Beacon - Preliminary Test Plan

Neema Boutorabi	***18694142***	***Luigi Sacco***	***21139143***
Andre Tertzakian	***17682148***	***Emre Pekel***	***48488134***
Alexander Ford	***18361148***	***Omar Dzick***	***36579143***
Konrad Izykowski	***26506148***		

Introduction

Beacon is an Android app that allows users to find events near their location in real-time, share events they are hosting with other Beacon users, and comment and post photos on the page of an event they are attending. The structure of our app presents several challenges in the testing portion of our project. For instance, all event information is user generated, so we need to create a system that can populate our database with spoofed events. User acceptance will be an important consideration, as our app will only be as good as the number of active users. This is why extensive validation with real users will be a crucial component of our testing plan.

Verification Strategy

Since Beacon is a crowdsourced application, we must make sure the users can interact with the framework easily via the tools provided to them in the GUI and its integration with the database. It is important that we get opinions from users who download the app, since testing it only by ourselves is not enough to ensure proper, fluid functionality. So, we will present the app to our friends, explaining what the app is, and tell them to download it to give it a try. As a form of feedback, we can link the users to a Google Form with questions regarding their thoughts about Beacon. The questions that would be asked are similar to what follows:

- What do you like about Beacon? *{short answer}*
- What do you dislike about Beacon? *{short answer}*
- Would you use Beacon? Why? *[absolutely not, maybe not, maybe, yes], {short answer}*
- If not, what could change so that you like the app? *{short answer}*
- Would you recommend Beacon to your friends? *[no, maybe, sure, yes], {short answer}*
- How comfortable are you with using the application? *{short answer}*
- What would you like to be changed? *{short answer} {short answer} {short answer}*
- What would you want to remain unchanged? *{short answer}*
- How would you rate your overall experience with the app? *[1=bad - 5=good]*

Non-Functional Testing Strategy

Test #	Requirement Purpose	Action / Input	Expected Result	Actual Result	P/F	Notes
1	Safety/Censorship (filtering out inappropriate content)	Expletive in event title	Event is not able to be created. User get's warning.			
2.1.1	Security (private shareable link generation)	The user creates a private event	A secure, shareable link is generated for the event. The link is <u>not</u> viewable to the public			
2.1.2	Security (public shareable link generation)	The user create a public event	A secure, shareable link is generated for the event. The link is viewable to the public			
2.2	Security (hiding event/contact details from public)	The user hides their phone number from public	Other users cannot see the user's phone number			
3.1	Usability (backend distance calculation and result finding given query results from database)	The user searches for an event	The result radius is adjusted as needed (up to a max) so that results are displayed. If no results are found within the max radius, nothing is returned.			
4.1	Performance (offline usability; more specifically, storing event data locally, and pushing changes to the server once the	The user writes to the database while offline	The change is automatically merged into the database once a connection is reestablished			

	user gets back network connectivity)					
4.2	Performance/ Scalability (responsiveness of the UI under heavy load)	Many mock events clustered around a certain area are created, with many mock users being online at the same time	The app operates smoothly and responsively for users in that crowded area			
5.1	Software Quality	Use JDeodorant to identify code smells in our code base	N/A, this is to ensure that our code is both scalable and intuitive.			JDeodorant will help us ensure that our code has as few bad smells as possible
5.2	Software Quality	Use EclEmma to optimize the code coverage of our tests	N/A, this is to ensure that the outcomes of our tests truly represent the correctness of our code.			EclEmma will make it easier for us to make our code coverage as close to 100% as possible

In order to test the app’s ability to load events, we will create a static class whose sole purpose will be to create fake data for testing purposes. This class will have methods for creating each of the fields associated with an event. In this fashion, we will be able to populate our database with a large amount of testable data.

Although we will generate random events, we will not create random users, as users have to be authenticated via a Google or a Facebook account. We will test the accessibility of our database via the Firebase tool for testing access to different levels of the database. In this fashion, we will be able to adequately test the read/write permissions of users, as all users are required to be authenticated.

Since each event requires an individual user to be the creator, and since we are not creating random users (as stated above), the random event generator will associate the created events with randomly selected administrative users in the database. Since administrators do not have any extra privileges when it comes to event creation, this will not affect our testing. Thus, we will be able to test for tracking events via users, or for tracking users via events.

This method for random data generation will allow us to test our database, as well as the parts of the application which depend on the database being filled with users. We will configure the random event generation to be chronological, so that they are created across random intervals. By creating the events across random periods of time, we will try and imitate real-life situations where users will be dispersed across many different time zones, and a variety of events which occur at different times.

