Design Documentation For Quest Till Done

Prepared By

Mun Khong Chin
Kevin Huang
Mark Scheid

At
Drexel University

Advisor
Yuanfang Cai

Stakeholder
Robert DiMarco, CTO of eLocal.com

Revision History

Version	Prepared By	Date
0.1	M.K. Chin	1/1/2014
1.0	The QTD Team	2/18/2014
1.1	The QTD Team	4/29/2014

Contents

1. Introduction

1.1 Purpose

The purpose of this document is to describe the implementation details of the QTD software. The performance, functionalities and various constraints, including graphical user interfaces, will be directly affected by these design decisions. This program will address the lack of effective tools to keep a complete log of a programmer's development process, expanding upon existing source control and bug tracking systems.

1.2 Scope

This document describes the software architecture and design for the initial release of QTD version 1.0.

1.3 Overview

This document contains three major components, system architecture design, data design and design rationale.

1.4 Definitions and Acronyms

Active Record: The Ruby On Rails base class that generates a relationship between the model, or class definition, and the relation, or database schema. All Ruby classes in this project inherit from Active Record

Adventurer: Synonymous with user.

Adventurer Level: A count of all levels the user has achieved since the creation of their account.

Campaign: Synonymous with Project.

Experience: A measure of user progress, relevant to Adventurer/Recent Level as well as Skill Points. Higher levels require more experience to level up.

Encounter: Synonymous with Time-block.

GitHub: Web based hosting service that uses Git revision control.

Group: A collection of users and their shared projects.

Level: A measure of the user's effort invested in projects in QTD.

Link: A record that represents a specific website, like a bookmark for a browser.

Model: A blueprint representing a single object.

Module: A collection of methods and constant. The methods appear as methods in a class when inserted into the class.

Record: A fundamental data for annotation of tasks. This could be a note, link, commit, or image to provide additional log information.

Project: A project is a group of tasks that are strongly connected to each other by dependencies. The top-most task in the dependency hierarchy defines the project name and details

QTD Site Administrator: One user account with maximum permissions, able to perform maintenance on member accounts as well as the QTD server and database.

QTD Group Administrator: A specific user who manage groups of QTD Members.

QTD Member: Registered users of QTD, with no special permissions.

QTD Group Member: A user registered as a part of a given group.

Quest: Synonymous with Task.

Recent Level: A count of all levels achieved by the user in a rolling 30 day period.

SCM: Source control management system, a system that is responsible for management of changes or revisions for computer programs. This term is interchangeable with VCS.

Skill-points: A representation of a user's time investment in a given task type, such as a programming language, on a scale of 0-20, measured by reading tags on completed tasks.

STI: Single table inheritance, a pattern used to simplify the database by using one table to represent several, related models

Tag: A label for classifying tasks and record.

Task: A task is some discrete, actionable item, the building block of the projects.

Timer: A timer functionality that allows the user to work on tasks for the duration of a time-block

Time-block: A user defined duration of work for grouping user progress on tasks during the given time period.

User: The generic term for a registered user of QTD. This includes QTD Members, QTD Group Administrators, QTD Group Members, and QTD Group Administrators. Will be used wherever the difference between these groups and their permissions are not relevant.

VCS: Synonymous with SCM.

Workflow: Representing all process of a work, including all setups, intermediate development, formal and informal work related to given task.

2. Application Overview

2.1 Technologies Used

2.1.1 Development

2.1.1.1. Ruby on Rails

Ruby on Rails is a powerful MVC framework that drives QTD web services.

2.1.1.2. jQuery and JavaScript

jQuery and JavaScript will power the front end of QTD, dynamically/asynchronously displaying content through web server based on user interaction.

2.1.1.3. Postgresql

Postgresql is the database that powers QTD website.

2.1.2 Testing

2.1.2.1. Cucumber

Cucumber will be used as the graphical user interface-testing library that helps verify GUI is working as defined.

2.1.2.2. Jasmine

Jasmine is used as the JavaScript testing library that helps verify JavaScript validity.

2.1.2.3. Test::Unit for Ruby on Rails

Unit testing for Ruby will ensure internal logic is correct with complete coverage.

