February 2018

One Plus Two Mobile App

Dimitar M. Vouldjeff
Worcester Polytechnic Institute

Frank Egan
Worcester Polytechnic Institute

OnePlusTwo Mobile App

Project Team

Dimitar Vouldjeff dmvouldjeff@wpi.edu
Frank Egan fegan@wpi.edu

Project Advisor

Prof. Wilson Wong wwong2@wpi.edu

Abstract

Students' engagement is an important part of campus life experience whether that is attending weekly meetings with small clubs on campus or attending large social events and mixers with larger fraternities off campus. Small clubs have difficulty reaching larger audiences to grow their member base and retaining active members. Larger organizations struggle tracking members for events which must be regulated. Our aim was to solve these issues by creating a progressive web application that allows anyone to organize events and share invitations. These invitations allow tracking attendance in real-time and building re-engagement campaigns. We used ReactJS and Redux to build the client and Firebase's real-time database, cloud storage, and static website hosting to build the server.

Table of Contents

1. Introduction

Student involvement on campus is a key part of student's success in school. WPI has taken several steps to help clubs and organizations on campus share and manage their events. Most recently, WPI renewed its contract with Campus Labs OrgSync club management software license. This provides yet another platform for event sharing in addition to Facebook, Twitter, Daily Campus Emails, and private group chats. In 2016 an MQP was formed to explore the needs of an event management application on university campuses (Lyttle, Ross, Malofsky, McCarthy, & Bennet, T). Taking this scattered market into consideration our team has developed a new event management product OnePlusTwo. OnePlusTwo provides event management tools to hosts, such as creating events that can be shared with a simple hyperlink, real time guests arrival tracking, and a ticketing system that allows you to remind and re-engage with your guests through targeted emails. For guests, OnePlustwo provides real time tracking of how many people are attending an event, a centralized place to find event information, and does not not require users to download any apps. OnePlusTwo was developed as a Progressive Web App (Progressive web app checklist.), this means our application meets the following definition:

- **Progressive** - Work for every user, regardless of browser choice because they are built with progressive enhancement as a core tenet.
- **Responsive** - Fit any form factor: desktop, mobile, tablet, or forms yet to emerge.
- **Connectivity** independent - Service workers are permitted to work offline, or on low quality networks.
- **App-like** - Feel like an app to the user with app-style interactions and navigation.
- **Fresh** - Always up-to-date thanks to the service worker update process.
- **Safe** - Served via HTTPS to prevent snooping and ensure content has not been tampered with.
- **Discoverable** - Are identifiable as "applications" thanks to W3C manifests and service worker registration scope allowing search engines to find them.
- **Re-engageable** - Make re-engagement easy through features like push notifications.
- **Installable** - Allow users to "keep" apps they find most useful on their home screen without the hassle of an app store.
- **Linkable** - Easily shared via a URL and do not require complex installation.

As the Progressive Web App (PWA) standard is still being adopted by browsers and platform developers, not all users will have access to every features but they are offered as progressive enhancements to users whose devices support them. Based on these requirements,we decided to use Facebook's ReactJS library to develop a rich responsive front end, and Google's Firebase Real Time Database for our backend.

We employed an iterative development process that was guided by interviews with potential users of our application. This allowed us to evaluate why people were using their current solutions and provided us with insight into what the shortcomings of these current solutions were.

2. Research

2.1. Identifying Users and their Needs

During our interview process we discovered two different users for our application. The first are event planners of larger organization who are hosting events with over two dozen members. An example of a large organization would be a Fraternity or Sorority. These users face three distinct challenges when they plan events. Because their events are big, they need to be able to coordinate help form their members volunteering to help with the event. They need to track their member involvement within the organization to ensure they are fulfilling their member responsibilities. Finally they need to track the number of guests attending their event.

Smaller organizations on the other hand need to remind their members of events they are holding that day. They also want to re-engage with the new members that attend their events which requires being able to track their members involvement over time.

There are additional features that both organizations require. Hosts want to be able to share invites to their events with as little friction as possible. Guests should not be required to log in. Additionally, hosts want to share a hyperlink because they can be opened and shared without installing any extra applications. They also want their guests to have one centralized place where they can go to get information about the event without it getting lost in a stream of group messages.

We took these problems into consideration when designing the use cases and software requirements for our application. OnePlusTwo will address the need for hosts of large organizations to track their members activity through an invite system that will allow these hosts to see which members actually attended the required events. Because of OnePlusTwo's invite system, hosts will also be able to see which guests scanned into their events. These invites will provide a means for smaller organizations to track their guests' participation over time as they will need to be scanned in at the event. In order to accommodate smaller clubs' need for engagement messaging we will provide reminder and re-engagement emails to their guests that have signed into the app. Additionally as OnePlusTwo is a Progressive Web App, any organization can invite its members to an event with a simple link that will open in the browser. Guests can then check back to that link at any time to see the event's page where all the information can be centralized.

2.2. Personas

Designing the User Experience of our application was one of the first steps in the design process. We were guided by our interviews with students and event organizers. We then used these discussions to decide what features we would focus on and who our audience was. One common technique in the user experience design process is to imagine “personas” for your different users. These personas are meant to describe what the different users of our application might “look like”. From a UX perspective we will differentiate among the three personas we’ve defined below. However, for the purposes of software requirements we will not differentiate and will treat them as a single role in unified modeling language (UML) diagrams. Writing these personas entails writing a brief background about a user, describing their goals, and listing typical behaviours. We designed our user cases and user interface around four main user personas (Affairs, Assistant Secretary for Public, 2013).

1. A new small club president recently held a few events where they expected 20-30 people to attend but instead fewer than 10 showed up. They had a good turnout for events early in the term and they knew that students really enjoyed attending. They also kept their credit cards and boarding passes in Apple Pay.
2. A junior or senior who has an executive position and is considered a leader in his/her fraternity chapter or large-size organization. She is involved in Student Government and wants to make sure her organization's events are well attended. She recently made a push for accepting donations for the organization’s / fraternity’s philanthropy through Venmo, and moved all communication to a Slack group. She also keeps her credit cards and boarding passes in Apple Pay.
3. A member of a large organization who is only a sophomore. He/her are not too involved on campus or in their fraternity but they are always happy to help out with an event when their chapter holds one. They know that they have to attend a few events each semester to still be in good standing, so they want to make sure their attendance is counted. They recently downloaded Slack to join their chapter’s group and think it is useful.
4. Lastly a guest who is just a freshman looking to get invited to cool parties and new clubs so they can meet people. He/her are happy to use whatever app or system the club is using. After creating these personas we were able to systematically consider how each user would interact with the application. This influenced several design details, such as providing the ability to create multiple invite links from the same screen, or adding push notifications for event updates.

2.3. Alternatives and Competitors

2.3.1. Alternatives

Based on the interviews we conducted we have identified four methods that clubs and organization use to keep track of their events. The most common method for organizations to track members was to send out a link to a public Google Sheet in which members could write their names. This has the advantage that guests and members can add their name to the spreadsheet by simply clicking a link and typing their name. Users do not have to create an account or download an app. OnePlusTwo works right in their browser equally well on mobile or on the desktop. The disadvantage is that these links tend to get lost in old emails or buried in group messages. Additionally, editing spreadsheets on your phone is a lot of overhead for counting the number of guests signed up, and even if someone signs up you would have to create an additional spreadsheet for internal use to actually check them off.
The second method was to send out a Slack message to their groups and people could respond saying if they were available or not. This method is convenient because everyone in the group that needs to know instantly gets notified of the event. The disadvantage is that once this message gets posted it easily gets lost as more messages pour into that channel. This approach also does not provide a count of how many people are coming to an event or what their volunteer position might are events.

The third method was creating an event on Facebook and sharing it with all their friends. This provides the benefit of reaching all their friends easily with option to have guests respond “Going, “Not Going”, or “Maybe”. This provides at least a crude estimate of the number of guests expected to attend. The disadvantage to managing events through Facebook is that you can not share events without having guests login with a Facebook account. Additionally you can not assign roles to members with Facebook events.

The fourth and final method was creating an Outlook invite and sharing that through email. This was only used for small meetings where all the guests were WPI students. This method was also somewhat polarizing as some organizers kept track of all their events through Outlook while some guests had never opened their Outlook calendar.

2.3.2. Competitors

We chose to research Eventbrite as our main competitor. They are another event management organization that offers a suite of applications to host events and share invitations

electronically which users can access from their smartphone. With Eventbrite, organizers create a page with information about their event. From there they can share and promote their event on social media or through Eventbrite's local event listings. They also provide delete online ticket sales for paid events and offer guest management tools such as ticket scanners and total ticket sales. Eventbrite is free for all event sizes and makes money by taking a portion of the ticket sale prices. Over $1 billion in tickets sales were processed by Eventbrite last year alone. They offer applications for Android, iPhones, and iPads. You can use their website on desktop or mobile to create and manage some event details but you cannot scan guests into an event without using one of their native mobile apps. Guest can accept event invitations and get a ticket from their website as well.

2.3.3. OnePlusTwo Competitive Advantage

Our team evaluated the pros and cons of each system discussed above to help us narrow down the software requirements for OnePlusTwo. Our applications will encompass many of these advantages such as ease of use and preserving a low barrier to entry. Specifically OnePlusTwo has a richer set of event management tools than Facebook, Outlook, or Slack. Additionally OnePlusTwo is more flexible and entirely web based application unlike Eventbrite which require hosts and guests to install their native apps to use all of their features such as ticket scanning.

2.4. Customer Development

Although there are many event management tools available, we failed to see even one of them widely adopted on-campus. Orgsync was used for purely administrative purposes, while most of the clubs resorted to Facebook events and Google Spreadsheets for attendance management. We spoke with club presidents and event organizers to see what their biggest challenge was whenever hosting an event. Everything we have covered so far in this chapter is based on those interviews we conducted.

Customer Development is a formal methodology used by startups and businesses and it assumes that early ventures have untested hypotheses about their business such as (who are the customers, what features they want, how much are they willing to pay, etc.) (Steve Blank,). Before developing a product, you must define what your hypothesis is, design an experiment and derive insight to either validate the hypothesis, invalidate it, or modify it.

We defined what our core assumption and conducted interviews with our target users to see if we were on the right track. Our core assumption was:

- **Event organizers have trouble making sure people show up.**

With the interview scenario that we designed, we were conscious not to hint in any way the purpose of the conversation or the answers we were looking for. Asking "will you be willing to use feature A", or "have you ever had this problem" already bias people to what we want to hear and thus get no valuable insight from the conversations. We opted for open-ended questions such as "Tell us about the last event you organized", "What was the biggest challenge there", "How did you solve this challenge" and asked many "Whys".

2.4.1. Interview Protocol

The following is the interview protocol we used. It includes notes for the interviewer to be reminded where to lead the conversation. The interviewee does not see any of those. We provide overview of the conversations we have had in the Appendix section.

Tell us about the last event you've organized?

- asked to get a general idea on the type of event (used to determine if this is our target audience)
- asked to prevent priming on what we are trying to figure out

What is the biggest challenge whenever you've organized an event (before, during, after)?

- ask many WHYs
- do not suggest anything
- clearly define it
- understand what makes it a challenge

How do you solve that challenge?

- understand what the typical behavior is (our solution has to fit in this and simplify)

What are the alternative solutions

- (e.g. for interlocutor only: alternative to getting a shake for morning commute is getting a donut)?

Why don't you "hire" those alternatives?

- understand why person dislikes alternatives
- ask many WHYs
- shows what our application has to avoid doing

How do you organize the guest list for you events?

- What tools do they keep organized with
- how strict are they with guest lists/invites

How can you distinguish between your loyal attendees (fans) and flakers?

- How do you know who's new and who's a usual
- Do they do anything differently for the two groups?

How do you spread word about an event and make sure people will come?

- ask many WHYs
- understand their current process and struggle
- clarify seeming facts

3. Methodology

3.1. Software Development Life Cycle

We used an Iterative Waterfall software methodology. This included four distinct phases: Requirements Process, Design Process, Implementation Process, and Validation Process . After the sequential completion of each of these steps, we would start back at the beginning using what we had learned in the last development iteration and use the new information to guide the evolution of our application.

3.2. Requirements

Once we had decided on tackling an event management applications, we began pre-development interviewing for potential users of both large and small organizations in order to determine what requirements our application should have. Our interview scripts can be found in the Appendix. These focus on why people are using the solutions they currently are, and why they have not "hired an alternative."

We then used the feedback from the interview process as a guide in defining simple use cases and user stories required to that solve the problems that our interviewees were facing with their current solutions. These requirements defined exactly what kind of user (guest or host) would use this feature, how they would interact with the application and what the result would be. Once we had written a list of a few dozen use cases and user stories, we sorted and prioritized them.

3.3. Design

During the design process we decomposed the project into components. We then compared various strategies for realizing these software components. Based on these discussions we developed our solution technology stack outlined in the next chapter.

3.4. Implementation

Our implementation phase is detailed in chapter seven. In summary over the course of three iterations we translated each of the components described in our design process into

Javascript. We closely followed all of our database entity relationship diagrams and client side architecture diagrams in the creation of the back-end Firebase database.

3.5. Validation

During the testing or validation phase we evaluated each of our implementations on sample data and wrote unit tests in Jest (Jest JavaScript testing). to ensure individual components were behaving as intended. Once we developed tests for individual components we wrote integrations tests that included multiple components communicating between server and client. We recorded the status of these tests with a regression test spreadsheet. With this spreadsheet we could validate which features were working on a specific date and revert to that version if needed.

