ENCE 688R Final Project Report

115090873, Shao-Hung, Lee

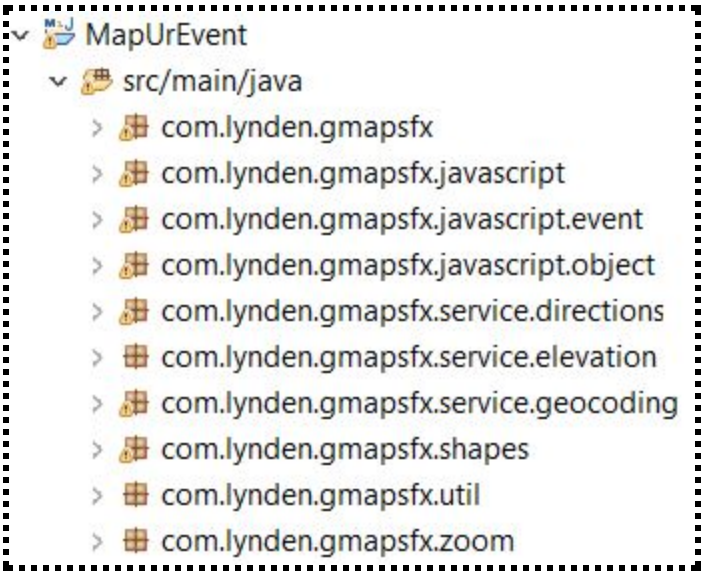
1. Problem Statement

Nowadays, people always have to plan things orderly to make their lives easier and there are lots of tools can help people to plan their schedules. For example, I usually utilizes to maintain my school stuff to manage my time. Therefore, I come up with the idea that if there is a program, which can help you manage your schedules visually on the map. I think the program will be useful.

Here are the problems will be addressed:

- Users can know where their events are directly on the map.
- Users can plan their events on the user-friendly map application.
- The program will make the robust management by storing the XML.

2. Software Architecture



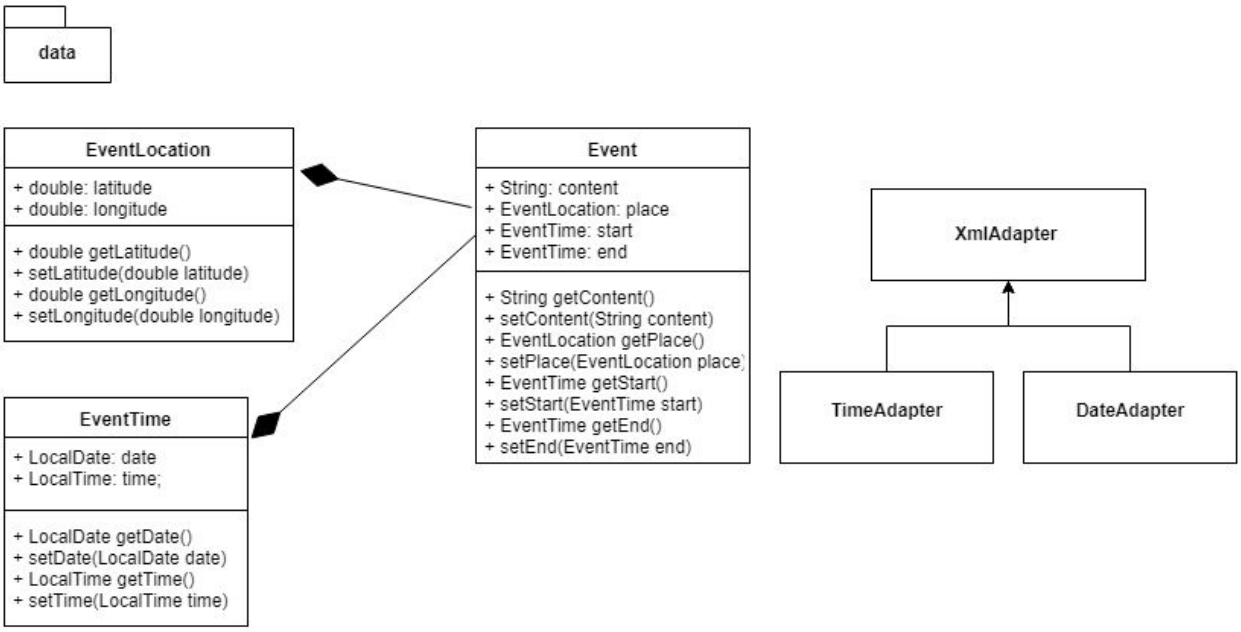
A. Reference Library

In this project, I reference two external libraries to help me finish my work:

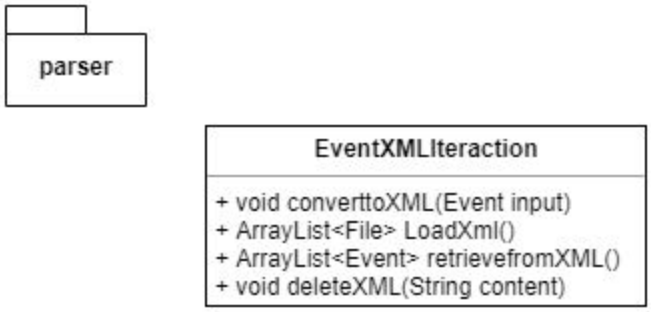
- **GMapsFX :** This is shown on the picture below. GMapsFX provides a wrapper to the Google Map's Javascript API, allowing the programmers to use and interact with maps using a pure Java API. Currently there is only support for a fraction of the Google Maps Javascript API, and the documentation is sparse, and this is opened sourced library. This is the reference: https://github.com/rterp/GMapsFX

- **JFoenix :** JFoenix is an open source java library, that implements Google Material Design using java components. This is the reference: https://github.com/jfoenixadmin/JFoenix

B. Structure of the Application - Packages



(The graph shows the structure of data package.)



(The graph shows the structure of parser package.)

date package: The package basically contains the formation of all the objects I define. Event is the basic unit for the data for the storage and the display of the data on the map. An event object consists of a primitive type String (content) and three objects containing an EventLocatoin object and two EeventTime objects. On the other hand, TimeAdapter and DateAdapter extends the XMLAdapter in order to make the objects transform into XML file.

parser package: This package is for the data transformation of both directions: Java object to XML and XML to Java object. It also maintains the data storage in the application. The functions and the implementation will be described more detailed in section C.

Other packages: As I mentioned before, they are used as reference libraries to construct the map.

C. Structure of the Application - Class

a. MainApplication.class:

It is under the package com.lynden.gmapsfx.

```
package com.lynden.gmapsfx;

import java.time.LocalDate;

public class MainApplicatoin extends Application implements MapComponentInitializedListener,
    ElevationServiceCallback, GeocodingServiceCallback, DirectionsServiceCallback {

    protected GoogleMapView mapComponent;
    protected GoogleMap map;
    protected DirectionsPane directions;

    //GUI
    private TextField contentText;
    private Label showClcik;
    private JFXDatePicker startDate;
    private JFXDatePicker endDate;
    private JFXTimePicker startTime;
    private JFXTimePicker endTime;
    private Button sendData;
    private Button delete;
    private Marker currentMarker;

    //----Data
    private String content;
    private double latitude;
    private double longitude;

    private int numofEvents;
```

This class contains the display of the graphical user interface and the main interaction of the map. This picture shows the declaration of the variable and these variables will play roles of interacting with the map.

```
@Override
//Set up the environment
public void start(final Stage stage) throws Exception {
    mapComponent = new GoogleMapView();
    mapComponent.addMapInitializedListener(this);
    mapComponent.setDisableDoubleClick(true);

    BorderPane bp = new BorderPane();
    bp.setCenter(mapComponent);

    setupRightTab(bp);

    Scene scene = new Scene(bp);
    stage.setScene(scene);
    stage.show();
}
```

