Assignment 4 report Explaining the functionality and the implementation of the project:

MongoDB database:

- For incorporating MongoDB in our web application, we have used “mongoose”, which is a MongoDB object modeling tool designed to work in an asynchronous environment.
- It can be installed by executing the following command in the root directory of our application in the command window.

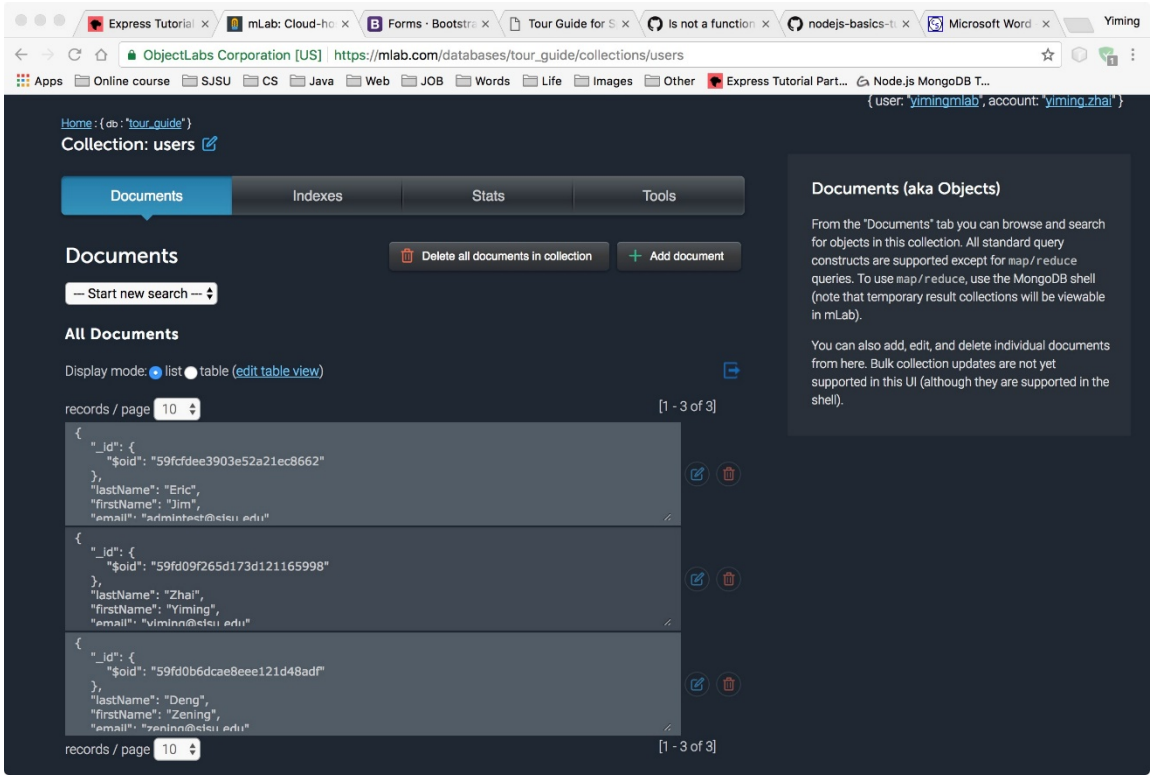
  `npm install mongoose`
- Connection to database using mongoose has been implemented as shown in the following figure:

```
var dbConfig = require('./db');
var mongoose = require('mongoose');
// Connect to DB
mongoose.connect(dbConfig.url);
```

- Online Database Host

```
module.exports = {
    'url' : 'mongodb://minglei:minglei@ds243295.mlab.com:43295/tour_guide'
    //'url' : 'mongodb://127.0.0.1:27017'
}
```

- Online Database



- User Schema: This schema consists of details of a registered user. The user details that are stored are: Username, password, email, first name of user and last name of user.

```
var mongoose = require('mongoose');

module.exports = mongoose.model('User',{
    id: String,
    username: String,
    password: String,
    email: String,
    firstName: String,
    lastName: String
});
```

- Review Schema: This schema has been designed for CRUD operations related to the reviews given by a user for a place he or she visits. It consists of Username (of the user giving the review), review given (the content), creation date (date when a new comment was given by this user), update date (the date on which a user updated one of their previously given comments).

