CPSC 476: ASSIGNMENT 3

SHARED EVENTS CALENDAR

CSPC 476 Assignment 3

Mussa Mubarki

893536896

DESCIPTION

The assignment architecture is Spring web MVC framework. It provides model-view-controller architecture , and It "is designed around a DispatcherServlet that dispatches requests to handlers, with configurable handler mappings, view resolution, locale and theme resolution as well as support for uploading files. The default handler is based on the @Controller and @RequestMapping annotations, offering a wide range of flexible handling methods."[1]

The central Servlet -DispatcherServlet- configure in web.xml as any other servlet:

```
<servlet>
      <servlet-name>EventsDispatcher</servlet-name>
      <servlet-class>
         org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
      <load-on-startup>1</load-on-startup>
      <init-param>
         <param-name>contextConfigLocation</param-name>
         <param-value>/WEB-INF/EventsDispatcher-servlet.xml</param-value>
      </init-param>
   </servlet>
```

[1] http://docs.spring.io/autorepo/docs/spring/3.2.x/spring-framework-reference/html/mvc.html

```
    <servlet-mapping>
        <servlet-name>EventsDispatcher</servlet-name>
        <url-pattern>/</url-pattern>
        <url-pattern>/Events</url-pattern>
        <url-pattern>/Events*</url-pattern>
    </servlet-mapping>
```

The contextConfigLocation initial parameter indicate where context can be found.

The context EventsDispatcher-servlet is contains all beans in Assignment3 application.

```
<mvc:annotation-driven />
<context:component-scan base-package="master.cpsc476" />
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
        <property name="url" value="jdbc:hsqldb:hsql:/localhost/test"/>
        <property name="username" value="A"/>
        <property name="password" value=""/>
        <property name="initialSize" value="10"/>
        <property name="testOnBorrow" value="true"/>
    </bean>

    <jdbc:initialize-database data-source="dataSource" ignore-failures="ALL">
        <jdbc:script execution="INIT" location="/META-INF/schema.sql" />
        <jdbc:script execution="INIT" location="/META-INF/test-data.sql" />
    </jdbc:initialize-database>

    <bean name="EventDAOImp" class="master.cpsc476.dao.EventDAOImp" >
        <property name="dataSource" ref="dataSource" />
    </bean>
    <bean name="UserDAOImp" class="master.cpsc476.dao.UserDAOImp" >
        <property name="dataSource" ref="dataSource" />
    </bean>
    <bean id="viewResolver"
                class="org.springframework.web.servlet.view.InternalResourceViewResolver" >
        <property name="prefix">
            <value>/WEB-INF/jsp/</value>
         </property>
```

```
    <property name="suffix">
      <value>.jsp</value>
    </property>
  </bean>
```

<mvc:annotation-driven /> : declares explicit support for annotation-driven MVC controllers such as @RequestMapping, @Controller.

<context:component-scan base-package="master.cpsc476" />: enable auto scanning feature in Spring to find out the Spring components.

```
<bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver" >
    <property name="prefix"><value>/WEB-INF/jsp/</value></property>
    <property name="suffix"><value>.jsp</value></property>
</bean>
```

ViewResolver is for resolve JSP name return string from methods in controller to actual JSP full path by adding prefix and suffix.

Controller

EventsServlets refactored to use EventsController. Add @Controller in header and @PostConstruct before init() method to initialize DAOs object.

```
@Controller
public class EventsController {
   @Autowired
   private ApplicationContext applicationContext;

   private EventDAO eventDAO;
   private UserDAO userDAO;
```

```
@PostConstruct
public void init(){
    eventDAO = (EventDAO) applicationContext.getBean("EventDAOImp");
    userDAO = (UserDAO) applicationContext.getBean("UserDAOImp");
}
```

Not like Servlet, in Spring controller we no longer need complex switched[2]. Instead, requests are automatically routed to the proper controller and methods using @RequestMapping.
Each methods in controller have ModelMap as parameter variable. ModelMap is map that uses to save data, and it is passed to view. Used model.addAttribute(key, object) to passing data.
Also, before show form view (EventForm.jsp or SignupForm.jsp): we have to add form backing object (e.g. EventDetail) to model as

```
model.addAttribute("eventDetail", new EventDetail());
```

And in view we have to use spring tags to build the form:

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
..
<form:form modelAttribute="eventDetail" method="POST"
action="${pageContext.request.contextPath}/Events/createEvent" name="createEventForm">
…
<form:input  path="title"/>
...
```

For “POST” form action, in controller’s methods: use @ModelAttribute to supply form backing object to a Controller method as following:

```
@ModelAttribute("eventDetail") EventDetail eventDetail
```

For parameters that are passed with the url : we use @PathVariable as following:

[2] Text Book: Professional Java for Web Applications page 356.

```
@RequestMapping(value="/Events/view/{id}", method = RequestMethod.GET)
   public String viewEvent(@PathVariable Long id, Model model, HttpSession session){
```

The Assignment3 application has the following routings:

- @RequestMapping(value={"/Events/list","/Events/","/Events"}, method = RequestMethod.GET): for list all active event with get method: to show all events list. It is the default action if action value is null. It is not required login.
- @RequestMapping(value="/Events/view/{id}", method = RequestMethod.GET): view event with get method: to show event detail by redirect to /jsp/EventView.jsp. Event Id is passed as parameter. It is required login.
- @RequestMapping(value="/Events/create", method = RequestMethod.GET) : create with get method: to show event create form by redirect to /jsp/EventForm.jsp. It is required login.
- @RequestMapping(value="/Events/createEvent", method = RequestMethod.POST) : createEvent with post method: to create new event. It is required login.
- @RequestMapping(value="/Events/createUser", method = RequestMethod.POST): createUser with post method: to create new user.
- @RequestMapping(value={"/Events/signup*"}, method = RequestMethod.GET) : signup with get method: to show signup form by redirect to /jsp/SignupForm.jsp.
- @RequestMapping(value="/Events/signin", method = RequestMethod.POST): signin with post method: to login process. Call UserDAOImp.isMach(User) then store user detail in HTTPSession that getted by declaration it in method header.
- @RequestMapping(value="/Events/signout", method = RequestMethod.GET): signout with get method: to logout by remove user bean form HTTPSession
- @RequestMapping(value="/Events/userHome/{id}", method = RequestMethod.GET): userHome + parameter id: to show user detail by redirect to /jsp/User.jsp. User Id is passed as parameter. It is required login.
- @RequestMapping(value="/Events/likeEvent/{userId}/{eventId}", method =

RequestMethod.GET): likeEvent with get method to add event to user interested list. User id and event id are passed as parameters. It is required login.

- @RequestMapping(value="/Events/unlikeEvent/{userId}/{eventId}", method = RequestMethod.GET): unlikeEvent with get method: to remove event from user interested list. User id and event id are passed as parameters. It is required login.

BUILD INSTRUCTIONS

To packaging and deploying this assignment using Maven, we have to do the following:

- Update Assignment 2 pom.xml by add dependencies.
- for package execute

  mvn package

- or for deploy execute after start tomcat manager

  mvn tomcat7:deploy

- Or because I set the default goal, you can just

  mvn

