Chronos Optim

Prakhar Sahay, Dewar Tan, Erica Chai

Motivation

We developed an instant-scheduling Android app because emailing back and forth, Doodle polls, and Microsoft and Google products rely on verbal heuristics. Our app leverages privileged, centralized access to every user's schedule without compromising privacy, to automatically find times at which the greatest number of people are available.

Introduction

The idea of our app derives from our everyday task of scheduling events based on our availabilities. For example, when you want to schedule a lunch with a friend, you would have to both compare your availabilities to see when both are free. Another example is scheduling a club meeting and completing a Doodle poll to pick the best time available.

However, this process can be simplified with the introduction of our app. The app stores a calendar of the user's personal events. Once an event is created, the app will list out all the possible times that the user and his team are mutually available, or list how many people are available.

The five major components to our project are 1) the view, 2) the local database, 3) the syncing layer, 4) the server and cloud, and 5) push notifications. This gives us a dual databasing system, a server with centralized access, and fast, secure communication.

In the front-end, the Events tab of the app appears as an ordinary list of events, that keeps track of a user's events. On the Team tab, it lists the teams or groups the users are in. Once a user is in a team, the team leader will be able to schedule an event for the team and this is where the back-end comes into play. The optimal times for the team members to meet will be calculated by an algorithm implemented on the PHP server and will be returned to the team leader for their one-tap selection. We've used the Parse service as our cloud database to store the events for each user, the teams, and the users themselves.

1. Front-End

-Permissions

We used INTERNET permission for communicating with the server and VIBRATE permission for our custom notifications. Parse push notifications also require ACCESS_NETWORK_STATE, WAKE_LOCK, GET_ACCOUNTS, and Google Cloud Messaging's RECEIVE.

-StarterApplication

In addition to Parse push notification boilerplate code, it has three functions, getTeam(), getEvent(), and setPair(). These are used to smartly update the MainActivity without leaving the TeamDisplayActivity.

-Recycler View

We used the ***Recycler View*** library over the traditional list-view because it offered more flexibility in the layout of our list. It is essentially like list-view but allows me the option of changing the arrangement of each view's if needed (example, layout as Cards). We choose this type initially because we weren't sure how we would ultimately arrange our views so we wanted the flexibility to change if needed. In addition it allows us to use the itemTouchHelper for swiping on an item for removal.

-Toolbar

This is a replacement for the ***action bar*** and allows us to add items onto the menu to execute functions as well as allowing me to manipulate and add views onto the bar. We used it for the tabs feature in our app to switch between two different lists.

-Tab
The tabs were for navigating between the teams to which a user belongs to or to the events list.

-ItemTouchHelper
For directional swiping on the Recycler View list for prompting removal. We also added a restriction when selecting the date separator so that the user cannot remove it. Removing a single item from a date separator will also remove the separator.

-alertDialogBuilder
This creates a prompt for us to alert/check if the user wants an item removed from their events list.

-Android Support Design Library
This library we added to our project gave us the animated features seen in our app. For example entering into the TextEdit will show an animation when selected. Tabs are also part of this library.

-Time Picker and Date Picker
We used a switch case(that we set as an int when chosen) for the time picker and date picker so that we could implement multiple pickers that establishes the start and end time along with the date of our event.

-Auto Complete Text View
When adding members to a user's team on TeamDisplayActivity, this text view is populated with the usernames of every user found on the cloud.

2. Local Database
DbHelper is a stylized and unusually highly heavy implementation of an SQLiteOpenHelper subclass. In the ChronosOptim.db it maintains a table of teams and a table of events. The columns reflect the cloud database table columns exactly. Methods are overloaded to provide a simple API for event and team CRUD, and DbHelper even handles most of the contact with SyncBuffer, providing an even higher abstraction. For such a heavy implementation, it manages to have only one non-final field, SyncBuffer. Safe insertion is a technique made necessary by the dual databasing system. Parse will generate a unique and unchangeable object ID for each object that reaches the cloud, and the communication is handled asynchronously, but locally changes need to occur instantly. So a unique temporary ID is generated locally and sent to the cloud, and the uponSync() callback will receive the old and new IDs in the server response and update the SQLite database object ID.

