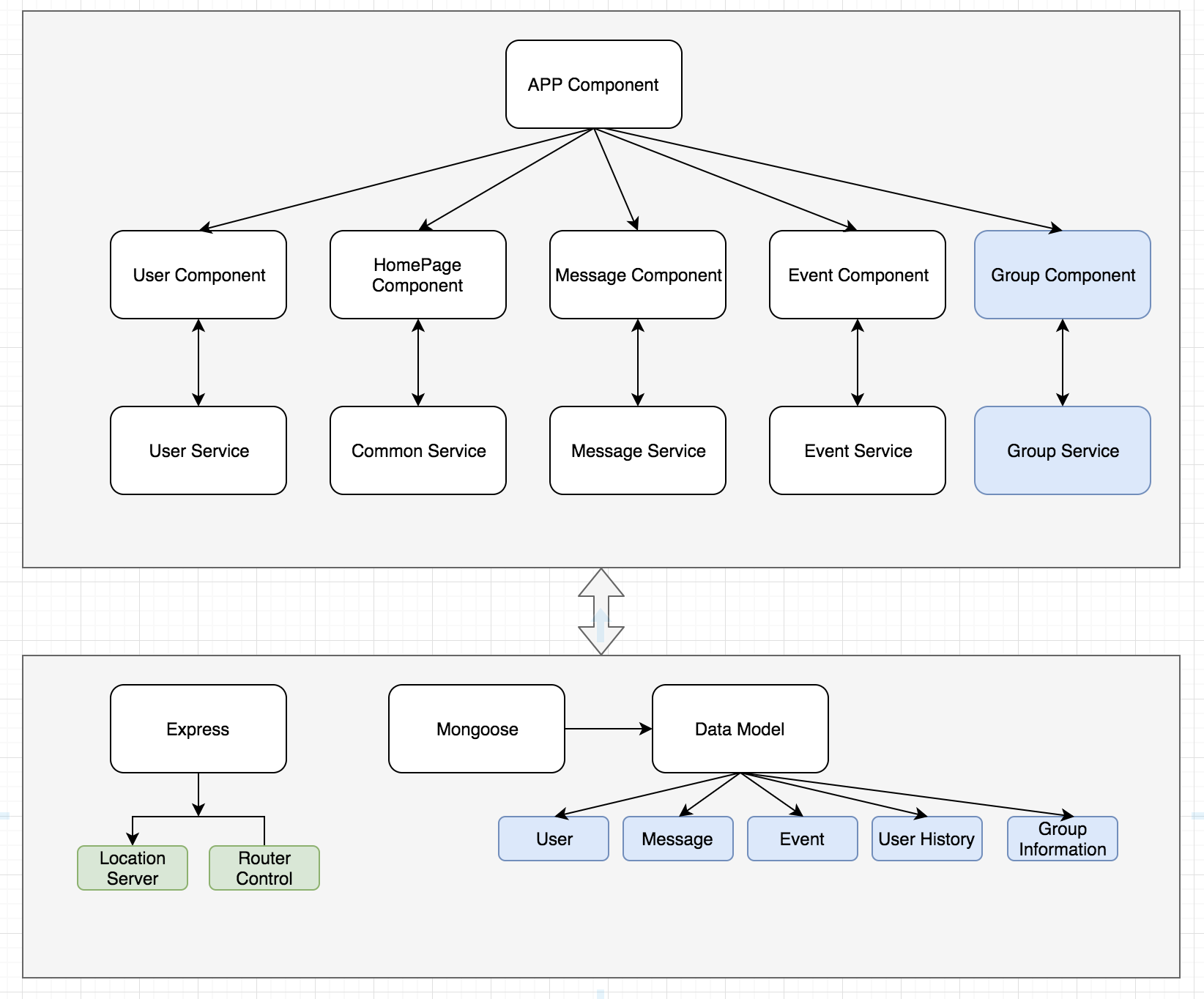
# **TravelMATE**

This community application includes Users, Messages, Events and Groups components. The component Users is for user login/logout, and component Events provides the contents which is shared by users or the information of login/logout, creating/updating/deleting groups. The Messages component has the functionalities to communicate between all users. The Group component also provides community collaboration which for user to create, update or delete travel groups that have same trip destinations. Here is the architecture design and data models.



There would be five or more Components/Views:

* Home Component

This component is to display all information includes users, messages, events and groups.

* User Component

To show all user action histories such as login, logout, join etc user related information. The user's location also will be stored in mongo.

* Message Component

This component is to show messages that users in this community communicated. Generally, the message in this view is read only, but it can be updated or deleted by the owner who generate this message. An 'x' will be shown on the right of each message which is created by the owner.

* Event Component

To show all events that users want to share with others. Same as message component, the event can be removed by the owner.

* Login Component

Very simple implementation with login/register together. A new user will be generated if this user name doesn't exist in our system, otherwise it will be logged in automatically.

* Group Information Component

This component provides some actions for user to create, delete, join and quit from the group. All actions will be sent to Event Component and generate an event immediately.