```
<defaultGoal>tomcat7:deploy</defaultGoal>
```

Note: sometime When I redeploy the application to the web server using Netbeans the .lck file is not properly released and the HSQL server cannot load the file. So we have to delete test.lck under ${tomcatHome}/bin/hsql/localhost to avoid this issue.

Prepare pom.xml:

1. Add Maven dependencies for spring MVC

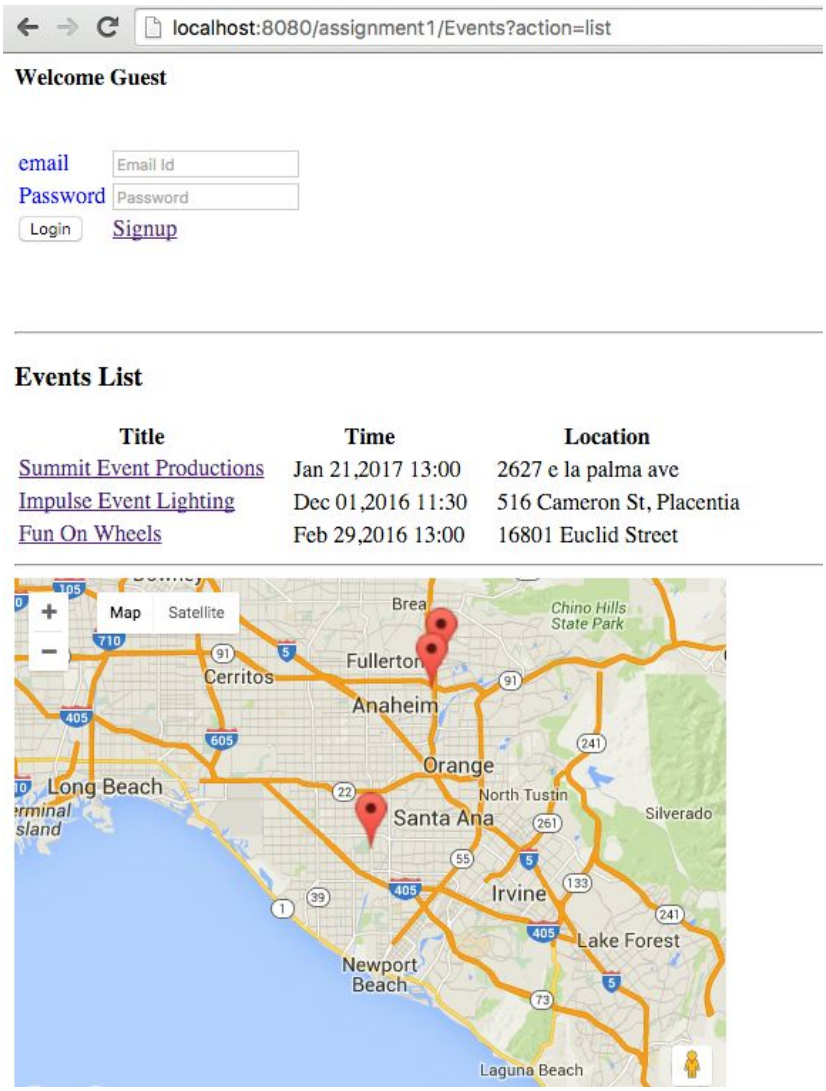
```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>${spring.version}</version>
</dependency>
```

DEMONSTRATING

Because nothing change in view layer, the screens are same as in Assignment 1.

Assignment URL is http://localhost:8080/assignment3