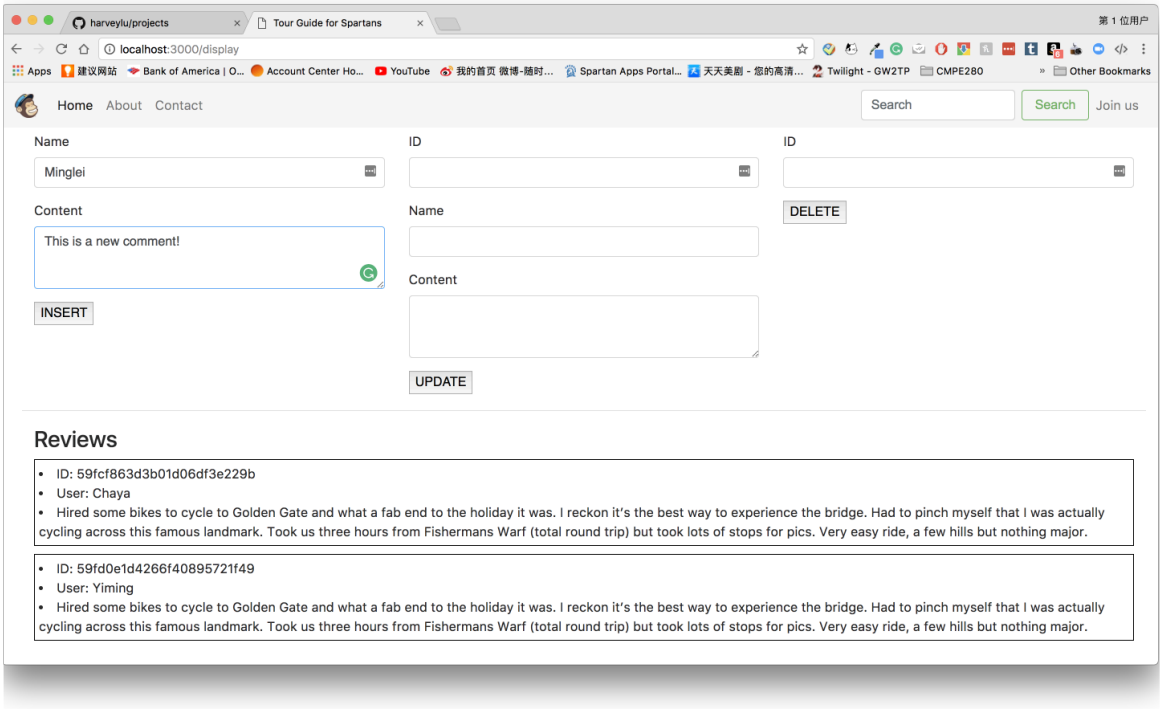
```
var mongoose = require('mongoose');

var UserReview = mongoose.model('UserReview',{
    name: {
        type: String,
        required: true
    },
    content: String,
    create_date: {
        type: Date,
        default: Date.now
    },
    update_date: {
        type: Date,
        default: Date.now
    }
});

module.exports = UserReview;
```

RESTful API:

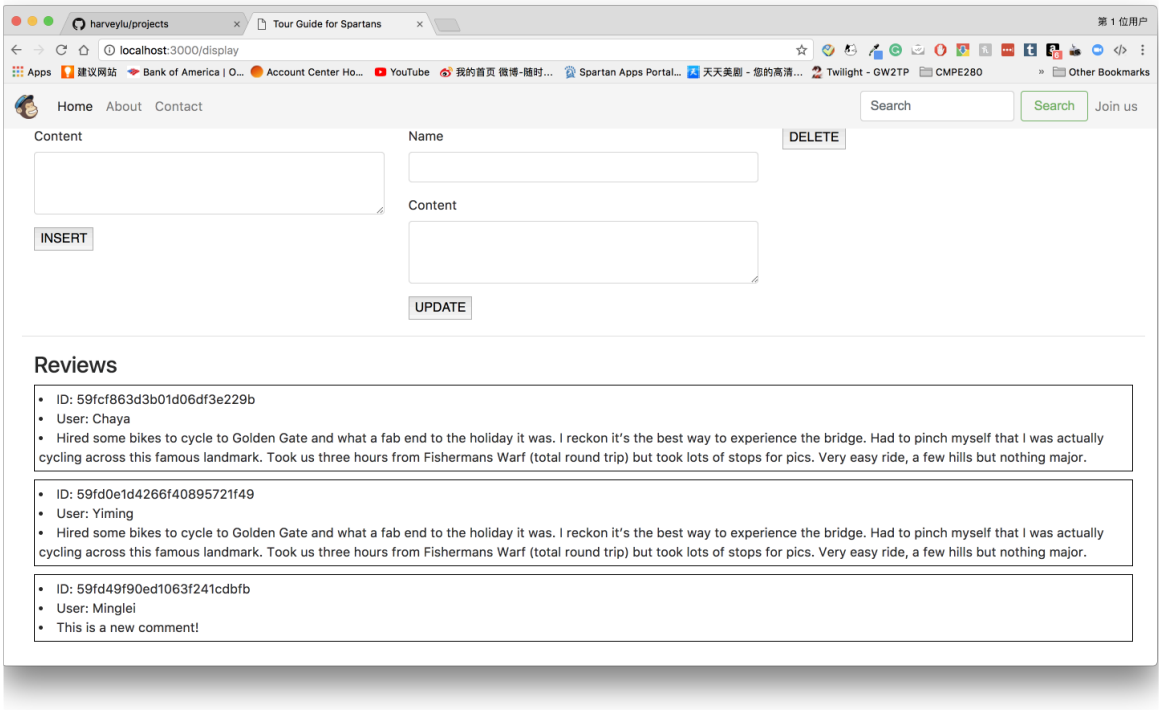
- We have used RESTful API in our web application to send requests and receive responses in our web application by using HTTP protocols GET, POST, PUT and DELETE.
- We are using RESTful API for user reviews, i.e., creation, deletion, display and updating of user reviews will be performed by our web application by using RESTful API.
- Performing Create operation using RESTful API and HTTP POST
    - Code for implementing:

```
router.post('/insert', function(req, res, next) {
  var item = {
    name: req.body.name,
    content: req.body.content
  };

  var data = new UserReview(item);
  data.save();

  res.redirect('/');
});
```

