Design Document for GeoStories project

Group#F11

MemberName: Forrest Scott

MemberName: Joel Veencamp

MemberName: Alex Rinehart

MemberName: John Egannhouse

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Change** |
| 0.1 | 10/25/14 | FS | Initial Document |
| 0.2 | 11/1/14 | AR | updated section 5 |
| 0.3 | 11/1/14 | JE | updated section 4 |
| 0.4 | 11/1/14 | JV | updated section 3 |
| 1.0 | 11/2/14 | FS | Completed the first version of the Document |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1 Introduction 3](#_Toc84977455)

[1.1 Purpose 3](#_Toc84977456)

[1.2 Scope 3](#_Toc84977457)

[1.3 Definitions, Acronyms, Abbreviations 3](#_Toc84977458)

[1.4 Design Goals 3](#_Toc84977459)

[2 References 4](#_Toc84977460)

[3 Decomposition Description 5](#_Toc84977461)

[3.1 Module Decomposition 5](#_Toc84977462)

[3.2 Concurrent Process 5](#_Toc84977463)

[3.3 Data Decomposition 5](#_Toc84977464)

[3.4 STATES 5](#_Toc84977465)

[4 Dependency Description 6](#_Toc84977466)

[4.1 Intermodule Dependencies 6](#_Toc84977467)

[4.2 InterProcess Dependencies 6](#_Toc84977468)

[4.3 Data Dependencies 6](#_Toc84977469)

[5 Interface Description 7](#_Toc84977470)

[5.1 Module Interface 7](#_Toc84977471)

[5.2 Process Interface 7](#_Toc84977472)

[6 Detailed Design 8](#_Toc84977473)

[7 Design Rationale 9](#_Toc84977474)

[7.1 Design Issues 9](#_Toc84977475)

[7.2 <Issue 1> 9](#_Toc84977476)

[7.3 <Issue 1> 9](#_Toc84977477)

[8 Traceability 11](#_Toc84977478)

# Introduction

## Purpose

To explain the software architecture of the GeoStories project and to give the rationale behind design choices.

## Scope

This document covers both the Android GeoStories Application and the GeoStories web server.

## Definitions, Acronyms, Abbreviations

|  |  |
| --- | --- |
| Term | Description |
| Zend Framework 2 | open source framework for developing web applications and services in PHP |
| Doctrine 2 | several PHP libraries primarily focused on database storage and object mapping |

## Design Goals

1. Reliability

The GeoStories project has many connections and data objects. The system must perform reliably in order to accomplish its core functions.

1. Security

It’s our responsibility to keep the user’s private data, including passwords, secure.

1. User Friendly

The GeoStoies project is a social application. In order to attract a user base of all ages, the system must be easily understood by an untrained user.

1. Maintainability

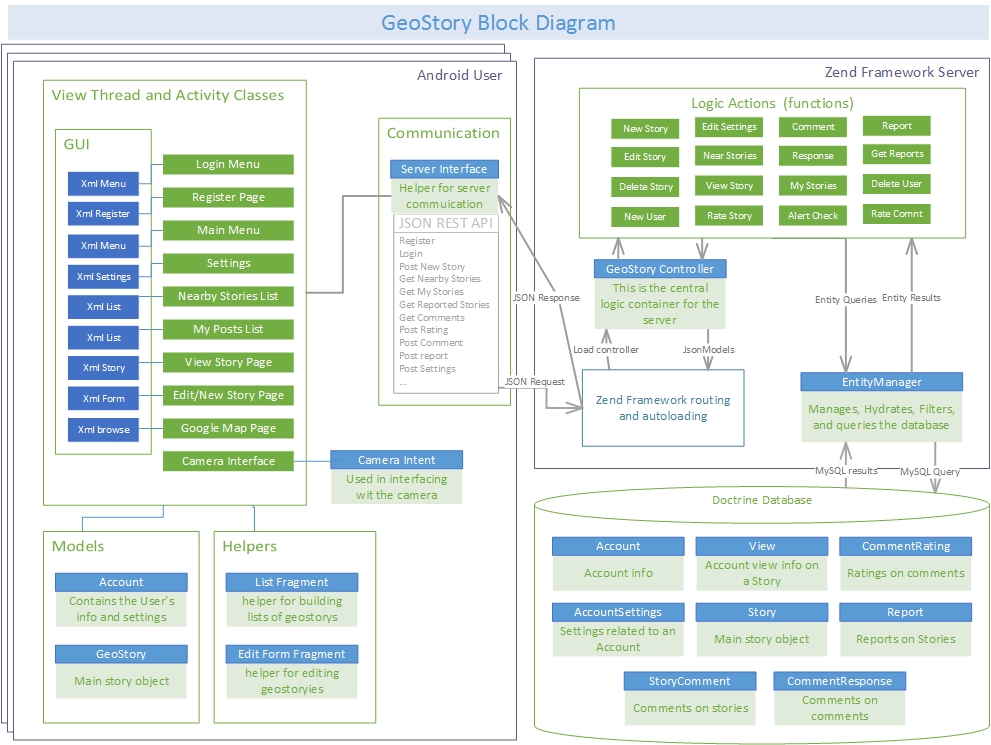
The system must be easily maintainable to keep up with new features and a growing user base.