4. Software Environment

Early on in our project we wanted to select project development tools that focused on two main criteria. The first and most important of which is having the flexibility to modify as we went along in our project. The second metric we used to evaluate our tech stack was familiarity. We wanted to choose tools that had as small a learning curve as possible. This allowed our team to get up and running as quickly as possible as we did not have to spend time learning new paradigms.

4.1. NoSQL vs. SQL

Design is a critical step in the software development process. In database design, there are a plethora of architectures to choose from all with their own advantages and disadvantages. The two we considered were NoSQL and MySQL database architectures. NoSQL maximizes flexibility with a very loose structure and hierarchy that are not strongly enforced by the database itself. This can cause errors with mismatched data types, and also complicates data queries. On the other hand MySQL has a very structured and well defined hierarchy that we must decide on early in the application development process. If we decided to change how we stored or structured our data we would need to migrate our entire database which can be a hassle. When it came to database design, we wanted an architecture that met our first design criteria of flexibility. This is why we settled on using a NoSQL architecture.

4.2. Database Development

For our backend we chose Google's Firebase (Firebase.). We decided that the Firebase real time database would would provide the flexible platform we needed to prototype various features such as live guest tracking, and permission controlled invites. We also considered using a MySQL database for a Heroku project. A SQL database has the advantage of providing a strict data validation check that ensure consistency throughout the entire collection. There are also some performance and conveniences offered by using a SQL database when you have nested data structures which our application uses throughout. However, these strict validation checks come at the price of flexibility and they necessitate that we settle on a single architecture early on as we will not be able to migrate to a new database architecture easily. Ultimately we decided that these restrictions would not be worth it.

4.3. Front End Development

To create the front end for our application, we decided upon Facebook's React JS (React - A JavaScript library for building user interfaces.). This is a very popular, actively maintained Javascript framework that utilizes composable modular "Components" to build rich web apps. The other framework we evaluated was Angular which uses a similar modular component-like approach but is written in a language called TypeScript, a variation of JavaScript that is strongly typed. Angular also abstracts a lot of the underlying JavaScript to expressive HTML-like tags called angular directives that update on the webpage automatically as data changes. The deciding factor for choosing React over Angular was that React provides a much more flexible framework for client side development whereas Angular has fairly strict guidelines about what design patterns developers must utilize with their build tools.

4.4. Version Control

For Software Version Control, our team chose to use GitHub. Our team has prior experience in version control using Git . We were also especially fond of GitHub's integration with Travis CI services (Continuous Integration) which ran a suite of tests any time a Pull Request is opened and only allow merging based on their success (GitHub, 2016) . We found it to be indispensable in our implementation process ensuring that we could roll out updates certain that core application features continued to be functional.

4.5. Organizational Tools

4.5.1. Project Management

For project management we compared two products, Jira and Trello. Both systems offered a wide range of plugins and customization that support a variety of development strategies. However, the authors were more familiar with Trello. Additionally, the lowest payment tier is $10/month while Trello's lowest tier services are free. For these reasons we chose to use Trello.

4.5.2. Team Communications

For communications, we needed a tool that would allow our international team to stay in constant contact. The tool would also have to be cross device with support for iOS, Android, and

Desktop. We considered two options for text based communications, Slack and Facebook Messenger. Slack and Facebook Messenger are nearly identical in terms of feature support. Slack has several additional features such as “Communication Channels” which allow groups to organize their discussion into separate threads to stay organized (Where work happens.), Slack). For our small two person team, we felt that these extra communication hierarchies would be overkill while Facebook Messenger’s simple single threaded interface would fit the bill (Facebook messenger.). For video calling, we decided on Google Hangouts as it integrates nicely with Google calendar invites, shareable links to video calls, and cross device support without having to install additional apps or plugins (Lawrence Mello, 2015). No other tool has these features making it a natural choice.

4.5.3. File Management

For file management, our team chose to use Google Drive. Google Drive’s 15GB file storage limit, simple file version control, collaborative editing features, and cross platform support have made it a natural choice for many teams working on collaborative projects. Our team has had great success using it on a previous projects such as CS 3733 Software Engineering, and our IQPs.

5. Requirements

We utilized the Customer Development interviews (as mentioned in chapter 4.4) to devise a set of functionality that will serve our target audience the best (defined via personas in 4.2). Additionally, we wanted to differentiate from our competitors which necessitated additional considerations for our requirements (e.g. we wanted our QR code scanner to work within the browser).

5.1. Use Cases

The following are use cases we developed for the initial version of the application.

Title	Steps	Entry Criteria	Exit Criteria	Exceptions
Create an Event	1. Event Organizer (Organizer) navigates to create event page. 2. Organizer enters event title. 3. [Add More Info] 4. [Set Maximum Attendees] 5. Organizer initiates event creation. 6. App creates new event. 7. App navigates to event page.	Event Organizer has an account and has logged in.	New Event is created. Event page is shown.	Organizer auto-invites attendees from a previous event via [Import from Previous Event].
Add More Info	1. Event Organizer (Organizer) enters event description. 2. Organizer enters event location. 3. Organizer picks a date for the event. 4. Organizer picks a time for the event. 5. Organizer uploads a banner.	Event Organizer has already navigated to create event page or chosen to edit an existing event.	Event details are updated.	
Set	1. Event Organizer enters	Event Organizer has	The event attendee	

Maximum Attendees	the maximum number of attendees allowed.	already navigated to create event page or chosen to edit an existing event.	limitation is set.	
Import from Previous Event	1. Event Organizer (Organizer) navigates to import from previous event. 2. Organizer selects an old event he is an owner of. 3. Organizer initiates import. 4. App creates an invite for each of the attendees of the previous event. 5. App sends out an email with link to the invitation to each of the imported attendees.	Event Organizer has already navigated to create event page.	All the attendees from the previous event have been sent an email invitation.	
Validate Pass	1. App requests access to Organizer phone's camera. 2. Organizer grants camera access. 3. App loads QR code scanner. 4. Organizer verbally asks Attendee to show show his QR code. 5. Organizer points scanner at Attendee's QR. 6. App recognizes the Pass from the QR. 7. App checks if Pass is valid. 8. App updates QR scanner screen to show the Attendee's name and whether he can	Event Organizer has an event and has already navigated to the event details page. Attendee has accepted an invitation and has its QR code shown on the screen.	Event Organizer is told if the Attendee can enter the event. The Attendee's pass is invalidated.	

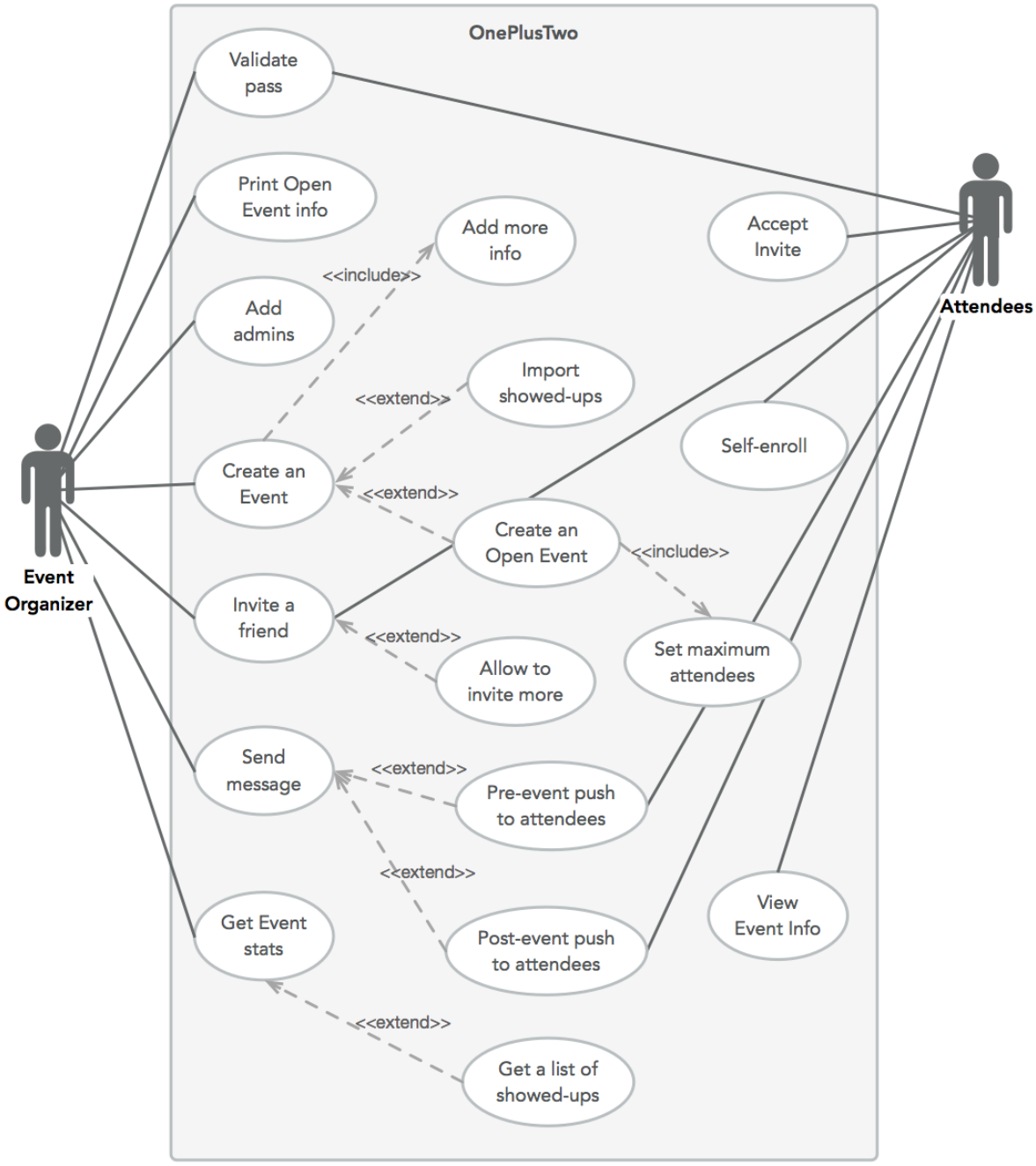
	enter the event. 9. App updates the Attendees screen to show that Pass has already been used and thus been invalidated.			
Invite a Friend	1. Event Organizer (Organizer) navigates to “invite friends” screen. 2. Organizer initiates a new invite creation. 3. App generates a new invite. 4. App shows the link the the invitation. 5. Organizer shares link with his friend (new guest) via a third-party text messaging app.	Event Organizer has an event and has already navigated to the event details page.	An invitation was created. New guest has received a link to invitation via text messaging app.	Event Organizer allows this new guest to bring his own friends.
Accept Invite	1. Attendee click on invitation link. 2. App shows the details of the event Attendee was invited to. 3. Attendee navigates to accept invite. 4. App shows [Sign-in / Create an Account] if Attendee isn’t already authenticated. 5. App creates a new Pass for Attendee. 6. App asks Attendee for notifications permission (if device permits). 7. Attendee grants or rejects notifications permission. 8. App marks the invite as used. 9. App navigates to event details screen.	New Guest (Attendee) has received a link to an event invitation.	Attendee has a Pass issued. Event details screen is shown.	

View Event Info	1. User navigates to an event she/he has been invited to. 2. App shows event details.	User is authenticated and has already accepted at least one Invite. User has navigated to list with his events.	Event details screen is shown.	
Send Message	1. Event Organizer (Organizer) navigates to new message page. 2. Organizer types in message. 3. Organizer initiates message delivery. 4. App adds message to list of all announcements. 5. App sends push notification to Android attendees who have given notifications permission. 6. App sends email to the rest of the attendees.	Event Organizer has an event and has already navigated to the event details page.	All attendees have received a push notification or email (depending on preference & device type).	
Get Event Stats	1. Event Organizer (Organizer) navigates to event stats page. 2. App computes event attendance by age groups and gender. 3. App computes peak arrival time distribution. 4. App generates charts to visually display stats. 5. App shows event stats page with charts.	Event Organizer has an event and has already navigated to the event details page.	Event statistics is generated and displayed.	
Sign-out	1. User navigates to sign-out page. 2. App destroys oath authentication token given by provider.	User has navigated to OnePlusTwo and has already signed into the app.	User has been signed out of the app.	

	3. App signs-out User.			
Sign-in / Create an account	1. User navigates to sign-in page. 2. User picks Facebook or Google as an authentication provider. 3. App redirects to respective provider. 4. User grants OnePlusTwo access via provider. 5. Provider redirects back to OnePlusTwo. 6. App creates an account if User does not exist in database. 7. App signs-in User.	User has navigated to OnePlusTwo, but has not signed into the app yet.	User has been signed into the app.	

5.2. Use Case Diagram

As we borrowed the user story technique from the agile methodology, we chose to employ use cases diagrams as a quick way to design and implement the application requirements. The following use case diagram represents the full functionality of the final version of the application with the exception of the Add admin use case.



Use Case diagram

5.3. Functional Requirements

There are several functional for our application, OnePlusTwo. Functional requirements are “what the system should do” (Eriksson, 2012).

5.3.1. Event Creation

Hosts will have accounts that allow them to create new events. They will be able to add information such as the event's name, address, and a general description of the event.

5.3.2. Event Sharing

The event host can then share this event's sign up link however they like, for example through a messaging app, Facebook, or email. Any guest who receives this invite link will receive a QR code that will be scanned at the event. This allows hosts to keep track of who exactly shows up to the event and who has been invited.