Main screen without login contains events list, and top page contains login form and link sign-up link. The title in events list is link to event detail.

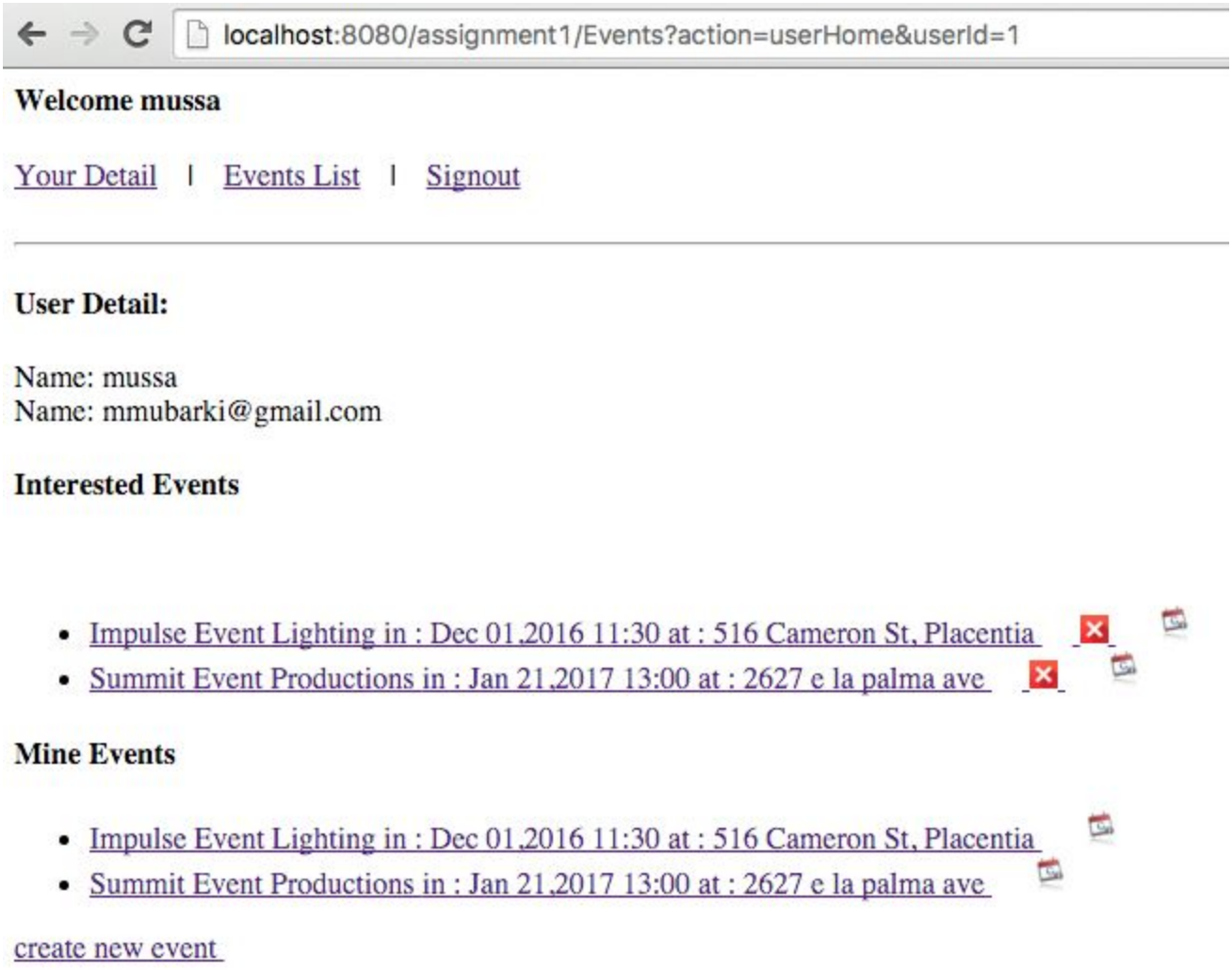


To login: user the following information:

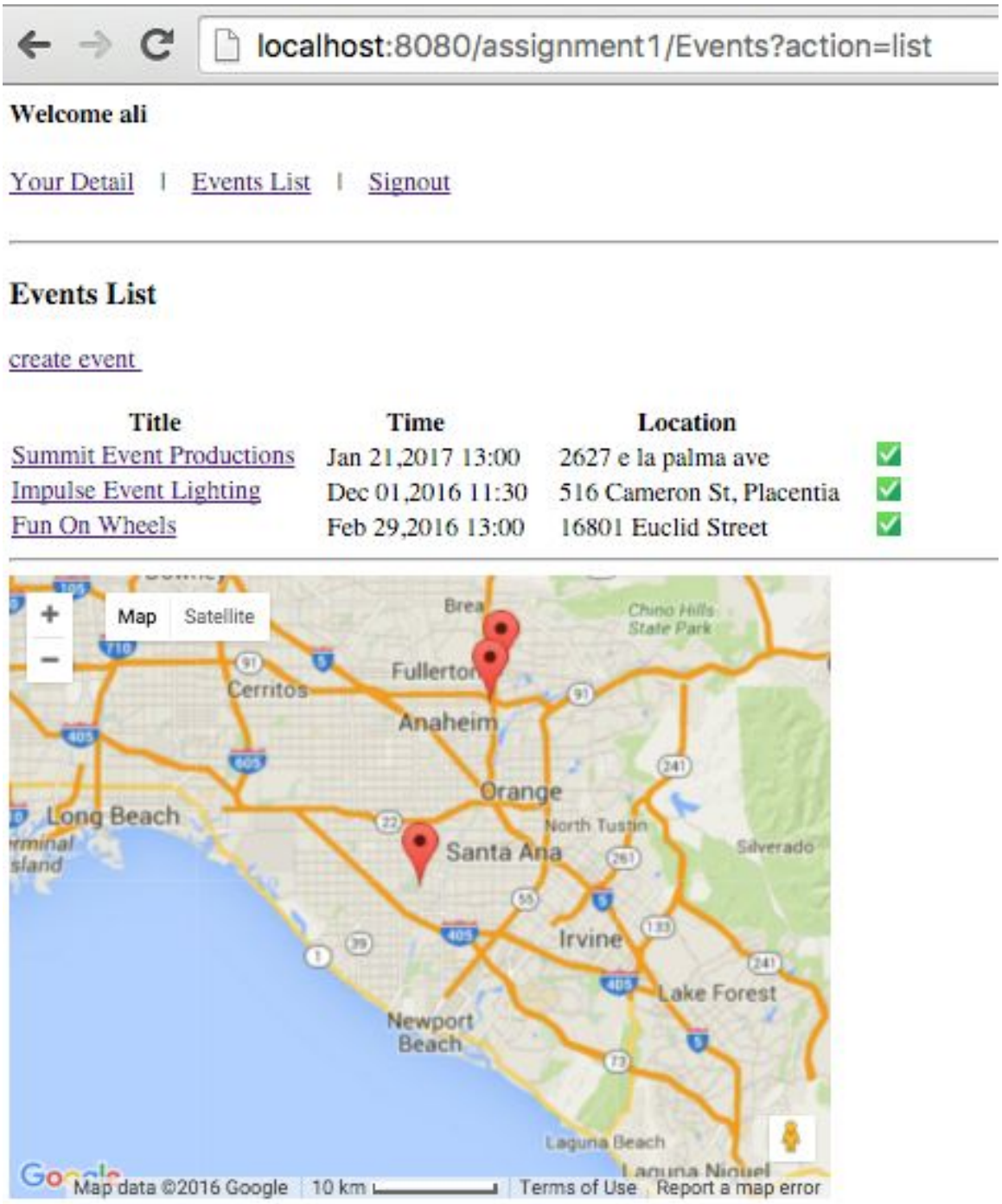
email: mmubarki@gmail.com

password: 123456