2.1.3 Library integration

2.1.3.1 Ruby Gems

Ruby's libraries are compiled nicely by the Ruby-Gems website. Details on any of the following gems can be found by imputing the name into the search field.
http://www.rubygems.org

The gems listed below are the technologies this software will leverages and utilize. Table 1 is the Gem listing table which briefly describes the functionality of the gems acquired from rubygems.org and its classification/role in the MVC framework.

Table 1 Gems Listing

Ruby Gem Name	Functionality Description	MVC Classification
acts-as-taggable-on	Adding Tag Functionality	M
attr_encrypted	Privacy for ActiveRecord fields	M
coffee-rails	CoffeScript Support	V

consul	Permission and power	C
cucumber-rails	Functional Testing	V
d3_rails	Dynamic visual display layer	V
database_cleaner	Advanced Database Support	M
delayed_job_active_record	Adding Queueable Jobs	M
devise	Single User Permission	M,V,C
execjs	Run JavaScript from Ruby	M,V
flipclockjs-rails	Timer	V
github_api	Github API Provider	M,V,C
haml	Support for HAML Format	V
high_voltage	Support for Static Page	V
jasmine-core	JavaScript Behavior Testing	N/A
jbuilder	JSON Ruby Helper	C
jquery-rails	JQuery Integration	V
jquery-ui-rails	JQuery UI JS integration	V
localtime-rails	Converts times to browser TZ	V
minitest-rails	Minitest Integration	N/A
pg	PostgreSQL Support	M
paperclip	File Upload	M,C
rack-mini-profiler	Optimization tool	N/A
rails 4.0.0	Rails Framework	M,V,C
sass-rails	Support for Sass Language	V
single-test	Testing tool	N/A
spork-rails	Testing Server	N/A
turbolinks	Browser Optimization	V
uglifier	JavaScript Runtime Tester	N/A
watir-webdriver	Backend for functional testing	N/A
will_paginate	Pagination Library	V,C
yard	Ruby Documentation Library	N/A

N/A in the table denotes not belonging to any of the MVC category. This means such gem is related to testing, documentation or other project management functions.

2.2 System Overview

QTD will be a web application developed in Ruby on Rails following the paradigm of MVC, model view and controller. It is designed to incorporate existing SCM tool to increase productivity through better logging by consolidating a user’s backlog

3. System Architecture

3.1 Architectural Design

QTD is designed to follow the Model-View-Controller pattern, which separates the representation of data and the user interface from the implementation and business logic. This design pattern is shown in Figure 1. This differes from the conventional MVC model, as the Rails framework by design routes interaction between model and view through the controller.

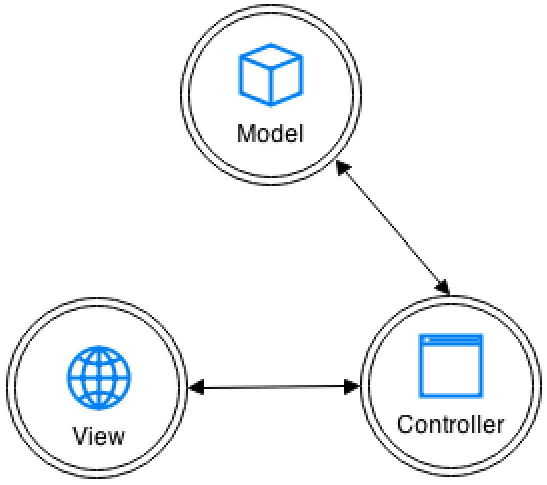


Figure 1 MVC overview

3.1.1 Context Diagram

The context diagram Figure 2 shows how the main components of QTD will interact with each other at a high level view.