5.3.3. Event Management

In addition to general event information like date and time the host will be able to set additional details about the event such as how many invites are available. They will also be able to specify whether guests can sign up for roles when they accept the invitation. This targets the users who are currently using google sheets to organize their events but can check back on OnePlusTwo at any time for up to date information about the event without having to skim through old emails.

5.3.4. Guest Interactions

Guests will be able to share invitations based on the hosts setting for the event. When they arrive at the event they can present their invitation in the form of a QR code to enter the event. They will also receive up to date email or push notifications as updates from the hosts are published on OnePlusTwo.

5.4. User Stories

Even though we define three personas that will use OnePlusTwo, for software development purposes we would refer to only attendees and event organizers. This is because small club presidents and large event organizers will utilize virtually the same features. Private events are ones which can only be attended with an invite, whereas open events allow people to self-enroll.

5.4.1. Event Creation

- [ST-1] As a Website Visitor, I want to register and create a Private Event, so that I can closely control the guestlist.
- [ST-2] As an Event Organizer, I want to create a new Private/Open Event and import people who showed up from a previous event I organized, so that I start exciting my audience from early on.
- [ST-3] As an Event Organizer, I want to create an Open Event where people need to bring a friend to be eligible to attend, so that more people attend and are excited about my Event because bringing a friend means they are more committed.
- [ST-4] As an Event Organizer, I want to set a maximum number of attendees for my Open Event, so that there is space and seating for everyone.

5.4.2. Event Sharing

- [ST-5] As an Event Organizer, I want to invite many people to my Private Event using Text & Facebook, so that I can easily expand the guestlist.
- [ST-6] As an Event Organizer, I want to allow some people to invite additional attendees to my Private Event, so that new people can come.
- [ST-7] As an Event Organizer, I want to get a printable file with information about my Open Event and instructions on how to self-enroll (e.g. QR with link), so that it is effortless to use OnePlusTwo.

5.4.3. Event Management

- [ST-8] As an Event Organizer, I want to give “admin” rights to other people, so that it is easier to manage a big event.
- [ST-9] As an Event Organizer, I want to add additional information to my Event (e.g. location, time, description…), so that people know what they are signing-up for and where they should be.
- [ST-10] As an Event Organizer, I want to validate attendee’s passes at the entrance, so that I can keep track of actual attendance and let only eligible people in.
- [ST-11] As an Event Organizer, I want to know how many people did attend my event as well as basic demographics, so that I know what my target audience is.
- [ST-12] As an Event Organizer, I want to send a short message to attendees before the event (via email and push notifications), so that I can remind and excite them to come.

- [ST-13] As an Event Organizer, I want to send a short message to the people who came (actual attendees), so that I can stay on their radar and make them anticipate my next event.
- [ST-14] As an Event Organizer, I want to get a list with all invited attendees and people who showed up, so that I can have a record to use for other non-related purposes.

5.4.4. Guest Interactions

- [ST-15] As a Person Receiving an Invite, I want to quickly accept the invite and add it to my iOS/Android Wallet, so that I am reminded about the event and have the pass readily available at the entrance.
- [ST-16] As a Person who sees an interesting Event Ad on-campus (or hears about it) and wonders whether to go, I want to subscribe for interesting updates (and even quickly sign up for the event), so that I do not lose the opportunity to go.
- [ST-17] As an Attendee to an Open Event, I want to invite a friend of mine, so that I become eligible to attend closed events.
- [ST-18] As an Attendee, I want to occasionally get news and updates about the event, so that I do not forget about it and do not rule it out as boring and useless.
- [ST-19] As an Attendee, I want to see which of my friends are also attending the event, so that I am not anxious to go alone.
- [ST-20] As an Attendee, I want to refer back to the event details (location, date…), so that I do not forget those.
- [ST-21] As an Attendee, I want to see live stats about the event (number of people actually inside), so that I have more motivation to join.

5.5. Non-Functional Requirements

There are several non functional requirements for our application, OnePlusTwo. Nonfunctional requirements “describe how the system works” (Eriksson, 2012).

5.5.1. Ease of Use

The most important nonfunctional requirement for our application is ensuring that users can intuitively perform the most basic operations. We want to ensure users see a natural user interface that feels familiar and responsive.

5.5.2. Performance

Another important consideration for our application is the performance. Performance is important in every aspect of our application. Page load times are an important consideration. According to Google's metrics 53% of users will abandon a website after 3 seconds of loading (Google, 2016). Taking this into consideration we will keep all page load times in the application to under 3 seconds for a slow 3G connection.

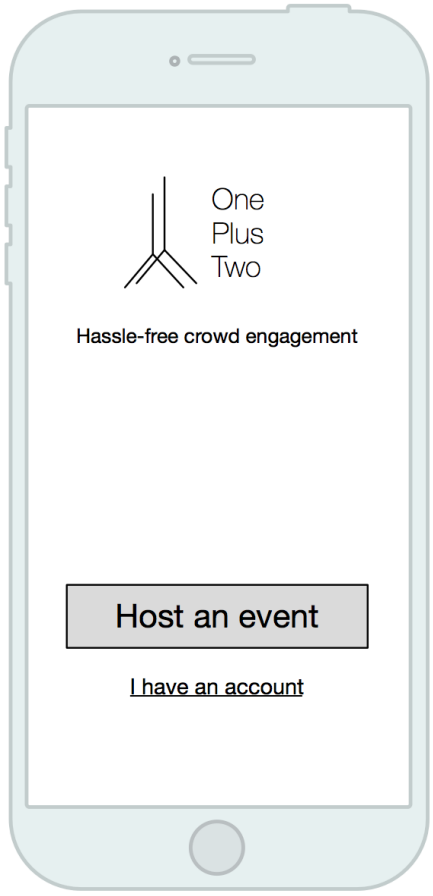
5.5.3. Scalability

OnePlusTwo is designed to be used by many organizations hosting events with potentially hundreds or thousands of guests all with their own tickets interacting on multiple devices accessing different events simultaneously. Our applications must continue to provide services and meet all the performance metrics measured above. The limiting factor is Firebase's service. OnePlusTwo is built on Firebase's spark plan, which includes a quota for 100 concurrent connections, and 1Gb of storage. However Firebase provides pricing plans that allow up to 100K concurrent connections and 1Tb of database storage.

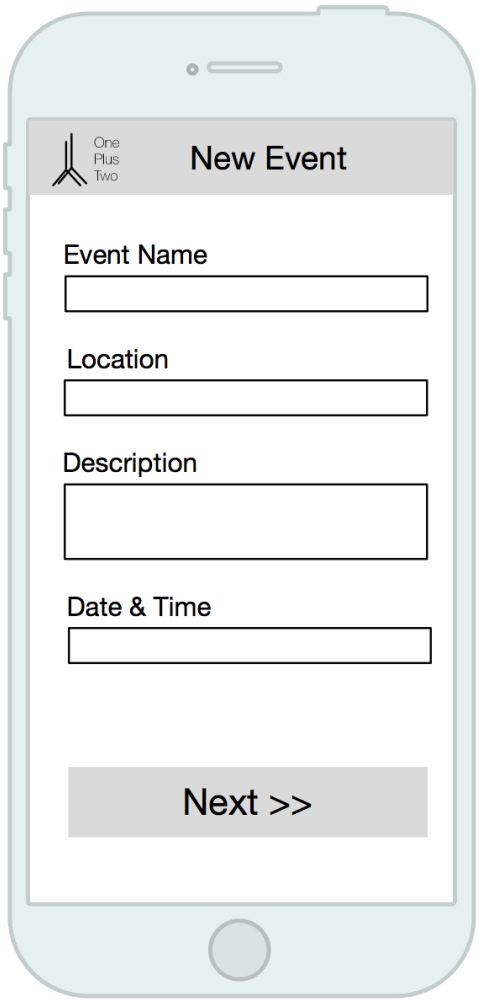
6. Design

6.1. Wireframes

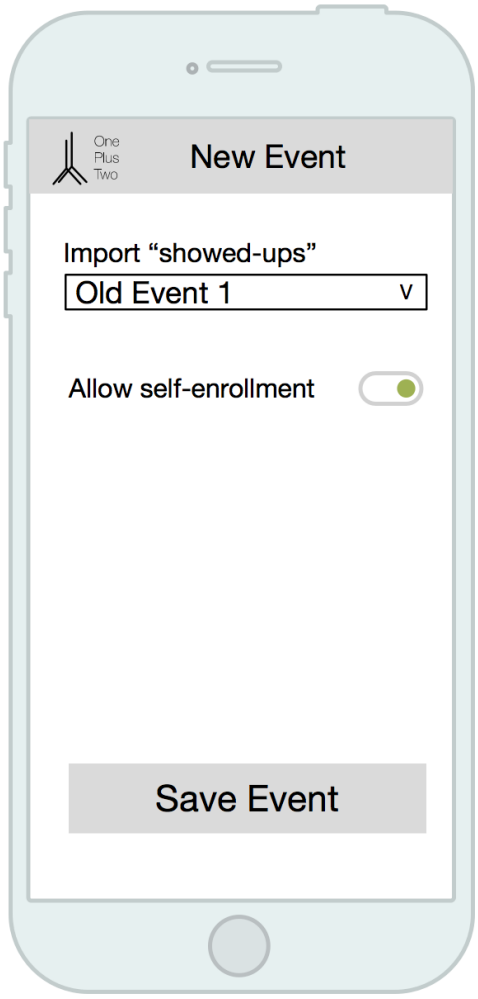
We used OmniGraffle with an additional UX stencil (plugin) to create the wireframes. When designing each screen we opted for a less clutter and visually highlighting the most important call-to-actions.


One
Plus
Two
Hassle-free crowd engagement
Host an event
I have an account

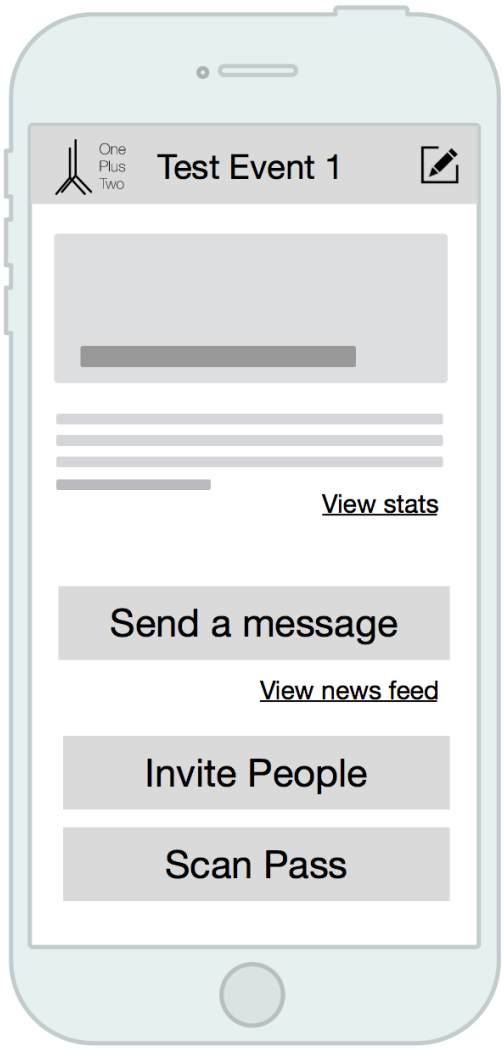
Home screen



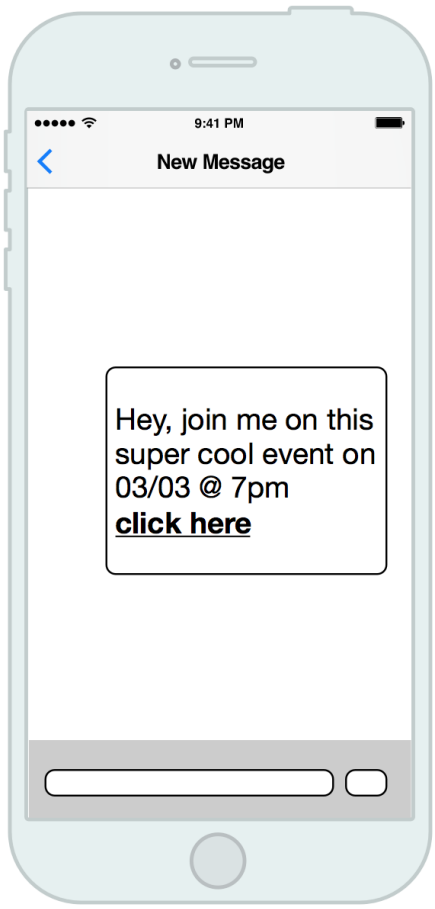
First step of creating an Event



Second step of creating an Event



Event Overview screen for the organizer

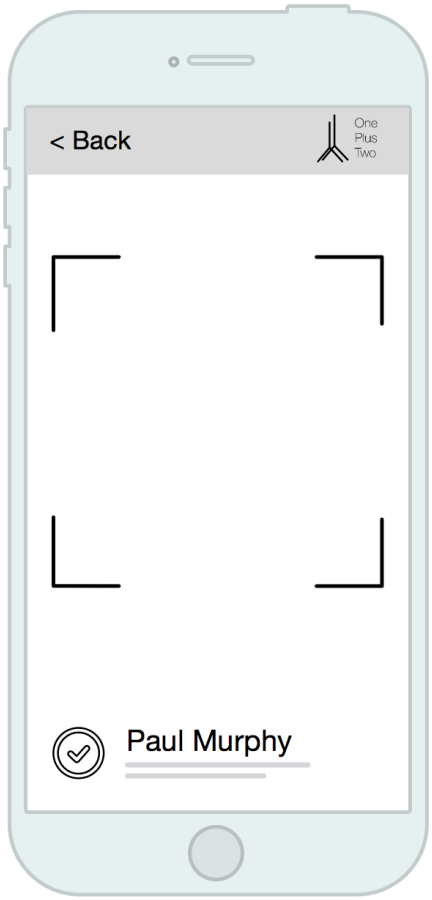


1. After clicking on Invite People, OnePlusTwo creates an Invite with link and auto-generates this message.
2. Link takes you to Event page with info; join with Facebook to attend.
3. Pass is generated and user adds it to Wallet
4. * If self-enrollment requires, user has to invite other friends to be eligible to attend.
5. * After friends join, Pass is activated.
6. Show Pass at entrance.