The function **void start()** will set up the map the GUI of the map. There is also the function called **setupRightTab(bp)** which is in charge of my display of user interface.

```
//Set up the GUI
private void setupRightTab(BorderPane bp) {
    VBox background = new VBox();
    Label evetLabel;
    //Event
    evetLabel = new Label("Event:");
    contentText = new TextField ();
    contentText.setText(" ");
    HBox hb = new HBox();
    hb.getChildren().addAll(evetLabel, contentText);
    hb.setSpacing(10);

    //Location
    showClcik = new Label();

    //Date
    startDate = new JFXDatePicker();
    endDate = new JFXDatePicker();

    //Time
    startTime = new JFXTimePicker();
    endTime = new JFXTimePicker();
```

```
//Container of date and time
GridPane gridPane = new GridPane();
gridPane.setHgap(10);
gridPane.setVgap(10);
Label label1 = new Label("From");
Label label2 = new Label("To");
gridPane.add(label1, 0, 0);
gridPane.add(label2, 0, 3);
gridPane.add(startDate, 0, 1);
gridPane.add(startTime, 0, 2);
gridPane.add(endDate, 0, 4);
gridPane.add(endTime, 0, 5);
//Submit
sendData = new Button("Submit");

//Delete
delete = new Button("Delete");

//Background Setting
background.setPadding(new Insets(0, 20, 10, 20));
background.setSpacing(10);
background.getChildren().addAll(evetLabel, contentText,
        new Label("Locatoin: "),
        showClcik, gridPane, sendData,delete);
bp.setLeft(background);
```

This is the **void setupRightTab(BorderPane bp)** and it can be observed how the elements form into the GUI. For the event section, the HBox() would combine the TextField() and the Label("Event"). Due to the mechanism, I decide to make Label() to show where the location is. In addition, GridPane is used to be the container of the start time and the end time. Finally, I make all the elements above to be the children of BoderPane() bp. Now, my right tab pane in my GUI application will show them.

```
@Override
public void mapInitialized() {
    Thread t = new Thread( () -> {
       try {
           Thread.sleep(3000);
           System.out.println("Calling showDirections from Java");
           Platform.runLater(() -> mapComponent.getMap().hideDirectionsPane());
       } catch( Exception ex ) {
           ex.printStackTrace();
       }
    });
    t.start();
    //Once the map has been loaded by the Webview, initialize the map details.
    LatLong center = new LatLong(38.986849, -76.944750);
    mapComponent.addMapReadyListener(() -> {
        // This call will fail unless the map is completely ready.
        checkCenter(center);
    });
```

```
MapOptions options = new MapOptions();
options.center(center)
        .mapMarker(true)
        .zoom(9)
        .overviewMapControl(false)
        .panControl(false)
        .rotateControl(false)
        .scaleControl(false)
        .streetViewControl(false)
        .zoomControl(false)
        .mapType(MapTypeIdEnum.ROADMAP);

map = mapComponent.createMap(options);

map.setHeading(123.2);//map.showDirectionPane();
```

The funcion **void mapInitialized()** in the MainApplication.class and obiovusly, by the meaning of words, this is used for the initializing the map. Therefore, many interaction mechanisms of the elements are defined in this function. There are mainly three interactions. First of all, it is the display of location. Every time I click on the map, one of the label of my right tab would display the location by (latitude, longitude). Secondly, when the user finishes typing the data he wants to input, he can click the button named submit. The program transforms the datas from Java objects to XML files. Finally, it is deletion of the data. When the user inputs the name of the event and click delete button, the program deletes the event data.

```
//Allow the label shows the location on the label and
//update the current marker by clicking the mouse
map.addUIEventHandler(UIEventType.click, (JSObject obj) -> {
    LatLong ll = new LatLong((JSObject) obj.getMember("latLng"));
    latitude = ll.getLatitude();
    longitude = ll.getLongitude();
    showClcik.setText("latitude: "+latitude+"\n"+"longitude: "+longitude);

    if(currentMarker!=null) {
        currentMarker.setVisible(false);
        map.removeMarker(currentMarker);
    }
    MarkerOptions currentMarkerOptions = new MarkerOptions();
    currentMarkerOptions.position(ll).visible(true);
    currentMarker = new Marker(currentMarkerOptions);
    map.addMarker(currentMarker);
});
```