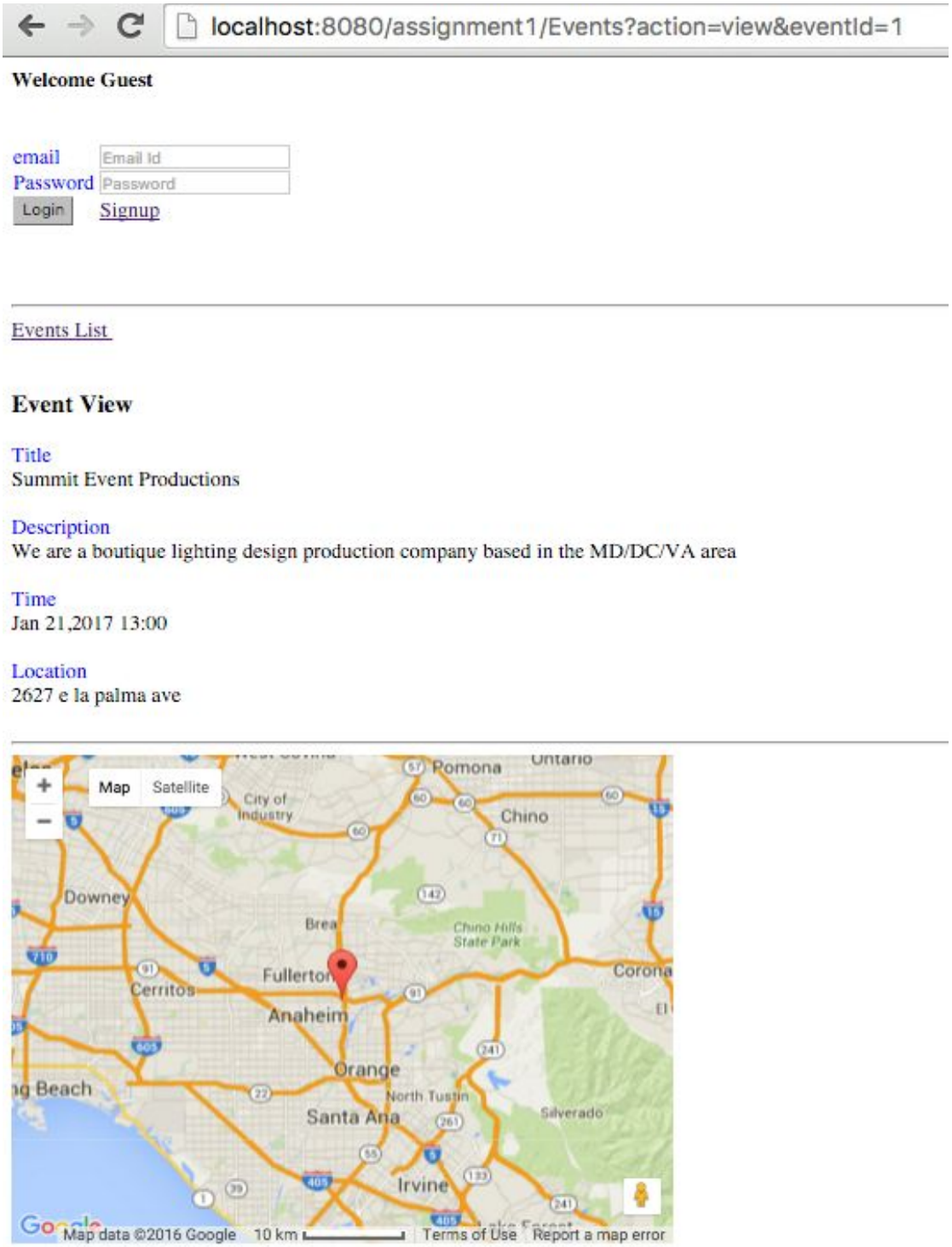
After login: controller method: signin() forward into user detail screen. It contains user information, user's interested events list, and user created events list. User has option to drop unlike events from interested events list. Also each event has link to its detail. Each