Sending an invite to an Event

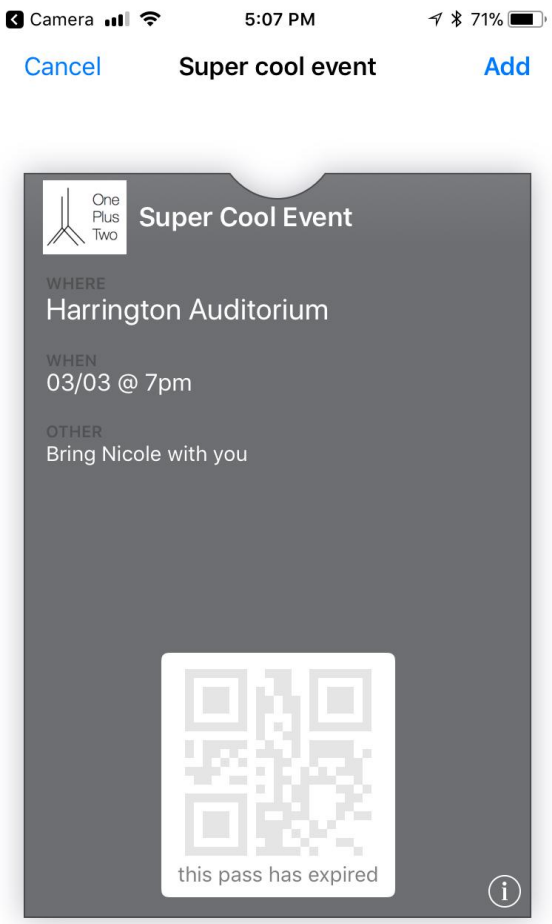
1. Short message is created and added to the news feed.
2. Attendees receive an email with the text.
3. Passes are "refreshed", thus small notification is shown on home screen.

Sending a message to all Attendees



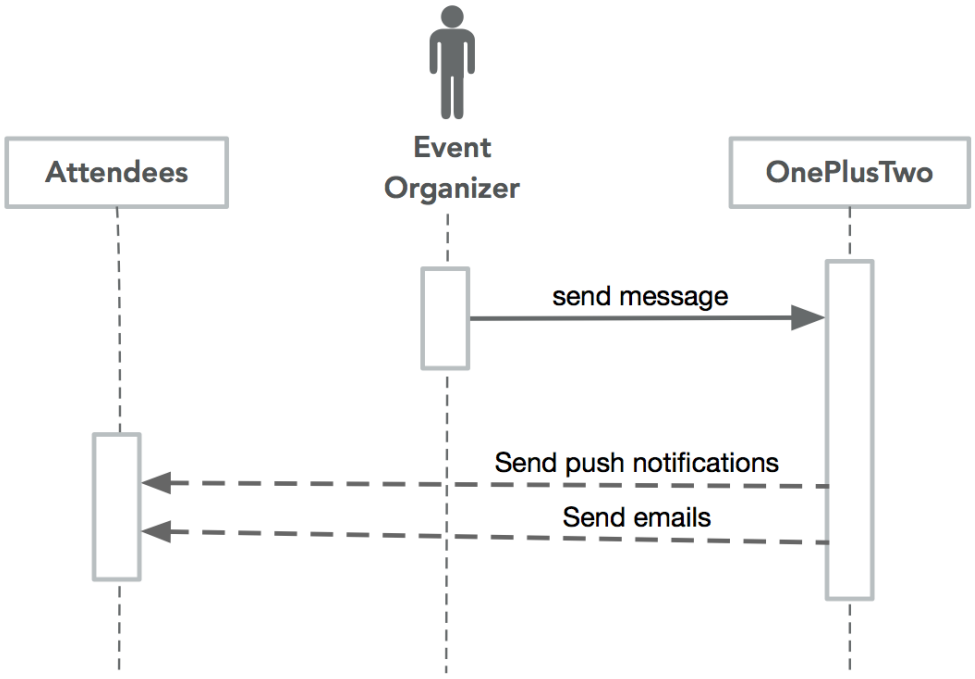
1. Attendee shows Pass at entrance.
2. Event Organizer scans the QR on the Pass.
3. OnePlusTwo validates against DB.
4. Recognize attendee; Invalidate Pass to prevent further uses; Push update to attendee's phone as well.

Scanning QR codes of Passes at entrance



Adding an Event's Pass to your iOS Wallet

6.2. *Sequence Diagrams*



Sending a message to all attendees

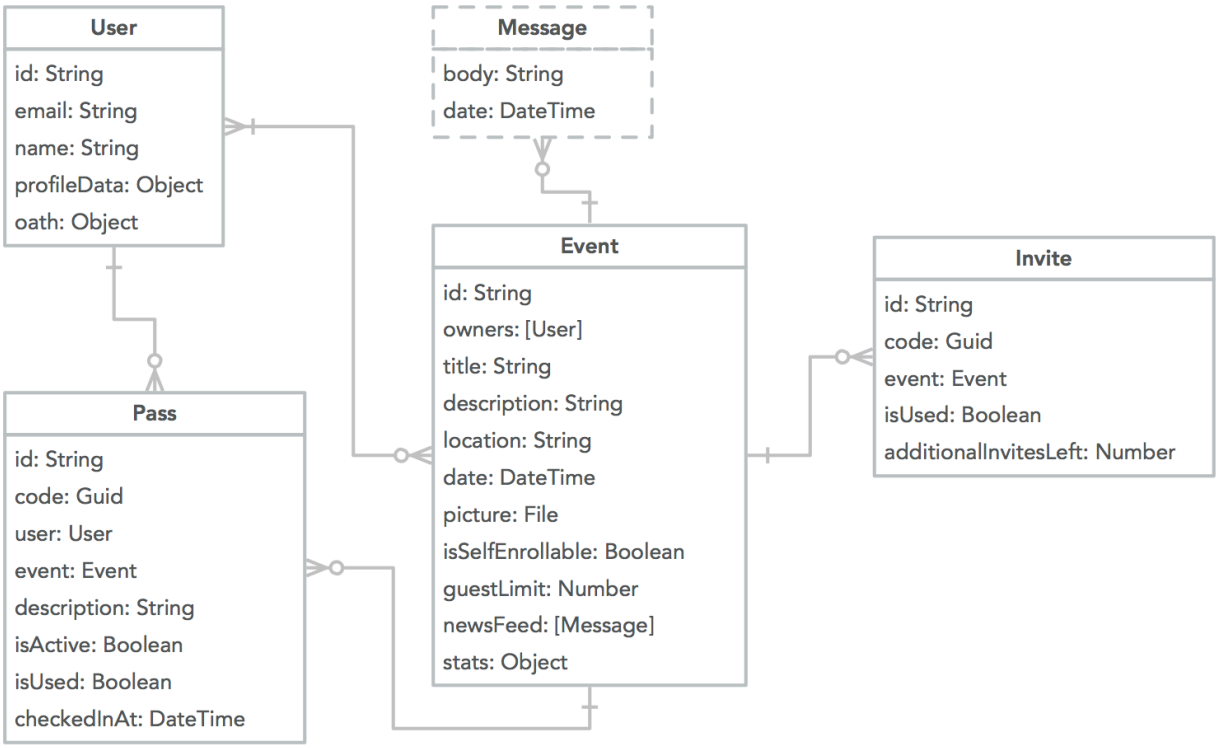
6.3. *Database Schema*

The main collection in our database holds Events. In addition to the basic fields (such as title, description, etc), each entry has an isSelfEnrollable field which tells if an event is private or public. Because Firebase does not support join operations, we denormalized our schema and decided to hold messages as embedded json objects within each event. Messages only contain body and the date when it was sent.

Both Event Organizers and Attendees are held as user objects in the User collection. Each user contains additional oath token information used to authenticate via Facebook or a Google Account. The profileData field holds gender and age information pulled from the authentication provider.

Whenever an event organizer wants to invite somebody an Invite object is created and stored in the Invite collection. Each invite has a unique code and is linked to its respective event. The additionalInvitesLeft field tells whether this invite grants the Invitee the opportunity to bring more friends (issue Invites). The isUsed field tells if the invite was already accepted by a user.

Finally, when an Invite is accepted a Pass object is created in the Pass collection. Each pass is linked to the respective event and invitee (user) owner; passes have a code (shown as a QR code during Check-in process). The isUsed field tells if the Attendee check-in at the event, thus rendering the pass no longer valid for entry to an event.

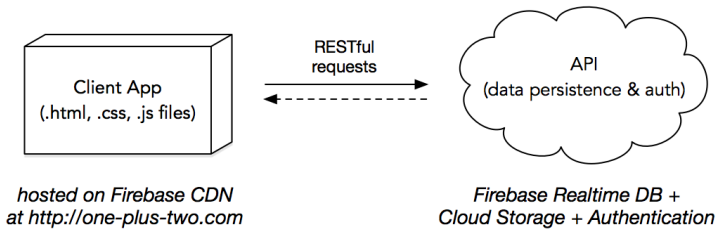


Entity Relationship Diagram

6.4. Architecture

We wanted our application to be easily accessible to the greatest number of users, therefore the application is built as a mobile-optimized web application instead of an iOS or Android native app. This means that our application is is run entirely inside the browser. Additionally as a single page web application, each page can be rendered with essentially no latency as we do not need to make an additional round trip to the server. Instead, routing is handled locally by caching all pages on the first server payload. This makes OnePlusTwo accessible from all mobile devices regardless of operating system, but also allows Event Organizers and Attendees to use OnePlusTwo without installing any additional apps.

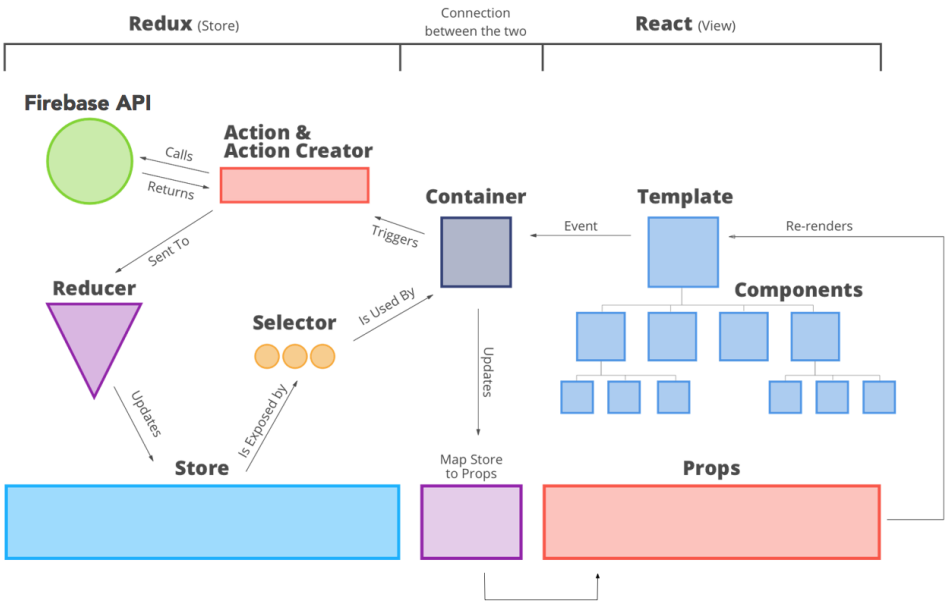
Our application is split into two -- a rich single-page-app (SPA) client and an API that handles all the data persistence as well as authentication. All static assets for the client are served by Firebase's content delivery network (CDN). Our API is implemented via Firebase which gives out-of-the-box create, read, update, and delete (CRUD) for its cloud-hosted NoSQL data storage.



Client-server architecture with Firebase

6.4.1. Client-app Architecture

The client is built using ReactJS (React), react-router and Redux. Comparing it to a typical model view controller (MVC) architecture, React is the View part of the application. Unlike MVC where the flow of processing is bidirectional, in a React + Redux architecture the flow of processing is strictly unidirectional which reduces dependencies and complexity.



Single page application (SPA) with React & Redux (Pini, 2016)

6.4.1.1. Components & Template

The component is part of the view and typically represents a button, a list or event an entire screen. Each components is rendered with a set of props (similarly to parameters) that get passed down from the parent component. The template is simply a synonym for top-level component for a certain page.

6.4.1.2. Container

The container is the glue that holds redux (the state) and react (the view) together. It filters only the necessary information to render the current page by applying *selectors* to the store and then passed that information as props. This is done via *map store to props* function.

6.4.1.3. Actions & Action Creators

Actions represent user triggered events or updates to the state of the application (e.g. load my passes, passes successfully loaded, user not found etc). In fact, action creators is the code called to create those actions, while the actions themselves are simply objects with event description and parameters.

