GROUPLINE

SPECIFICATION DOCUMENT

OCTOBER 27TH , 2017

**1. INTRODUCTION**

*1.1. Overview*

This document provides the Software Requirements Specifications for ‘Partner with me” project and is intended to detail the:

* Technical Architecture
* Class Diagram
* Database Design

*1.2. Scope*

Server: NodeJs

Database: PostGresSQL

Web-Framework: Express

ORM: Sequelize

Template Engine: Jade

Languages: Javascript, Python

*1.3. Reference Material*

Other system documentation for this system should include:

* System Requirement Specification

*1.4. Document Conventions*

All diagrams, charts and tables are generated using standard UML guidelines.

*1.5. Competitive Landscape*

http://appcrawlr.com/ios-apps/best-apps-group-activities

**2. SYSTEM OVERVIEW**

*2.1. Overview*

This web-application is an MVC project consisting of an N-layer architecture to separate and decouple each layer to act dependently:

* UI-Layer
* Controller/ Configuration-Layer
* Service/Business-Logic-Layer
* Data-Access-Layer

In addition to, the project will include a subset of layers required to connect and process the data transferred amongst each layer. These layers include:

* Models
* Model-View
* Utilities/common

*2.1.1. UI-Layer*

This is the front-end client-side facing layer. Its purpose is to provide the end-user with a graphical interface to interact with, e.g. Firefox, Chrome. Within the web-browser interface, the end-will have the following views:

Stakeholder Type:

Guest User:

* Log-In
* Log-Out
* Sign-Up

Registered User:

* View All Profiles
* View All Posted Activities
* View Activity Detail

Administrator:

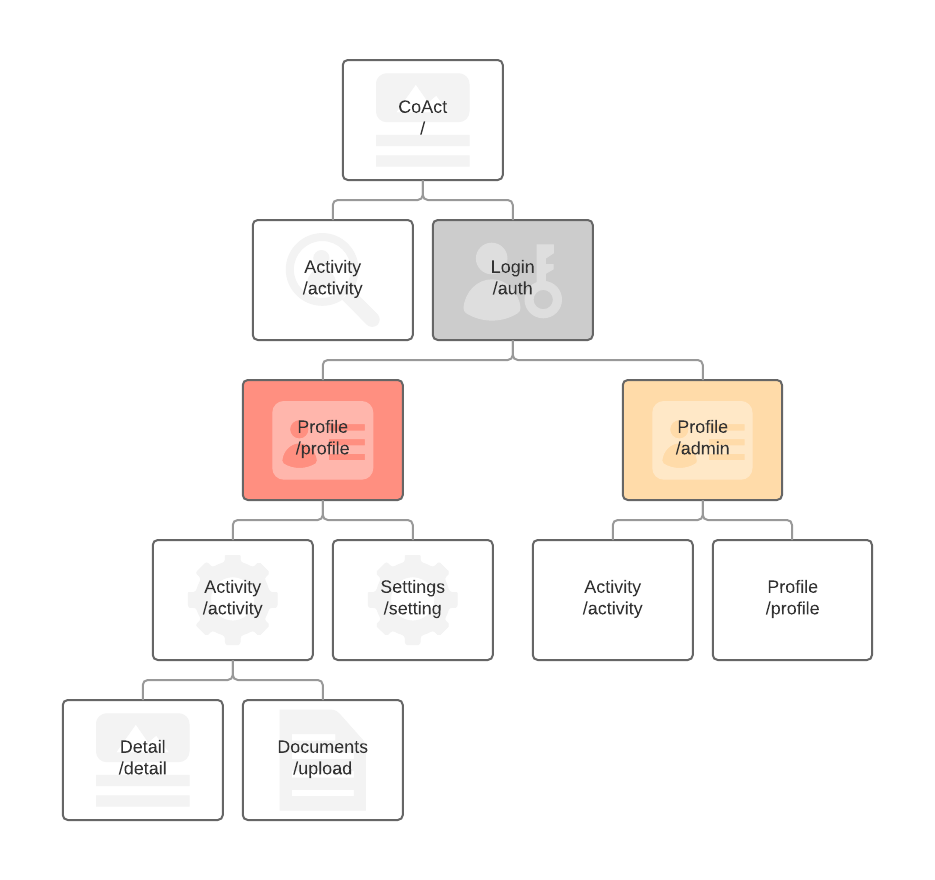
* Log-In
* Log-Out
* View All Profiles
* View User Profile

*2.1.2. Controller/Servlet/Configuration-Layer*

The controller is responsible for receiving HTTP requests and managing sessions from an end-user request and also forwarding that request to the appropriate layer, i.e. Service, where the request is processed before responding back the client.