- Screenshots showing implemented functionality:
  Before Creating a new review



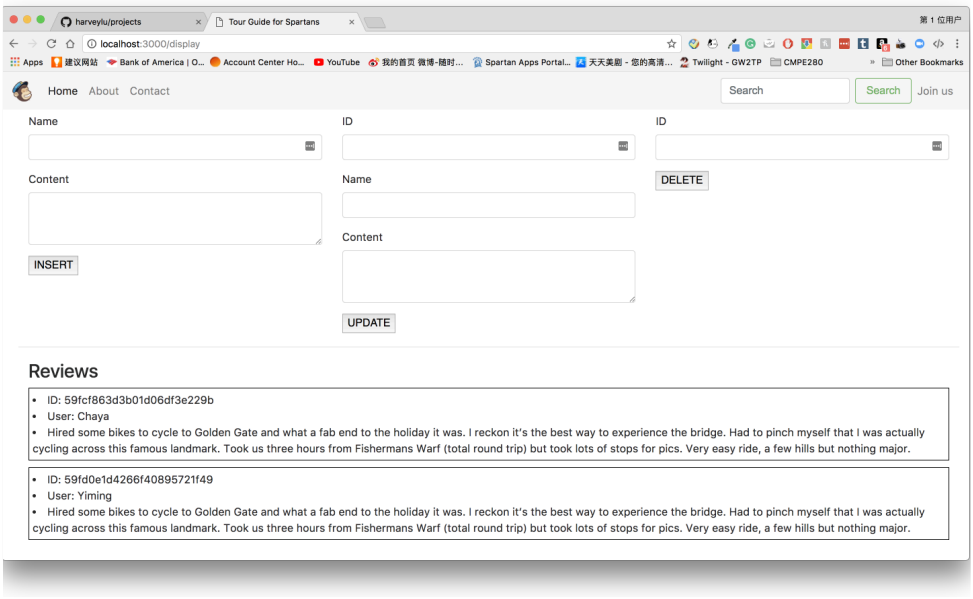
After Creating a new review



- Performing Read operation using RESTful API and HTTP GET
  - Code for implementing:

```
router.get('/display', function(req, res, next) {
  UserReview.find({}, function(err, doc) {
        res.render('display', {items: doc});
    });
});
```

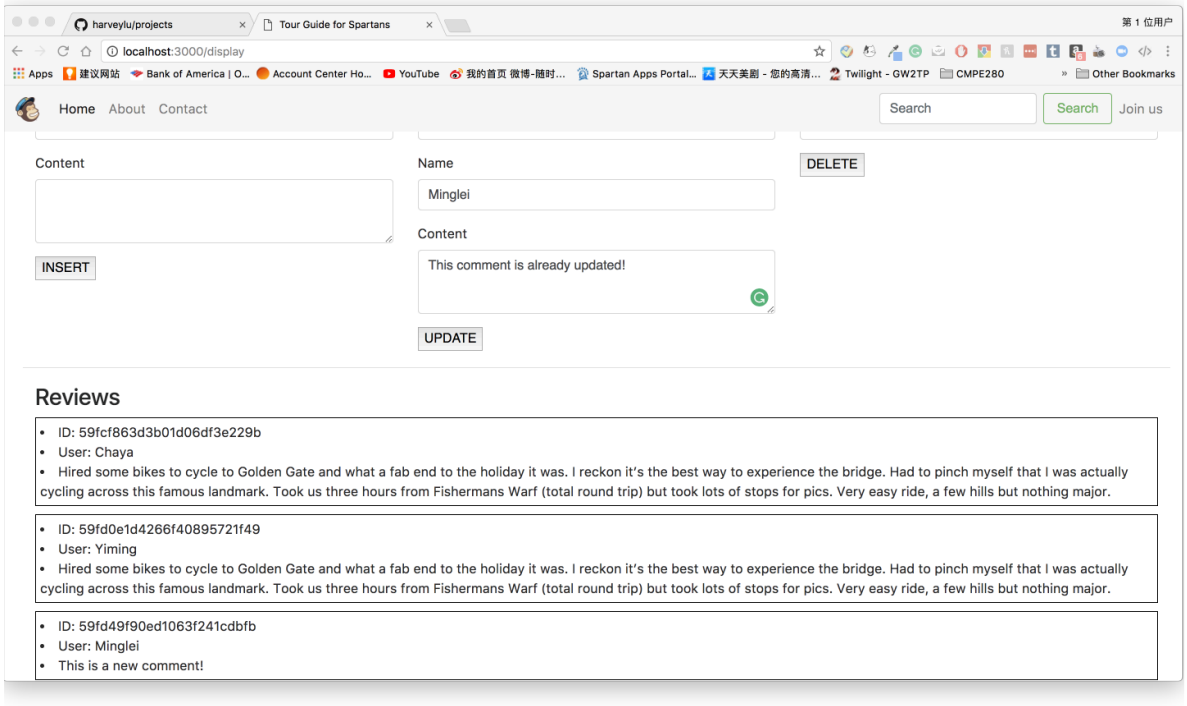
  - Screenshots showing implemented functionality:
    Displaying Review



