.target;
      const text = target.text.value;
      // Insert a message into the collection
      Meteor.call('saveMessage',text);
      // Clear form
      target.text.value = '';
```

Figure 27 Instant Messaging.

Figure 27 show how the event listeners capture the text from text field, delete events and helpers that retrieve data from the database. The data from the message collection is published in the message collection and subscribed so that the text message could be displayed in the UI through handlebars.

4.10 Map

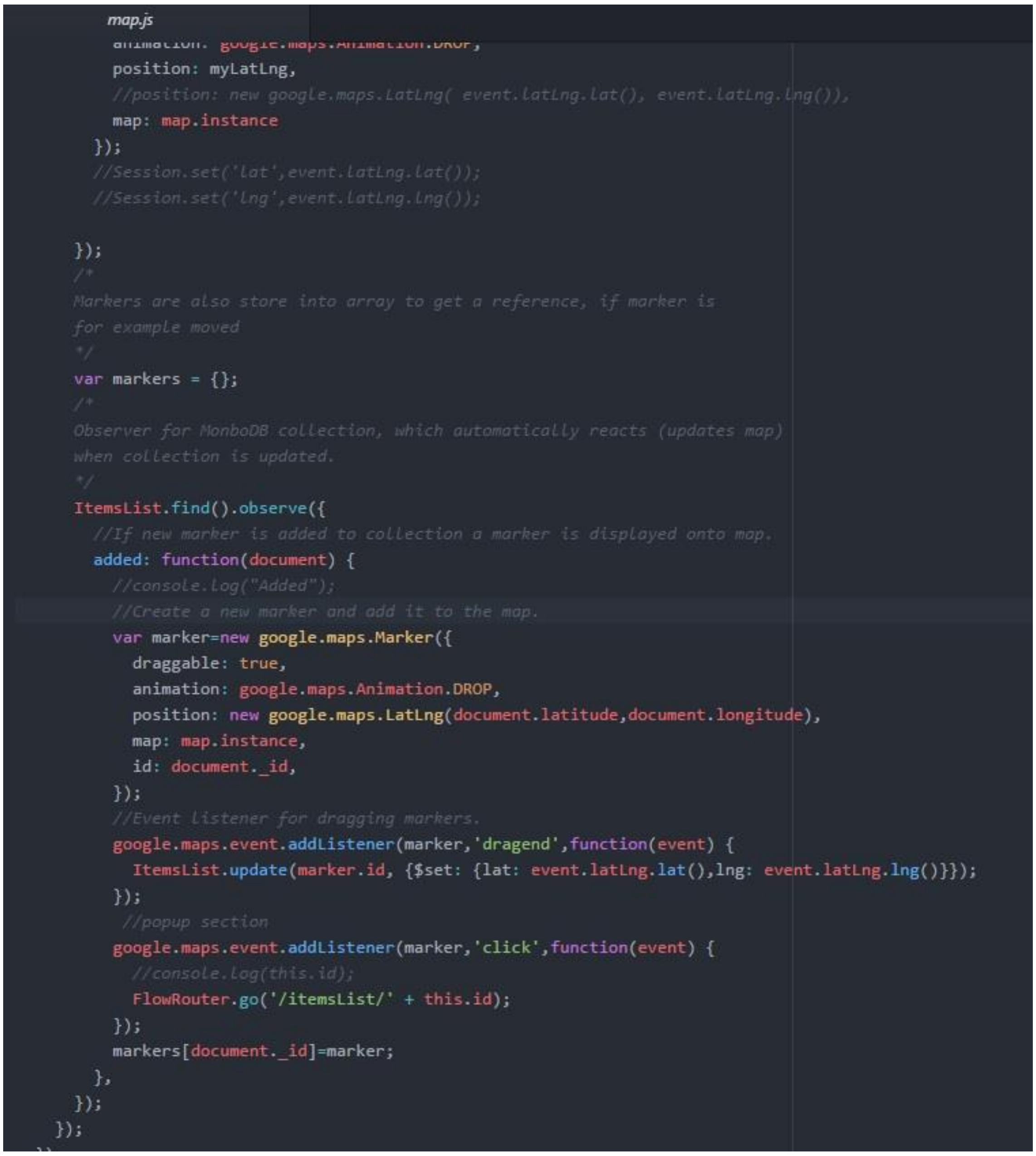
The Google map is loaded upon startup through GoogleMaps.load() function. From the figure 28 the size and center of the map is described, this enables the map to load fully on any displaying page. The map gets the latitude and longitude coordinates of current position to display the map from the current location. The collection that contains the data collected from the add section is imported.



