Hometown Heroes

CS 361 - 400, Group 6, Homework 3

July 20, 2017

Jon Austin
developer
austinjo@oregonstate.edu

Valerie Chapple
developer
chapplev@oregonstate.edu

Kenny Lew
developer
lewke@oregonstate.edu

Gregory Niebanck
developer
niebancg@oregonstate.edu

Charlotte Murphy
developer
murpchar@oregonstate.edu

Benjamin Rodarte
client
rodartec@oregonstate.edu

ABSTRACT

In this paper, we describe the architecture for the Hometown Heroes mobile app.

CCS Concepts

• **Software and its engineering → Data flow architectures**
• **Software and its engineering → Feature interaction.**

INTRODUCTION

Our group selected two high-level architectures to examine: Repository and Pipe & Filter.

HIGH-LEVEL ARCHITECTURES

We assessed repository and pipe and filter architectures against our key quality attributes of Usability, Reliability, Integrity and Efficiency. Below are our findings.

Dataflow Diagram #1: Repository

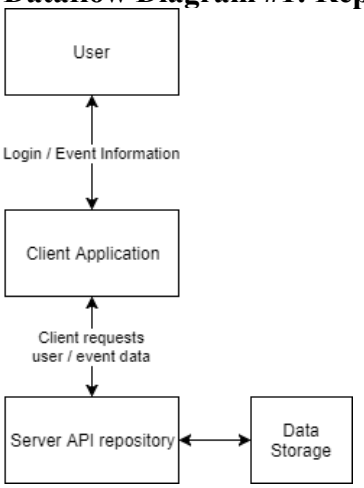

Figure 1: Repository architecture diagram

Quality Attributes: Repository

Usability

Client application communicates with Server API for all user requests. Event data can be stored and retrieved later from any user's device that has the client application. User and event information are stored in data storage for user access or social publishing. Does not address social media (third party) communications.

Reliability

Data storage should be setup with redundancy. Event data can be updated by from any client application and is not dependent on any one user or device. Server uptime needs to be addressed. Does not address social media (third party) communications.

Integrity

All user and event data is stored and tracked in a central location, the database. Client application does not have to store and maintain the data. Does not address security of authorizing logging in through social media. All communications between client application, server, and data storage should be secure.

Efficiency

Data requests between the client application and server may take additional time over the network versus storing it locally. Client application receives data on an as-needed basis by making the request to the server.

Dataflow Diagram #2: Pipe and Filter

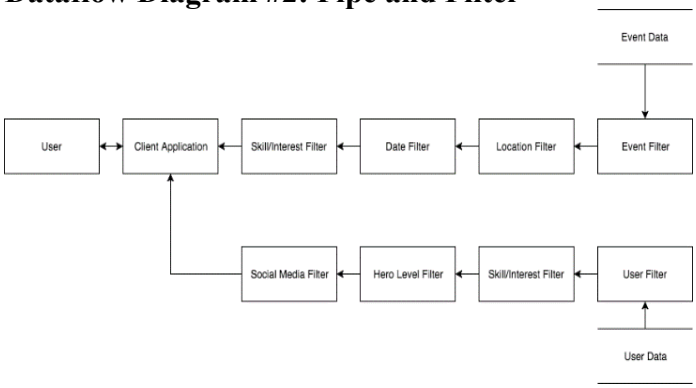

Figure 2: Pipe and Filter architecture diagram

Quality Attributes: Pipe and Filter

Usability

Does provide functionality to filter events based on their preferences. Does not address creating events or user registration. Does not address participating in events. Does not address social media (third party) communications.

Reliability

Data storage should be setup with redundancy. All access to data relies on the robustness of the filtering service. Does not address social media (third party) communications.

Integrity

Users can get exactly the data they request. If any one pipe or filter fails, the whole application fails. All user and event data is stored and tracked in its own storage. Client application does not have to store and maintain the data. Does not address social media (third party) communications.

Efficiency

Requires an intermediate service to filter relevant data from the server to the client. Users cannot get new events as a "push" notification.