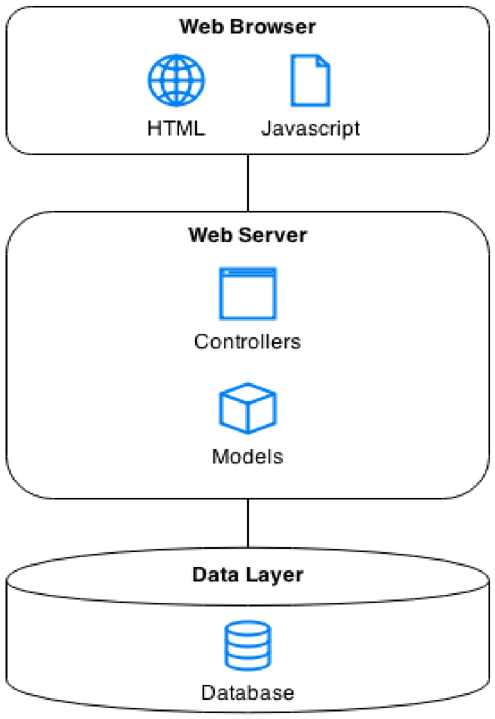


Figure 2 Context Diagram of QTD

3.1.2 Site Diagram

Site diagram in Figure 3 shows how the major pages will fit into the QTD system via navigation between pages.

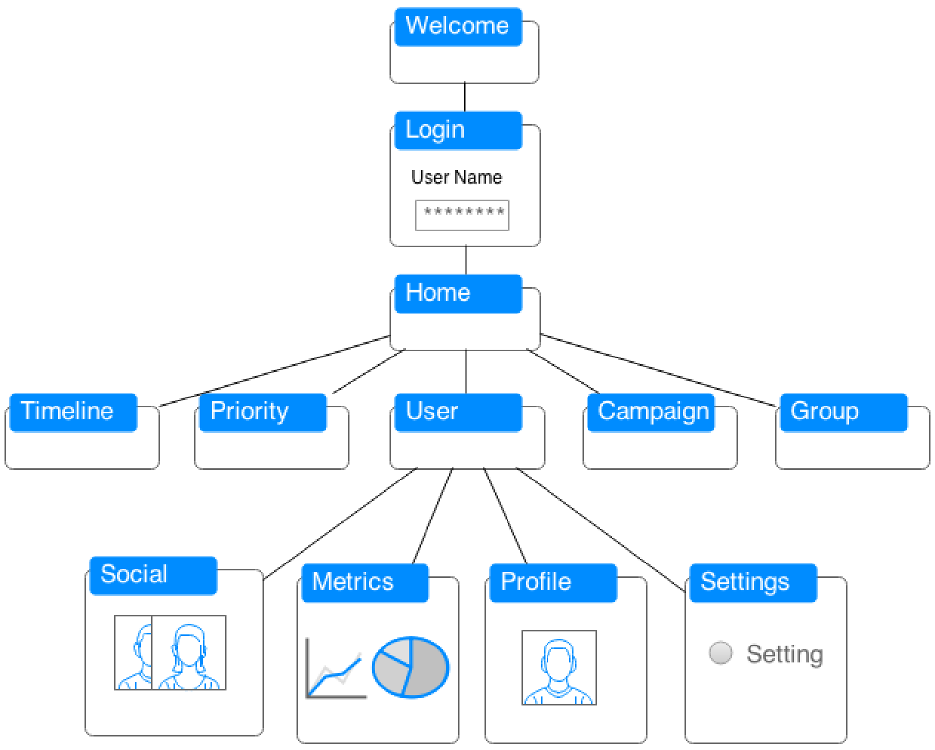


Figure 3 Site Diagram

3.1.3 Architectural Diagram

Architectural diagram Figure 4 shows the overall architecture of the entire QTD system, with each major component in each major section listed.