(This is for the interaction of location label.)

```
// Clicking submit
sendData.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent arg0) {
        System.out.println("In the Submit......");
        content = contentText.getText();
        LocalDate date_s = startDate.getValue();
        LocalDate date_e = endDate.getValue();
        LocalTime time_s = startTime.getValue();
        LocalTime time_e = endTime.getValue();
        // Avoid users mistakenly input data
        if( content.equals(" ") || latitude==0 || longitude==0 ||
            date_s == null || date_e == null ||
            time_s == null || time_e == null ||
            (date_s.compareTo(date_e) > 0) ||
            (date_s.compareTo(date_e) == 0 && time_s.compareTo(time_e) >=0)) {
            Alert alert = new Alert(AlertType.INFORMATION);
            alert.setTitle("Input Error");
            alert.setHeaderText("You should enter the right input foramt.");
            alert.setContentText("1. Content must be filled in.\n"+
                                 "2. Location must be pointed on the map.\n"+
                                 "3. Time duration must be selected.\n"+
                                 "4. Start time must be smaller then finish time.");
            alert.showAndWait();
        }
```

```
intln("Convert into XML......");
ew Event(content, new EventLocation(latitude,longitude),
ventTime(date_s, time_s),
ventTime(date_e, time_e));
etText(" ");

teraction.converttoXML(add);mapInitialized();
ption e) { e.printStackTrace(); }
```

(This is for the interaction of submit button.)

```
// Clicking delete
delete.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent arg0) {
        content = contentText.getText();
        if( content.equals(" ") ) {
            Alert alert = new Alert(AlertType.INFORMATION);
            alert.setTitle("Input Error");
            alert.setHeaderText("You should enter the right input foramt.");
            alert.setContentText("Content must be filled in.");
            alert.showAndWait();
        }
        else {
            EventXMLIteraction.deleteXML(content);
            mapInitialized();
        }

    }
});
```

(This is for the interaction of deletion button.)

```
//Set up the event information on the map
private void EventsShowonMap() {

    ArrayList<Event> events = EventXMLIteraction.retrievefromXML();
    numofEvents = events.size();

    for(int i=0; i<numofEvents;i++) {
        Event e = events.get(i);

        MarkerOptions temp = new MarkerOptions();
        LatLong markerLatLong = new LatLong(e.getPlace().getLatitude(),
                e.getPlace().getLongitude());
        temp.position(markerLatLong)
        .visible(true);

        Marker tempMarker = new Marker(temp);

        map.addMarker(tempMarker);
```

```
EventTime ss, ee;
ss = e.getStart();
ee = e.getEnd();

InfoWindowOptions infoOptions = new InfoWindowOptions();
infoOptions.content("<h3>Event: "+e.getContent()+"<br><br>"+
        ss.getDate().toString()+" "+ss.getTime().toString()+" to"+"<br>"+
        ee.getDate().toString()+" "+ee.getTime().toString()+"</h3>")
        .position(markerLatLong);

InfoWindow window = new InfoWindow(infoOptions);
map.addUIEventHandler(tempMarker, UIEventType.click, (JSObject obj) -> {
    window.open(map);
});
```

The function **void EventShowMap()** is called at the **MapInitialized()**. This function mainly purpose is to extract the data stored in the XML to the Java objects and add a marker on each event. When the program initializes the map, the map should update the events on the map no matter how the map increases or decrease events. Therefore, I called this function in the MapInitialized().

b. EventXMLIteraction.class:

It is under the parser package and this class is utilized to process the transformation between the XML to Java objects.

```
public static void converttoXML(Event input) throws Exception{
    JAXBContext contextObj = JAXBContext.newInstance(Event.class);

    Marshaller marshallerObj = contextObj.createMarshaller();
    marshallerObj.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

    Event e = input;
    File saveFolder = new File("enentData");
    saveFolder.mkdir();
    File saveData = null;
    int num = 1;
    while(true) {
        saveData = new File(saveFolder,"event"+num+".xml");
        if(saveData.exists())
            num = num + 1;
        else break;
    }
    marshallerObj.marshal(e, new FileOutputStream(saveData));
}
```