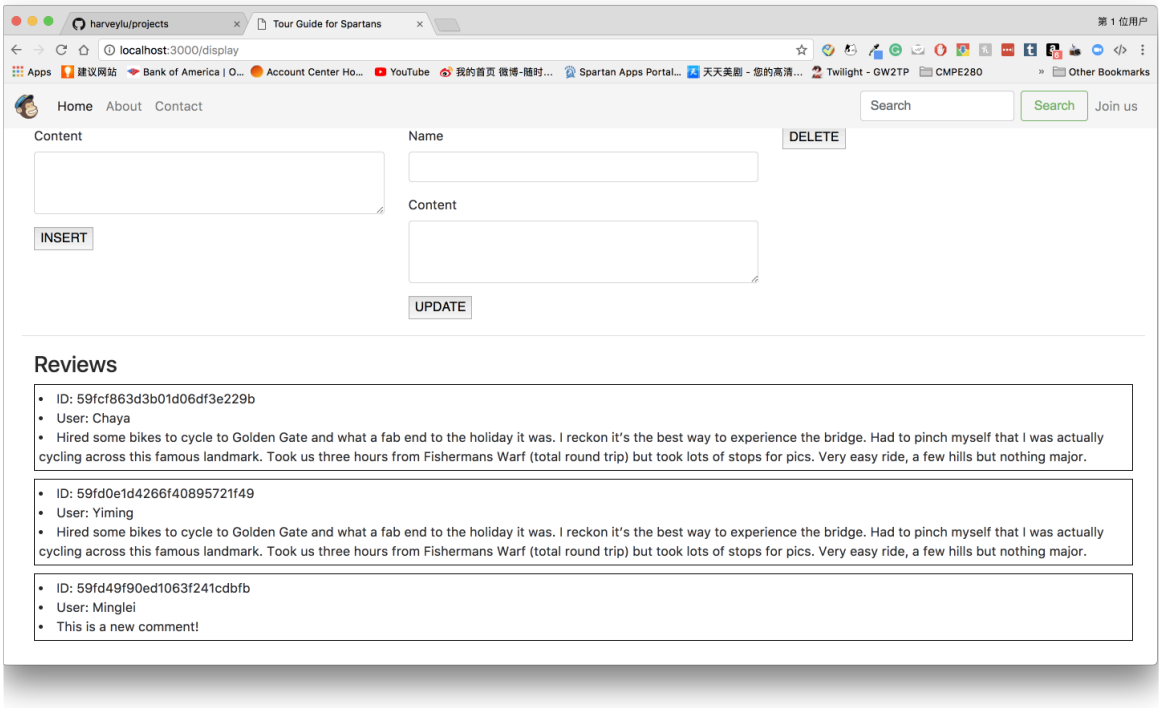
- Performing Update operation using RESTful API and HTTP PUT
  - Code for implementing:

```
router.post('/update', function(req, res, next) {
  var id = req.body.id;

  UserReview.findById(id, function(err, doc) {
    if (err) {
      console.error('error, no entry found');
    }
    doc.name = req.body.name;
    doc.content = req.body.content;
    doc.save();
  })
  res.redirect('/');
});
```

  - Screenshots showing implemented functionality:
    Before Updating a review



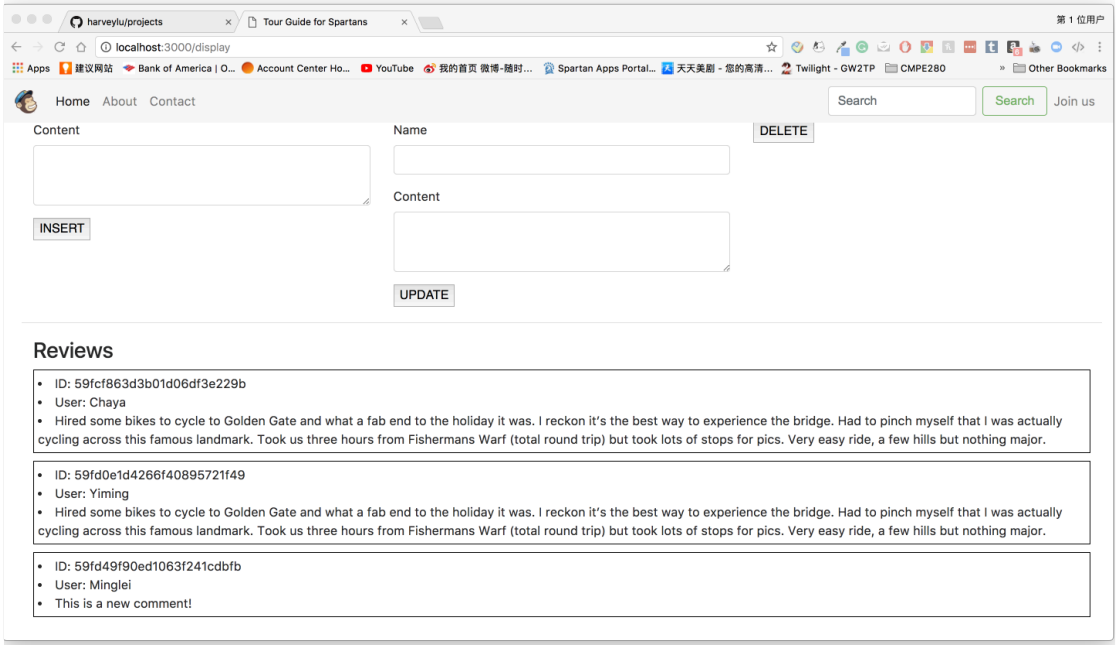
After updating a review

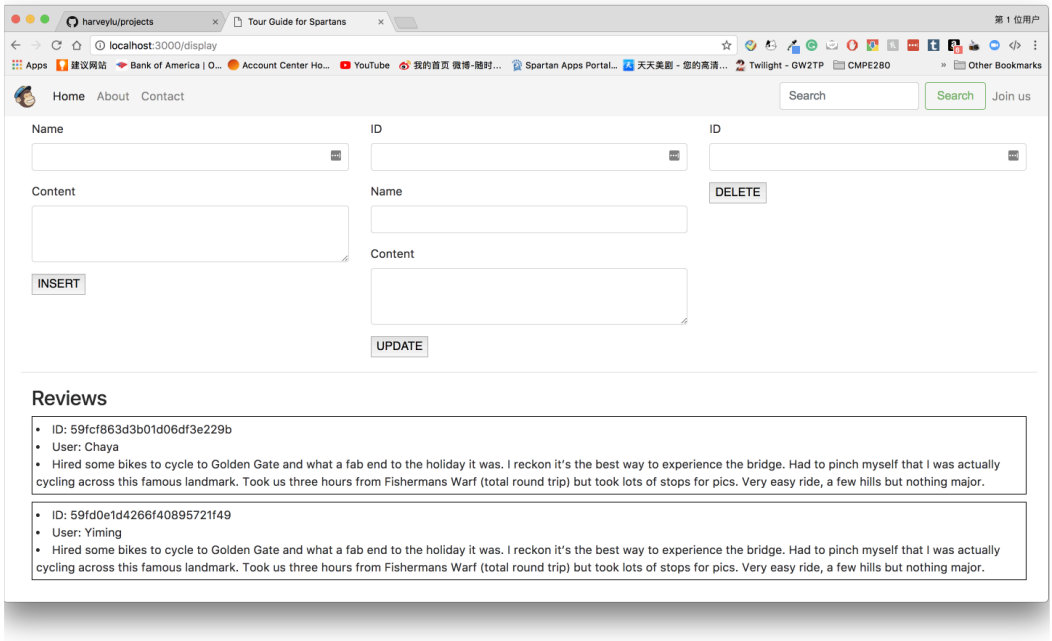


- Performing Delete operation using RESTful API and HTTP DELETE
    - Code for implementing:

```
router.post('/delete', function(req, res, next) {
  var id = req.body.id;
  UserReview.findByIdAndRemove(id).exec();
  res.redirect('/');
});
```

    - Screenshots showing implemented functionality:
      Before Deleting a review





After Deleting a review

User authentication:

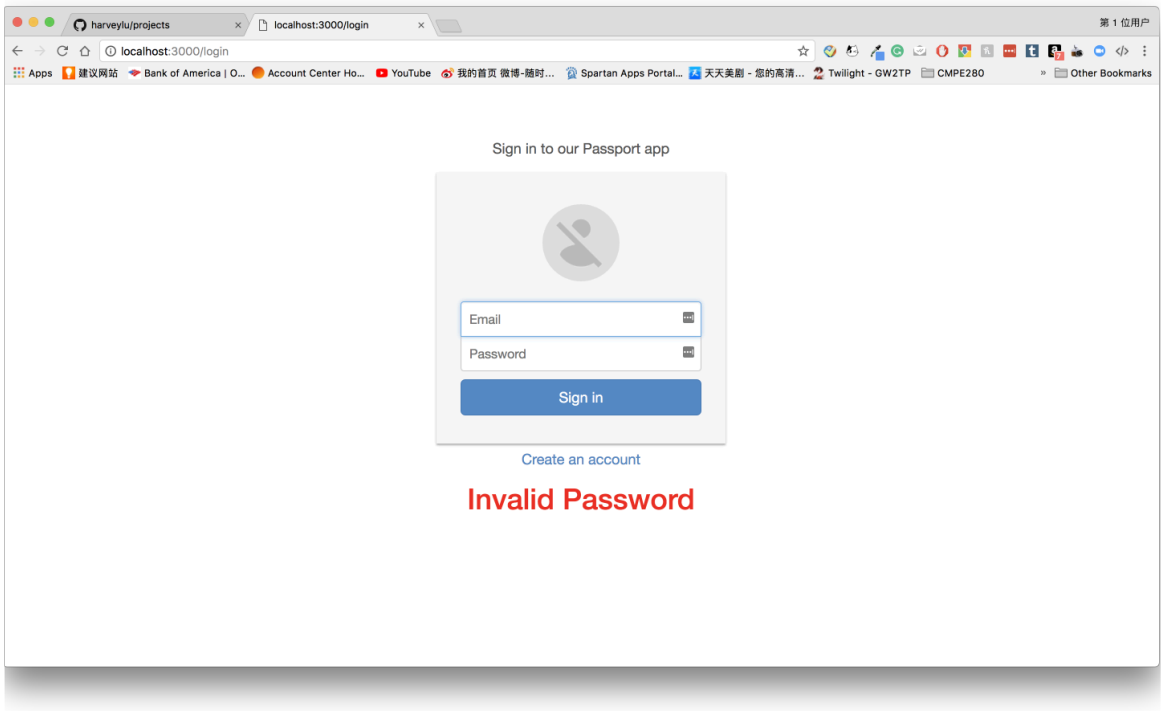
- For user authentication, we have made sure that the already registered users are able to login by providing valid credentials as input.
- The users who try to login and are not yet registered receive an error message on the same page: “User not found”.
- Also, if a user tries to register with a user name that already exists, they can an error message on the screen “User Already Exists”
- The screenshots below demonstrate our “user authentication” functionality and the code written for its implementation.
    - Code for implementing user authentication while user registration:

```
function(req, username, password, done) {

    findOrCreateUser = function(){
        // find a user in Mongo with provided username
        User.findOne({ 'username' :  username }, function(err, user) {
            // In case of any error, return using the done method
            if (err){
                console.log('Error in SignUp: '+err);
                return done(err);
            }
            // already exists
            if (user) {
                console.log('User already exists with username: '+username);
                return done(null, false, req.flash('message','User Already Exists'));
            } else {
                // if there is no user with that email
                // create the user
                var newUser = new User();

                // set the user's local credentials
                newUser.username = username;
                newUser.password = createHash(password);
                newUser.email = req.param('email');
                newUser.firstName = req.param('firstName');
                newUser.lastName = req.param('lastName');

                // save the user
                newUser.save(function(err) {
                    if (err){
                        console.log('Error in Saving user: '+err);
                        throw err;
                    }
                    console.log('User Registration succesful');
                    return done(null, newUser);
                });
            }
        });
    };
    // Delay the execution of findOrCreateUser and execute the method
    // in the next tick of the event loop
    process.nextTick(findOrCreateUser);
})
```