6.4.1.4. Reducers

For every dispatched action all reducers are called with the action as a parameter, then its up to the individual reducer to "process" the action or not. Finally, reducers update the store in some way.

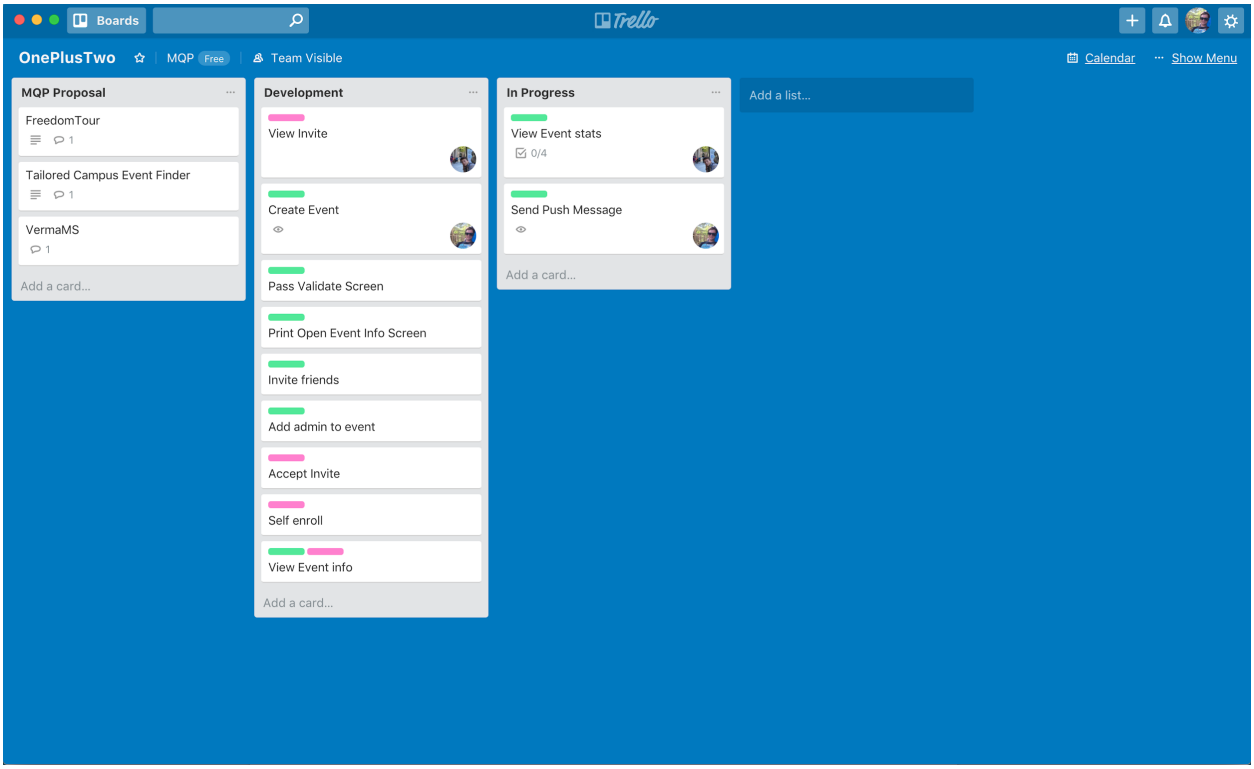
6.4.1.5. Firebase API

Some actions require communication with the Firebase API (e.g. load an event, authenticate a user, check-in a pass). When an action creator is called, it can "pull" some information from the Firebase API before passing along that result as a parameter inside the action object.

7. Implementation

7.1. Trello Board

Team communication and task organization was the cornerstone to keeping our project on track. Keeping track of our progress on the dozens of requirements outlined during the design process can be very challenging. To overcome this our team decided to borrow a tool from the Agile development methodology, Trello. Over the course of our second term we made over 150 commits in our GitHub repository and archived more than 50 cards on Trello. Between these Trello cards and Facebook Messenger we were able to keep our team constantly informed into what the other one was doing. As each of us accomplished tasks on Trello we could assign ourselves to the next task without overlapping the other's work.



A screenshot of our Trello during the implementation phase.

7.2. Development

Throughout our development term we had three distinct waterfall iterations that we detail below. The first iteration we worked to create a minimum viable product with a skeleton of each screen we would need and just enough server side logic to serve our project securely. The

second iteration we discovered several architectural issues that required substantial refactoring, and we completed most of the user stories. Finally in the last iteration, we cleaned up the user interface, did some refactoring and added unit and integration tests for our project.

7.2.1. Early Development Process

During this iteration, we laid most of the groundwork for the future two iterations. By the end of this iteration, we were able to create events, retrieve a list of events from Firebase, and open an event detail screen. The functionality was basic as there were no permission restrictions (so everybody could meddle with the database) and there was no visual styling. The most important milestones are listed below.

We used Facebook's *create-react-app (Create React App)* to initialize our project and set up the boilerplate. We set up the app's architecture as well as conencting React components, actions, reducers and Redux middleware to one another. We created a FirebaseService on the web client that linked it to Firebase. We set up the automated build process, test process and deployment via Travis CI. The process included:

- Generating compressed assets and single javascript package
- Running a simple test to ensure app could start
- Deploying cloud functions to Firebase
- Deploying the entire client with its assets to Firebase CDN
- Deploying database permission rules

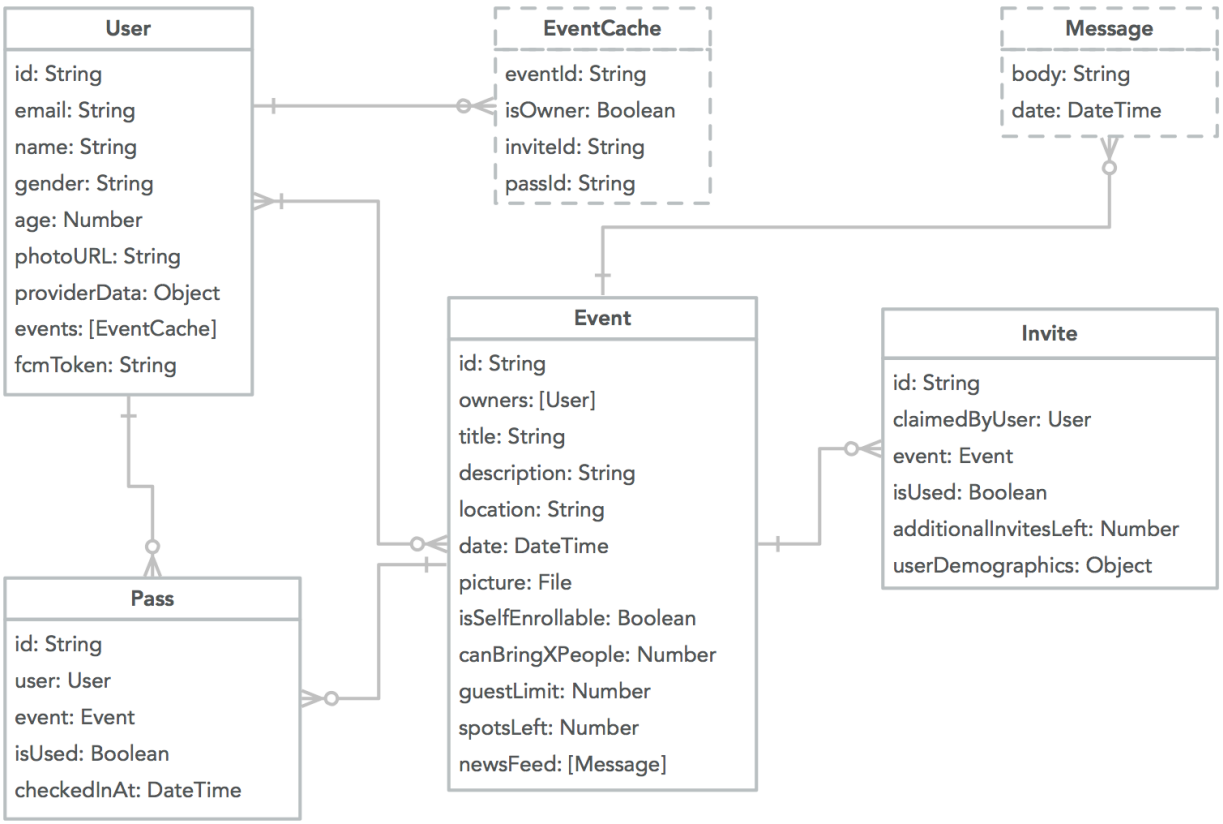
We added functionality for fully-fledged user authentication & session management via Facebook or Google account. We also added support for creating events, listing a user's events, and adding event details. Finally, we added simplified invites creation and accepting functionality.

7.2.2. Extending the Application

During the second iteration, we implemented permissions for Firebase and completed the invitation system. Using our initial database schema, invitations contained no way to identify which user claimed it. Additionally because of Firebase's limited querying tools we were not able to find our users' passes within the pass table. After discovering these limitations we had to make several changes to our database schema as well as change much of how our backend logic worked. These changes centered around the fact that Firebase does not support using permissions as filters. For example, in Firebase you can not give a user read permission for a specific event and then allow any user to query the Event table and receive only permitted

events. Instead, the entire read will fail once you reach an event for which you do not have the appropriate permissions. To overcome this shortcoming, we added an EventCache to the User table. This includes a list of permissions for each event the user is involved in. Now all event editing is controlled by server side functions that validate the user requesting the event has the necessary permissions.

Fortunately because of our decoupled client side architecture components, all of the client side changes were confined to a single file that interacts with Firebase. Everything else such as our actions, reducer, and store remained the same. The only change was to the endpoint we query within the FirebaseService module.



Updated Database schema -- major change was the introduction of EventCache due to Firebase limiting the query engine with the introduced permissions

7.2.2.1. Cloud Functions

We ended up rewriting most of our permission-sensitive code as Firebase Cloud Functions, Some of them are triggered by database updates, others were triggered by HTTP POST requests. The following is a list of the application's Firebase Cloud Functions:

- **AcceptInvite** -- Accepting an invite involves changing the Invite itself, creating a Pass and updating user demographics. It is both insecure and inefficient to let the client application handle all of this. We opted for no-write permissions on the client and extracted all the functionality as an HTTP endpoint.
- **CheckInPass** -- similarly to accepting an invite this is a complex operation, so we opted for implementing it as an HTTP endpoint cloud function.
- **GenerateNewInvite** -- similar to the previous two Firebase Cloud Functions.
- **GetEventStats** -- computation intensive function which also needs access to sensitive user data. We preferred to keep no-read-no-write to the data and have an HTTP cloud function return the aggregate results.
- **GetInviteInfo** -- to properly render an invitation we need the event details. However, events should be kept locked because otherwise private addresses could be visible. Firebase does not support fine-grained permissions so we opted for moving all this into an HTTP cloud function.
- **ImportGuests** -- this database trigger is run whenever a new event is created.
- **SendEmail** -- a database trigger which sends out emails to all people who have accepted an invite and are not subscribed for push notifications.
- **SendMessage** -- a database trigger which sends out push notifications to all people who have accepted an invite.
- **UpdateEventOwners** -- because Firebase cannot return “a list of all events I’m invited to” due to permissions limiting the query engine, we had to create an EventCache array which held all events a certain user can see with the respective invites and passes. This was implemented as a database trigger.

7.2.2.2. Other packages we integrated during this stage

- **ChartJS** -- for the Event Statistics screens we needed a library that renders pie charts as well as line graphs.

- **Normal distribution generator** -- we implemented our own function to analyze the times at which attendees checked-in at an event. With the calculated mean and standard deviation the function generated 100 data points for the line graph.
- **Material-ui** -- we felt the optimal way to ensure consistency in the UI is to use a library basic UI components. Material UI, designed by Google, creates all the buttons, menus, popups and loading bars.

7.2.3. Final Application & Testing Process

The testing phase of our application is made of three pieces. Automated unit tests for the client side javascript, integration tests for the server side code, and manual tests based on our regression test spreadsheet. The client side unit tests were painless to write due to the purely functional nature of the Redux reducers.

For the server side tests we mocked the database with an in-memory JavaScript object that contained test data. Each cloud function was then mocked based on this test data. The most heavily tested part of our application were the Firebase Cloud Functions.

Finally for the manual regression test, we developed a spreadsheet with all of our use cases, and validated each feature worked with a timestamp so we could revert to a previously known working state at any point. Towards the end of the project we used this spreadsheet to evaluate our application and logged our most recent fully functional version on February 13th 2018.

	Feb 7th 2018	Feb 13th 2018
Auth		
Sign in from home screen	✅	✅
Sign out from home screen	✅	✅
Events		
Select create event from home screen	✅	✅
You can import from previous events you've hosted when you create an event	✅	✅
Enter mock data for each field and submit event	✅	✅
Redirected to new event detail page?	✅	✅

Data loads for new event page?	✅	✅
Send Message and Invite guests buttons are present for new event?	✅	✅
Invite guest link can be generated	✅	✅
Sent messages appear in news feed	✅	✅
Guests receive push notification and or email for each new message	❌	✅
Events list shows all events you are involved in	✅	✅
You can view event stats for events you host	✅	✅
Invites		
Invite links you receive open to the event detail page	✅	✅
Clicking accept prompts you to either sign in or receive a pass	❌	✅
You can display your pass	✅	✅

8. Evaluation

During the development phase of our project we implemented all four of the functional requirements (event creation, sharing, management, and guest interaction) defined in our project specification. Each use case we implemented is detailed below. Additionally we evaluated each of the three non-functional requirements (ease of use, performance, scalability) defined in our project specification.

