Developing an adaptable mobile app for group scheduling

By: Victoria Nielsen
Advised by: Holly Rushmeier

Abstract:

Today, scheduling a meeting with an acquaintance often involves a back-and-forth dialogue. One party requests to meet, another asks for availability, and a string of replies must follow. Discussing and deciding on dates and times that work for all participants is a hassle, particularly when it comes to group scheduling. To simplify this process, many scheduling applications have been developed, and work to address a multitude of problems – deciding on a meeting time, syncing calendars, avoiding scheduling conflicts, and so on. Their implementations also vary. Some require logins for existing applications, such as Google Calendar or Microsoft Outlook, while others require a unique login, or no login. Each method comes with pros and cons – requiring a Google login is convenient for Google users, but limits the application to Google users only. Meanwhile, requiring no login is simple and inclusive, but does not link users to a personal account. This project’s idea was crafted in the hopes of providing a middle-ground – an application that is accessible by all users, but with an easy login to store all account information. The result of this project is TimeSlots, a group scheduling iOS app designed to be compatible with several popular calendar services, in order for users with varying calendar service preferences to integrate their meetings into their default calendars. TimeSlots accomplishes this by emailing .ics event invitations to group participants, allowing them to quickly import events into their desired calendars. This paper discusses the thought process and implementation of the app, as well as limitations and barriers encountered throughout its development.

Introduction:

With various scheduling options available – digital calendars, paper calendars, emails, and beyond, it is often difficult to find a scheduling app that caters to each user’s preferences. This is particularly an issue with group scheduling, where it is likely that group members prefer different scheduling methods. Some may use Google Calendar, others may use Outlook, others may prefer an email, etc.

Many scheduling apps exist today, aiming to improve logistical planning for groups. Some applications help groups pick a time to meet, others help groups confirm a time slot and sync to digital calendars. Apps that sync to digital calendars often only sync to one calendar, such as Google calendar, and require that users log in with a Google calendar. This becomes disadvantageous for users who do not use Google calendar primarily, or who do not have a Google login. Meanwhile, other applications require no login at all. This is generally convenient for groups, since users do not need an account to access the scheduler. However, without a login, data is difficult to keep track of, reduced to various links and sent via email.

The goal of TimeSlots is to build a simple scheduling interface for users to login, create and sign-up for events within a group, and sync events to their personal calendar of choice.
Process & Challenges:

*TimeSlots* was developed on iOS 9 using Swift 2.2. The Udemy tutorial “iOS 9 and Swift 2: From Beginner to Paid Professional” by Mark Price provided a foundation for iOS 9 and Swift 2, influencing the project’s file organization and Firebase database.

Login and Auth:

Users are able to log in with a phone number. Phone numbers are verified using Fabric’s Digit Kit through Cocoapods, requiring the user to enter a verification code that is texted to the inputted phone number. This is implemented such that *TimeSlots* can remain as much in the background as possible, removing the need for the user to remember a new username. Once a phone number is verified, Firebase’s built-in user authentication is used to generate unique user ID’s and authentication tokens.

Firebase Database:

Firebase is a real-time, NoSQL database service with built-in user authentication, proving convenient for first-time iOS app developers. Because the data is hosted on a cloud server, it can be updated by many users in real-time, with no extra work for the developer. However, while convenient, this also means the server is out of the developer’s control, which can be a disadvantage. For the scope and timeline of this project, the convenience of a cloud database with built-in authentication outweighed this disadvantage.

Firebase stores all data as JSON, accessible via asynchronous listeners attached to database references. Firebase works best with flattened data (rather than nested), since Firebase must iterate through JSON in order to retrieve data. This means data redundancy is best practice in order to quickly access data with two-way relationships. The *TimeSlots* database structure is therefore represented in several categories, with some data redundancy for quick reference access: groups, phone numbers, sign-ups, and users. These categories contain group information for loading group calendar data, phone numbers for access to the associated user ID, sign-ups based on user ID to track a user’s events, and user information associated with a user’s profile.