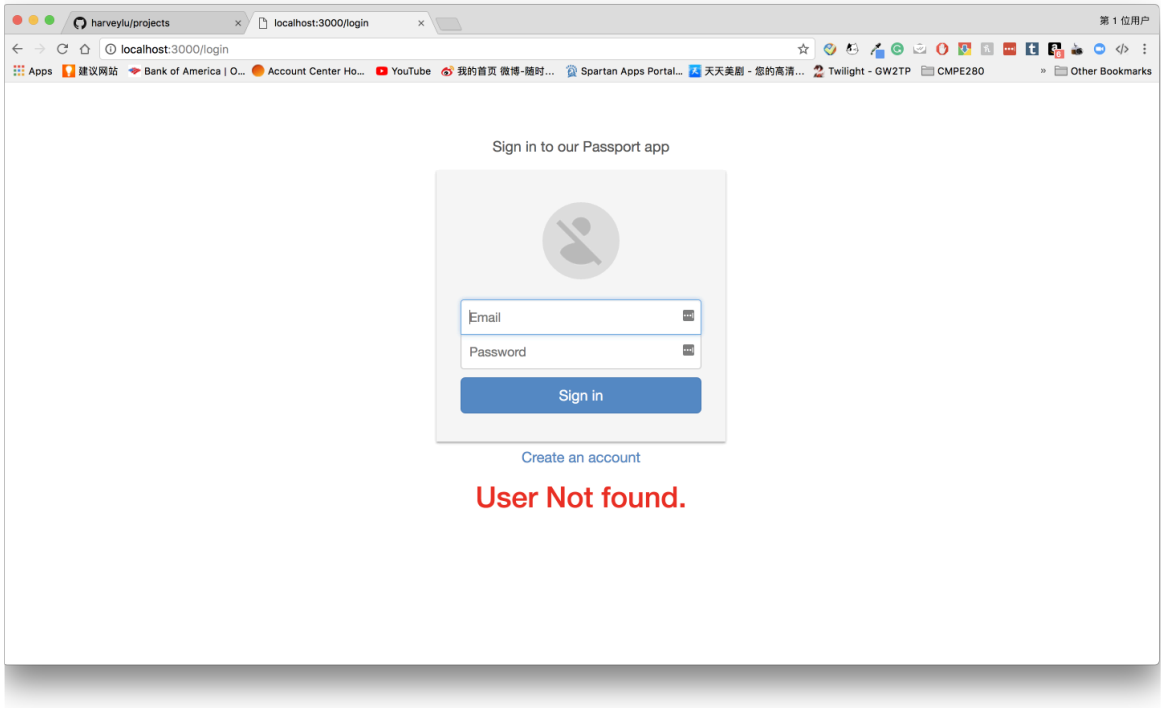
- Code for validating user login credentials or verifying user:

```
var LocalStrategy   = require('passport-local').Strategy;
var User = require('../models/user');
var bCrypt = require('bcrypt-nodejs');

module.exports = function(passport){

    passport.use('login', new LocalStrategy({
            passReqToCallback : true
        },
        function(req, username, password, done) {
            // check in mongo if a user with username exists or not
            User.findOne({ 'username' :  username },
                function(err, user) {
                    // In case of any error, return using the done method
                    if (err)
                        return done(err);
                    // Username does not exist, log the error and redirect back
                    if (!user){
                        console.log('User Not Found with username '+username);
                        return done(null, false, req.flash('message', 'User Not found.'));
                    }
                    // User exists but wrong password, log the error
                    if (!isValidPassword(user, password)){
                        console.log('Invalid Password');
                        return done(null, false, req.flash('message', 'Invalid Password')); // redirect back to login page
                    }
                    // User and password both match, return user from done method
                    // which will be treated like success
                    return done(null, user);
                }
            );

        })
    );


    var isValidPassword = function(user, password){
        return bCrypt.compareSync(password, user.password);
    }

}
```