1. Response Time

The system communicates through the web often. Because it’s a user based application, response times must be kept to a minimum to appease its user.

# References

(If any)



# Decomposition Description

## Module Decomposition

See previous figure “Geostory Block Diagram” for Block Diagram.

### <Module 1-8: Includes Login Menu, Register Page, Main Menu, Settings, Nearby Stories, MyPosts, View Story, Edit/New Story (These are described more in section 3.3> These 8 modules interact with each other through the use of intents, but also act as views. The geostory block diagram in the previous assignment shows the specific interaction between each of these modules, and are also stated in section 3.3.

### <Module 9: Server Interface Code> This module bridges the gap between the View Theads and the Server. Activities interact with it by building the async thread and registering. The server module also interacts with the server.

### <Module 10: Controller> This includes the logic and JSON parsing that is done between the app the server. This is very important to sort and interpret the data the is being sent to and from it.

### <Module 11: Database> This holds the queries that happen when a request is needed from the server. This module is the library holding all the information need (account info, Stories, ect…) that are needed to be queried (see diagram above.

## Concurrent Process

### Android View Thread Description

### Use-Case: User opens application, logs in, goes to settings page

### Uses one thread, where it is used to go through the different activities, such as described above in the use-case

### Server

### Use-Case: User opens application, logs in, creates new story.

### This Process can have two theads. When the application makes a call to the server, an asyncrounous thread is created to deal with the JSON and interactions between the application and the server.

## Data Decomposition

### <Login Menu> This class is the default opened if a user is not automatically logged in. This allows the user to log into their account, in which the server checks if there is an account with the given specications which sends them to <Stories Nearby> or go to <Register Page>.

### <Register Page> This class allows the user to register for an account, which will send the user data to the server, and promptly login the user, sending them to the <Main Screen>

### <Main Screen> This class has a list of actions the user can do (go to <Settings>, <Stories Nearby>, <My Posts>, <Edit/New Story>. There will also be text telling the user how many stories are nearby.

### <Settings> This class holds the various settings one can change, such as push notifications, ect…

### <Stories Nearby> This class gives a list of the stories that are nearby. This class can redirect to <Individual Story>.

### <My Posts> This class shows the lists of posts the user has made. A User can click on a post to go to <Individual Story>.

### <Individual Story> This class shows the current story, it’s textual representation, as well as a photo (optional). It will also show comments, and a rating. If it is the user’s post, they can redirect to <Edit/New Story> to edit such.

### <Edit/New Story> This class allows the user to edit/create a story. The user can send open the camera interface to take a photo, or choose one from a browser interface. When posted, the server fetches the story and saves it in the database.

## STATES

### non-blocked state Description When the view thread is not blocked, the user can input touch presses and change activities freely.

### blocked state Description Whenever the Application is sending or updating data TO the server, the UI will bill blocked in order to guarantee that the server handled the data correctly. Note: whenever the server is asking for data, the view UI thread will be non-blocked.

# Dependency Description

## Intermodule Dependencies

### Create Post Module

### The create story model depends on the GeoStory class. When done creating story the Create Post Model must rely on the Servers JSON interface to send the message to the server “create story” asynchronous thread is used here. This module also depends on internet access to send the Story.

### 4.1.2 My Post Module

### In order for this module to populate it list it must make a call to the sever module to get posts created by the user and the server must send back all the posts created by this user from the database module. This module also depends on internet access.

### Notifications Nearby

### This module is another list view also dependent on a call to the server module to communicate with the database module. This module also depends on the Location model for the users coordinates to populate the Stories nearby. This module also depends on internet access.

### Map

#### The map module depends on the Stories Nearby model it then uses this data to populate a Google Map. This module also depends on internet access to update your location

### Settings

### The settings menu is not dependent on other modules, but module such as notifications rely on setting to whether they will function.

### Server Controller Module

### The server controller module depends client modules to send correct JSON requests. It also depends on the database module after it parses its instructions from the app

### Database Module

### The database module is only linked to the Server controller, it sololy functions off of calls from this module.

## InterProcess Dependencies

### Location process also depends on internet access to actively update.

### Maps

### Maps depend on the Google maps api to create a map, also using this module is making an implicated call to Stories nearby.

### Settings

### Modules such as notifications and stories nearby depend on this.

## Data Dependencies

### Server data dependency

### All user info and post sent to the server are store in the database.

### 4.3.1 App data

### The app may cache data on the device such as location and maps. It also stores user info such as login info.

# Interface Description

## Module Interface

### Create Post Interface

#### When creating post, a JSON object will be created and passed to the server module via the use of listeners and asynchronous threads. The server module will then send it to the server using the HTTP protocol.

