**2.**     **Technical Design**  
The team identified different components in the project and generated alternatives for those components. This section describes the lists of design alternatives and final solution for each component in the project. This section also describes the system-level overview and module-level description of the final design.  
  
**2.1.**   **Possible Solutions and Assessment of Alternatives**

This section describes the possible solutions and chosen alternative for the following components:

1. Data storage
2. Communication between Android application and server
3. Data interchange format between web and android applications

**2.1.1.**      **Alternatives for Data Storage**

The team has generated two alternatives for data-storage: MySQL database on web-server and SQLite database on android phone.

**Table 2.1.1 Alternatives for Data Storage**

|  |  |  |
| --- | --- | --- |
|  | MySQL Database on Web Server | SQLite Database on Android |
| Definition | Data is stored on MySQL database on the web-server. Data is communicated between android app and web-server using a networking protocol. | Data is stored on SQLite database on the android phone. Data is communicated between android app and web-server using a networking protocol. |
| Advantages | 1. MySQL database is automatically updated as user enters data using web-interface 2. Data entered through the user is sent across to the web-server, and MySQL database is updated | 1. Faster response time |
| Disadvantages | 1. Response time is slower, as the data needs to be communicated between the android app, and web-server 2. Requires constant connection to internet | 1. When data is entered by user through web-interface, the android users need to be notified to update their SQLite local database on the phone 2. Data entered by user using android app, requires to be sent to other android users and web-server, which involves complex implementation and synchronization logic 3. Potential outdated information on the android app |

One disadvantage of using web-server is that the app requires internet connection. But the number of mobile users subscribing for mobile broadband has increased significantly in the last five years [6]. From the table 2.1.1, advantages of using a web-server outweigh the disadvantages of local data storage on the phone. Therefore, the team has chosen *MySQL database on web-server* for data storage.

**2.1.2.**         **Alternatives for Communication between web and android applications**

The team has generated two alternatives for communication between web and android application: Hypertext Transfer Protocol (HTTP) and File Transfer Protocol (FTP).

**Table 2.1.2 Alternatives for Communication between web and android applications**

|  |  |  |
| --- | --- | --- |
|  | HTTP | FTP |
| Definition | HTTP is a protocol to transfer data across a network using request-response messaging pattern [7]. HTTP can be used to communicate between web and android applications. Android SDK provides packages for HTTP client communication. | FTP is a protocol used for transporting files across network [6]. FTP can be used to download and upload files between server and android application. Java FTP client library provides APIs that can be used to implement this alternative. |
| Advantages | 1. Allows transfer of data across network 2. Files and documents can be accessed using hyperlinks | 1. Easy to upload and download files |
| Disadvantages | 1. Downloading and uploading files require complex implementation. | 1. Can only be used for files, and not data |

One disadvantage of HTTP is that the implementation of file download/upload is complex. But as files can be viewed through hyperlinks, the effects of this disadvantage are minimal. Therefore, the team has chosen *HTTP* as the means of communication between web and android application.

**2.1.3.**         **Alternatives for Data Interchange Format**

The team has generated two alternatives for data interchange format: JavaScript Object Notation (JSON) and Extensible Markup Language (XML).

**Table 2.1.3 Alternatives for Data Interchange Format**

|  |  |  |
| --- | --- | --- |
|  | JSON | XML |
| Definition | JSON is a data-interchange format, which is programming language independent. The web application queries information from database and encodes the query into JSON objects. The JSON encoded output is then sent to android application. Android SDK provides API that supports extracting data from JSON objects. | XML is a markup language used for transferring and storing data. Web application queries data information from database, and writes the result in XML format, which is sent to the android application. The XML structure is then parsed on the android application. |

The implementation of JavaScript Object Notation (JSON) and Extensible Markup Language (XML) are quite similar on both web and android application. However, one major advantage of JSON over XML is that PHP provides API for encoding queries from MySQL directly into JSON format. On the other hand, the query obtained from MySQL needs further processing to be encoded into XML format. Therefore, the team has chosen *JSON* format for supporting data interchange between web and android application.