The function is for the users to convert the Java objects to XML files. When the user clicks the submit button. The program enters the interaction section of the submit button and it will call **void converttoXML(EventtoXML)** to convert the data into XML. I use JAXB to reach the goal of transforming. In this function, I also name every data and do the precautions to avoid the duplicate naming. It will maintain the files to be name as

event1.xml, event2.xml… to event(data.size).xml. In addition, every time the program inserts a new data, it detects whether the naming is being used. If the naming is being used, the iterator num will increase by 1 and finish the iteration.

```
private static ArrayList<File> LoadXml() throws Exception{
    File folder = new File("enentData/");
    File[] listOfFiles = folder.listFiles();
    ArrayList<File> xmlfiles = new ArrayList<>();

    for(int i = 0; i < listOfFiles.length; i++){
        String filename = listOfFiles[i].getName();
        if(filename.endsWith(".xml")||filename.endsWith(".XML")) {
            xmlfiles.add(listOfFiles[i]);
        }
    }
    return xmlfiles;
}
```

```
public static ArrayList<Event> retrievefromXML() {
    ArrayList<Event> events = new ArrayList<>();
    try {
        ArrayList<File> test = LoadXml();
        for(int i=0; i<test.size();i++) {
            File temp = test.get(i);
            JAXBContext jaxbContext = JAXBContext.newInstance(Event.class);
            Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
            Event e = (Event) jaxbUnmarshaller.unmarshal(temp);
            events.add(e);
        }
    } catch (Exception e) { e.printStackTrace(); }
    return events;
}
```

ArrayList<File> LoadXML() returns the ArrayList<File> which references to all the XML files. In addition, this function is called in the function **ArrayList<Event> retrievefromXML()** which utilizes the technique of JAXB to transform every XML file into Java object. Finally, the function store all the data from XML to the ArrayList of Event objects. It is called in the function **void EventShowMap()**, and then add marker on each event object displaying on the map. By this way, the user can visually see the information of events on the map.

```
public static void deleteXML(String content) {
    ArrayList<Event> events = new ArrayList<>();
    try {
        ArrayList<File> test = LoadXml();
        int deleteIndex = 0;
        for(int i=0; i<test.size();i++) {
            File temp = test.get(i);
            JAXBContext jaxbContext = JAXBContext.newInstance(Event.class);
            Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
            Event e = (Event) jaxbUnmarshaller.unmarshal(temp);
            //events.add(e);
            if (e.getContent().equals(content)) {
                deleteIndex = i+1;
                break;
            }
        }
        if(deleteIndex != 0) {
            File deleteFile = new File("enentData/event"+deleteIndex+".xml");
            deleteFile.delete();
        }
    } catch (Exception e) { e.printStackTrace(); }
}
```

When the user want to delete the events, he/she can specify which events and clicks delete button. Next, the program enters the section of delete button and call this function to help to identify the specified XML file. The function **void deleteXML(String content)** also utilizes JAXB to implement the deletion function and helps the data maintenance.

c. Event.class, EventLocation.class and EventTime.class:

They are all under data package and form the basic object in the program. Event represents one event and it consists of an EventLocation object and two EventTime objects.

```java
import javax.xml.bind.annotation.*;

@XmlRootElement(name = "Event")
public class Event {

    private String content;
    private EventLocation place;
    private EventTime start;
    private EventTime end;

    public Event(String content, EventLocation place, EventTime start, EventTime end) {
        this.setContent(content);
        this.setPlace(place);
        this.start = start;
        this.end = end;
    }

    public Event() {}

    @XmlElement
    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }
```

This is a fragment of Event class and the rest of it is similar. Getter and setter method for every variable in the class. In addition, the"@Xml..." is the notation of the usage of JAXB.

```java
@XmlRootElement(name = "EventLocation")
public class EventLocation {
    private double latitude;
    private double longitude;

    public EventLocation(double la, double lo) {
        this.setLatitude(la);
        this.setLongitude(lo);
    }
    public EventLocation() {}

    @XmlElement
    public double getLatitude() {
        return latitude;
    }

    public void setLatitude(double latitude) {
        this.latitude = latitude;
    }
    @XmlElement
    public double getLongitude() {
        return longitude;
    }

    public void setLongitude(double longitude) {
        this.longitude = longitude;
    }
}
```

```java
@XmlRootElement(name = "EventTime")
public class EventTime{

    private LocalDate date;
    private LocalTime time;

    public EventTime(LocalDate d, LocalTime t) {
        this.setDate(d);
        this.setTime(t);
    }
    public EventTime() {}

    @XmlElement
    @XmlJavaTypeAdapter(DateAdapter.class)
    public LocalDate getDate() {
        return date;
    }
    public void setDate(LocalDate date) {
        this.date = date;
    }

    @XmlElement
    @XmlJavaTypeAdapter(TimeAdapter.class)
    public LocalTime getTime() {
        return time;
    }
    public void setTime(LocalTime time) {
        this.time = time;
    }
}
```