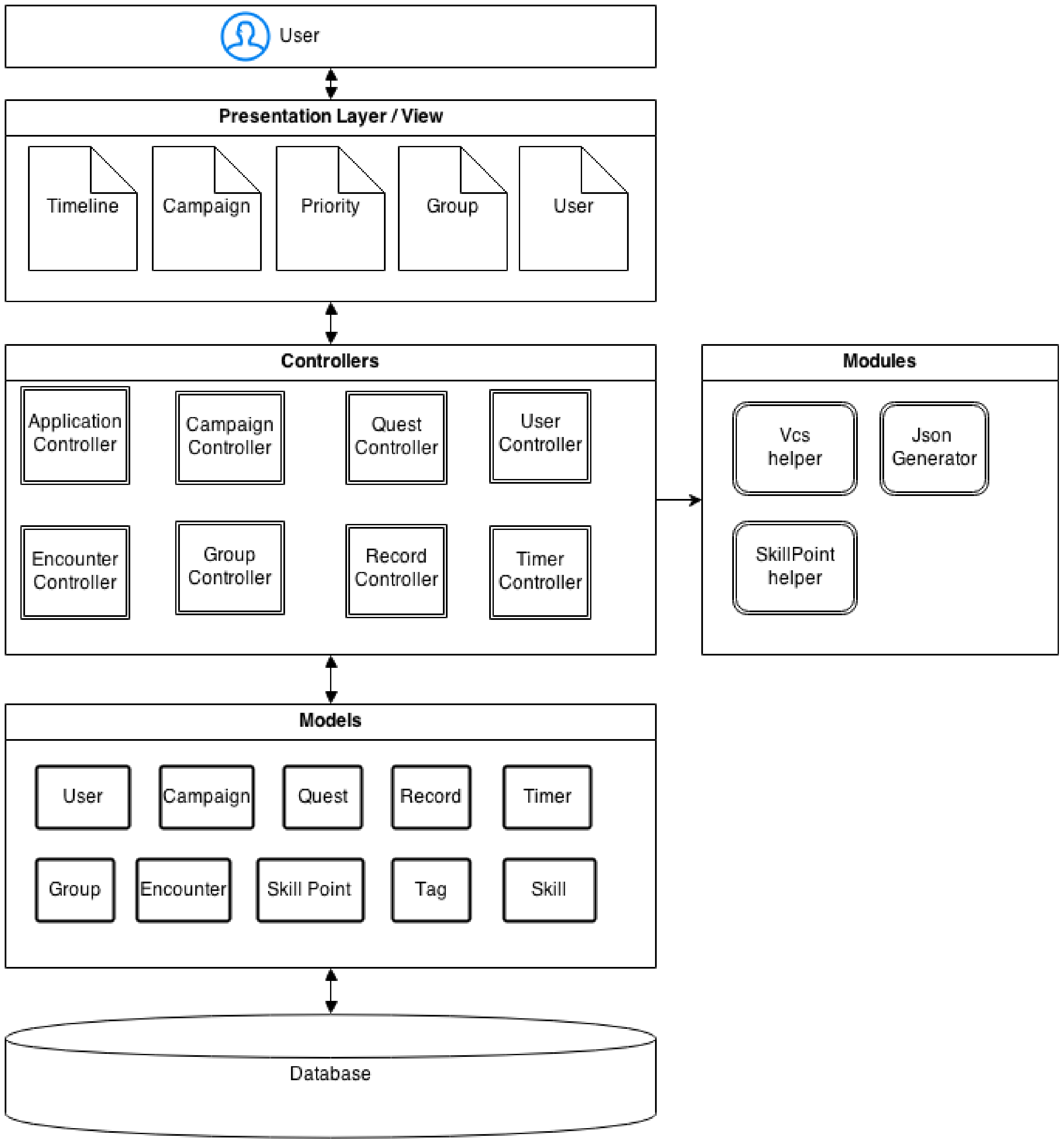


Figure 4 Architectural Diagram

3.2 Decomposition Description

3.2.1 User Account

3.2.1.1 Login

The login feature will allow the user to login and authenticated using their password. The authentication process is handled by the *Devise* gem. This includes login and logout functionality, and stores passwords as a hash. A user helper class is designed to facilitate more detailed username length control and password strength definition. This is handled by user model.

3.2.1.2 Logoff

As referenced in 3.2.1.1 in SRS the logout feature is handled by *Devise* gem. Once a user has logged out, user session will cleared, and any part of the website which requires authentication will be inaccessible.

3.2.1.3 Register

A user can sign up by filling out a form detailing desired Username, Email Address and Password. The system will validate that the email address is unique in the database. After the user has registered, an email will be sent to the user to verify successfully registration. These are handled by notification controller, user model, and user controller.

3.2.1.4 Modification

The user can go to profile view as per section 5.2.1 in SRS and modify their Email Address, and Password. This is handled by *Devise* gem and user model.