In the “Activity Partner” project, an HTTP request to a specified URL is forwarded to the responsible controller for that URL end-point. Whereby, when the HTTP request is received, that controller forwards the process to the appropriate Service-Layer to validate and process the request, such as asking the data-layer for profile data.

The URLS for the controllers begin with the root folder location as the default prefix, i.e. [http://{project-name}/](http://libsource/) and the URL mappings for the controllers are as follows:



2.1.2.1. Registered User URLS:

* “/” Request: GET
  + Displays home page
* “/auth/signup” Request: GET, POST
  + Displays signup page
* “/auth/login” Request: GET, POST
  + Displays login page
* “/auth/logout” Request: GET, POST
  + Displays logout page

2.1.2.2. Guest User URLS:

* “/activity/” Request: GET
  + Displays list of all activities
* “/activity/:catid/ Request: GET
  + Displays details of a single activity

2.1.2.3. User Activity URLS:

* “/usr/:id” Request: GET
  + Displays logged-in user profile
* “/usr/activity/:catid /like Request: POST
  + upvotes, likes an activity (user logged-in)
* “/usr/activity/:catid/add” Request: GET
  + Add the activity to user request
* “/usr/activity/:catId/favorite” Request: GET, PUT
  + Add activity as favorite
* “/usr/activity/” Request: GET, POST
  + Add new activity category
* “/usr/activity/:actId/delete Request: DELETE
  + delete an existing activity listing
* “/usr/activity/:actId/update Request: PUT
  + update an existing activity listing

2.1.2.3. User Activity Tag URLS:

* “/usr/activity/:actId/tag/ Request: GET, POST
  + view activity tags and add new tags
* “/usr/activity/:actId/tag/:tagId/delete Request: DELETE
  + delete a tag that exists
  + must be owned by self

2.1.2.2. Admin URLS:

* “/admin/login” Request: GET, POST
  + Displays login page
* “/admin/activity/all” Request: GET, POST, PUT, DELETE
  + Displays all activities details
* “/admin/activity/:id” Request: GET, POST, PUT, DELETE
  + Displays single activity
* “/admin/profile/users” Request: GET, POST
* “/admin/profile/edit” Request: GET, PUT

*2.1.3. Service-Business-layer*

This layer is responsible for validating any request received by the Controller. For example, when an HTTP request is received by the controller, it is routed/forwarded to the Service-layer, whereas for example – in a login request, the input of the user may be authenticated before forwarding the request to the Data-layer – preventing further processing if the credentials are invalid.

Furthermore, each business object (BO) model of the application will consist of a service implementation. This implementation will validate most requests that require data, thus preventing invalid and/or malicious data from entering the data-layer and/or database.

*2.1.4. Data-Access-Object-Layer (DAO)*

The Data-Layer contains the data-manipulation-language (DML), data-definition-language (DDL), and data-control-language (DCL) logic required to execute queries for the database. Each CREATE, READ, UPDATE AND DELETE request to this layer attempts to perform one of the fore-mentioned operations.

Once the operation has been performed, the result of the operation is returned through the layers of the application.

*2.1.5. Models-Layer*

In this application, the model layer will consist of classes/objects that the application can translate different data–types received by another layer to understand. The model layer contains class objects that enable the application to store temporary data received from an input form, database or some utility class, for example.