The pictures above are classes of EventLocation and EventTime. There are "@Xml..." notation for the use of JAXB. Therefore, in my function **void**

converttoXML(EventtoXML) which I previously described can detect the corresponding value and extract the value from the Java object to XML.

d. DateAdapter.class and TimeAdapter .class

The reason why I create these two functions is that I use the objects of LocalDate and LocalTime to form the EventTime object. Therefore, in order to utilize JAXB to complete the transformation. I have to create the corresponding adapter to each class. Both of the classes all extend XmlAdapter which can help the programmer to finish the JAXB transformation.

```
public class DateAdapter extends XmlAdapter<String, LocalDate> {
    @Override
    public String marshal( LocalDate date ) throws Exception {
        return date.toString();
    }
    @Override
    public LocalDate unmarshal( String date ) throws Exception {
        return LocalDate.parse( date );
    }
}
```

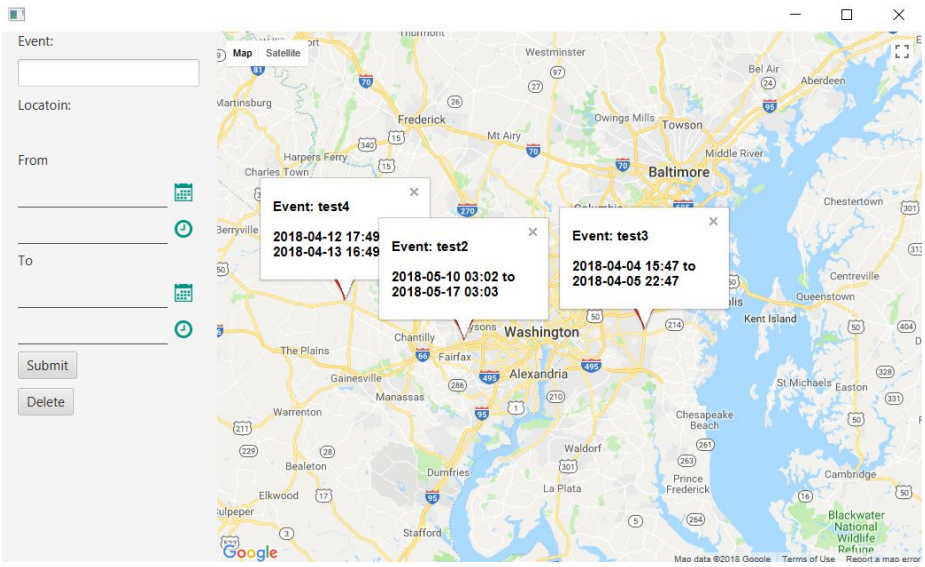
```
public class TimeAdapter extends XmlAdapter<String, LocalTime> {
    @Override
    public String marshal( LocalTime date ) throws Exception {
        return date.toString();
    }
    @Override
    public LocalTime unmarshal( String date ) throws Exception {
        return LocalTime.parse( date );
    }
}
```

By creating these kind of adapters, the program can recognize the LocalTime and LocalDate and transform them to the corresponding XML style.

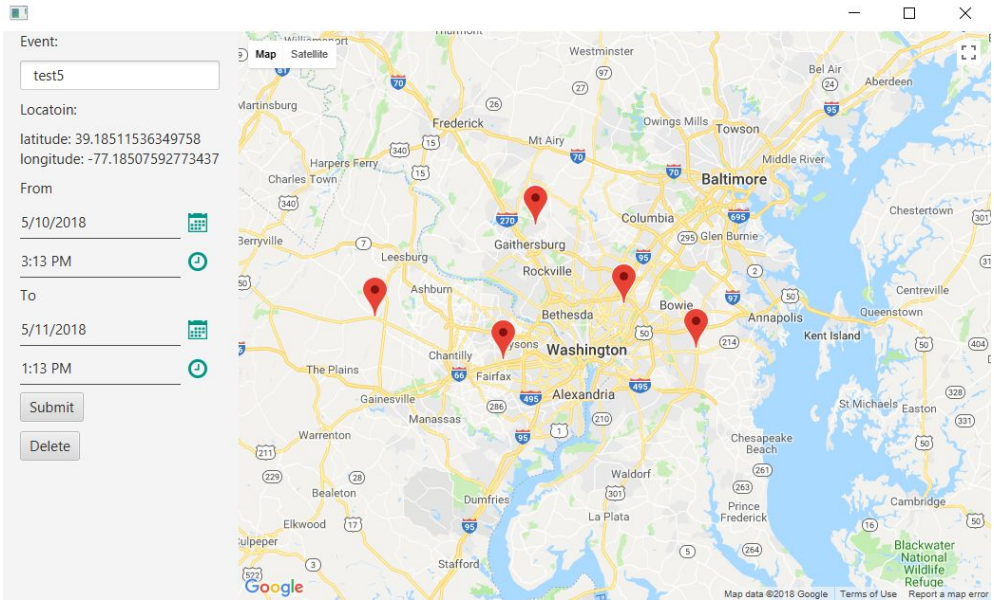
3. How to Use the Application?