FAILURE MODES

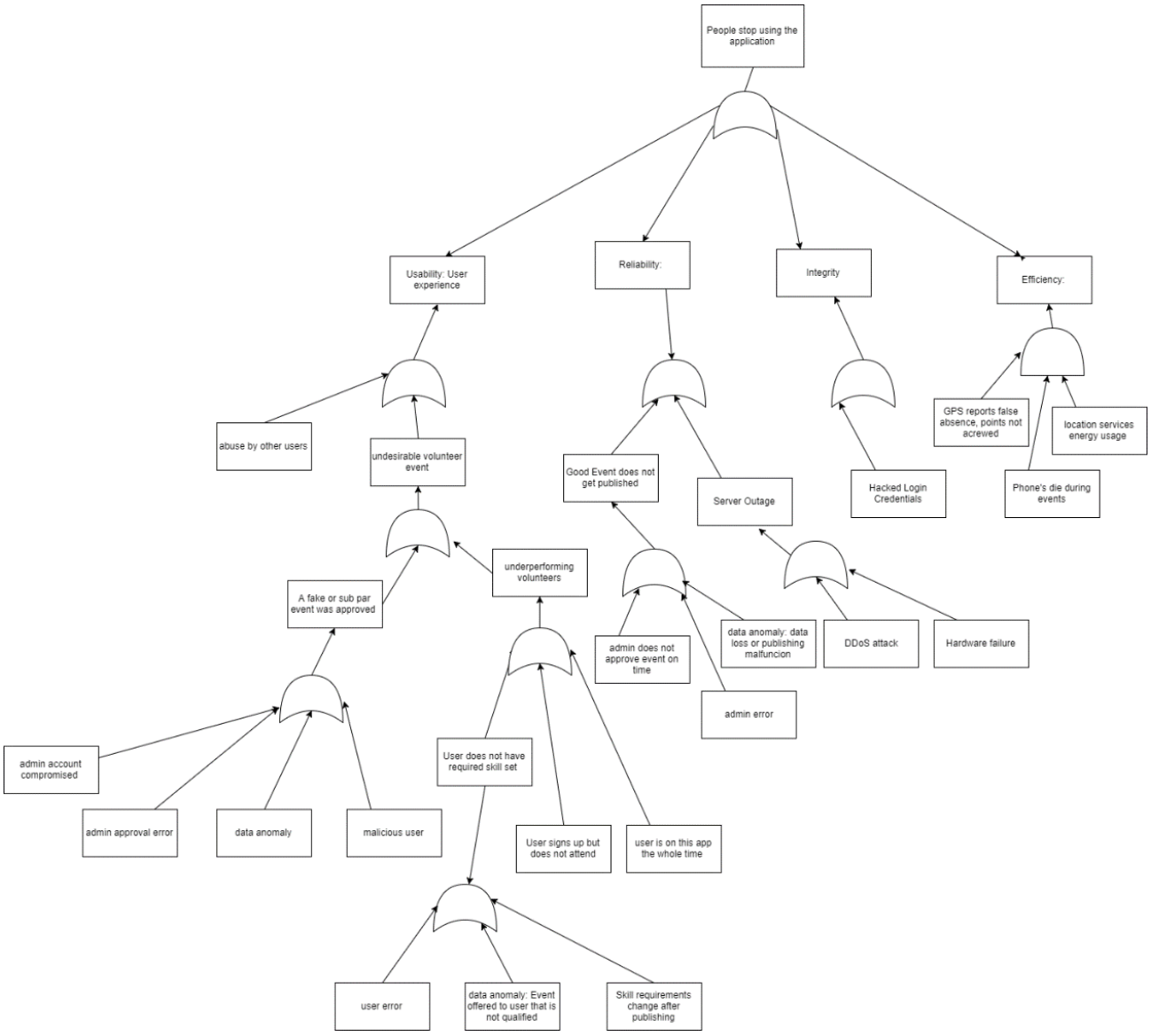

Figure 3: Failure concepts related to quality attributes

Identifying Vulnerability

Our failure discussion started out as a list of failures that seemed likely and would also create a significant disruption in terms of effecting the most important quality aspects of the application.