During the first iteration of our application we realized that having both the concepts for Open and Private events made OnePlusTwo confusing to our users. This is why we decided to not implement the UI controls needed for an Open Event (even though the backend supports Open Events). The following represent what we were able to implement in the time frame of the project. The components that were not implemented also serve as suggestions for future work.

- [ST-1] As a Website Visitor, I want to register and create a Private Event, so that I can closely control the guestlist.
    - ✓ Implemented.
- [ST-2] As an Event Organizer, I want to create a new Private/Open Event and import people who showed up from a previous event I organized, so that I start exciting my audience from early on.
    - ✓ Implemented.
    - ✖ No UI for Open Event.
- [ST-3] As an Event Organizer, I want to create an Open Event where people need to bring a friend to be eligible to attend, so that more people attend and are excited about my Event because bringing a friend means they are more committed.
    - ✖ No UI for Open Event nor functionality to “unlock” invites; just regular ones.
- [ST-4] As an Event Organizer, I want to set a maximum number of attendees for my Open Event, so that there is space and seating for everyone.
    - ✓ Implemented. Working for Private events as well.
- [ST-5] As an Event Organizer, I want to invite many people to my Private Event using Text & Facebook, so that I can easily expand the guestlist.
    - ✓ Implemented.

- [ST-6] As an Event Organizer, I want to allow some people to invite additional attendees to my Private Event, so that new people can come.
    - ✓ Implemented.
    - ✖ No UI to fine tune (e.g. person X can bring 2 friends, person Y only 1)
- [ST-7] As an Event Organizer, I want to get a printable file with information about my Open Event and instructions on how to self-enroll (e.g. QR with link), so that it is effortless to use OnePlusTwo.
    - ✖ Not implemented.
- [ST-8] As an Event Organizer, I want to give "admin" rights to other people, so that it is easier to manage a big event.
    - ✓ Backend supports it.
    - ✖ No UI.
- [ST-9] As an Event Organizer, I want to add additional information to my Event (e.g. location, time, description…), so that people know what they are signing-up for and where they should be.
    - ✓ Implemented.
- [ST-10] As an Event Organizer, I want to validate attendee's passes at the entrance, so that I can keep track of actual attendance and let only eligible people in.
    - ✓ Implemented.
- [ST-11] As an Event Organizer, I want to know how many people did attend my event as well as basic demographics, so that I know what my target audience is.
    - ✓ Implemented.
- [ST-12] As an Event Organizer, I want to send a short message to attendees before the event (via email and push notifications), so that I can remind and excite them to come.
    - ✓ Implemented.
- [ST-13] As an Event Organizer, I want to send a short message to the people who came (actual attendees), so that I can stay on their radar and make them anticipate my next event.
    - ✓ Implemented.
- [ST-14] As an Event Organizer, I want to get a list with all invited attendees and people who showed up, so that I can have a record to use for other non-related purposes.
    - ✓ Implemented. Can import attendees when creating a new event.

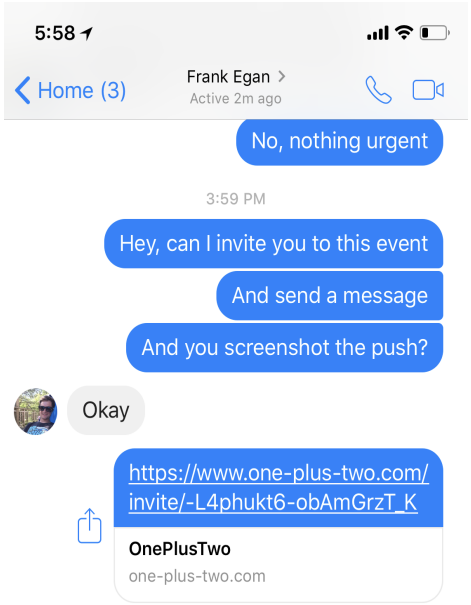
    - ✖ Cannot export attendees to .csv file.
- [ST-15] As a Person Receiving an Invite, I want to quickly accept the invite and add it to my iOS/Android Wallet, so that I am reminded about the event and have the pass readily available at the entrance.
    - ✓ Implemented.
    - ✖ No iOS Wallet support.
- [ST-16] As a Person who sees an interesting Event Ad on-campus (or hears about it) and wonders whether to go, I want to subscribe for interesting updates (and even quickly sign up for the event), so that I do not lose the opportunity to go.
    - ✖ Not implemented.
- [ST-17] As an Attendee to an Open Event, I want to invite a friend of mine, so that I am eligible to attend the event.
    - ✓ Some attendees can invite friends.
    - ✖ No “unlocking” of an invite.
- [ST-18] As an Attendee, I want to occasionally get news and updates about the event, so that I do not forget about it and do not rule it out as boring and useless.
    - ✓ Implemented.
- [ST-19] As an Attendee, I want to see which of my friends are also attending the event, so that I am not anxious to go alone.
    - ✖ Not implemented.
- [ST-20] As an Attendee, I want to refer back to the event details (location, date…), so that I don’t forget those.
    - ✓ Implemented.
- [ST-21] As an Attendee, I want to see live stats about the event (number of people actually inside), so that I have more motivation to join.
    - ✖ Visible only for Event Owners.

We were able to implement the core features that allow guests and hosts to easily create, share, and join in events without installing any additional applications. Even though not every use case in our project specification was implemented, the two remaining use cases were supplemental goals which although desirable are not critical to the application.

We fulfilled our non-functional requirements by ensuring that the load time for each page was kept below three seconds. Additionally because we developed our application using Firebase, our application is able to scale to thousands of concurrent users.

9. Future Work

What we would like to see further developed is the ability to update the invitation page's metadata (title, description, og:image etc), so that whenever an invitation is shared it is rendered nicely instead of a as a simple link (see screenshot of current rendering below). Additionally, hosts should be able to designate more people as owners of an event. This was supported on the backend. However, we had no time to implement the accompanying user interface. This would allow more than one person to scan QRs at the entrance. Also, our backend supports allowing invitees to bring more friends, however, we had no time to implement the user interface to manage and fine tune those parameters, e.g. if X can bring 2 or 3 people. Users should also be able to invite friends from their Facebook friends list or the contacts from the Google account. This would make the invitation process more effortless for users. Additionally, we would have liked to have more comprehensive end-to-end tests that simulate user clicking on buttons and navigating the application to ensure all components are properly linked together. Users should be able to add their passes to iOS Wallet. To implement this we would have had to register under Apple's developer program which would have been outside the scope of this project. Finally, we believe generating tracking sheets of all attendees after an event would be useful for some of the on-campus organizations as mentioned in the research.



Poorly rendered invitation due to lack of page metadata.

10. Conclusion

Even though not every use case in our project specification was implemented, the two remaining use cases were reach goals which although desirable are not critical to the application. However, we did meet all functional, and non-functional requirements in the development phase. Our application follows a high standard for data security, permissions and efficiency. It can easily scale to thousands of concurrent users due to its architecture. The most important modules were extensively tested at our Continuous Integration after every commit and then automatically deployed to Firebase.

Our team employed an iterative waterfall SDLC and gained experience with customer development, requirements gathering, software design, application development, and software validation with regression tests.

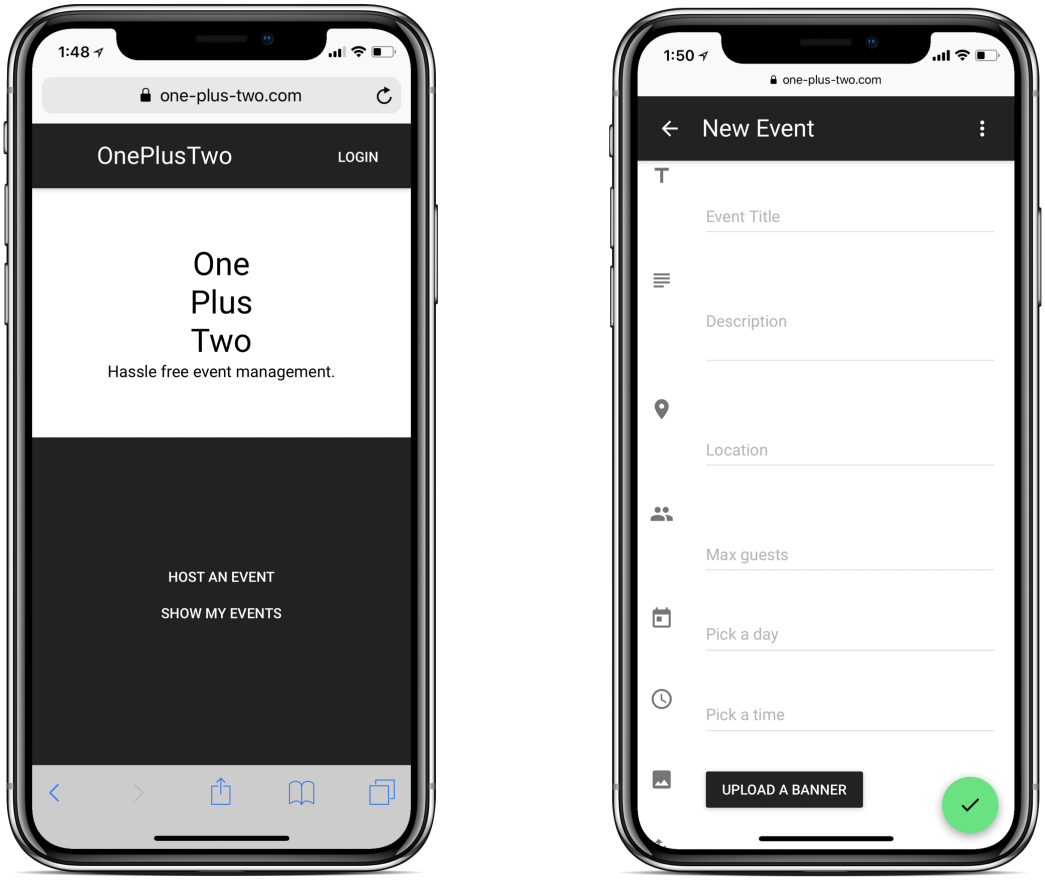
Ultimately we produced an application built on modern web technologies that meets all the requirements of a progressive web application. As a result of developing with Firebase, OnePlusTwo is capable of scaling to thousands of concurrent users.