Group Calendar Interface:

The Group Calendar is implemented using a UICollectionView, with horizontal and vertical scrolling enabled, as well as a sticky header for dates to remain at the top of the view. A Brightec blog post (source embedded in source code) was used as the starting point of the calendar, and customized to create the desired interface. The resulting calendar is in table format, beginning at the current date and displaying the following two weeks of calendar data. If desired, the number of weeks could be easily modified without any alterations to the database.

Rather than a calendar interface that shows 24 hours of each day, the *TimeSlots* calendar organizes events in a table in order of start time. Not only is this a relatively simple implementation using UICollectionView Cells, but it is also an intuitive way for users to view data simply, without scrolling through all hours of each day.
The implementation of a horizontally and vertically scrolling table view proved to be challenging when loading information into the table. Since requests from Firebase are asynchronous, the data had to be stored in an external data structure until the request was completed, upon which a table reload occurs. The size of the table also had to be fetched before the page view loaded, so the maximum number of events are stored as a key-value pair in Firebase in order to retrieve the size of the table without waiting for all event data to be retrieved.

*Calendar API’s:*

To sync to users’ personal calendars, *TimeSlots* was originally intended to provide log-ins for Google calendar, Outlook calendar, and request access to iCal. An interface was built such that on sign-up, the user could indicate which calendar(s) they’d like to sync to.

Requesting access and adding / removing from iCal was successful using EventKit, and remains implemented in the final version. The Google Calendar API was also successful in creating a login and adding events to the user’s calendar. However, after initial testing, users were unable to exit out of the Google login API if they decided to cancel the login process. While most users would choose to proceed with the Google login rather than canceling out, it remained a problem that the app would become inaccessible to users who chose to deny access to their account. Finally, Outlook’s API required a Microsoft Azure Active Directory Authentication, which provided restricted login access to the API based on a client ID. This could not be extended to the general user, and was never successfully implemented into the project.

*Email Implementation:*

Because of the limited functionality of the app syncing thus far, a different approach was taken. Instead of logging into Google / Outlook using their respective API’s, *TimeSlots* sends an email containing the event invitation. The event invitation is an .ics file that is formatted in order to be compatible with Google Calendar and Outlook. While this adds an extra step for the user (the user must accept the invitation via email rather than it importing directly through *TimeSlots*), it is compatible with all of the major calendar services and is consistent with a fully-functioning app. Meanwhile, it still offers the convenience of a one-click addition to the user’s calendar of choice, with all event data immediately recognized in the .ics file.

One challenging aspect of email implementation was the varying formats accepted by Google calendar versus Outlook calendar. While the .ics formats are similar, Gmail reads each event in UTC time and converts it to the user’s local time zone, while Outlook reads each event in the time zone specified in the .ics file. This means that different .ics files had to be created for each calendar service. This is an inconvenience for a user who wishes to receive events compatible with both Google calendar and Outlook calendar – two .ics files must be attached, and are distinctively titled “google_event.ics” and “outlook_event.ics”.

A second inconvenience is that Apple’s MFMailComposeViewController has emailing regulations to prevent apps from automatically sending emails, for security and anti-spam purposes. This means the user must click “send” on the email form that appears when s/he confirms an event, rather than automatically receiving the desired email.
One final caveat is that emailing using iOS and Swift can only be tested on a real device – and thus cannot be tested using Xcode’s simulator. This does not affect its usability, but is useful to know for further testing purposes.

**Final Result:**

The final iteration of *TimeSlots* presents an interface for logging in, creating groups, adding and accepting events, syncing to iCal, and emailing event invitations. Aspects of the final result are detailed below.

*Login/Create Account:* A user creates an account using his/her phone number. (See Fig 1.A)

*Sync/Email Preferences:* The user is able to indicate if s/he would like to sync events to iCal, or if s/he would like to be emailed event invitations for Google/Outlook upon accepting an event. If yes, *TimeSlots* will request access to the user’s iCal, and/or prompt the user to send emails when an event is confirmed. (See Fig 1.A)

*Group Creation:* Groups are created by adding phone numbers of friends. Groups can be of any size, so in addition to coordinating large groups, the app could also be used to simply schedule a lunch between two people, providing an alternative to back-and-forth texting/emailing.