1. Volunteer minutes not correctly saved to user.
2. Event data changed after approval process.
3. Events are offered to user-volunteer that contradict the user's filters.
4. User cannot create account through client application.
5. User-organizer creates an event that requires zero volunteers.
6. Login data and/or social media login data compromised.
7. Event approved but never published to users.
8. Event not approved, but still gets published to users.

This list indicates that failures affecting the data accuracy or the ability to for volunteers to participate would be most disruptive.

The failure tree in figure 3 represents an exploration into possible failures related to each quality attribute. The first tier are the most important quality attributes for our application. The usability subtree shows indicates vulnerability in terms of data, user participation, and administrator error. The reliability subtree indicates failure related to data accuracy and administrator responsibility. The integrity subtree mentions data security. Our system does not contain valuable information specifically, but social media integration represents a significant collateral threat. The efficiency subtree focuses on client-side energy use tied to location services. Efficiency could also be represented in terms of the number of client-server interactions for the most important use cases.

Many of the sub-failures can be put into three categories: User malice or negligence, data failure, and administrator failure. User negligence or malice in many cases are unavoidable in terms of architecture, but negligence related to creating events is managed directly by administrators.

Failure Mode 1: Missing or False Data

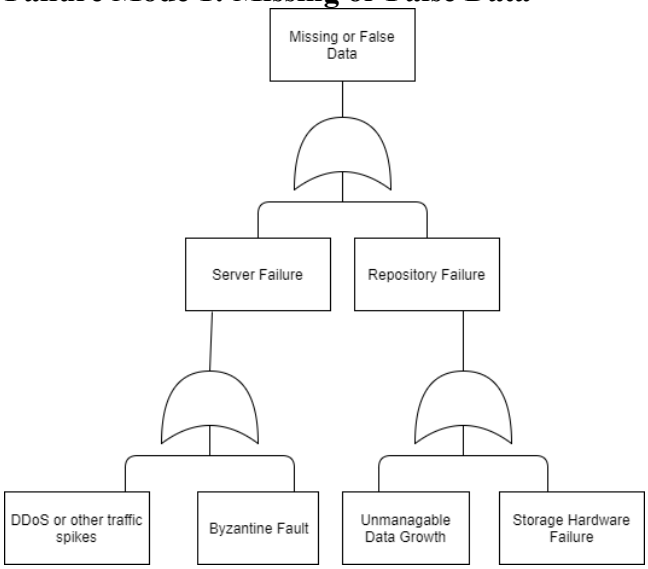

Figure 4: Data Failure Fault Tree

Every single use case for the application require multiple server interactions that retrieve data from the repository. A failure or data anomaly at any step renders the application completely dysfunctional. On the server interaction side, systems need to be in place to impose rate limits for managing traffic or detecting a DDoS attack early. At the very least, have a plan to handle a DDoS attack should it occur. On the repository side, systems need to be in place to backup data. A well-managed database and research to predict usage details are important to preventing failure in this system. The system need to be able to handle the load in terms of traffic and data storage.