11. Appendix

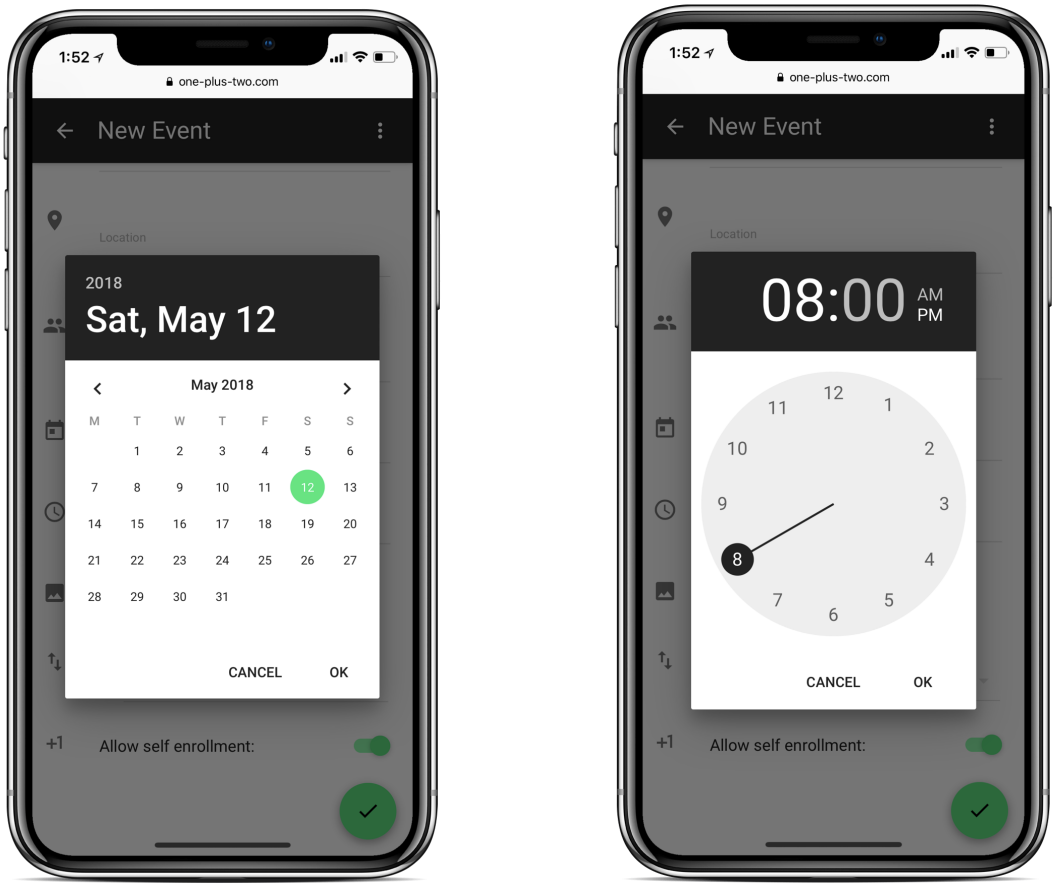
11.1. User Manual

11.1.1. Creating a new Event

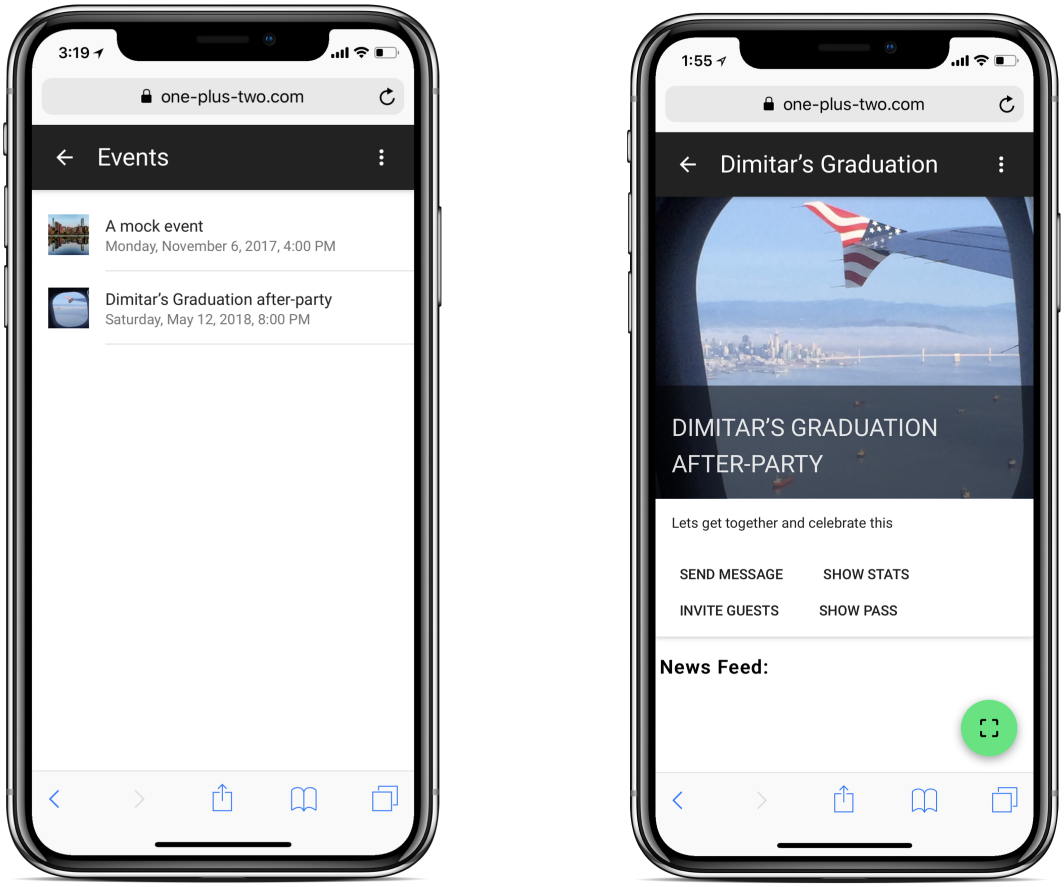
1. Open https://one-plus-two.com .
2. Click “Login” and use either your Facebook or Google account.
3. Click “Host an Event”.
4. Name your event, pick date & location, upload a picture and click the green button.



Home Screen without being logged in (on the left). New Event Screen (on the right).



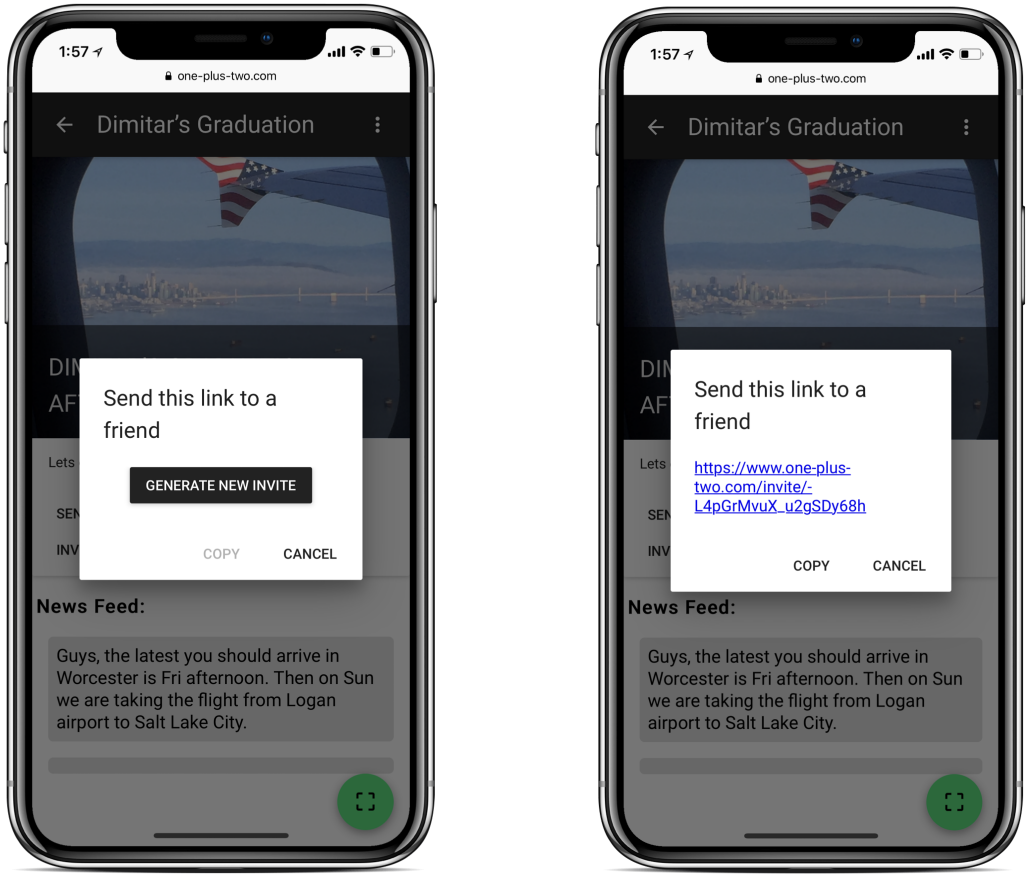
Date Picker & Time Picker



List with all event I've created or was invited to (on the left).
Newly created event as viewed by an event owner (on the right).

11.1.2. Inviting Guests

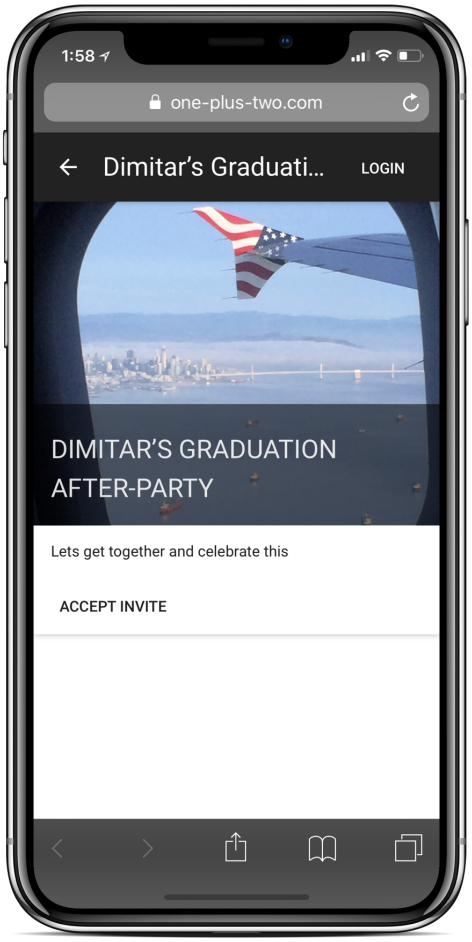
1. Open an event you are an owner of and click "Invite Guests". This will also work on some events where the owner has allowed you to bring friends.
2. Click "Generate New Invite" in the popup.
3. Copy the link and send it to your friend via text or your preferred communication.
4. ** Your friend has to accept the invite (process described below). Depending on the event settings he/she might be allowed to invite additional friends.



Invitation popup opened (on the left) and generated link (on the right).

11.1.3. Accepting an Invite

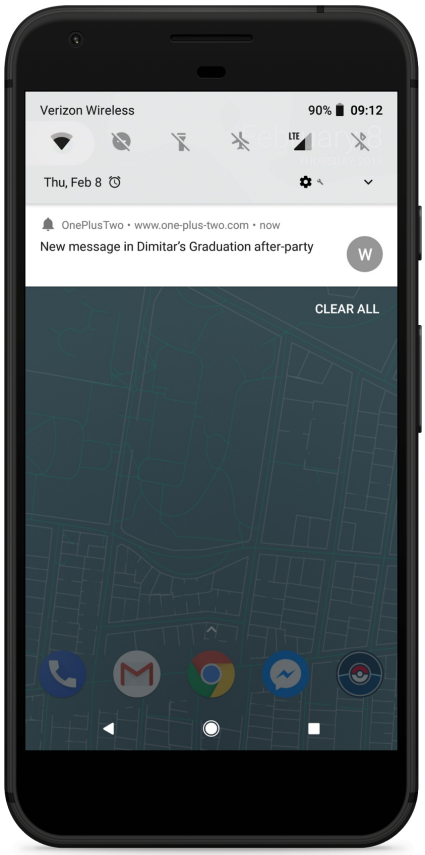
1. Open the link your friend sent you. You will see info on what all of this is about.
2. Click "Accept Invite".
3. If you have not used OnePlusTwo before you will be asked to create an account via your Facebook or Google account. After creating the account, click "Accept Invite" again.
4. You will be redirected to the event details page where you will also see any messages on the News Feed (as previously shown above).



Invite information screen

11.1.4. Sending a Message to all Attendees (if you are an owner)

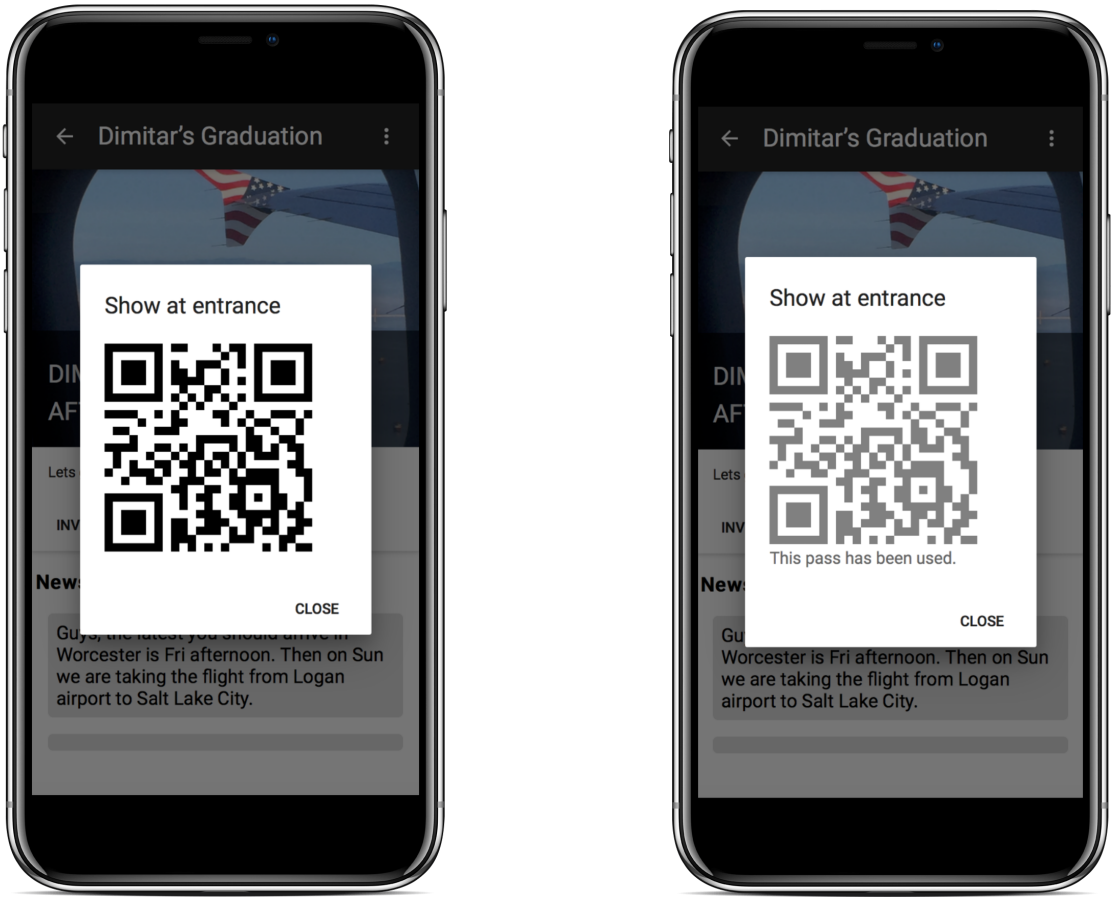
1. Open an event you are an owner of and click "Send Message".
2. Type it in and click "Send".
3. ** All Android users will get a push notification. iOS and other users will get an email.