The application is user-friendly, but I would still explain how it works step-by-step with pictures.

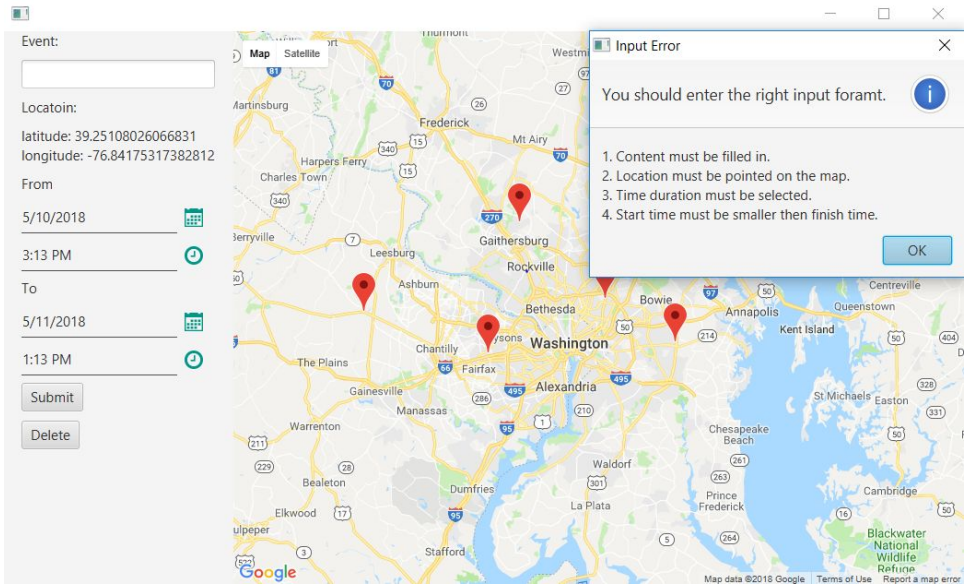
- Step 1 : Under the package com.lynden.gmapsfx. MainApplication.java, run the program, and it will show the following window. The red marker is for the previously stored event. The user can click the markers on the map and see the information on the application.



- Step 2 : Insert the desired input, and the location needs clicking on the map. The label on the right tab also shows the location value. After that, user can click the submit button and the map will refresh. However, you should insert the appropriate input value. For example, the start time should be earlier than the end time. I create the alert message if user fails to make the appropriate input value.

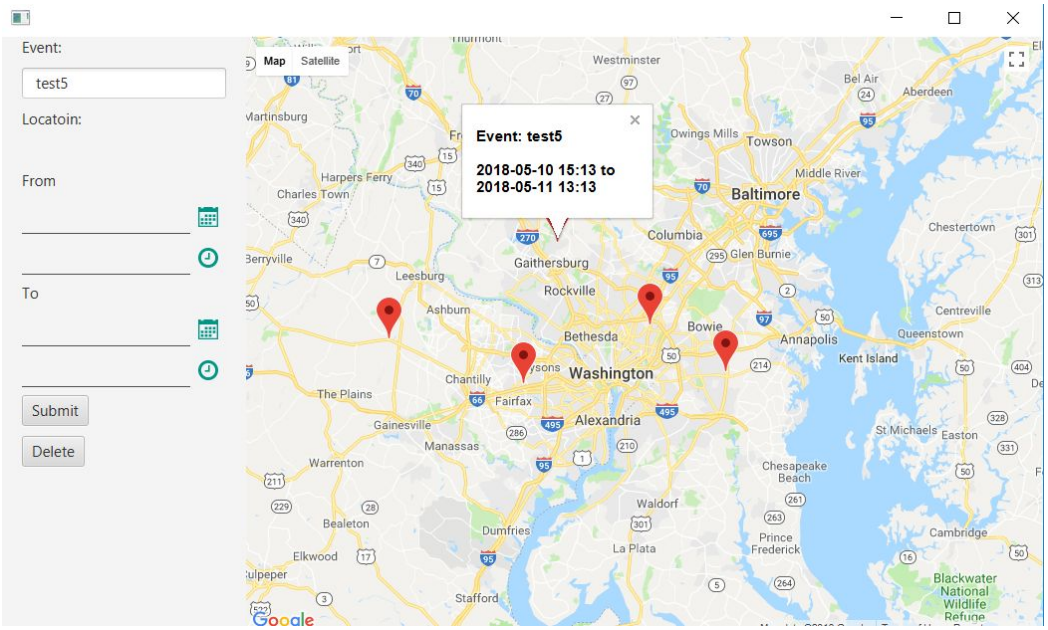


This is the example that user inserts all the input values.

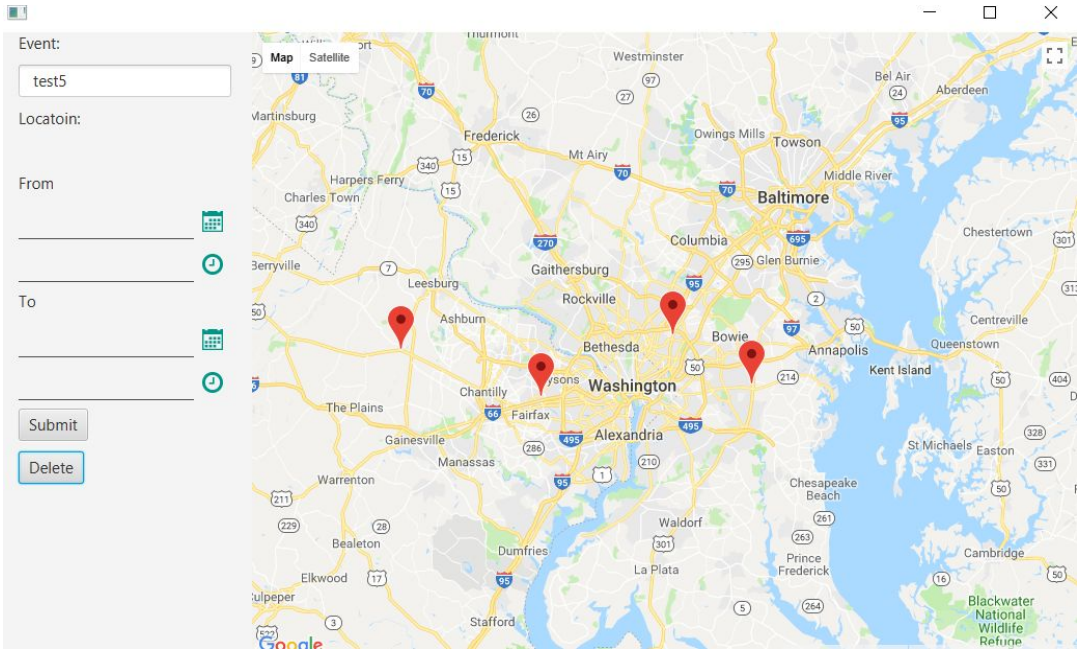


This is the example that user does not input the appropriate input.

- Step 3 : If the user wants to delete the event, he/she can input the event content and click the delete button.



Before click the deletion button



After clicking the delete button

4. Future Work

If I have more time to work on the project, I think I will keep adding several functions below:

- Make the GUI more customized. For example, according to the user's habit, the map application can change the style of the markers.
- Add the function of sorting up the events depends on the time or something else.
- Add the funcion of marking priority of the events.
- Make the representation of the location from (latitude, longitude) to the street name in the real world.
- Of course, continue trying to design more user-friendly GUI.