Failure Mode 2: Administrator Failure

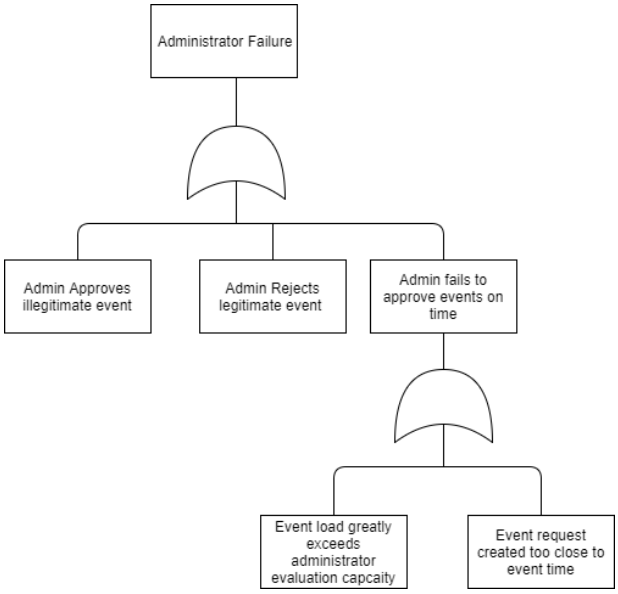

Figure 5: Administrator Failure Fault Tree

It doesn't matter how many volunteers are willing and available if there are no events to volunteer for. The participation process requires an event proposal to be approved by an administrator. For this system, event administrators will be staff and/or credible users determined by their event history or hero score. This reveals several vulnerabilities regarding error in the approval process and the capacity to evaluate events in a certain period of time. Systems need to in place to standardize the approval process, and verify the experienced users are adhering to that process. One solution is a system that dynamically adjusts the capacity to create an event based on the current administrator workload. Ie. The client application calendar interface would grey out days for event creation based on a calculated prediction of when an event could possibly be approved.

QUALITY AND FAILURE ASSESSMENT RESULTS

We have selected to repository architecture for its efficiency and uncomplicated design. Failure assessment indicated that faults likely to occur in repository architecture are highly documented with many available solutions. Pipe and filter architecture carried a lower efficiency standard in regards to client side data storage. Therefore, the amount of data transferred at each filter represents a higher likelihood of data loss at each stop.

DECOMPOSITION

The repository architecture will be further structured by using feature-oriented decomposition for our client application. The core services are:

1. Login Service to implement a user login process
2. Create Event Service to manage events from creation to approval to publication
3. Browsing Service for users to search and filter events
4. Participation Service to allow track of event signups and participation
5. Sharing Service to tag users in Hometown Hero events, as well as post user updates.

We chose to decompose the Create Event Service and the Participation Service, as these are two the most complex interfaces to implement.

Dataflow Diagram for Event Creation

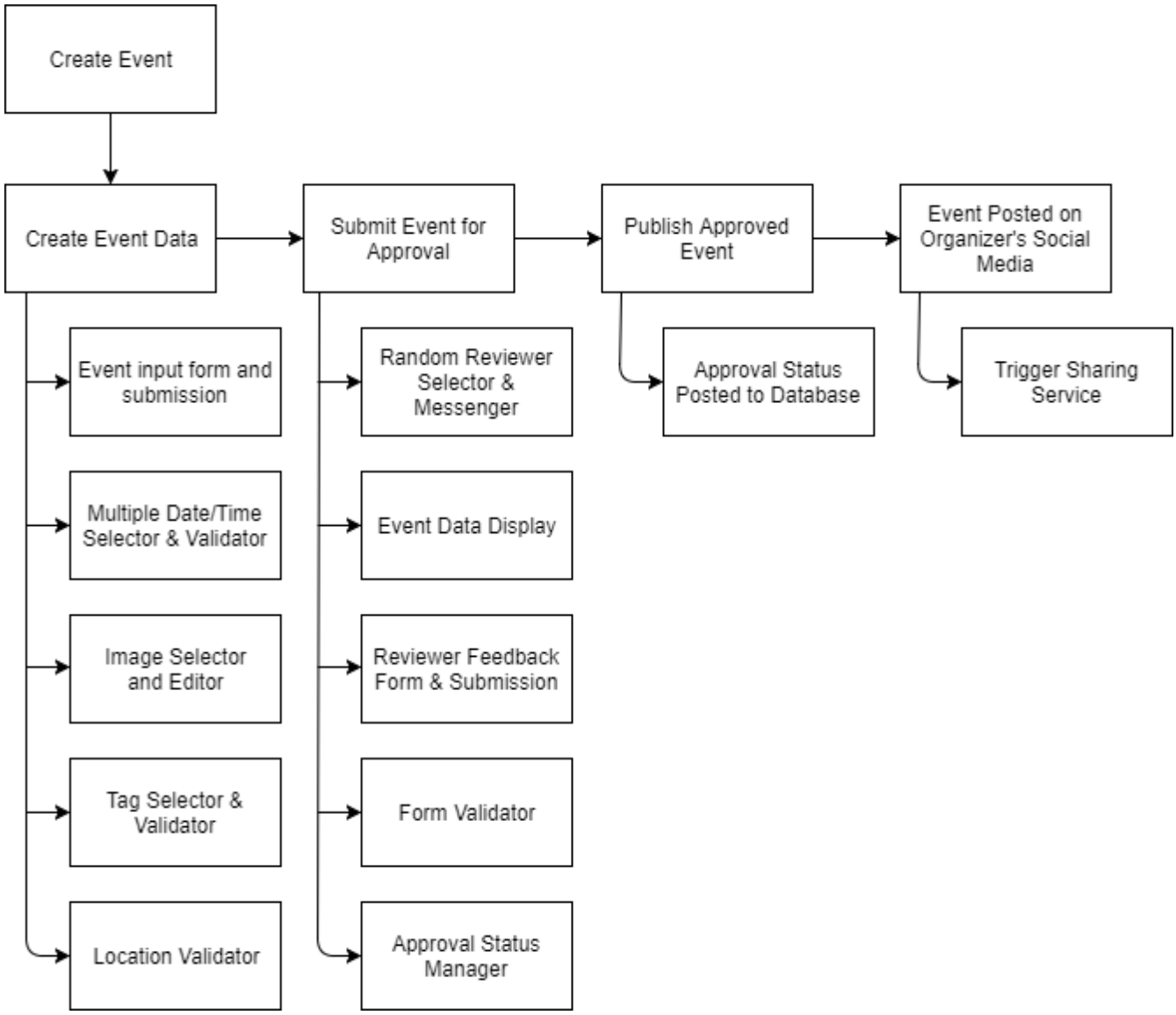

Figure 6: Event Creation use case features

Dataflow Diagram for Event Participation

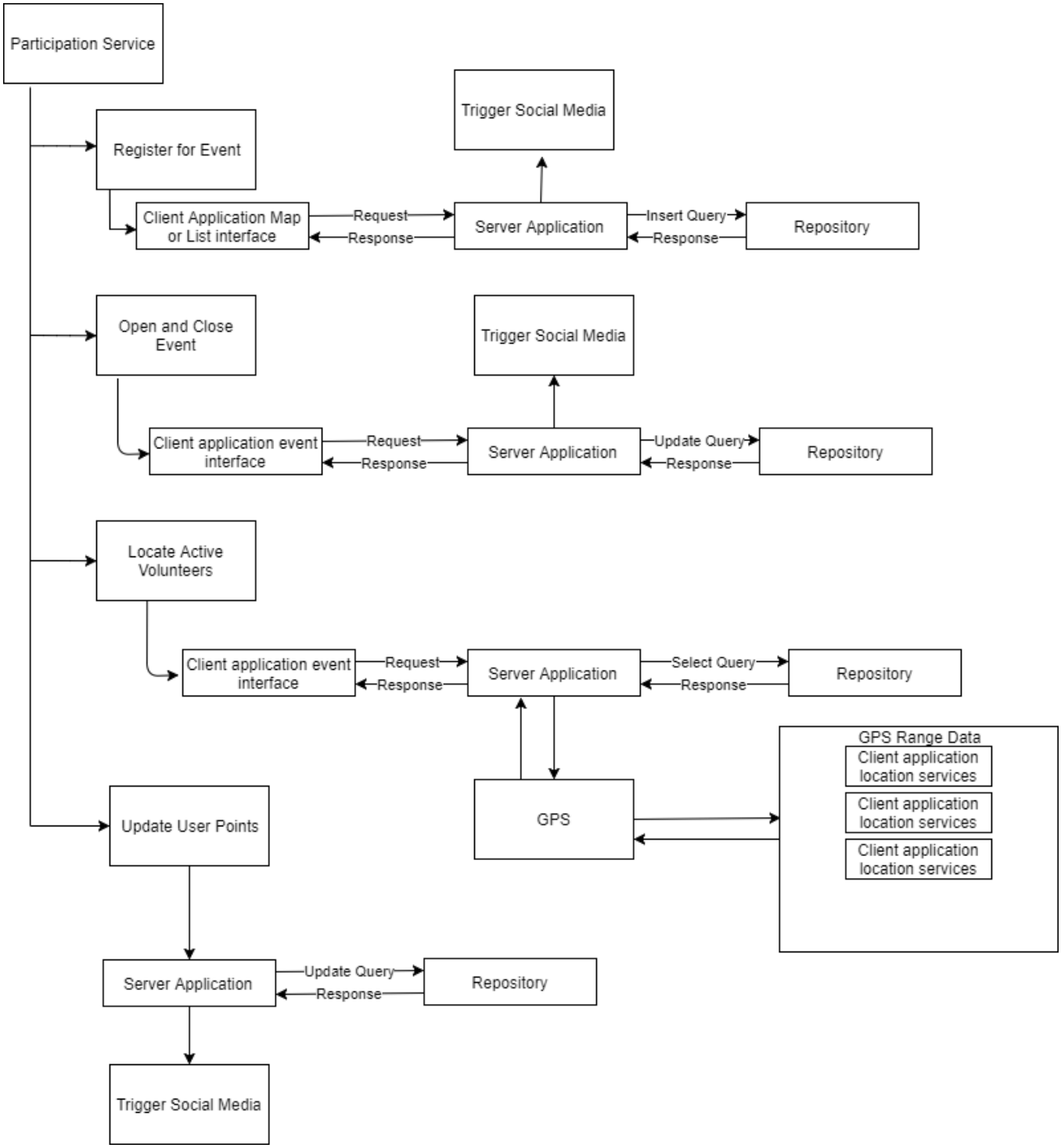

Figure 7: Event participation use case features

ARCHITECTURE VALIDATION

The following walk-throughs describe how the repository architecture supports each of the four main use cases: Create an Account, Create an Event, Register for an Event, and Participate in an Event.

Use Case: Create an Account

A description of the steps necessary to create a Hometown Heroes account in the context of a repository architecture:

1. User makes a request to the server to create a new account with the client application.
2. The client application shows the account creation page to the user.
3. User authorizes the client application to use social media for login information or enters their own account information.
4. User account information is stored in a database.
5. Client application shows the skill questionnaire to the user.
6. User fills out questionnaire or enters skills manually into client application.
7. Profile and skill information is sent to the database for later retrieval.

Use Case: Create an Event

A description of the steps necessary to create a Hometown Heroes event in the context of a repository architecture:

1. User (Organizer) enters login information into client application.
2. Client application sends login information to the server.
3. Server verifies the login information with data storage. If the login information is valid, the server sends notification to the client application that the login information was valid and the system proceeds to step 4. If the login information is not valid, the server sends notification to the client application that the logic information was not valid and the system returns to step 1.
4. Client application shows User (Organizer) default dashboard page, User requests to create an event from the calendar page.