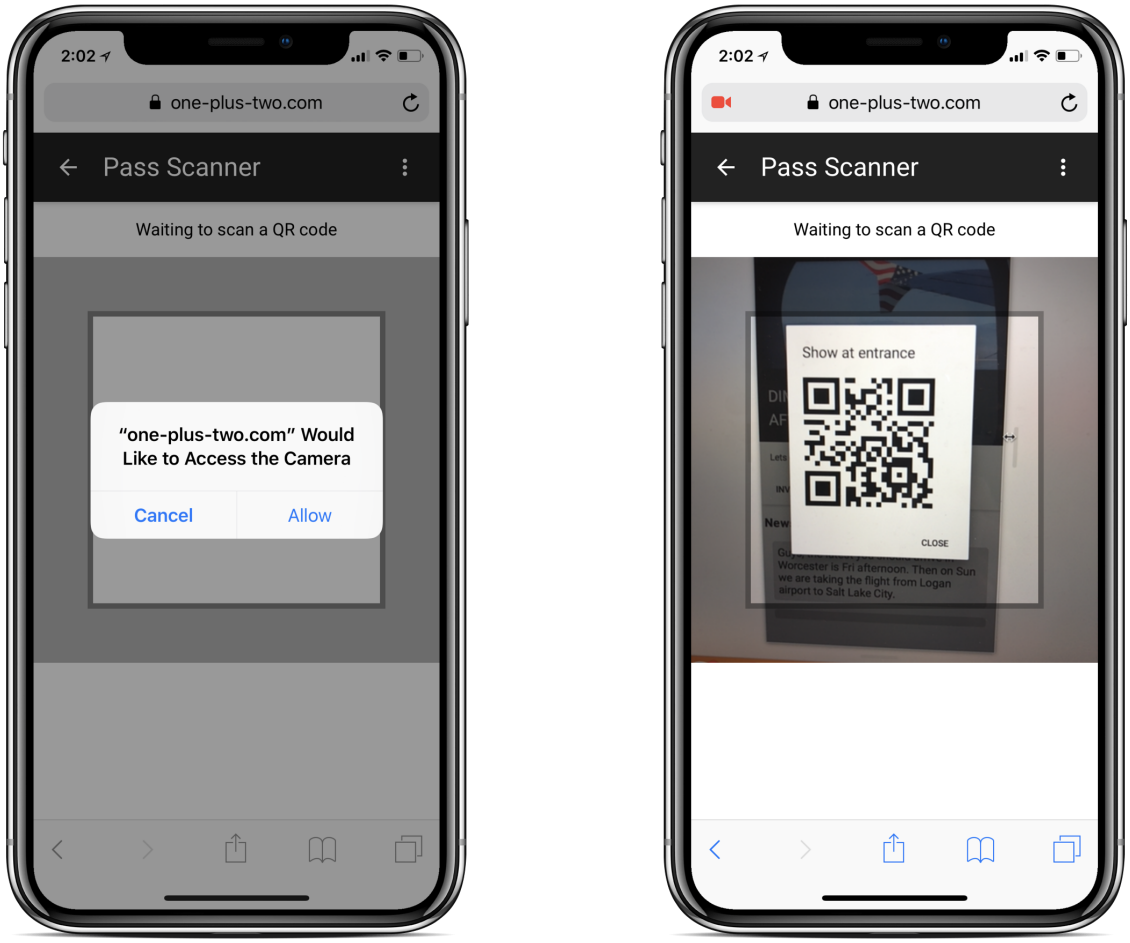
An invitee received a notification.

11.1.5. Checking-in people with invites

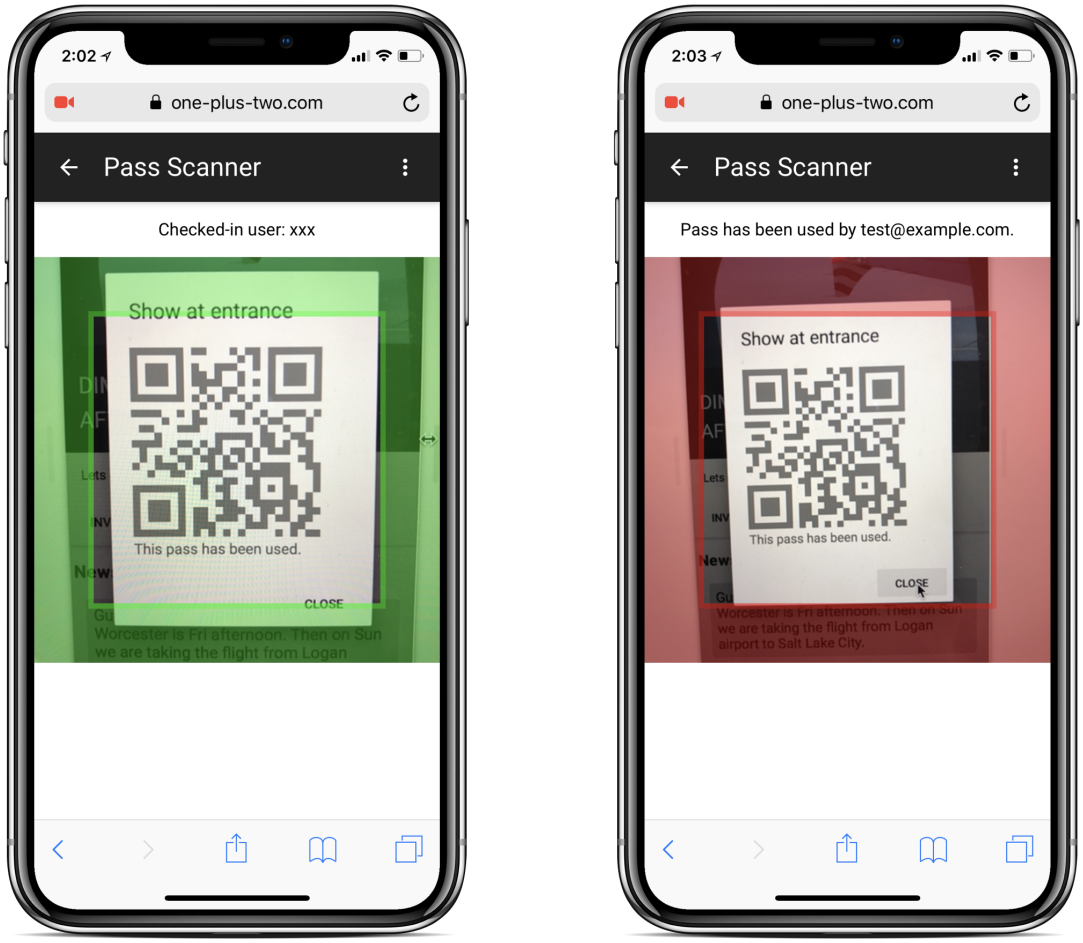
1. Invitee should open the event details and click “Show Pass”.
2. Event Owner should click on scan button (green button at the bottom).
3. Event Owner gives access to camera.
4. Event Owner points camera at Invitee’s QR code.
    a. If the pass is valid the Owner’s scanner will flash in green.
    b. If the pass is invalid or already used the scanner with flash in red.
    c. The Invitee screen should say if the pass was already used up.



Invitee QR code screen -- valid on the left; already used up on the right.



Event Owner's scanner screen.



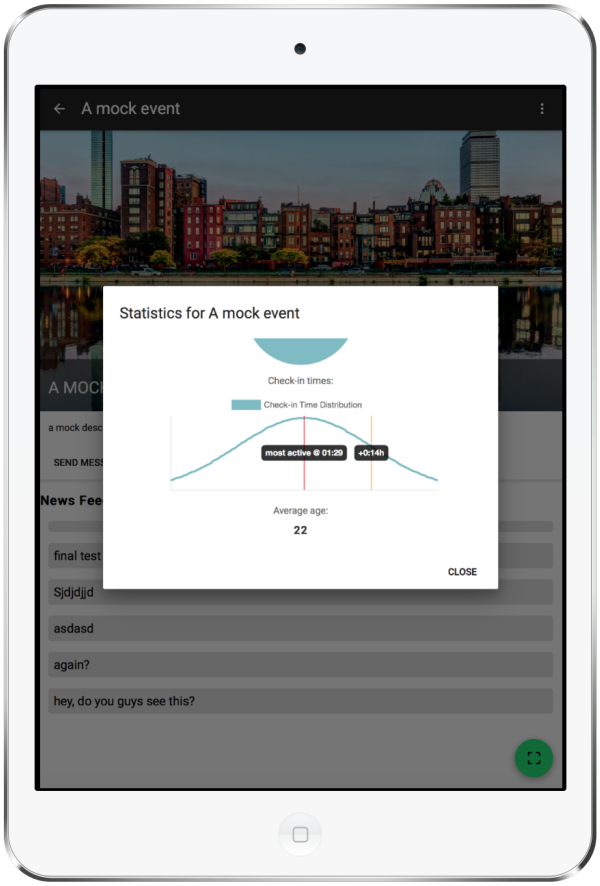
Valid pass (on the left) and invalid pass (on the right).

11.1.6. Viewing Event Statistics (if you are an owner)

1. Open the event details and click “Show Stats”.
2. The data you see represent:
    a. Number of people invited vs. number of accepted invites vs number of actual attendees.
    b. Split by gender of all attendees.
    c. Average age for all attendees.
    d. Time distribution showing peak time of check-ins.

Graph showing attendees vs. accepted invites vs. invites sent.

Graph showing attendees by gender.



Graph showing peak time of check-ins. Average age is shown under the graph.

11.2. Interview Response #1 (summary as it was a conversation)

Tell us about the last event you've organized?

- Recently Planned ΒΘΠ's Homecoming Events
- Recently organized for a guest speaker to present for the Society of Environmental Engineers

What is the biggest challenge whenever you've organized an event (before, during, after)?

- Sending out email reminders the day of to get people to show up
- making sure the rooms were booked

How do you solve that challenge?

- Emailing WPI aliases
- Scheduling on Outlook

What are the alternative solutions (e.g. for interlocutor only: alternative to getting a shake for morning commute is getting a donut)?

- Messaging Slack channels with reminders

Why don't you "hire" those alternatives?

- Too much setup to make a slack for a small group that does not meet more than a few times a month
- Not very organized

How do you organize the guest list for you events?

- Voluntary sign up on Google Form
- Just send out a link, anyone can add their name
- *MOST* people show up but some flake

How can you distinguish between your loyal attendees (fans) and flakers?

- It was open only to brothers

How do you spread word about an event and make sure people will come?

- Word of mouth in small classes
- Emails day of to remind people

How can you distinguish between your frequent comers (fans) and flakers?

- Currently they have no method for differentiating

Notes

- Smaller events are interested in pre-engagement to make sure they get an turnout
- They have not considered re-engagement (are there any clubs doing this really?)
- They like google sheets because link sharing is so easy it can be done on mobile or desktop very quickly
- Brothers need to attend a certain number of events and one chair is in charge of keeping track of how many events each brother attends

11.3. Interview Response #2

Tell us about the last event you've organized?

- LCA
- WPI Sailing Team

What is the biggest challenge whenever you've organized an event (before, during, after)?

- Coordination of all 100+ brothers for various steps (tabling, day of, clean up)

How do you solve that challenge?

- Google sheets sign up links

What are the alternative solutions

- Slack channels for coordination

Why don't you "hire" those alternatives?

- We do for more day of pick up and small jobs but they get lost in the channels message stream too easily

How do you organize the guest list for you events?

- Google sheets/Slack

How can you distinguish between your loyal attendees (fans) and flakers?

- We do not keep record of attendance for anyone

11.4. Interview Response #3

Tell us about the last event you've organized?

- SiStory -- successful entrepreneurs from Bulgaria share their stories & troubles onstage
- Erasmus Exchange party in a nightclub

What is the biggest challenge whenever you've organized an event (before, during, after)?

- "PR"-ing the event; making sure enough people do hear about the event, but most importantly hype them up! If they do not know who the speaker is SiS has to educate people to make them come
- Following up with the attendees and making them become part of our group in Facebook, so that our next events SiStory are for bigger crowds
- Sometimes Facebook attending say we would have 1000 people, but only 700 end up coming
- How do you make sure the proper target of people comes -- in the case with Erasmus Exchange students do we end up with 100 exchanges and 300 randoms, or 300 exchanges and 100 of their friends

How do you solve that challenge?

- Properly target when PR-ing the event

How do you organize the guest list for you events?

- Those events are with registration, but the reg is open to the public; We use automated online solutions (eventim, ticketmaster, eventbrite)

How can you distinguish between your loyal attendees (fans) and flakers?

- The ones that are second comers are loyal. Also who were brought by a friend.

12. References

Affairs, Assistant Secretary for Public. (2013). Personas. Retrieved from /how-to-and-tools/methods/personas.html

Cohn, M.Iterative waterfall model. Retrieved from https://www.mountaingoatsoftware.com/blog/an-iterative-waterfall-isnt-agile

Facebook messenger. Retrieved from https://www.messenger.com/features

Firebase. Retrieved from https://firebase.google.com/

Google, I. (2016). How mobile latency impacts publisher revenue. ().

Jest JavaScript testing. Retrieved from https://facebook.github.io/jest/index.html

Lawrence Mello. (2015). Google hangouts

Lyttle, Z., Ross, J., Malofsky, B., McCarthy, M., & Bennet, T. Tailored campus event finder.

Pini, C. (2016). React + redux: Architecture overview. Retrieved from https://articles.coltpini.com/react-redux-architecture-overview-7b3e52004b6e

Progressive web app checklist. Retrieved from https://developers.google.com/web/progressive-

web-apps/checklist

React - A JavaScript library for building user interfaces. Retrieved from https://reactjs.org/index.html

Steve Blank.What is customer development? Retrieved from http://www.startuplessonslearned.com/2008/11/what-is-customer-development.html

Website continuous integration with travis CI, jekyll, gulp, and GitHub. (2016). Retrieved from http://scholar.aci.info/view/14f3b8e116900100004/1531c842ff100014c0b

Where work happens. Retrieved from https://slack.com/