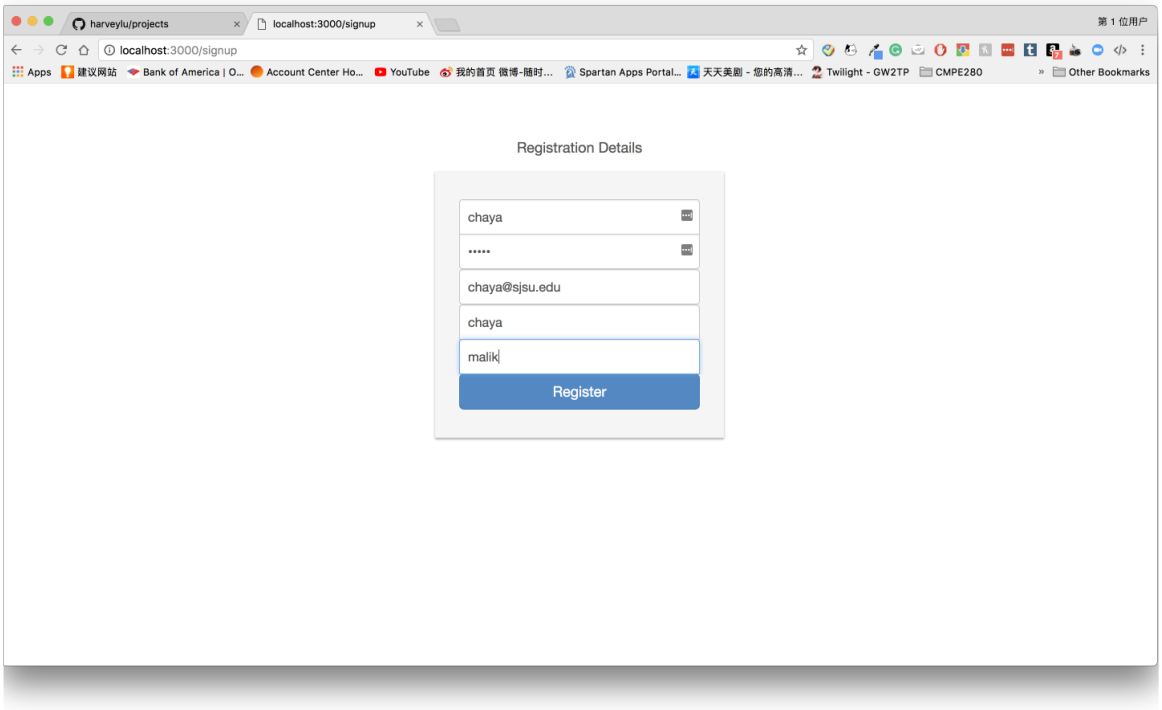
- User Authentication Functionality: “Invalid Password”, a user cannot login if the password provided by them is incorrect.



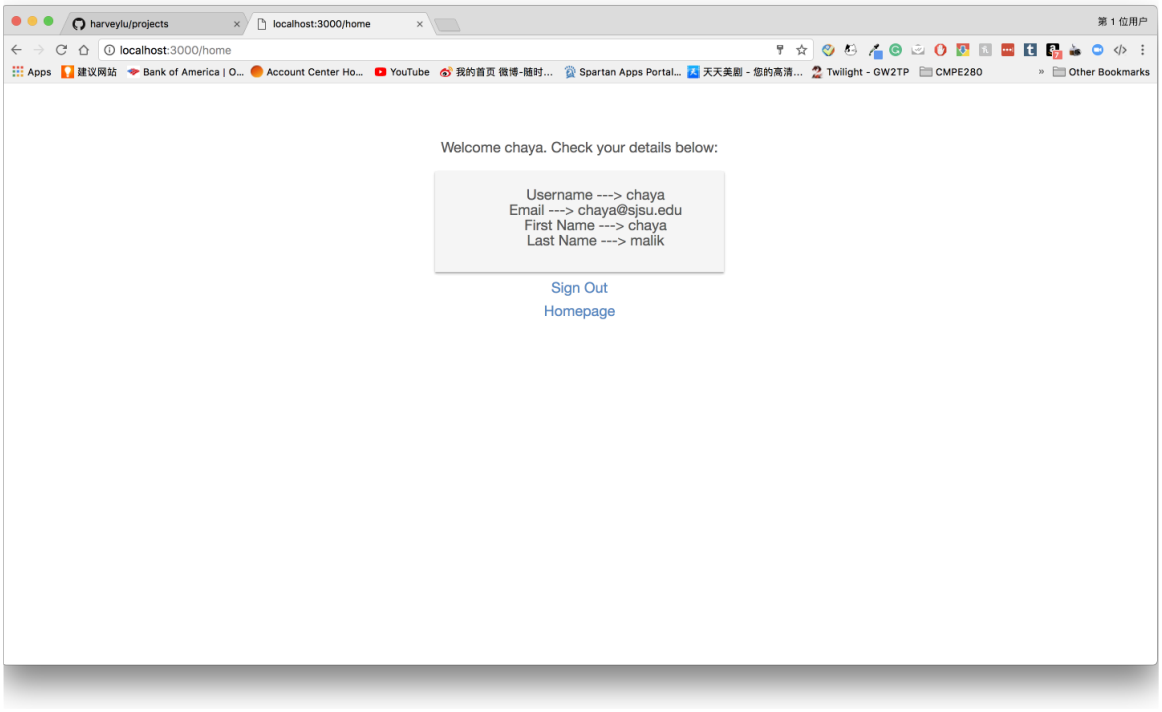
- User Authentication Functionality: “User not found”, a user cannot login using a user name, if they are not registered with the application using that user name.



- User Authentication Functionality: “Registration of a user with the application”