event in interested or created can be add to google calendar.



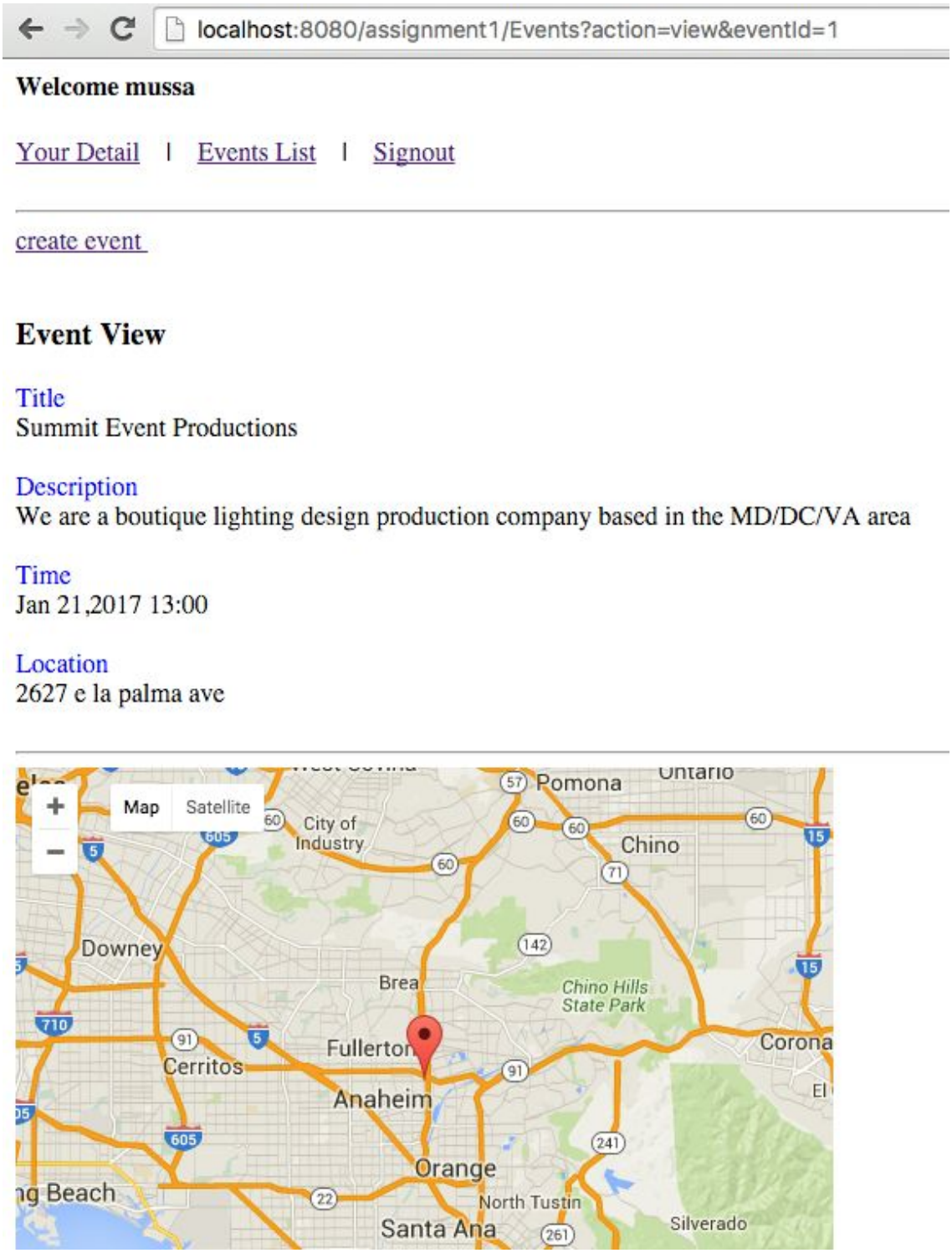
Main page if login contains event list and link to create new event, and top page contains three links: to user detail page, to events list, and sign-out link.



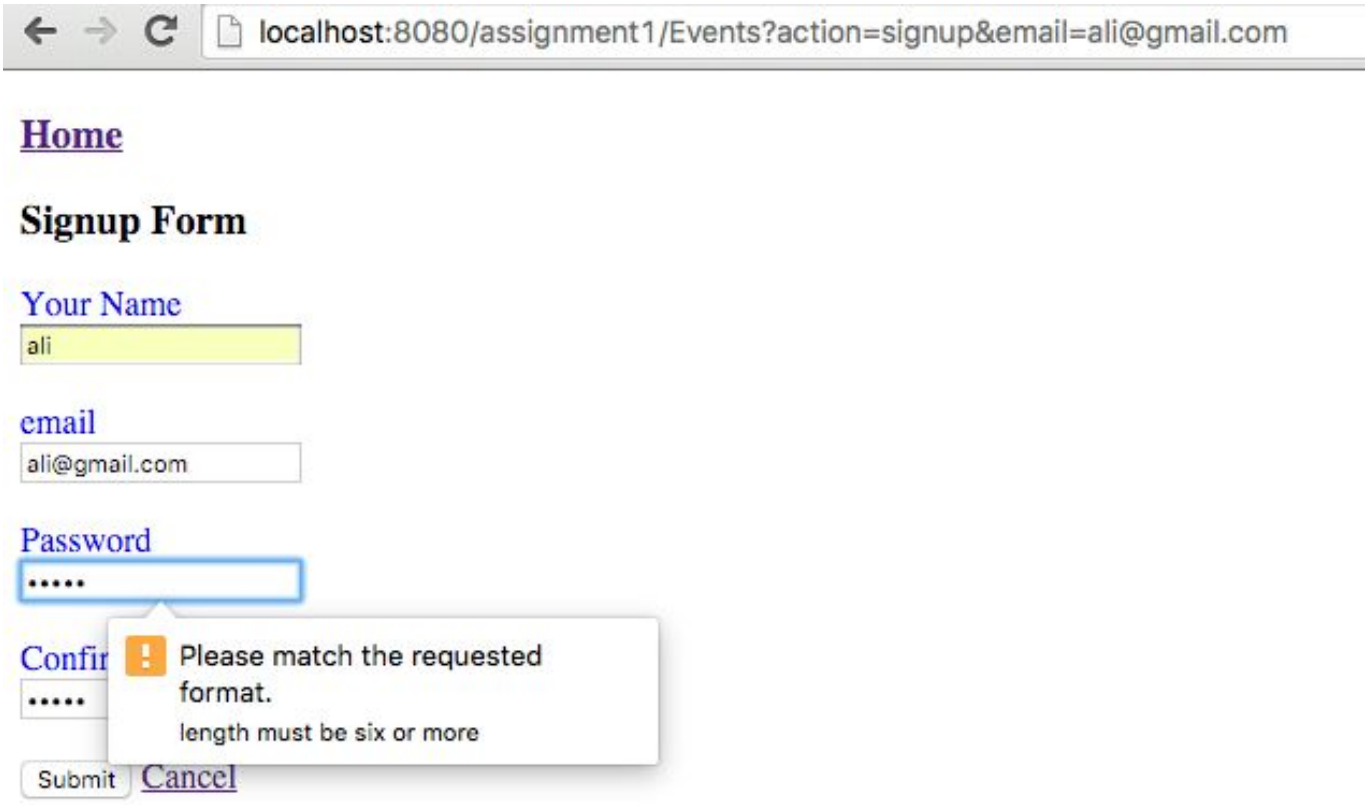