Finally, we will ensure that the random-generation classes will create the latitude and longitude components of the event so that they are dispersed randomly within a 100km radius of Vancouver. This will allow us to test the radius size option for filtering events. Later on, we will expand this to test the entire globe, so that we can test for users everywhere.

Functional Testing Strategy

In order to test the functions we create, we will use the JUnit testing framework. We will test all the functions before we incorporate them into the overall design. As we connect functions together, we will use smoke tests in order to test/cover the most important functionalities of our system. These smoke tests will be implemented approximately every sprint, or multiple times during a sprint if we are connecting several modules during that time. Moreover, as the system gets more complex, we will use integration testing (an example of this would be creating an event, and another user seeing it/adding it to their favourites - this would be testing several modules at once to make sure they are interacting appropriately).

An example of a functional test we may implement is to have a clock that speeds up time to see if events are deleted within the appropriate time frame (and that all users get updated by this). Once the app is fully implemented, if the app crashes, a crash report will be sent to and stored in our Firebase database (Google Firebase offers a crash log feature). This will help our team locate and fix bugs that cause the app to crash (since crashes are high priority bugs). We are using Github’s issue tracking feature to help delegate tasks associated with bug fixes and app improvements. Github allows developers to add tags to various issues such as (“bug”, “enhancement”, “question”) and issues can be prioritized by associating them with different milestones.

Adequacy Criterion

We want to ensure that the portions of our code that are unique to the app are tested extensively and that the boilerplate code/library code is tested less extensively. For instance, testing real-time database insertions and deletions takes precedence over testing code that involves the Google Maps API. We believe this is a more pragmatic testing strategy, since making sure that every single line of code is run in at least one test is somewhat unrealistic. Therefore, we have come up with the following adequacy criterion for our app:

- All use cases have an associated test, (this will involve finding a way to mimic user input since the UI can't be simulated in a test environment). We believe this is a reasonable benchmark since the use cases outline all the ways in which the user can interact with the app. Ensuring that these interactions behave as expected internally will help mitigate bugs.

- Every utility method has an associated test. Since utility methods deal exclusively with data manipulation, the consequences of improper code may not be immediately visible (for instance, creating an incorrect location stamp for an event may affect which users can see the event). Therefore it is imperative that these functions are tested for various inputs.

- The UI should look consistent across all Android devices. Since Android is run on devices with various screen sizes (including tablets and phones), we should verify that our UI design looks consistent on different devices. This falls into the category of QA testing rather than unit/integration testing.

Test Cases and Results

Test # codes correspond to the particular code given to that step in our use cases defined in our "Requirements, Vision, and Scope" document.

Test #	**Requirement Purpose**	**Action / Input**	**Expected Result**	**Actual Result**	**P/F**	**Notes**
1.2	Populate list view	Events within 30km radius	List of events ordered by their distance from the user			Must handle case where no events are nearby
1.2.1	Search within list view	User query	List of events that match the user's query and are nearby			Must handle case where user query returns no results

1.2.2	Search for Beacons in map view	User query	A map view populated with Beacons that match the user’s query			Must handle case where user query returns no results
1.4	Given a user selects an event, event page information will be displayed	Event	Event page; including title, description, images, time, and location details.			
2.3	Given a user has selected an event, they can write a comment and it will appear on the event page	Event page and user comment	Event page and database have been updated with a viewable comment			Consider testing text comments and comments with photo attachments independently.
1.4.2	Given a user has favourited an event, it will appear on their favourites page	User favourites event	Event is added to user’s favourites list			
3.4.3	Given a private event is created, only those with a direct link are able to view the event page	Private event created	Specifics not viewable on map or list view of nearby users. Other users who use the link can see the page.			

1.2a.1	User has no nearby events/there are no events in their search area	User queries the database	User is prompted to create an event			Consider testing case where user is in an area with no events
1.2.1a.1	User queries the database and no results are available	User queries the database	User is told that there were no results, and asked to enter a new search			
3.4.4	Given a public event has been created it should be synched to the database and be viewable by all nearby users	Public event created	Updated map and list view for all nearby users showing the new event page			
3.4.2a	A user should not be able to create an event outside of a 40km radius	User input location beyond 40km	Event is not created, user is prompted to select a new location			
4.3.1	Host of event wants to change their page	Host requests to change page	Changes host made to event are visible to all other users			