*Group Administration:* For each group, administrators are selected in order to control for who can add events to the group’s calendar. Only administrators have access to event creation.

*Event Creation:* Administrators add events by specifying a title, description, location, start date/time, and end date/time. Finally, the administrator can specify the number of slots available. If a group administrator wanted to fill three slots in a single shift, for example, then that can be easily implemented using this feature. (See Fig 1.A)

*Accepting/Canceling an Event:* Any member of the group can accept an event, or cancel their acceptance. Upon accepting an event, the event will be synced to their iCal and an email will be sent with an event invitation (if the user has specified this in his/her sync preferences). Upon cancelling their acceptance, the event will be removed from the user’s iCal. (See Fig 1.B)

*Managing preferences:* At any time, a user can choose to opt-out of emailing and iCal syncing.

**Testing and Feedback:**

User testing upheld the functionality of the app, and also exposed some possible improvements.

Eight users contributed to testing the application. Prior to testing, users were informed that the app was a group scheduling app, and were given dummy phone numbers that could be used to add group members during group creation. All users were able to successfully create an account, create groups, specify administrators, add, accept, and cancel events, and receive email invitations. When asked if the application was functional as a group scheduling app, all users responded yes and reported no bugs. However, the majority of applicants requested additional functionalities, such as ability to import contacts, invite friends outside of the app, and receive push notifications prior to an event occurring. These are all features that were originally detailed in the project proposal, and could be implemented with a more flexible timeline. One user also requested an export feature that would allow a user to dump all events on the group calendar into their iCal. When asked if they would use the application, the majority said that if the above functionalities were included, then yes. Some reported that their use would depend on the app gaining traction among friends.
Fig 1.A: Fabric Digits phone number verification (left); Calendar Sync Options (center); Event Creation interface (right).

Fig 1.B: Calendar of events for Group “Yale Developers” (left); Event Details page (center); Email page when user accepts time slot (right).
Another note is only two of the test users relied on digital calendar services for personal scheduling. Both of these users used Google calendar, and found the emailing method easy to use and convenient for scheduling. It would be interesting to test the app with users who rely on a wider range of calendar services.

**Discussion & Conclusion:**

*TimeSlots* is successful in its goal to provide a group interface for users to schedule and sign up for events, in a way that is unbiased toward any single calendar service. However, several limitations and improvements remain.
First, this application is only available on iOS devices, which immediately limits non-iOS users, defeating the purpose of an adaptable app. For such an app to be effective, it would have to be developed as an Android app as well, which was beyond the scope of this project. Furthermore, the app is currently limited to users who have already made an account with TimeSlots. As per user feedback, a text messaging system should be implemented to prompt the user to create an account with TimeSlots if one does not already exist. Other improvements, such as group messaging, user-to-user actions, location services, and those mentioned in user feedback, should also be implemented in order to be competitive with other scheduling applications. Lastly, additional user testing and automated testing should be performed for a more robust application.

However, within the scope of this project, TimeSlots successfully performs the basic capabilities of a group scheduling app, and allows users on different calendar services to export events to their calendars via email. Moreover, this project’s process of researching API’s, .ics file formatting, and various Swift 2 blogs/tutorials exposes the complexity behind a seemingly simple iOS app. While today’s calendar services provide useful and essential developer API’s, the inconsistencies between them force developers to cater to each API separately. If similar API’s were standardized, perhaps an app like TimeSlots would not have to rely on emailing .ics attachments, and instead could seamlessly access the various calendar services and import an event directly into the user’s calendar.

Advances such as these remain to be seen, but in a world of constantly updating technologies, consistency across similar API’s would be a useful step for developers.

**Acknowledgements:**

Thank you to my project advisor, Professor Holly Rushmeier, who met with me weekly throughout the course of this project and has been an invaluable advisor throughout my time at Yale.