Event view without login. It contains link to events list.



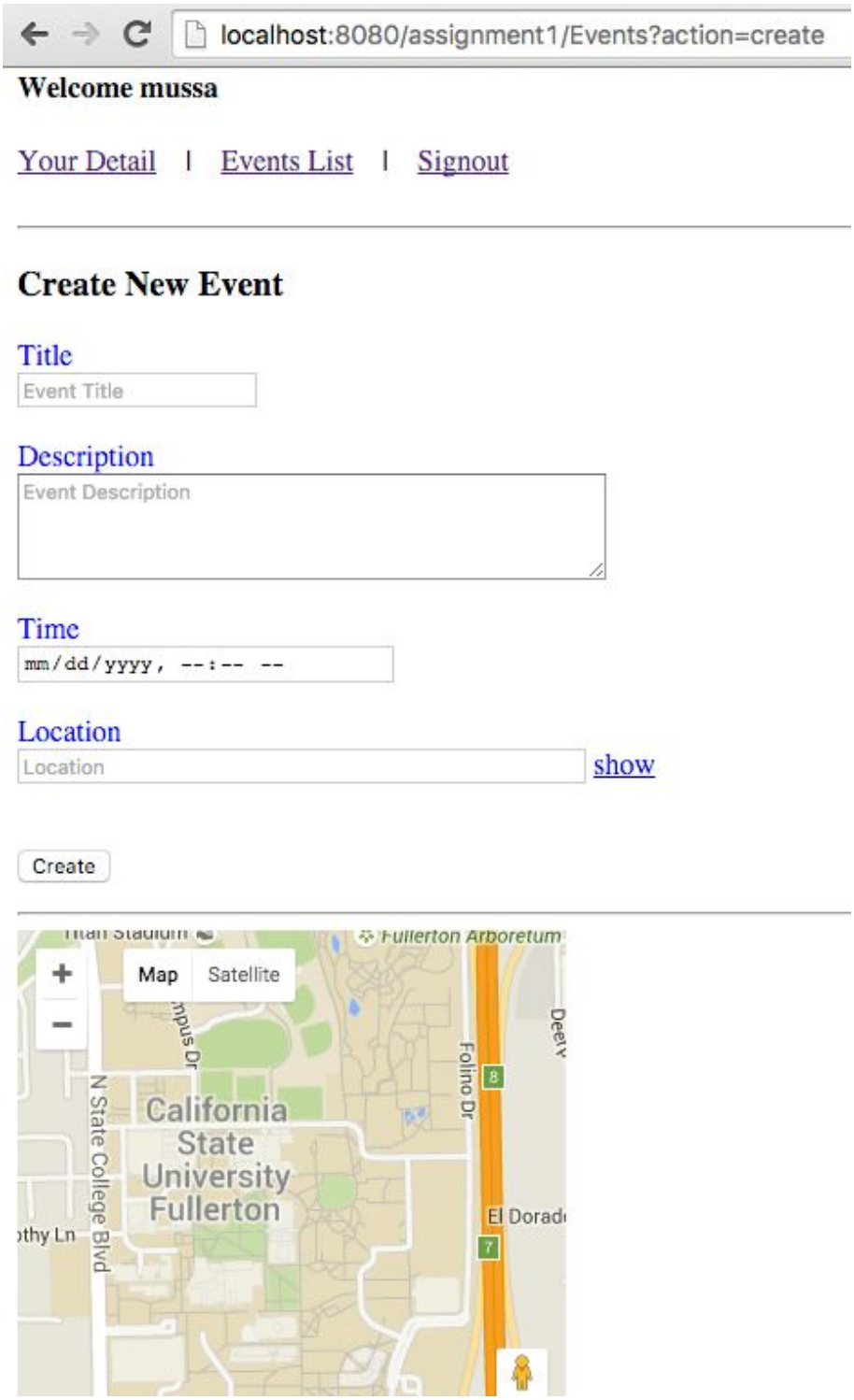
Event view if login. It contains link to create event.



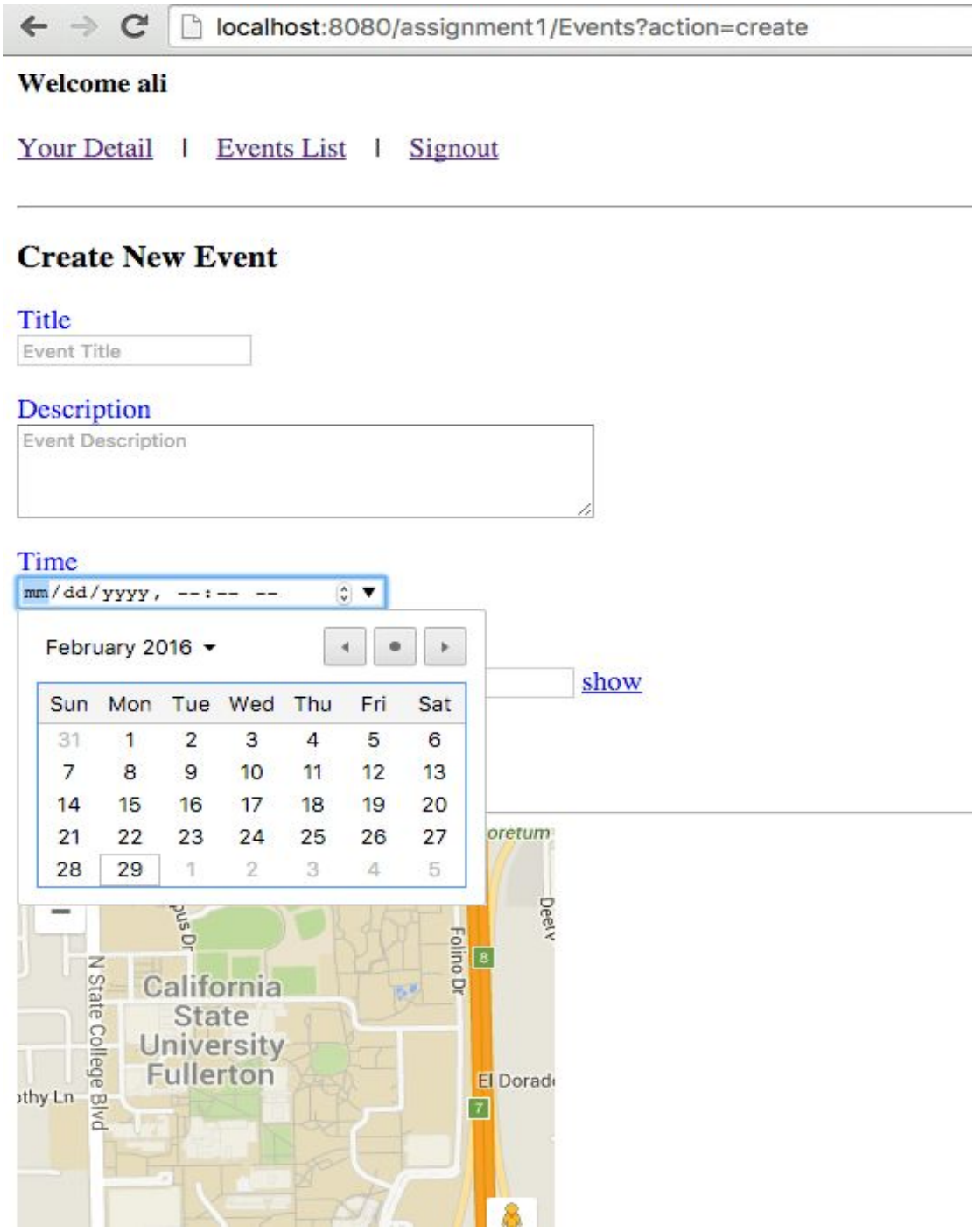
Sing-up form with validation



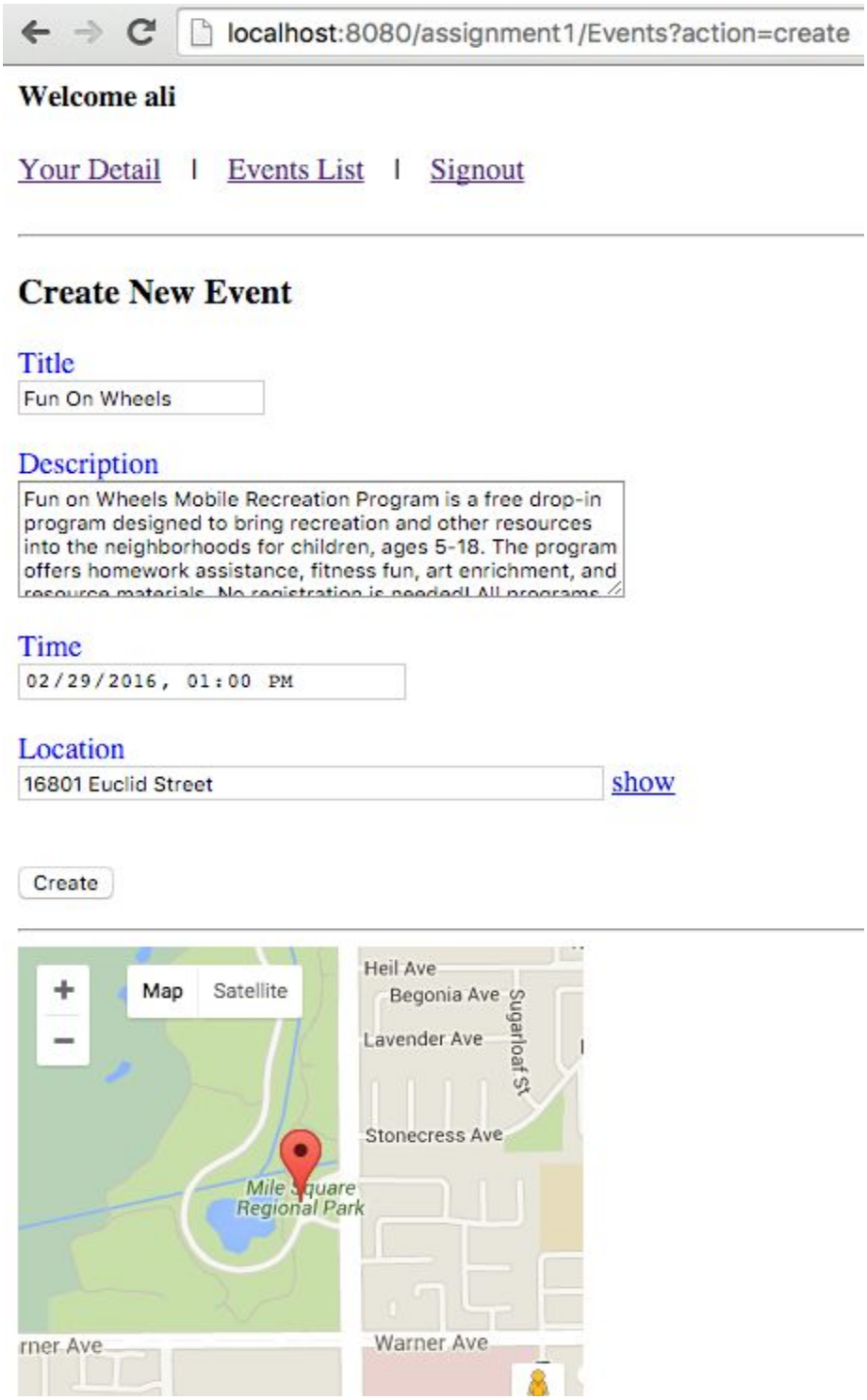
create new event form



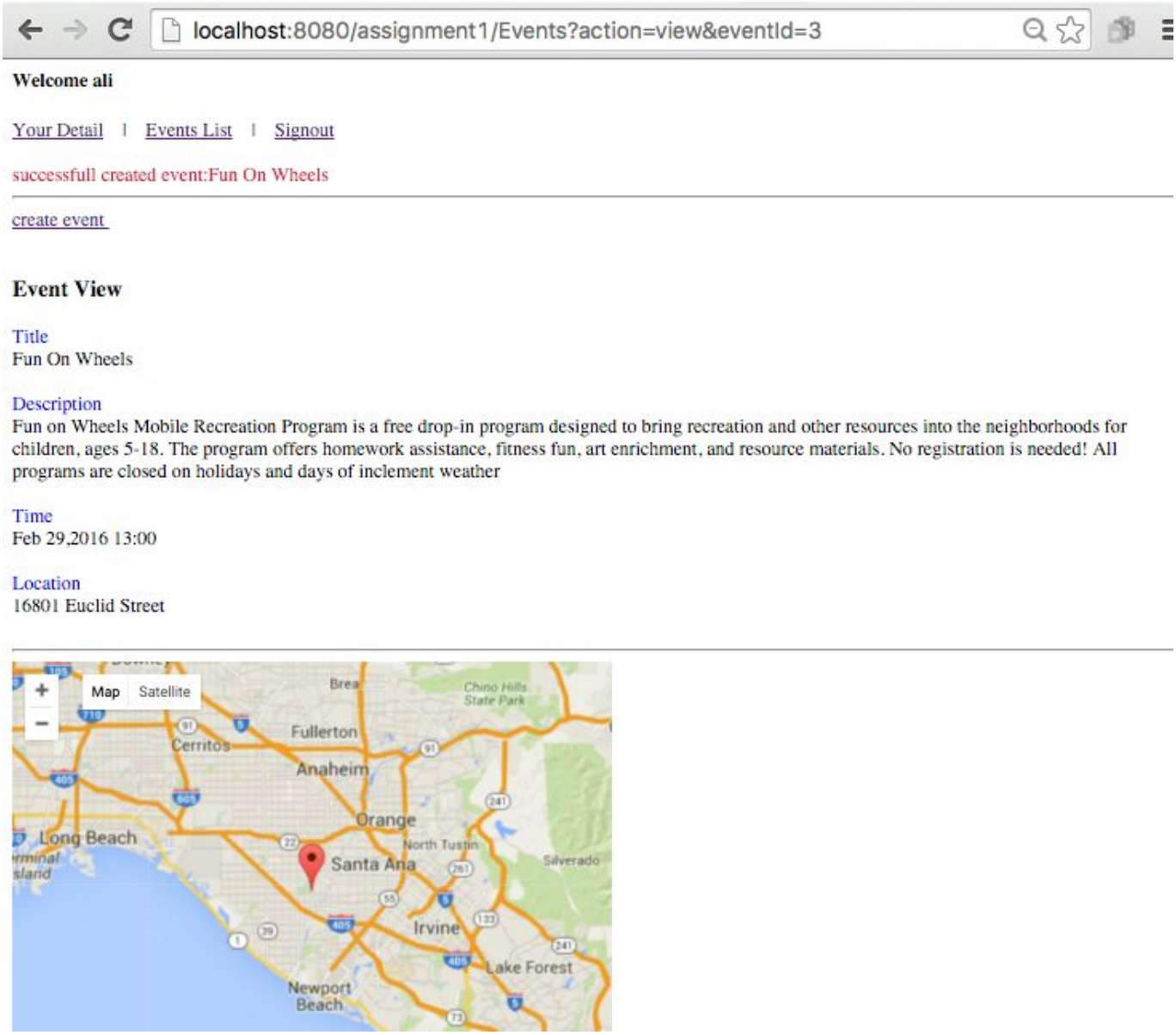