```
map.js
import { template } from 'meteor/templating';
import { ItemsList } from '../api/itemsCollection.js';
import { FlowRouter } from 'meteor/kadira:flow-router';

import './map.html';

if (Meteor.isClient) {
  Meteor.startup(function() {
    GoogleMaps.load();
  });

  Template.map.onRendered(function() {
    //This is called when window is resized.
    $(window).resize(function(event) {
      //var center=GoogleMaps.maps.map.instance.getCenter();
      var center=GoogleMaps.maps.exampleMap.instance.getCenter();
      var window_height=$(window).height();
      var navbar_up_height=$('.navbar-header').height();
      var navbar_bottom_height=$('footer .container-fluid').height();
      var height=window_height-navbar_up_height-navbar_bottom_height;
      $('.map-container').height(height);
      google.maps.event.trigger(GoogleMaps.maps.exampleMap.instance,'resize');
      GoogleMaps.maps.exampleMap.instance.setCenter(center);
    });
    //These are executed when page is displayed.
    var window_height=$(window).height();
    var navbar_up_height=$('.navbar-header').height();
    var navbar_bottom_height=$('footer .container-fluid').height();
    var height=window_height-navbar_up_height-navbar_bottom_height;
    $('.map-container').height(height);
    google.maps.event.trigger(GoogleMaps.maps.exampleMap.instance,'resize');
  });

  Template.map.helpers({
    geolocationError: function() {
      var error = Geolocation.error();
      return error && error.message;
    },
```

Figure 28 Map.

This will enable the creating of the marker. Find and observe method checks the collection for the items individually and get the longitude and latitude coordinates as shown in figure 29. These coordinates enable the creation of a marker. Therefore, every time an item is added a marker is created based on those coordinates. The map also has an event listener. The marker is clicked uses the id of the marker and refers to the item in the collection from which the marker originated. Therefore, the item is displayed on the item list.



```
      animation: google.maps.Animation.DROP,
      position: myLatLng,
      //position: new google.maps.LatLng( event.latLng.lat(), event.latLng.lng()),
      map: map.instance
    });
    //Session.set('lat',event.latLng.lat());
    //Session.set('lng',event.latLng.lng());

  });
  /*
  Markers are also store into array to get a reference, if marker is
  for example moved
  */
  var markers = {};
  /*
  Observer for MonboDB collection, which automatically reacts (updates map)
  when collection is updated.
  */
  ItemsList.find().observe({
    //If new marker is added to collection a marker is displayed onto map.
    added: function(document) {
      //console.log("Added");
      //Create a new marker and add it to the map.
      var marker=new google.maps.Marker({
        draggable: true,
        animation: google.maps.Animation.DROP,
        position: new google.maps.LatLng(document.latitude,document.longitude),
        map: map.instance,
        id: document._id,
      });
      //Event listener for dragging markers.
      google.maps.event.addListener(marker,'dragend',function(event) {
        ItemsList.update(marker.id, {$set: {lat: event.latLng.lat(),lng: event.latLng.lng()}});
      });
       //popup section
      google.maps.event.addListener(marker,'click',function(event) {
        //console.log(this.id);
        FlowRouter.go('/itemsList/' + this.id);
      });
      markers[document._id]=marker;
    },
  });
});
```

Figure 29 Creating a marker.

5 DISCUSSION

The objective of the thesis was to create a demo for learning purposes by developing a web application using Meteor.JS framework. The main requirement was to implement the functionality of the application. The other requirement was to create a responsive web application to enable a wide range of usage. Developing a native application is quite complex and requires more time, therefore the use of framework was considered. The benefit of framework is that it speeds up the development process. Apart from Meteor.JS there are other framework such as Ionic that can also be used to develop a web application.

Among other framework, Meteor.JS environment support the developers to accomplish the development process quickly also, embracing file structure and the use of latest technologies result into a better and reliable end-product. MongoDB is quite easy to use unlike the relational database. Packages such as Bootstrap and node.JS provides quick development of both front-end and back-end. Blaze.js and Flow-router is easy to implement especially for someone familiar with JavaScript. These technologies contribute largely to the development process also enable a quality product. For a beginner, it might be difficult to develop a product with less faults.

The development process provided an opportunity for learning and gaining experience. The use of different technologies recommended by Meteor framework increases the knowledge on JavaScript,Node.js MongoDB and Bootstrap also by using these technologies it enable one to understand how they connect and relate to one another. The documentation provided more information regarding the use and the best practice. The experience from the development process improves programming skills. Therefore, at the end of the process using other framework related to Meteor.JS can be much easier.

The whole process of these thesis report can be a challenge especially for one who is not familiar with programming skills. The guidance from the documentation including sources from internet provided great assistance in the theory and development process. Meteor.JS framework proofed as easy and quick for development process. Software frameworks is beneficial to anyone who want to learn and familiarize with the concept of software development. Therefore, these demo could be used as a study material during lectures or even for self-study, The demo consist of several functionalities, these provides a wide scope of learning material.

Regarding the future development of the application, security issues could still be improved. Currently a user can enter any value or submit and empty value to the application when adding a new item. These has to be checked whether the value has to be string, Boolean or integer so that the user does not give false information. Another improvement is the authentication, these application uses default authentication which is installed from account package. Meteor default authentication does not verify if the email address is legitimate.

REFERENCES

Blaze 2017a. Introduction. Cited 15.3.2017, http://blazejs.org/guide/introduction.html

Berners-Lee, T. 2005. Uniform Resource Identifier (URI): Generic Syntax. Cited 27.3.2017, https://tools.ietf.org/html/rfc3986

Hochhaus, S. & Schoebel, M. 2015a. Meteor in action. The story behind Meteor. Cited 13.3.2017, https://manning-content.s3.amazonaws.com/download/4/47f88a7-bff9-4144-b834-b5ecbe6c0c17/sample_ch01_Hochhaus_Meteor.pdf

Hochhaus, S. & Schoebel, M. 2015b. Meteor in action. The Meteor stack. Cited 15.3.2017, https://manning-content.s3.amazonaws.com/download/4/47f88a7-bff9-4144-b834-b5ecbe6c0c17/sample_ch01_Hochhaus_Meteor.pdf

Hochhaus, S. & Schoebel, M. 2015c. Meteor in action. Processing in the browser: running on distributed platforms. Cited 10.3.2017, https://manning-content.s3.amazonaws.com/download/4/47f88a7-bff9-4144-b834-b5ecbe6c0c17/sample_ch01_Hochhaus_Meteor.pdf

Hochhaus, S. & Schoebel, M. 2015d. Meteor in action. Core projects. Cited 2.3.2017, https://manning-content.s3.amazonaws.com/download/4/47f88a7-bff9-4144-b834-b5ecbe6c0c17/sample_ch01_Hochhaus_Meteor.pdf

Hochhaus, S. & Schoebel, M. 2015d. Meteor in action. Core projects, Tracker. Cited 6.3.2017, https://manning-content.s3.amazonaws.com/download/4/47f88a7-bff9-4144-b834-b5ecbe6c0c17/sample_ch01_Hochhaus_Meteor.pdf

Meteor 2017a. Atmosphere vs. npm. Cited 21.02.2017, https://guide.meteor.com/atmosphere-vs-npm.html

Meteor 2017b. URLs and Routing. Client-side Routing. Cited 12.3.2017, https://guide.meteor.com/routing.html#flow-router

Meteor 2017c. Collection and Schemas. Cited 12.3.2017, https://guide.meteor.com/collections.html#server-collections

OamkLabs 2017. Oamk LABs Research Cited 15.02.2017, http://www.oamklabs.fi/research/

Robinson Josh, Gray Aaron, Titarenco David 2016. Introduction to Meteor, MongoDB and NoSQL. Cited 6.4.2017, http://proquest.safaribooksonline.com.ezp.oamk.fi:2048/book/web-development/9781430268352

Sahand Sadjedee, 2013. Meteor framework, a new approach to web development: an experimental analysis. http://liu.diva-portal.org/smash/get/diva2:691485/FULLTEXT01.pdf

Strack, I. 2015a. Meteor cookbook. Introduction. Cited 15.3.2017, http://proquest.safaribooksonline.com.ezp.oamk.fi:2048/book/web-development/web-services/9781783280292/2dot-customizing-with-packages/ch02lvl1sec22_html

Strack, I. 2015b. Meteor cookbook. Adding Meteor packages. Cited 29.3.2017, http://proquest.safaribooksonline.com.ezp.oamk.fi:2048/book/web-development/web-services/9781783280292/2dot-customizing-with-packages/ch02lvl1sec22_html#X2ludGVybmFsX0h0bWxWaWV3P3htbGlkPTk3ODE3ODMyODAyOTIlMkZjaDAyczAyX2h0bWwmcXVlcnk9

Strack, I. 2015c. Meteor cookbook. Inserting templates with Spacebars. Cited 10.4.2017, http://proquest.safaribooksonline.com.ezp.oamk.fi:2048/book/web-development/web-services/9781783280292/3dot-building-great-user-interfaces/ch03s02_html