5. Client application shows the User (Organizer) the create event page.
6. User (Organizer) inputs event information into client application.
7. Client application sends the event information to the server.
8. Server sends the event information to User (Reviewer) client application for approval.

Event is not approved:

9. User (Reviewer) does not approve event.
10. User (Reviewer) client application sends notification to the server that the event was not approved.
11. Server sends notification to User (Organizer) client application that the event was not approved.

Event is approved:

9. User (Reviewer) approves the event.
10. User (Reviewer) client application sends notification to the server that the event was approved.
11. Server stores event information to data storage.
12. Server sends notification to User (Organizer) client application that the event was approved.

Use Case: Register for an Event

A description of the steps necessary to register for a Hometown Heroes event in the context of a repository architecture:

1. User enters login information into client application.
2. Client application sends login information to the server.
3. Server verifies the login information with data storage.
4. If the login information is valid, the server sends notification to the client application that the login information was valid and the system proceeds to step 5. If the login information is not valid, the server sends notification to the client application that the logic information was not valid and the system returns to step 1.
5. Client application shows User default dashboard page.
6. User filters events in client application and selects and event.
7. User registers for an event in client application.
8. Client application sends user event registration information to the server.
9. Server stores the user event registration information to data storage.
10. Server sends event registration confirmation to the client application.

Use Case: Participate in an Event

A description of the steps necessary to participate in a Hometown Heroes event in the context of a repository architecture:

1. Client application reads smartphone GPS location information.
2. Client application checks GPS location information against event location information.
3. If GPS location is within event location, client application sends check-in information to server.
4. Server logs event arrival time and sends check-in confirmation notification to client application.
5. If GPS location falls outside of event location, client application sends check-out information to the server.
6. Server logs event departure time and sends check-out confirmation to client application.
7. Server stores user participation information to the database.
8. Server sends user participation summary information to the client application.

ARCHITECTURE IMPLICATIONS

Use Case: Create an Account

The account creation process is the most initially robust use case in our architecture. Since this is a fairly common process (even

with using social media accounts instead of creating an account from scratch), there are a lot of examples of other systems that use it. This gives us a good jumping off point and a lot of confidence that a reasonable system architecture for this type of system would easily support the account creation process. We did overlook one step in our use case, though. Since integrating social media is a major focus of Hometown Heroes, we need to make sure our use cases explicitly allow for that. Our current use case does not have a step covering the possibility that a user would sign up with their social media credentials. This could cause a failure to identify faults associated with using that process instead of creating an account from scratch.

Use Case: Create an Event

The robustness of the event creation process is vital to the overall success of the Hometown Heroes system, and therefore must be treated with extra care and attention. Our analysis of the event creation use case showed a couple missing links. Since we chose to use a repository architecture, we need to have the server that the application talks to verify any information with the database that will be storing application information. This includes login information. Any time a user wants to use the application, they will have to log in, so there needs to be a check between the system server and the database that stores the information.

We also need to consider that the system will show the user the "create event" page when going through our use case for creating an event. This may seem trivial to us, but leaving it out of the use case diagram may lead to some faults that are not considered.

Finally, our verification pointed out several steps in the event approval process that were not accounted for in the use case diagrams. It could be, however, that event approval is an entirely different use case, and event creation only relies on the data returned by the approval process. In either case, our use case needs to be updated to reflect the change.

Use Case: Register for an Event

Our walkthrough for the event registration process that will be performed by the user showed some similar issues to the event creation process. We still need interaction between the server and the database, and a step that shows the user the interface for event registration.

There is also a debatable sequence in the event registration use case. The "request to register for event" arrow is before the step where users actually look for an event. This may not generate a fault, however; there simply could be a "register for an event' button that takes the user to the event selection screen. Alternatively, users could browse events, then choose to register for one. This could just be a user experience matter.

Use Case: Participate in an Event

Our walkthrough of the user participation system showed a couple issues that could be overlooked with GPS functionality. Firstly, the application needs to read the user's GPS location. This is important because users need to opt in to GPS tracking for any application. Users can Our walkthrough for the event registration process that will be performed by the user showed some similar issues to the event creation process. We still need interaction between the server and the database, and a step that shows the user the interface for event registration.

There is also a debatable sequence in the event registration use case. The "request to register for event" arrow is before the step where users actually look for an event. This may not generate a fault, however; there simply could be a "register for an event' button that takes the user to the event selection screen. Alternatively, users could browse events, then choose to register for one. This could just be a user experience matter.

Also, manually disabling GPS completely or on an app-by-app basis will break the core functionality of Hometown Heroes. The app also needs to check the user's location with the location stored for the event in the database.

TEAM CONTRIBUTIONS

Jon Austin

1. Attended group video conference Monday, July 14
2. Architecture validation
3. Architecture implications

Valerie Chapple

1. Attended group video conference Monday, July 14
2. Created high-level architecture diagrams
3. Collaborated on lower-level dataflow diagrams

Kenny Lew

1. Attended group video conference Monday, July 14
2. Collaborated on assessing high-level architectures through quality attributes
3. Collaborated on architecture validation

Gregory Niebanck

1. Attended group video conference Monday, July 14
2. Created failure mode discussion and fault trees
3. Collaborated on participation services decomposition diagram

Charlotte Murphy

1. Attended group video conference Monday, July 14
2. Collaborated on lower-level dataflow diagrams
3. Wrestled with ACM format in Google Docs