3. Syncing Layer
-Simulation
The algorithm is explained in great detail in the supplementary document, "Syncing Algorithm". The Java simulation provides a proof-of-concept, modeling a multi-threading environment in which the SyncBuffer class fulfills the promises made in the document. A device and action are randomly selected, and as seen in the database dump, the buffered actions reach the server and database in order although they are sent asynchronously.

-Syncing Buffer
This class is the fundamental abstraction of front-end, back-end communication. The client simply instantiates the class and calls the send() method, and implements the uponSync() callback for responses.

-Client Device
This abstract class greatly simplifies the interactions each Activity has with the Syncing Buffer by providing basic methods for accessing SharedPreferences and requiring that the uponSync() callback be implemented.

-Server Ping
Many APIs were deprecated in Android 6.0 (Marshmallow) relating to android.permission.INTERNET and HTTP connections, so this is written essentially from scratch, and modularized enough so that a single instance sends up a single buffer of POST requests to the server. This class subclasses AsyncTask, as network operations are required by Android to work on a background thread to keep the main (UI) thread from hanging.

4. Back-End

-Parse Cloud Database
The Parse cloud service maintains all user data in a second location, so that the optimization algorithm can access any user's data without contacting their phones. There are four tables: Installation, Event, Team, and User. The Installation table is automatically managed by the push service, but column names are as follows:
Event: objectId, userId, title, description, location, startTime, endTime, date, subtitle
Team: objectId, name, members, description
User: objectId, uname, installation

For users and teams, we keep track of relatively little data by design. The events are the central model of the app, encapsulating the most data. We wrote ignite.php to simplify the interaction and Parse PHP SDK setup and API.

-Server
We implemented our server in PHP, due to its gigantic native libraries, unlike Perl or Ruby which require series of setup and installations for more powerful functions. For any data, it communicates directly with Parse and does not access the client SQLite databases. The three interacting scripts are core.php, optim.php, and class.php. core.php takes in the POST request, parses it into the individual POST requests, and handles them sequentially, using a switch statement to determine which function to use. These functions do Event and Team CRUD as well as pulling the entire list of users, recognizing an installation, aliasing a user, and broadcasting a chosen meeting time. optim.php holds the optim() function and all of its components, and class.php holds the Block and Weight class for optim().

-Cross-Listing Algorithm
This algorithm, implemented in optim.php, takes a team name, gets the users, gets the events of each user, squashes the schedules, pools the events together, runs the main function, and finally constrains the data. In squashing, the event start times and end times are sorted and merged, and as they are looped through, a Block (see classes.php) is created for every region where the user is in at least one meeting. In the main function, the event start times and end times are again sorted and merged, and as they are looped through, a Weight (see classes.php) is created for every region where a specific number

of people are in meetings. (In these loops, the key logic is that at some time point, all start times indicate that users have gone into a meeting and end times indicate that they have left a meeting. To prevent double-counting users who register for concurrent events, a preliminary squashing of each individual schedule is necessary.) Finally, the output is constrained to 9am to 5pm, at least an hour long of available time, and no more than 25 results.

5. Push Receiver

Notification pushing requires many background classes and support in the Gradle scripts and permissions in AndroidManifest, but Parse's SDK does the heavy-lifting. Every time a suggested meeting time is selected, pushes are sent to all other members of that team as invitations, and a push is sent to this user as a confirmation if the function completes successfully. Tapping the invitation notification automatically adds the event to the team member's schedule. Thus getNotification() and onPushReceive() are overridden when the notification is an invitation from a team member, and a custom notification is generated which has the desired Intent attached, to launch MainActivity and bundle the pre-created event. This custom notification is created by the Notifier class.

Design Workflow

What these five components culminate in is an app with two ListViews and three menu options: Create Event, Create Team, and Change Username. These three create changes in the SQLite database or SharedPreferences, and the Parse cloud database. Changing the username allows the user to be found and recognized more easily. Events are sorted by day, under a header, in the RecyclerView. Once a team is created, the user can search in the AutoCompleteTextView for members to add, and finally hit the "Find a Meeting Time" button. This button runs the optimization algorithm on the server, and returns a list of an incarnation of the PHP Weight class. With a single tap, all of his teammates receive an invitation with an embedded data and implicit Intent for automatically creating the event if the notification is opened. The user himself also receives a confirmation push notification.