In create new event form: uses dateTime-local input type. It is not supported in Firefox and Internet Explorer.



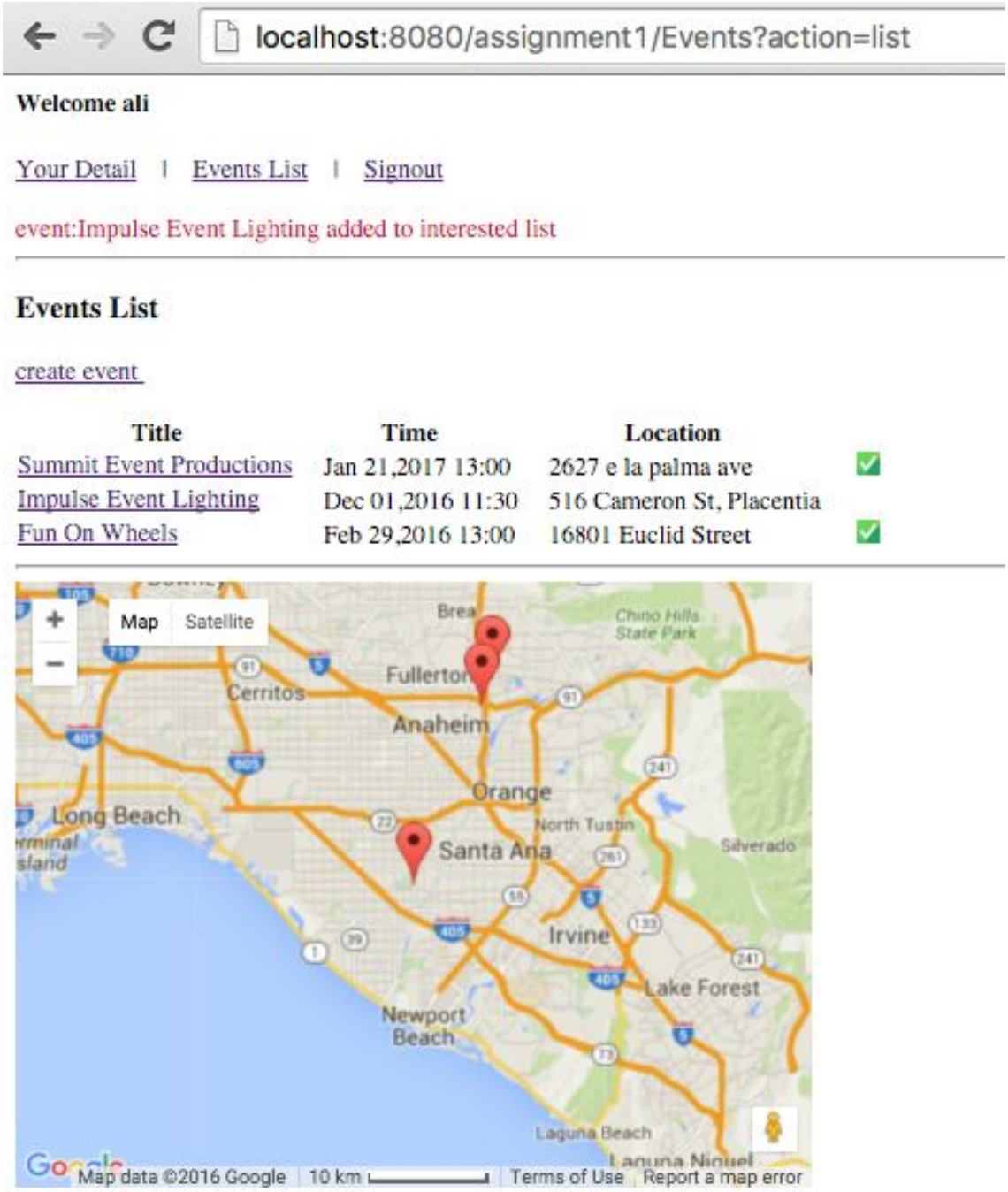
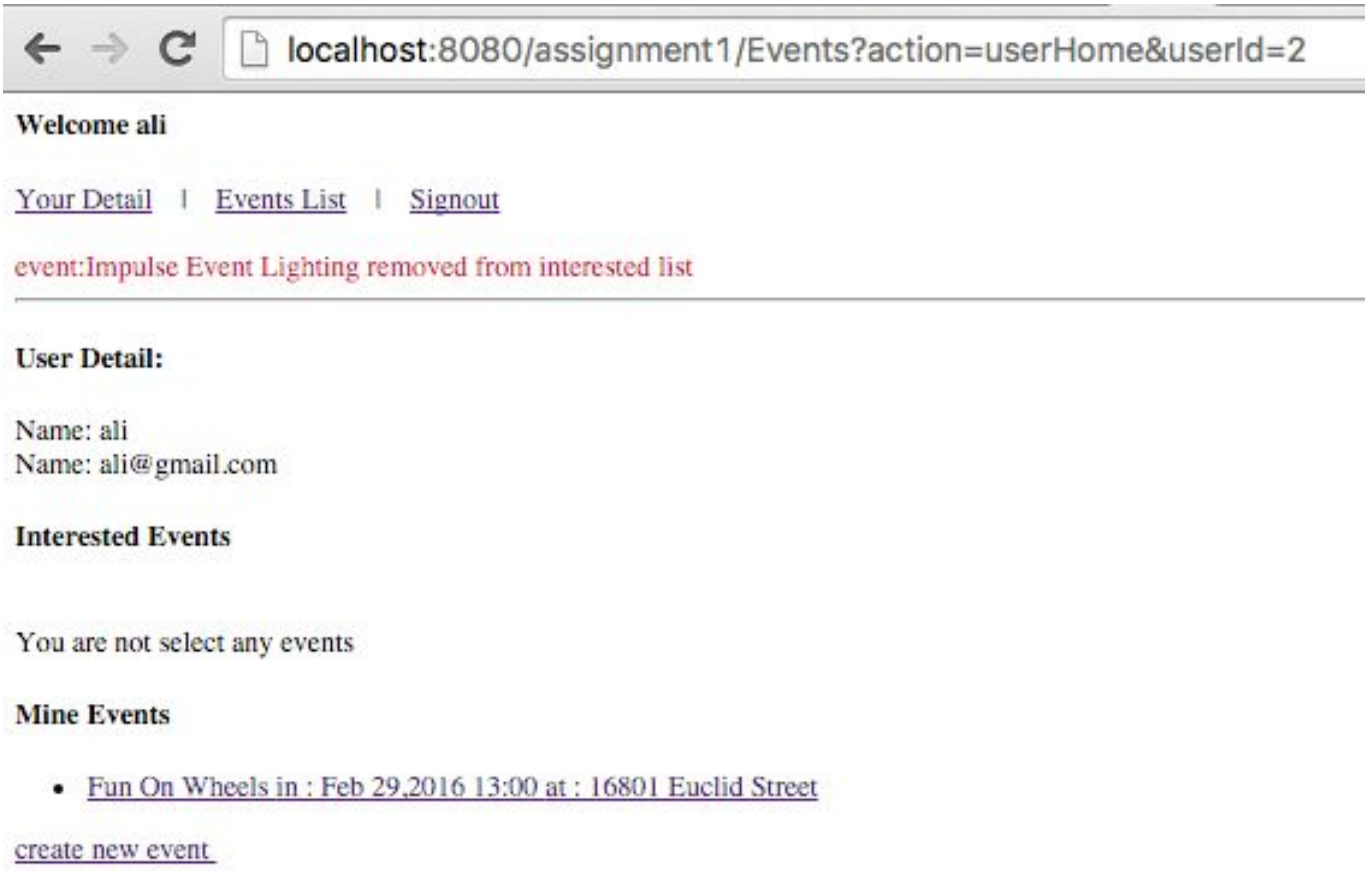
Create new event after filled



View event after creation



Add event into interested list



Drop event from interested list