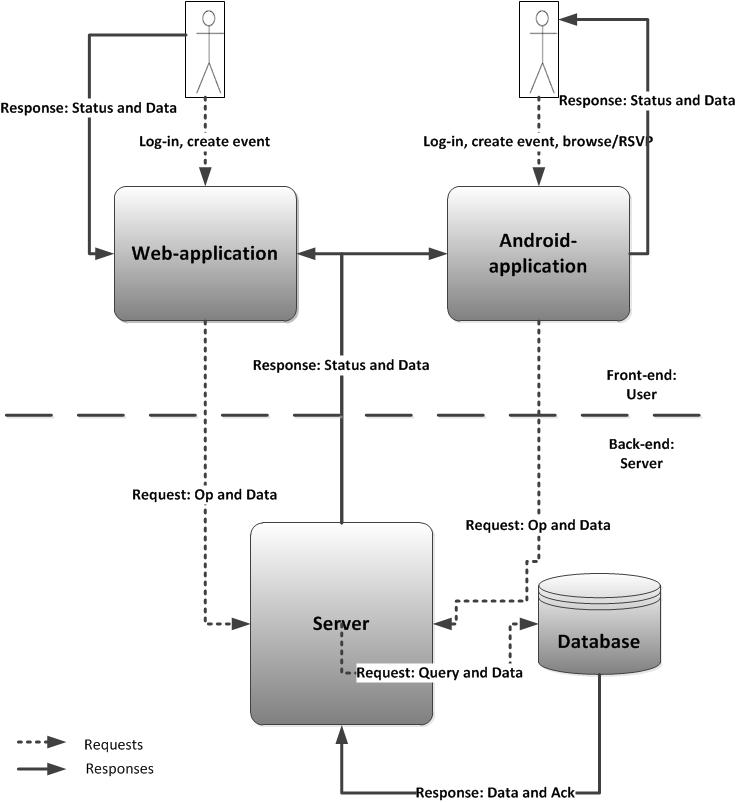
**2.2.         System-Level Overview**

This section provides the high-level description of main modules and sub-modules involved in the project. The project comprises of four main modules: *web-application, android-application, server,* and *database*. The *android application* is used by event attendees to browse through events and RSVP as desired. The *web* and *android applications* are used by event organizers to create events. The *server* interacts with web and android applications, and processes requests from these applications. The *database* is used to store and maintain data, and is controlled by the *server*.

These four modules have been further subdivided into a total of twelve sub-modules. Some features remain same across both *web* and *android applications*, therefore certain modules have similar functionality. Table 2.2.1 and Figure 2.2.1 provide a high-level description of the modules and sub-modules. The detailed descriptions are provided in Section 2.3 (Module-level description section).

**Table 2.2.1 High-level Module Description**

|  |  |
| --- | --- |
| **Module** | **Description** |
| **Web-application** Sub-modules: Register, Log-in, EventCreator | Provides a web-interface between event organizers and *server* to create account, to log-in, and to add, modify, or delete events. |
| **Android-application** Sub-modules: Register, Log-in, EventBrowser, EventCreator | Provides an android-interface between users (both event organizers and attendees) and *server* to create account, to log-in, browse events, RSVP, and to add, modify, or delete events. |
| **Server** Sub-modules: Get, Set, Validate | *Server* processes requests from the *web* and *android applications*, and converts requests into corresponding internal operations (Get, Set or Validate). Requests from these applications could be of the form: create-account, log-in, browse/RSVP events, create/modify events, or view. The *server* updates the contents of database if the operation is Set, and fetches data from the *database* if the operation is Get or Validate. |
| **Database** | Stores and maintains the user and event data. The *database* is managed by the *server* module |

**Figure 2.2.1 System-Level Overview**

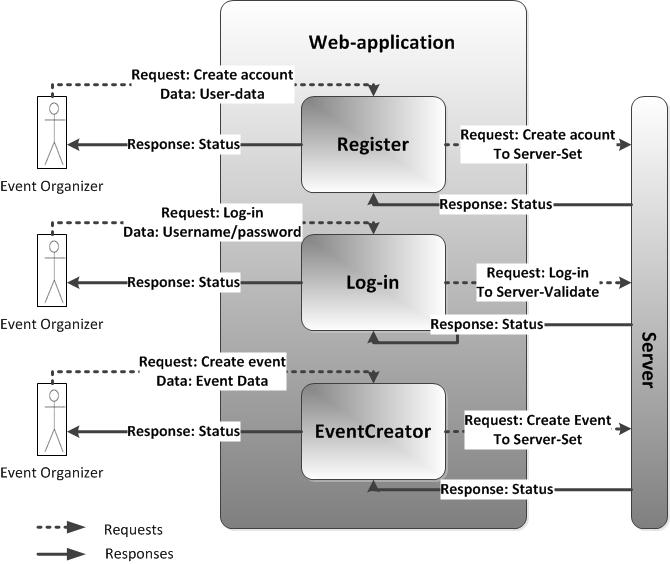
**2.3. Module-Level Description**

This section provides detailed description on the four modules and sub-modules discussed in section 2.2. Figures are included to better explain the flow of inputs and outputs per module.

* + 1. **Web-Application**

The *Web-Application* module has three sub modules: *Register*, *Login* and *EventCreator*. *Register* sub-module is responsible for signing up a new event organizer. *Login* sub-module allows an event organizer to login, and *EventCreator* sub-module allows an event organizer to create or update an event. The following table lists out the all the input-output interactions that can occur between these sub-modules and other modules.

**Figure 2.3.1 Web-application**

****

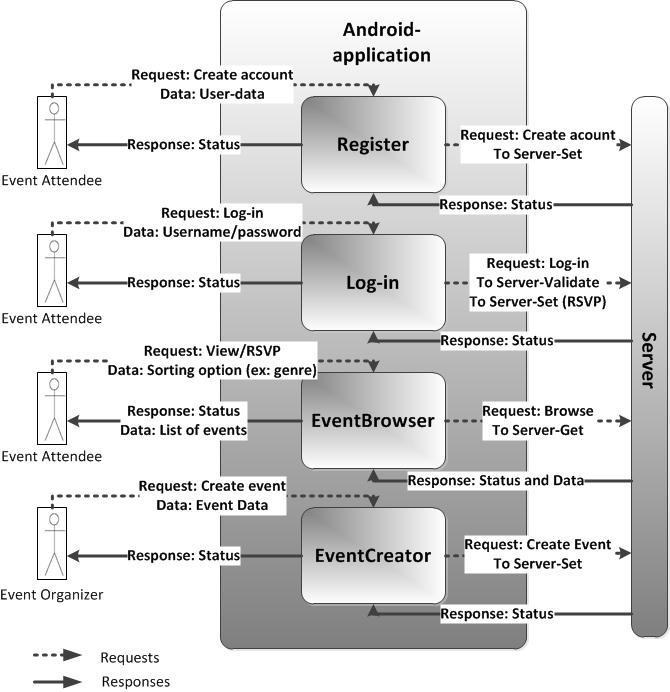
**Table 2.3.1 Web-Application module**

|  |  |  |
| --- | --- | --- |
| **Inputs** | **Outputs** | **Description** |
| **Register sub-module** | | |
| User-data Event-Organizer 🡪 Register | User-data  *Register* 🡪 Set | The user is required to make an account before he/she can use the service of the app. To do this, a user must choose the *Register* sub-module and enters their login name and password. The data packet is then sent to the Set sub-module in the Server. |
| Status  *Set* 🡪 *Register* | Status  *Register* 🡪Event-organizer | The *Set* sub-module responds back with a status message of success/failure which is forwarded back to the Event-organizer. |
| **EventCreator sub-module** | | |
| Data to create event  Event-Organizer🡪 *EventCreator* | Data to create event  *EventCreator* 🡪 *Set* | For an organizer to request an add/update of an event. He/she has to choose *EventCreator* module. *EventCreator* forwards this request to the *Server*, which invokes the *Set* sub-module. |
| Status  *Set* 🡪 *EventCreator* | Authentication message  *EventCreator*🡪 Event-organizer | The *Set* responds back with a status message of success/failure, which is sent back to the event-organizer. |
| **Login sub-module** | | |
| User info  Event-Organizer 🡪 *Login* | User info  *Login* 🡪*Validate* | The attendee/organizer is required to log in before he/she can use the app to create or update events. The user enters their user info, i.e. login name and password by choosing *Login* sub-module. *Login* forwards this request to the *Server*, which invokes the *Validate* sub-module. |
| Status  *Validate*🡪 *Login* | Status  *Login* 🡪 Event-organizer | The *Validate* sub-module responds back with an status message of success or failure which is forwarded to the Event-organizer. |

* + 1. **Android-application**

The *Android-Application* module has one sub-module in addition to *Web-Application* sub- modules: *Register*, *Login*, *EventCreator* and *EventBrowser*. *Register*, *Login* and *EventCreator* sub-modules function as described in section 2.3.1. *EventBrowser* allows an event attendee to view and RSVP to on-going events. The following table lists out the all the input-output interactions that can occur between these sub-modules and other modules.

**Figure 2.3.2 Android-application module**



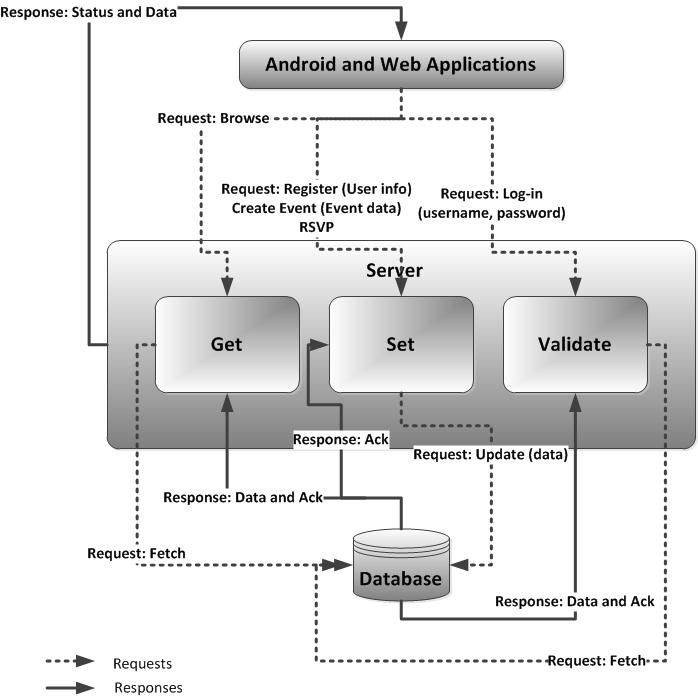
**Table 2.3.2. Android-Application module**

|  |  |  |
| --- | --- | --- |
| **Inputs** | **Outputs** | **Description** |
| **Register sub-module** | | |
| User-data Event-Attendee 🡪 Register | User-data  *Register* 🡪 Set | Refer to Table 2.3.1 |
| Status  *Set* 🡪 *Register* | Status  *Register* 🡪Event-Attendee | Refer to Table 2.3.1 |
| **EventCreator sub-module** | | |
| Refer to Table 2.3.1 | | |
| **Login sub-module** | | |
| User info  Event-Attendee 🡪 *Login* | User info  *Login* 🡪*Validate* | Refer to Table 2.3.1 |
| Status  *Validate*🡪 *Login* | Status  *Login* 🡪 Event-Attendee | Refer to Table 2.3.1 |
| **EventBrowser sub-module** | | |
| Event Metadata / RSVP request  Event-Attendee 🡪 *EventBrowser* | Event Metadata / RSVP request  *EventBrowser* 🡪*Get* | In order to view a list of events the event-attendee must go through *EventBrowser* sub-module. Event-attendee may provide event metadata to view specific type of events. *EventBrowser* forwards the event meta data to the server which invokes the *Get* sub module. RSVP request invokes the *Login* sub module. |
| Status and Event list  *Get* 🡪 *EventBrowser* | Status and Event list  *EventBrowser* 🡪 Event-Attendee | The *Get* responds back with a status message of success/failure and the list of events based on the meta data provided by the event-attendee. Both are sent back to the event-attendee. |

* + 1. **Server**

The server module is divided into two parts: *AndroidServiceEngine and WebAppServiceEngine.* These two engines have similar sub-modules which are dedicated to perform different tasks based on the input being received from either the *Android-application* or the *Web-application*, respectively. The following table lists out the all the input-output interactions that can occur between these sub-modules and other modules.

**Figure 2.3.3 Server Module**



**Table 2.3.3 Server module**

|  |  |  |
| --- | --- | --- |
| **Inputs** | **Outputs** | **Description** |
| **Validate sub-module** | | |
| User info  *Login* 🡪 *Validate* | Query  *Validate* 🡪 *Database* | The user enters their user info while signing in, i.e. login name and password which invokes the *Validate* module. This module checks if the login-password combination exists by sending a query to the database where all the user info is stored |
| Status  *Database* 🡪 *Validate* | Status  *Validate* 🡪 *Login* | The *database* responds back with status message of success or failure which is forwarded to the user |
| **Set sub-module** | | |
| RSVP  *EventBrowser* 🡪 *Set* | Query to update  *Set* 🡪 *Database* | The attendee requests to “RSVP” to a certain event, which in the form of an update query is sent to the *database* |
| Status  *Database* 🡪 *Set* | Status  *Set* 🡪 *EventBrowser* | The *database* responds back with a status message of success or failure which is forwarded to the user |
| User info  *Register* 🡪 *Set* | Query to update  *Set* 🡪 *Database* | The user enters their user-info while creating an account, which invokes the *Set* sub-module. The request in the form of an update query is sent to the *database* |
| Status | Status  *Set* 🡪 *Register/EventCreator* | The *database* responds back with a status message of success or failure which is forwarded back to the user |
| Data to create event  *EventCreator* 🡪 *Set* | Query to update  *Set* 🡪 *Database* | The organizer requests to add/update an event. This invokes the *Set* sub-module. The request in the form of an update query is sent to the *database* |
| **Get sub-module** | | |
| Browsing/viewing request  *EventBrowser*/ *EventCreator* 🡪 *Get* | Query to retrieve data  *Get* 🡪 *Database* | The attendee/organizer makes a request to browse or view event. This invokes the Get sub-module. This request is sent in the form of a query to the *database* where are all the events are stored.  *NOTE: The browsing request can only be made through the android-application interface.* |
| Data and status  *Database* 🡪 *Get* | Data and status  *Get* 🡪 *EventBrowser*/ *EventCreator* | The *database* responds back to the *Get* sub-module with the data and the status message with a token of either success or failure. This data and status is sent to the attendee/organizer. |

* + 1. **Database**

The database is responsible for executing and returning the results (data and status) back to the server.

**Table 2.3.4 Database Module**

|  |  |  |
| --- | --- | --- |
| **Inputs** | **Outputs** | **Description** |
| Query and Data *Server* 🡪 *Database* | Data and status  *Database*->*Server* | The *Server* will send a query (and data) to the *Database*. The *Database*, on executing the query will send Data and status. |