*2.1.6. Model-View*

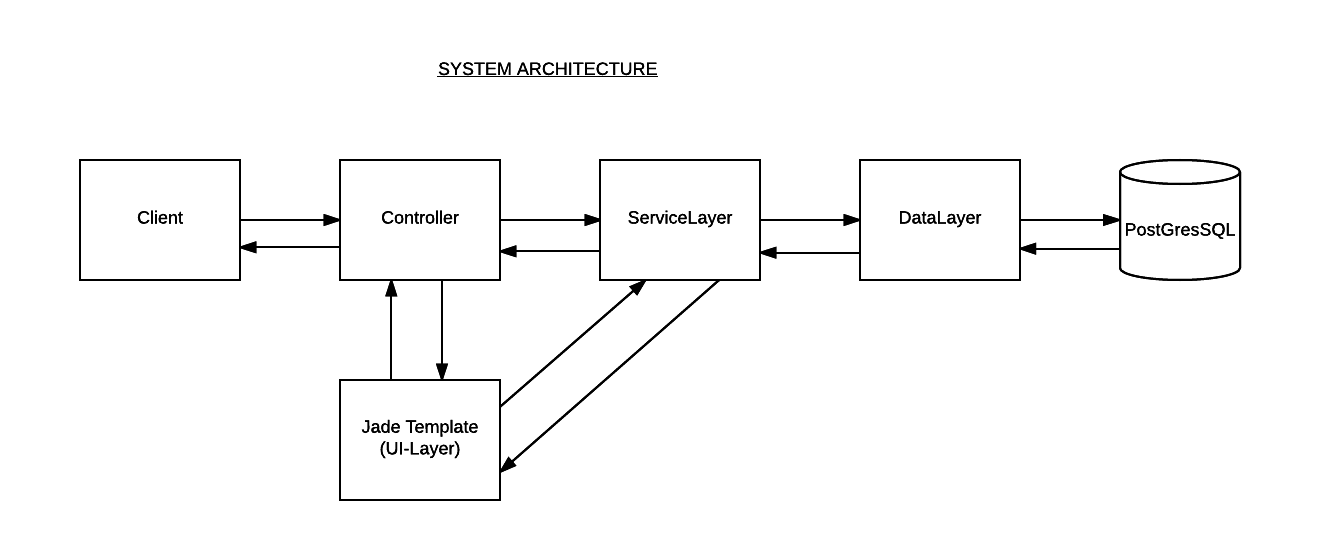
Content the user is viewing can require many different fields and, because of that, require special objects to construct it. This requires changing the default class or object to conform to the view. Therefore, rather than changing the default construction of the object to conform to the view requirement, this Model-View-View-Layer will be utilized to construct specialized views to display the data properly within a view.

*2.1.7. Utilities/Common-Layer*

Will consist of generic helper and common objects that may be utilized by many different Objects from other layers.

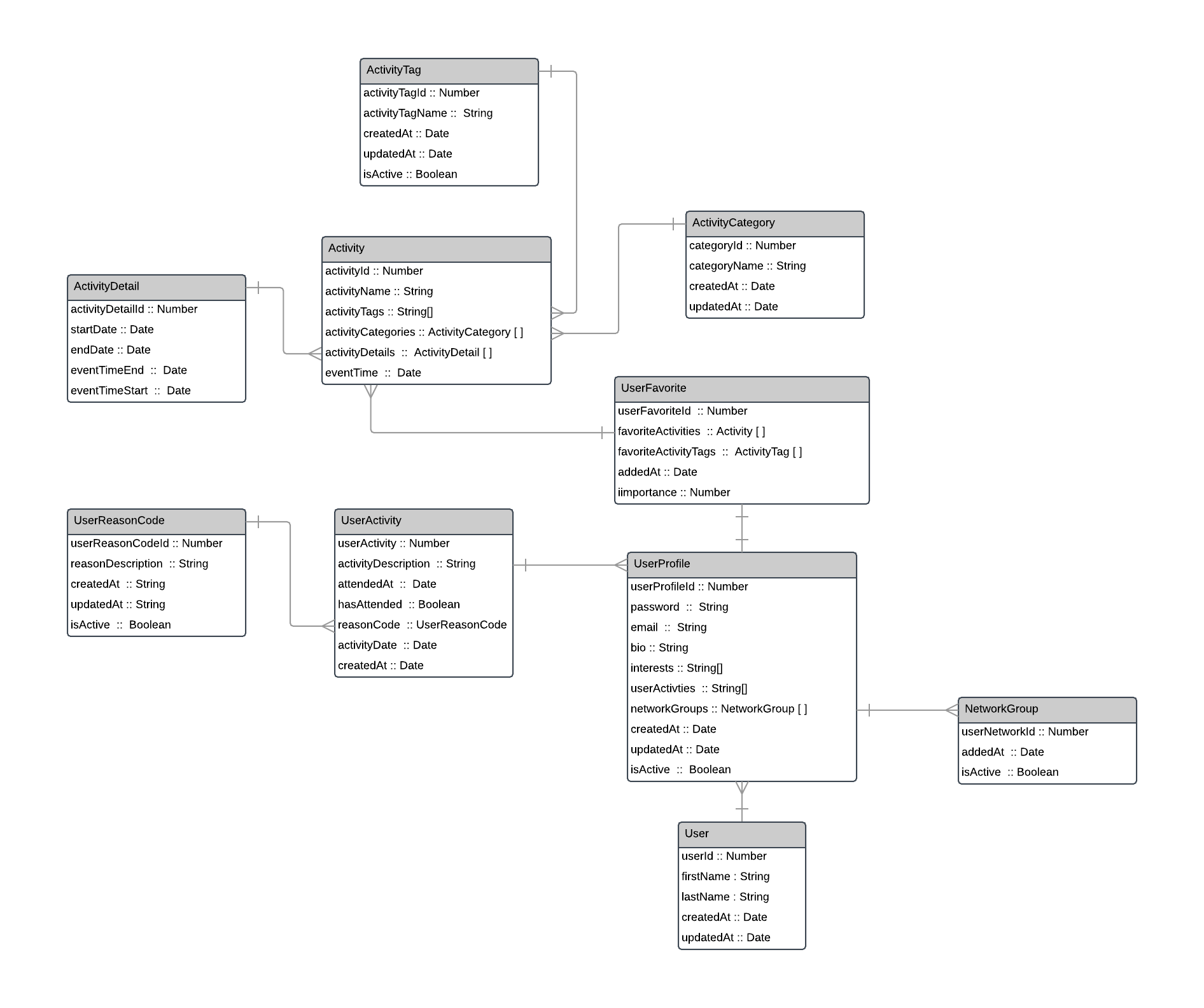
**3. SYSTEM ARCHITECTURE**

*3.1. Architectural Design*



*3.2. Decomposition Description*

This section decomposes the Models of the software application and the general implementation of its members.



*User Domain*

*3.2.1. User*

The User is the basic composition of a user object.

* userId
* firstName
* lastName
* createdAt
* updatedAt

*3.2.2. UserProfile*

The user-profile object composes of members that would represent a profile of a user.

* userProfileId
* password
* email
* bio
* interests
* userActivities
* networkGroups
* createdAt
* updatedAt
* isActive

*3.2.3. UserActivity*

User activity object represents the basic composition of an activities user has participated or engaged with.

* userActivityId
* activityDescription
* attendedAt
* has Attended
* reasonCode
* activityDate
* createdAt

*3.2.4. UserReasonCode*

User Reason Code object represents the basic composition of reasons that describe whether the user participated or did not participate in the activity.

* userReasonCodeId
* reasonDescription
* createdAt
* updatedAt
* isActive

*3.2.5. NetworkGroup (Experimental)*

Network object represents the basic composition of users in a group, following.

* userActivityId
* activityDescription
* attendedAt
* has Attended

*3.2.6. UserFavorite*

User Favorite object represents the composition of activities and items that a user views as a favorite.

* userFavoriteId
* favoriteActivities
* favoriteActivityTags
* addedAt
* importance

*3.2.7. Reputation*

Reputation object represents the users given cred for engaging with activities.

* reputationId
* reputationValue

*Activity Domain*

*3.2.7. Activity*

Activity object represents the composition of an activity that a user can participate in

* activityId
* activityName
* activityCategories
* activityDetails
* eventTime

*3.2.8. ActivityCategory*

Activity Category object represents the composition of categories that group a specific activity.

* categoryId
* categoryName
* createdAt
* updatedAt

*3.2.9. ActivityTag*

Activity Tag object represents labels in which an activity can be identified to a have an association with.

* activityTagId
* activityTagName
* createdAt
* updatedAt
* isActive

*3.2.10. ActivityDetail*

Activity Detail object represents the details that identify the time and location of an activity

* activityDetailId
* startDate
* endDate
* minActor
* maxActor
* eventTimeStart
* eventTimeEnd