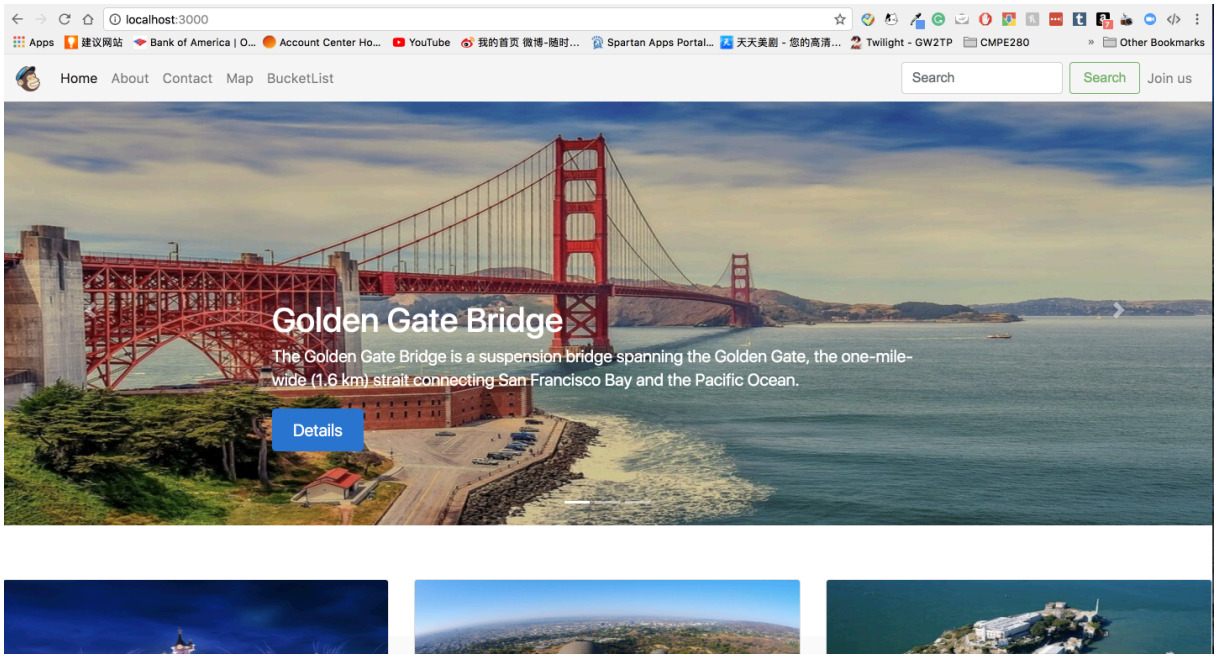


- User Authentication Functionality: “Logged in User”

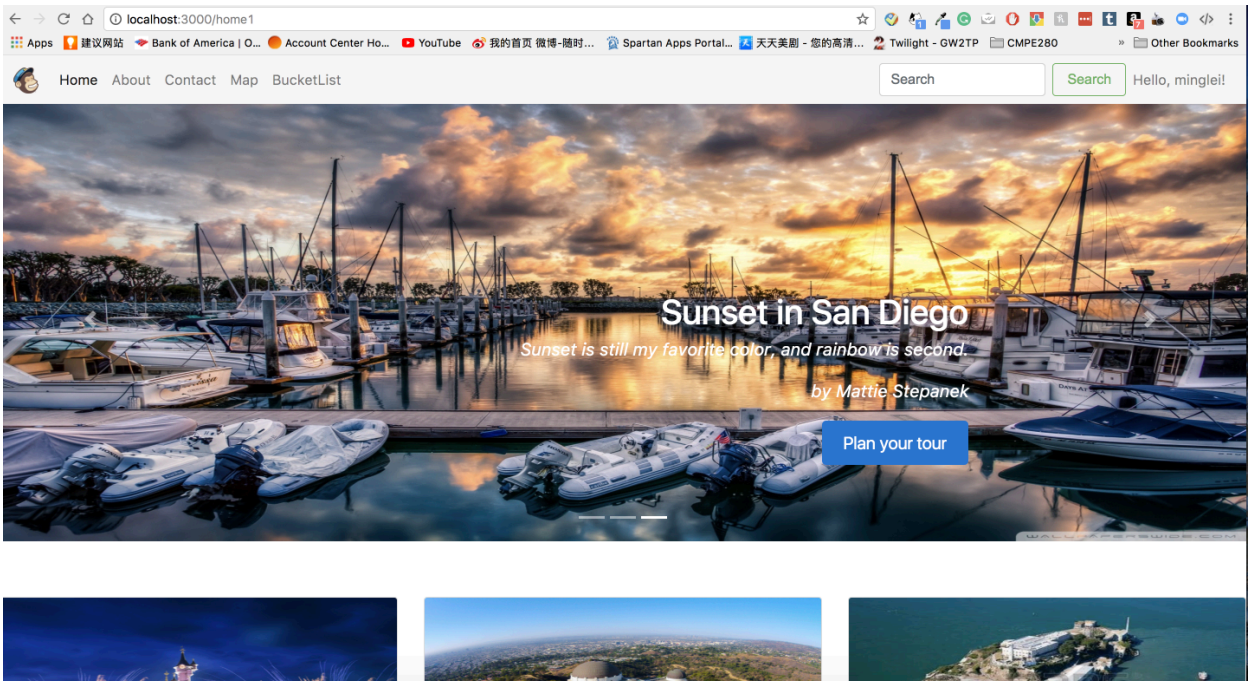


Session Management:

- We have implemented Session Management in our web application to provide a smooth user experience. The user does not have to login every time they access our application.
- By implementing Session Management in our web application, every time a user makes a request our application validates that it is the same user who started the session earlier and does not ask the user for login credentials for every request made.
- The following screenshots show our session management functionality and implementation:
    - Before Login



- After Login (The difference is on the upper-right side, it shows the user's first name)



- Code implementing Session Management in our application
  - Session Management code to check if user is logged in

```
var isAuthenticated = function (req, res, next) {
    // if user is authenticated in the session, call the next() to call the next request handler
    // Passport adds this method to request object. A middleware is allowed to add properties to
    // request and response objects
    if (req.isAuthenticated())
        return next();
    // if the user is not authenticated then redirect him to the login page
    res.redirect('/');
}
```

- Session Management code for page display for an already logged in user (a user who has already initiated the session)

```
    /* Handle Login POST */
    router.post('/login', passport.authenticate('login', {
        successRedirect: '/home',
        failureRedirect: '/login',
        failureFlash : true
    }));

    /* GET Home Page */
    router.get('/home', isAuthenticated, function(req, res){
        res.render('home', { user: req.user });
    });
```