* Common Service

This service will provide or keep some informaition that could be shared by all components. One of them is the geo location. Just simplely using [ipinfo.io](http://ipinfo.io/) to fetch user's current location information.

getLocation() {

return (

this.http

.get('https://ipinfo.io/json')

.pipe(map((response) => response || {}))

.subscribe((res) => {

console.log('currecnt location: ', res);

this.currentLocation = res;

}),

(err) => {

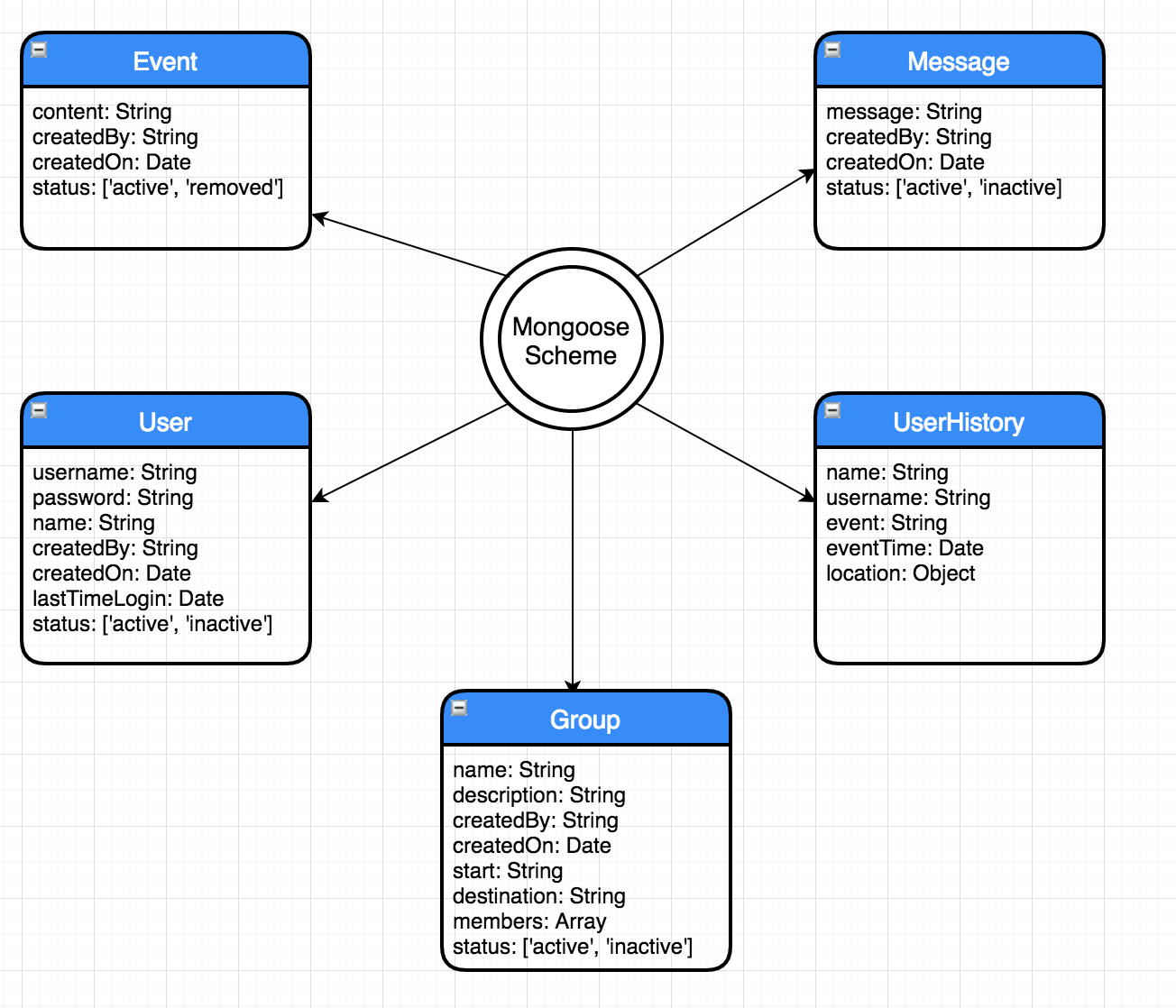
console.log('add user error:', err);

}

);

}

Here is the data model design.



There are multiple data models used in this application.

### **Event**

let Event = new Schema(

{

content: {

type: String,

required: true,

},

createdBy: {

type: String,

},

createdOn: {

type: Date,

default: Date.now,

},

status: {

type: String,

enum: ['active', 'removed'],

},

},

{

collection: 'event',

},

);

### **User**

let User = new Schema(

{

username: {

type: String,

required: true,

},

password: {

type: String,

required: true,

},

name: {

type: String,

default: this.username,

},

createdBy: {

type: String,

default: 'admin',

},

createdOn: {

type: Date,

default: Date.now,

},

status: {

type: String,

enum: ['active', 'inactive'],

default: 'active',

},

lastLogin: {

type: Date,

default: Date.now,

},

},

{

collection: 'user',

},

);

### **Message**

let Message = new Schema(

{

message: {

type: String,

required: true,

},

createdBy: {

type: String,

},

createdOn: {

type: Date,

default: Date.now,

},

status: {

type: String,

enum: ['active', 'removed'],

},

},

{

collection: 'message',

},

);

### **UserHistory**

let UserHistory = new Schema(

{

name: {

type: String,

required: true,

},

username: {

type: String,

},

event: {

type: String,

},

eventTime: {

type: Date,

default: Date.now,

},

location: {

type: Object,

},

},

{

collection: 'userHistory',

},

);

### **Group Information**

let Group = new Schema(

{

name: {

type: String,

required: true,

unique: true,

},

description: {

type: String,

required: true,

},

start: {

type: String,

required: true,

},

destination: {

type: String,

required: true,

},

members: {

type: Array,

},

createdBy: {

type: String,

},

createdOn: {

type: Date,

default: Date.now,

},

status: {

type: String,

enum: ['active', 'removed'],

},

},

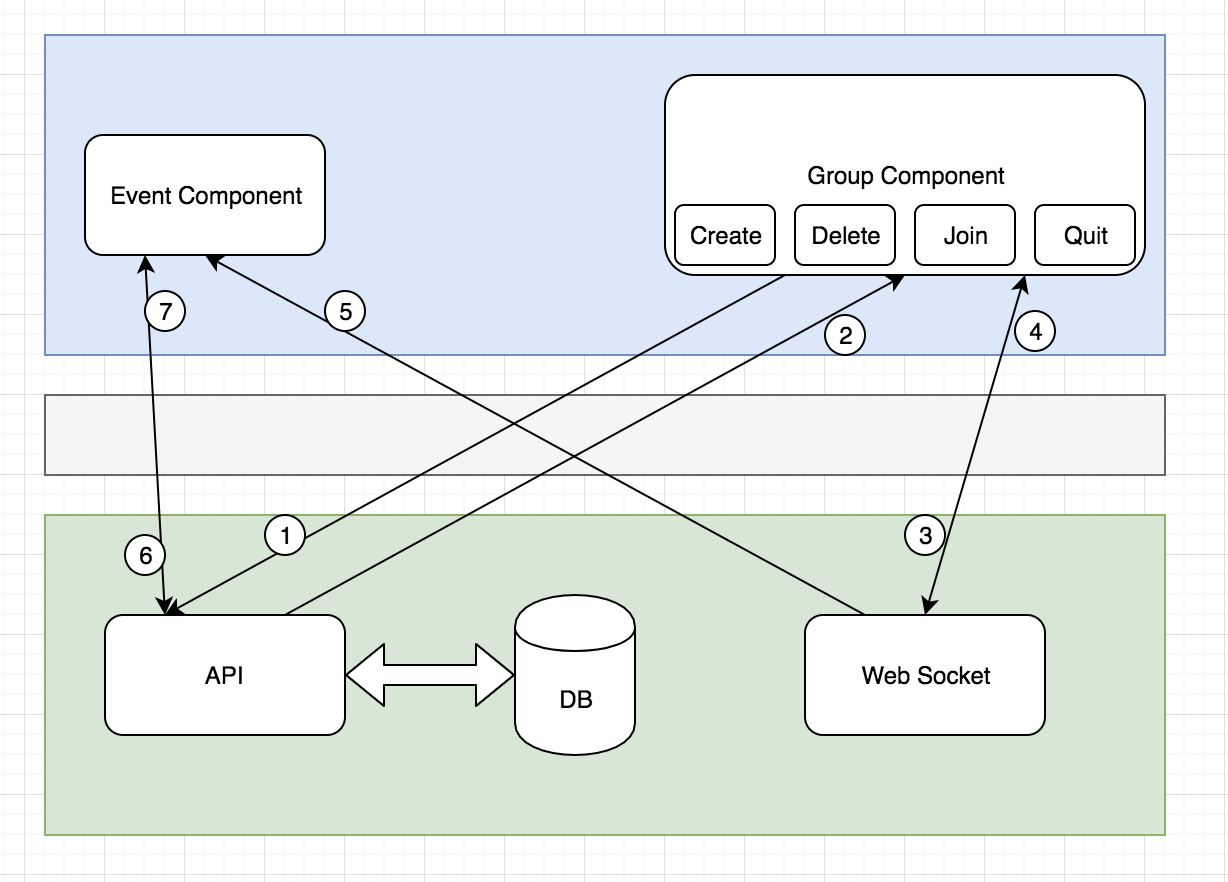
{

collection: 'group',

},

);

This application also uses websocket to communicate between front-end and backend.



Every time any action that apply to group, it will also add an event data to database. Here are the sequences that happens between group and event.

* User click create, delete, join or quit
* Group Component sends a HTTP request to API server
* API server returns response to Group Component
* Group Component sends a message to Web Socket
* Web Socket broadcasts a message to Event Component
* Event Component receives the message
* Event Component sends a HTTP request to API server
* API server processes the request
* API server sends the response back to Event Component
* Event Component shows event list

This community application uses MEAN stack and several third-party libraries. The front-end modules are implemented by Angular 6, while backend server uses Node.js version 11.0 and express framework. All data are stored in MongoDB with designed data models.