### My Post Interface

### When checking “My Posts”, the application will send a SQL request to the server via HTTP through the server module. The server will respond via HTTP and a list of stories with the current user as their author will be displayed.

### Notifications Nearby

### When checking for nearby notifications, the application will send a SQL request to the server via HTTP through the server module. The server will respond via HTTP and a list of stories which have a location within a predetermined radius of the user’s location will be displayed.

### Settings

### When changing settings, variables that affect the behavior of the rest of the applications will be modified. These variables will be adjusted via the use of set() methods.

### Controller

### The server will have a controller module which is responsible for handling incoming commands. This includes parsing incoming JSON from the application, passing appropriate data to the Database module, and sending data back to the application

### Database Module

### The database module takes SQL commands and queries the database, returning results to the Controller, which will return them to the application where the request was made.

## Process Interface

### Server Interface

The activity running on the application would create a JSON object, and start an asynchronous thread with that object as a parameter. The thread would send this data to the server via HTTP, and the server would convert the JSON into a function, which it would then call with parameters derived from the JSON. If the call was a SQL query, it will return the results to the activity, which will parse the results and update the GUI appropriately.

### Location Interface

### When the application makes a request requiring location, a request is made over GPS or Wi-fi to provide the GPS coordinates. These are reported back to the activity which called them, where they can be used or added to a JSON object.

# Detailed Design

NOT REQUIRED <Java Docs to be used instead>

# Design Rationale

## Design Issues

## Thread Scyncronisation

### Within the Android Application, The View Thread launches a JSON client thread to handle server interactions. There is a problem however with figuring out how to update the view thread WITHOUT blocking it until the JSON thread resolves.

### Factors affecting Issue

### Prevent the view thread from being blocked

### Transfer the data safely between the view thread and the JSON thread

### The JSON thread could potentially fail

### Alternatives and their pros and cons

### Use a Singleton

###### A Singleton could behave like a mutexed data object, allowing both threads to access the data atomically.

###### One problem with a singleton is that the view thread would have to continuously check if the singleton updated.

###### Another problem is when the view thread changes activities. What if the data in the singleton is updated, but the view doesn’t need it anymore? A singleton would create a lot of issues of data being passed/accessed by the incorrect activities.

### Resolution: We ended up using a design which includes listener functions. Essentially, the JSON thread calls a listener function of its own creator. This way the result is return to the *correct* activity as soon as the result is ready. Updates to the view are done on the view thread via a runOn

## LOGIN SESSION TRACKING

### Description: Keeping track of the login credentials of the app

### Factors affecting Issue

### Keep the data secure

### Remember the data between application uses

### Alternatives and their pros and cons

### Send credentials with each JSON request

###### This is less secure as a sniffer attack would have a better chance at finding user login details

### Build a custom module for handling Session data

###### This is an important piece of code to leave to inexperienced programmers and would take an additional amount of time.

### Resolution: Using our “Android Asynchronous Http Client” library, it allows the use of Persistent cookies that save into our app’s SharedPreferences object.

## GPS coordinatES

### Description: How to denote GPS coordinates throughout the system. Usually GPS coordinates are from 90 degrees North to South, and 180 degrees East and West.

### Factors affecting Issue

### representation needs to be easily parsed from a string

### data should be ideally compact

### Alternatives and their pros and cons

### Use standard format of (x degrees [W|E) y degrees [N|S]

###### This is hard to parse. and doesn’t query very well from a database

### Resolution: We decided on a format “x,y” with x being the latitude from -90 to 90, equivalent to 90 south and 90 north. y repesents longitude ranging from -180 to 180, which represents the degrees from -180 west to 180 west.

## APPLICATION LANGUAGE

### Description: We have to decide on what platform to design on: IOS, android, or windows mobile.

### Factors affecting Issue

### Availability of development devices

### Number of potential client devices

### Development license price

### Team’s familiarity with target device programming language.

### Alternatives and their pros and cons

### IOS

### IOS has a large client population.

### One of our group members is very familar with objective-c development

### Our team doesn’t own many IOS devices for development

### Development Keys are expensive, and each member would be required to purchase.

### Windows Mobile

### Windows Mobile has a small client population

### Group members are familiar with c++ development.

### Our team doesn’t own many windows devices for development

### Development Keys are free, but publishing an app is difficult.

### Resolution: We decided on developing on Android for the following reasons:

### Android has a large client population

### Group members are familiar with Java development

### All group members own an Android device

### Development Keys are free, but publishing an app is easy.

# Traceability

|  |  |  |
| --- | --- | --- |
| **No** | **Use Case/ Non-functional Description** | **Subsystem/Module/classes that handles it** |
| 1 | N/A |  |
| 2 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |