Project Globus

Senior Design I

David Crane

Jesse Miller

Kelsey Crea

Taylor Olson

**ABSTRACT**

Project Globus is a group management system designed to work for a variety of groups, both casual and professional. Whether it’s a group of friends planning a weekly board gaming session, student’s working on a school project or a business team working on a company’s next biggest product, project Globus will provide an innovative, simplistic and versatile experience to keep the group organized via document sharing, group notifications, group calendars and group messaging.

1. **Introduction**

Project Globus is our senior design project that is a group management system. We decided to implement an Android application for senior design I and then make the application cross platform for senior design II on IOS and as a web page. We have implemented the Android platform for senior design I. Throughout this paper, we discuss how the project started and how we planned the project through the waterfall approach with a dedicated timeline, as well as the design of the user interface and how the primary application features were developed. Finally, we conclude with issues and problems that we encountered while we were working on this project in senior design I as well as our use cases for the application that we have developed. Project Globus is a group management system designed to work for a variety of groups, both casual and professional.

1. **Origins, Preplanning and Planning**

This project idea started in early September of 2014 with the needs of one of our group member’s business fraternities, Alpha Kappa Psi, needs for a better group collaboration application. As a group, we decided to stem the project from Android to IOS and web page and that we would tackle the Android application during senior design I and the IOS application and web page in senior design II. We decided on the name “Project Globus” because it incorporates both group and globe into one word, Globus, which is how far our application can span.

As a team, we sat down and discussed how to strategically plan what aspects we were looking for in the project. We decided that we wanted a versatile application that could be aimed at both professional and causal applications (see use case’s citation or something here). From this broad target audience, we then discussed the individual features that we wanted the application to be capable of having. We broke the project up into multiple sections in order to adhere to a waterfall approach of the project. The planning process was the first step that we took as a group in the project.

The first planning meeting that we had, discussed the advantages and disadvantages of different aspects of the application. We decided that we wanted a central calendar that the application would revolve around. The application would start off with a typical login page that most other applications have, leading into a group selection page, and from there, a centralized master group calendar, Google drive link, attendance check, blanket messages, message feed that we deemed ‘The Whiteboard’, and a permissions section would be included in the application. We then decided that the central database would be the backbone of the project and started discussing how we would structure the database. This first communication about the database was to develop table structures and relationships. Once the database development was underway, we started writing the Android application in eclipse. We developed the Android application almost until completion and will link the application with the database over winter break and complete the app. The Android application will be completely done by January 1st, 2015.

1. Database flow chart and description (DAVID)

The first definitions of our database were designed such that we could keep as many objects as separate as possible, to ease creation of our classes on the application side. In our design, there were five major tables that controlled the flow of information: the tables for *Users, Groups, Messages, Calendar,* and *Calendar\_Events*. These tables held the primary keys that would help validate data stored in the other tables using foreign keys. Those tables were *Group\_Members*, which tied users to groups, and *Event\_Members*, which is used for attendance and organization of events. See Image 1 for a complete database implementation table.

Setting up the database on a SQL server proved quite easy using Amazon’s Relational Database Service (RDS). Creation of the database, necessary tables, and granting remote access proved to be an easy task. The next step was writing the initial code for our application to interact with the database. We used a SQL driver issued by Oracle, and coded access functions that worked directly with the database, manipulating it as necessary to populate and control information within the tables. We knew this was a naïve implementation, but this first step was necessary for proof-of-concept connection with the database. Eventually, we would need to create a server-side application to handle requests from the clients for us, allowing for greater security and faster access times.

1. Application design (TAYLOR) and flow chart (KELSEY)

The design of the application tried to follow some very basic, but overlooked ideas of user interface design. When designing the user interface, we wanted to follow several fundamentals that would be strictly enforced throughout the entire product. Those fundamentals are: know the audience, consistency, keep it simple, and empower the user. Knowing the audience involved us sitting down and analyzing our target audience, which turned out to be very large. This meant that we didn’t have the luxury of catering the design to one specific type of user. Rather, we had to make an interface that wouldn’t be “too dumbed down” and “too simple,” while at the same time it could not be too complicated that non-technical users would be able to use it without having to spend much time learning. Consistency was a no-brainer for us. We wanted to make sure the user was familiar with all aspects of Globus and no one area of the system was extremely different than another. Keeping it simple goes back to knowing the audience. We decided to go with a simple color scheme of blue and gray. This would allow us to have a nice contrast while still keeping the user engaged on what’s important – the content. Finally, empowering the user was very important. We wanted the user to feel as if they were running a group system, not a group system running them. Therefore, we focus the interface of Globus to be very non-protruding to the group’s productivity. We are also keeping a very open mind and are going to get feedback from users when we are in the beta testing phase.

1. Login group pages and database interaction (JESSE)

When a uses logs in to Globus, we will need to connect to the Globus database and pass it the username and password that the user provided. If the user has been listed in the database and their password matches the stored password, than the application will immediately log him/her in and send the device he/she is using the id associated with the user. If the user creates a new account, then the user will have to fill out there information and pass it to the app. At this point, the app will query the database to determine if there is already an account with the username/email address given by the user. If there is, the app will not create the account and instead it will inform the user that the username is already in use. If it isn’t, then a new entry will be added to the users table in the database containing the given information and the user will be informed that his/her account has been created. The user will also be logged in to his/her new account.

1. Group selection pages and database interaction (JESSE)

After logging in, the user will be sent to a fragment containing a list of groups he/she is currently a member of. This list will be obtained through the database, querying it for all unique groups listed with this user’s id in our group members table (the table contains a list of group ids attached to user ids, showing what users are members of specific groups. The group names, ids and descriptions will all be pulled from the database, and fed into the list for the user to see. At this point, the user can either select one of his/her current groups, create his/her own group or join a new group. If the user chooses to create a new group, they will have to input the name, description and password of the group they wish to create. The app will then query the database to make sure the group name has not already been used. If it has, the app informs the user that he/she needs to change the name. If it hasn’t, the app will add a new entry into the groups table of the database, including not only the information provided, but also the user’s id which will be used in the future to give the creator administrator privileges. The user will then be sent to the Whiteboard If the user wishes to join a group, they have to enter the group’s id number into the app. The app will then query the database to see if there is a group with the given id, and ask the user if they wish to join the group if there is. Otherwise, the user will be prompted that no groups could be found, and they will be asked to enter in a new id. After the user either creates, joins or selects a group he/she is already a member of, the app will send the user id and group id to the main activity of the application which will initially take them to the Whiteboard display.

1. Main application portions that are completed (JESSE)

In its current state, our Android implementation of the Globus application can be considered a good concept and (or) rendition of the final product. In the app, the login, create account, group selection, create group, join group, Google Drive and Whiteboard fragment have been completed in most every aspect except for database integration. That is to say, as soon as we are able to connect the app to the database, the majority of code we will need to add will be interacting with the database. Aside from database work, the only code that will be written for these fragments will be for fixing bugs and updating the user interface for its initial launch.

Along with these sections of the application, we also have the beginnings of our calendar fragment (the fragment that will allow users to see all group events and add new events to the group). The reason we consider this fragment more of a mock-up than a fully implemented fragment is because the majority of this code relies on the database, whether we need to pull a list of calendars to find the correct calendar to display, find all the events listed under a certain group or add new events to the group’s calendar. Until the database can be connected to the application, it will be very difficult to test the functions of this screen.

1. Database problems (DAVID and TAYLOR)

Once the database code was written, the next step was to use the code with the android application. This proved to be a very significant issue. At first, the entire android application would crashing, returning many errors, both android-based and SQL-based. After many hours of debugging, a trip to Jim Ward was necessary. He pointed us in the right direction by pointing out that our database adapter was compiled for regular Java rather than Android Java. Once a proper adapter was found on Oracle, only one error was given. That error was a connection issue that was given by our own database code. This turned out to be an issue with Android security not allowing the connection over the port. Because of the issue, we are developing server-side code to run all the functions that will feed to the mobile applications.

It was a serious setback to learn that Android could not maintain persistent SQL connections—one that made a server-side handler for database queries immediately necessary. However, there is no template or default code in Java to handle such tasks readily available. As such, we’re currently working to design our own client-server code to create and process requests efficiently.

The current code, still in progress, runs as an intermediary that creates a new thread for each client. Each thread then receives, validates, and executes the SQL commands on behalf of each client, before returning resulting information (if any). At the moment, we’re suffering from some sever concurrency issues, mostly involving correct thread termination after connections time out.

1. Additional problems (TAYLOR, KELSEY, and JESSE)

The navigation of the primary fragment of the application turned out to be a little less than trivial. This buttons ended up causing a memory leak within the app. It turned out that this was because the images were sized too large. Resizing them makes the memory leak cease, but the images are not as crisp as we would like. Further research into this will take place later on.

While working with the calendar fragment, many problems arose. First, we had to start learn about how Android systems store there user calendars and how to add new calendars. Aside from this, we also discovered that Android does not include a calendar object capable of showing events tied to certain days on the object itself. In order to allow users to see all group events in their corresponding days and times on the calendar object, we will either have to make our own custom calendar or download a third-party calendar. In the meantime, we have kept the current calendar object inside the fragment to give users an idea of what the completed fragment will look like.

1. Work for winter break (TAYLOR)

While initial development of the Android app comes to a close, winter break is going to be the time when primary development of the iOS app is going to take place. The difficult part of the mobile applications was designing the initial application and deciding where everything would go and how we wanted the user interface to be designed. This means that writing the iOS app will not be as planning-intensive because the design aspects of the application have already been decided. The primary goal of the iOS app is to match the Android app as closely as possible. This is to keep a consistent interface to for users within a group, no matter what operating system is used on their mobile device.

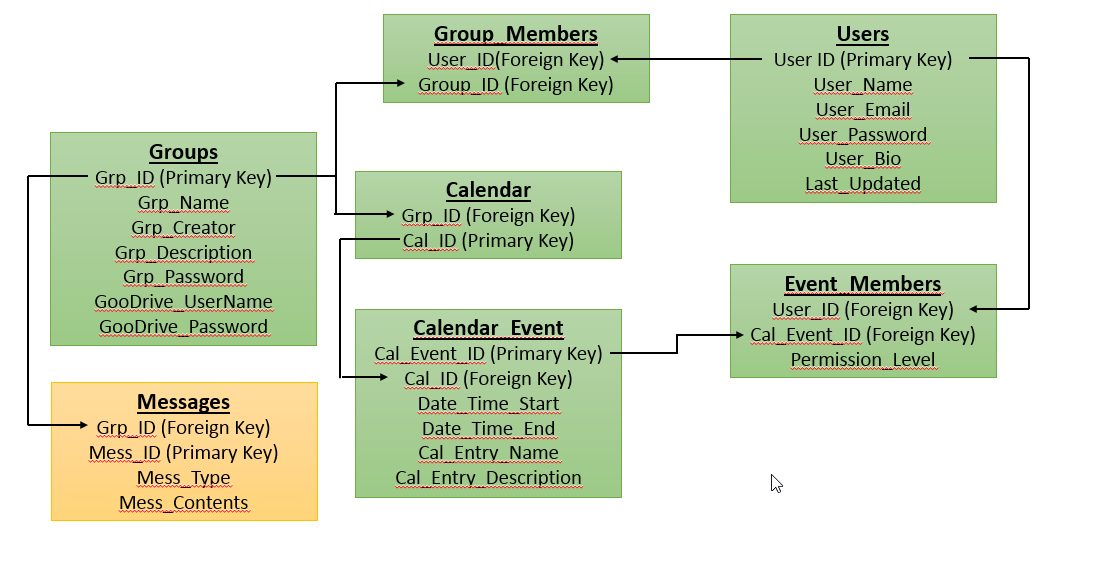
1. Time line (KELSEY)
2. Mobile project vs Senior Design I (TAYLOR, JESSE, and KELSEY)
   1. Database  
      Implementing the database for Mobile programming ended up not being a part of the equation. While attempting to work on database implementation, as we mentioned earlier in the paper, we ran into some issues. This in turn ended up being completely removed from the scope of our Mobile project. Rather we used hard-coded credentials, etc, for example purposes. While using this skeleton, we plan to implement our server-based database that would allow for the group management system to be available everywhere.
   2. Globus Features

In the Mobile Programming submission, the app is a very under developed version of the final product. It will not contain, or even discuss user privileges or attendance tracking/mandatory attendance. These are purely for the Senior Design version of our app. It will also not grow beyond the current state of the Globus app (this means it won’t ever have a functioning group calendar, or the ability to communicate with other users).

1. Use Cases (EVERYONE should have at least 1)
2. Plan for Senior Design II
   1. iOS App

As mentioned previously, the iOS application will be written. Globus is going to be a cross platform system, and that will allow users to use Android, iOS or a web interface. These will all focus on a really focused user interface that will allow the users to focus on their productivity and content, rather than the technology behind it.

1. Conclusion
2. Images

Image 1. Database Table Design

1. Works Cited