3.2.1.5 Deletion

The user can request to remove their account using the settings view as per section 5.2.2 in SRS. The user model handles this feature. Once the user is deleted the username won't be available for future users.

3.2.1.6 Reset Password

When a user forgets his/her password, he/she has the option to restore password by resetting password. In this case a randomly generated password will be sent to the registered email address. The User model and mailer class handles this functionality.

3.2.1.7 Active Session

A user will be considered active if no interaction with the system for 60 minutes, as recorded in the session table, further described later in this document. Client side, the browser detects the inactivity, and if session is passed, user will be kicked out. The automatic logout function is handled by *Devise* gem.

3.2.1.8 User Type

User types, as described in the SRS, are defined in the system by the relationships provided by the Group model, and the related groupsusers and adminsgroups tables. All new users belong only to a unique group which shares their user-name, which serves only to preserve the privacy of their personal projects. They are considered QTD Members at this point.

Upon navigating to the groups page, a user can create a group, whereupon they become a QTD Group Administrator for that project. Any other members they choose to share access with will be given QTD Group Member status or QTD Group Administrator status for the group in question, as per the current QTD Group Administrator's instructions. Futher details of this system follow in the definition of groups, and in the documentation for permissions in the Power model.

The QTD Site Administrator represents a user with administrator access to all groups, and thus the ability to manipulate and appropriately support all users with issues in their data. This functionality is all managed through the Group model and controller, as well as the Power model.

3.2.1.9 Friends

Users can add friends to their friend list. This process is handled by the friends_controller and linked with user model. See class documentation for more details.

3.2.1.10 Skill and SkillPoints

The user can gain skill points during regular usage of QTD. The skill points will be mapped by a Skill table which shows the user's progress and achievements. These are handled by skill_point and skill models, respectively.

3.2.2 GitHub Integration

3.2.2.1 GitHub Integration

The user can register their GitHub account in their profile setting page. The user will be prompted for their GitHub username and password to connect to their GitHub account. These settings are modifiable through the same setting page, and removable and emptying the input boxes. Once removed, the system will not be able to fetch any new updates from the user's GitHub via web API.

3.2.2.2 GitHub Synchronization

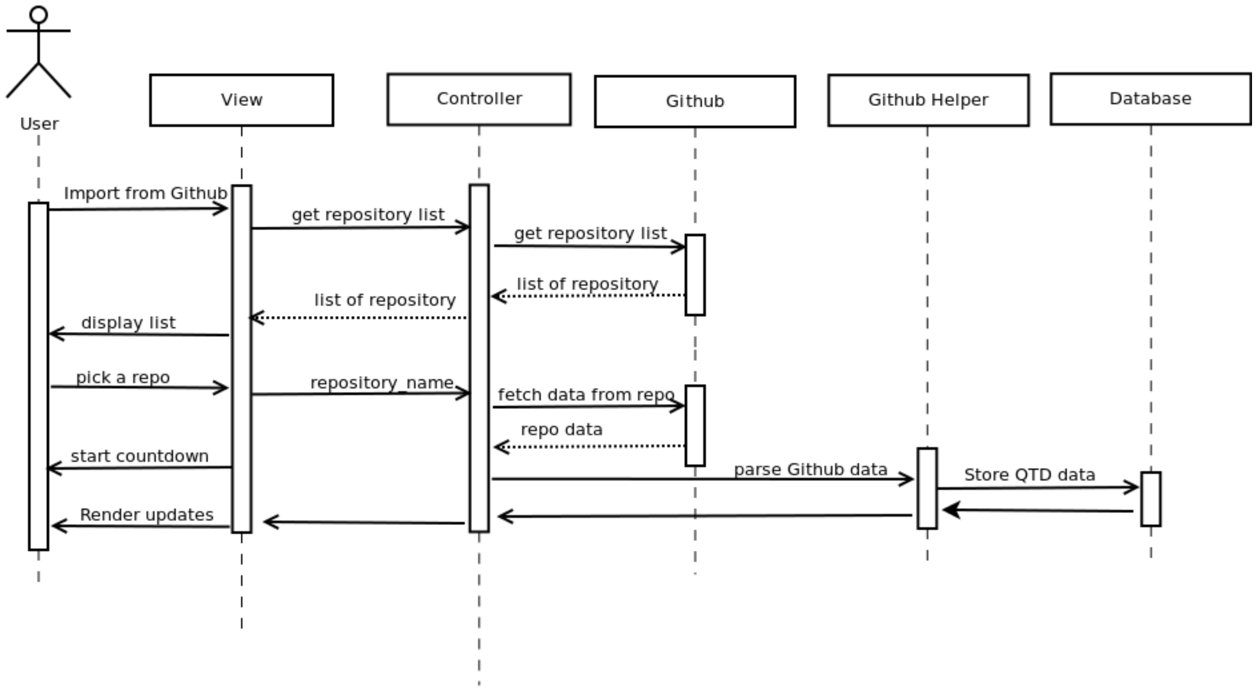


Figure 5 GitHub Synchronization Diagram

The user synchronizes and fetches his/her GitHub repo into a QTD project through the synchronization repository option. The system will fetch a list of user's repositories hosted on GitHub via GitHub API and lets the user pick from the list the repository they want to pull from. Once a repository is selected, the system will fetch the commit history, issues and related comments from the repository via the GitHub API. Then the data will be parsed into a QTD file format, which will be then parse as actual model and save it to the database as a new campaign.

3.2.3 Project Management

3.2.3.1 Create Project

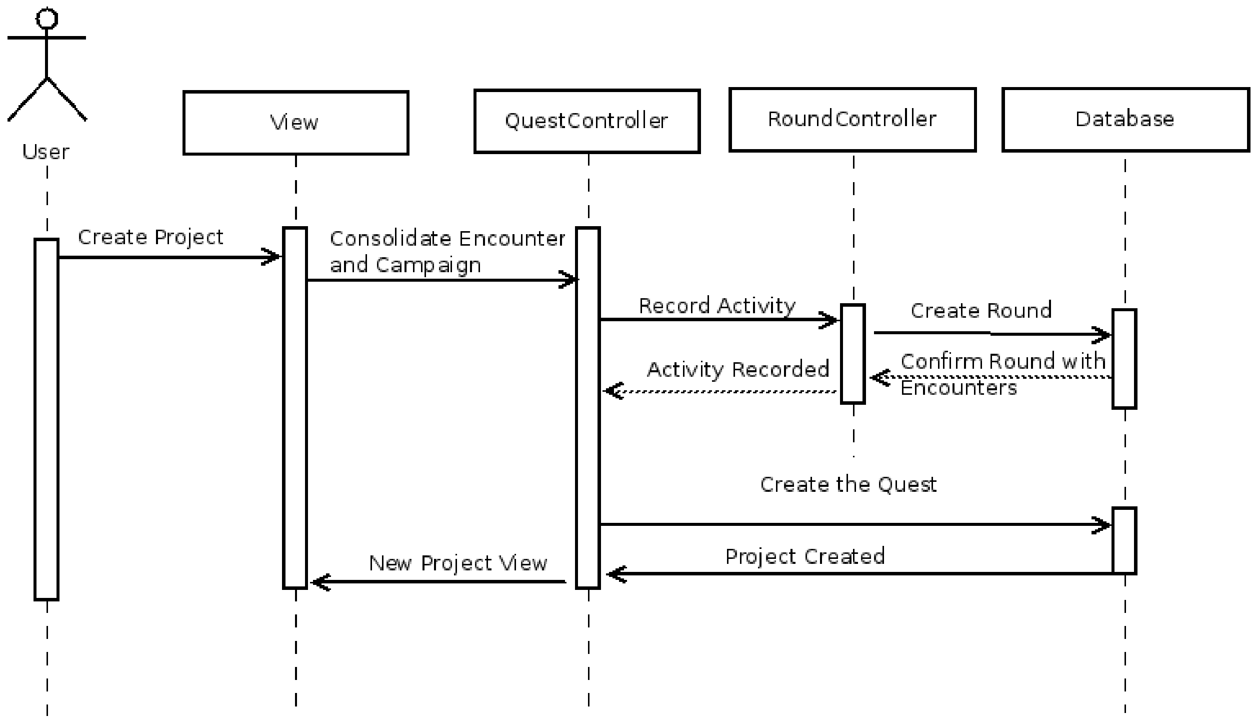


Figure 6 Create Project Sequence Diagram

A user creates a new project by using the create project option. The creation process validates the details for project before submitting. If the model is valid, a round will record the activity, then the quest is submitted for creation, and the new project will be created and saved into the database. The user will finally be redirected to the newly created project page.

3.2.3.2 Modify Project

The user modifies an existing project by using the edit project option. The system will validate any changes to the project to make sure the changes are valid and following the project model's constraints. Once validated, the new changes will be saved and updated in the database and reflected in the view.

3.2.3.3 Delete Project

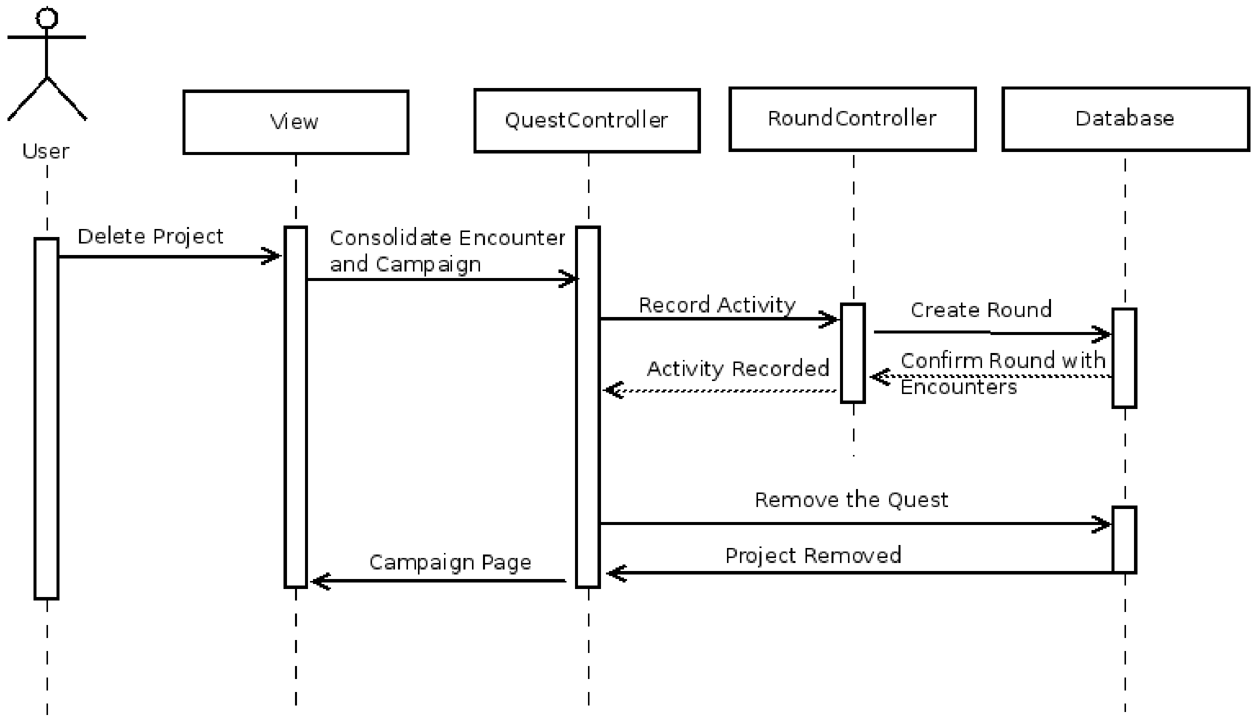


Figure 7 Delete Project Sequence

The user deletes a project through the delete project option. The system will prompt the user for confirmation before deletion. Once confirmation is received, the system will first remove association any group has with the current project. Then the system will remove all tasks in the project. Finally, the project itself will be removed from the database. Once done, the user will be redirected to the project index page.

3.2.3.4 Import Project

The user imports a project through the import project option. The user will be prompted to upload a QTD format file to be parsed and import as a project. Once the file is uploaded, the system will parse the QTD format file and store it as a new project in the database.

3.2.3.5 Export Project

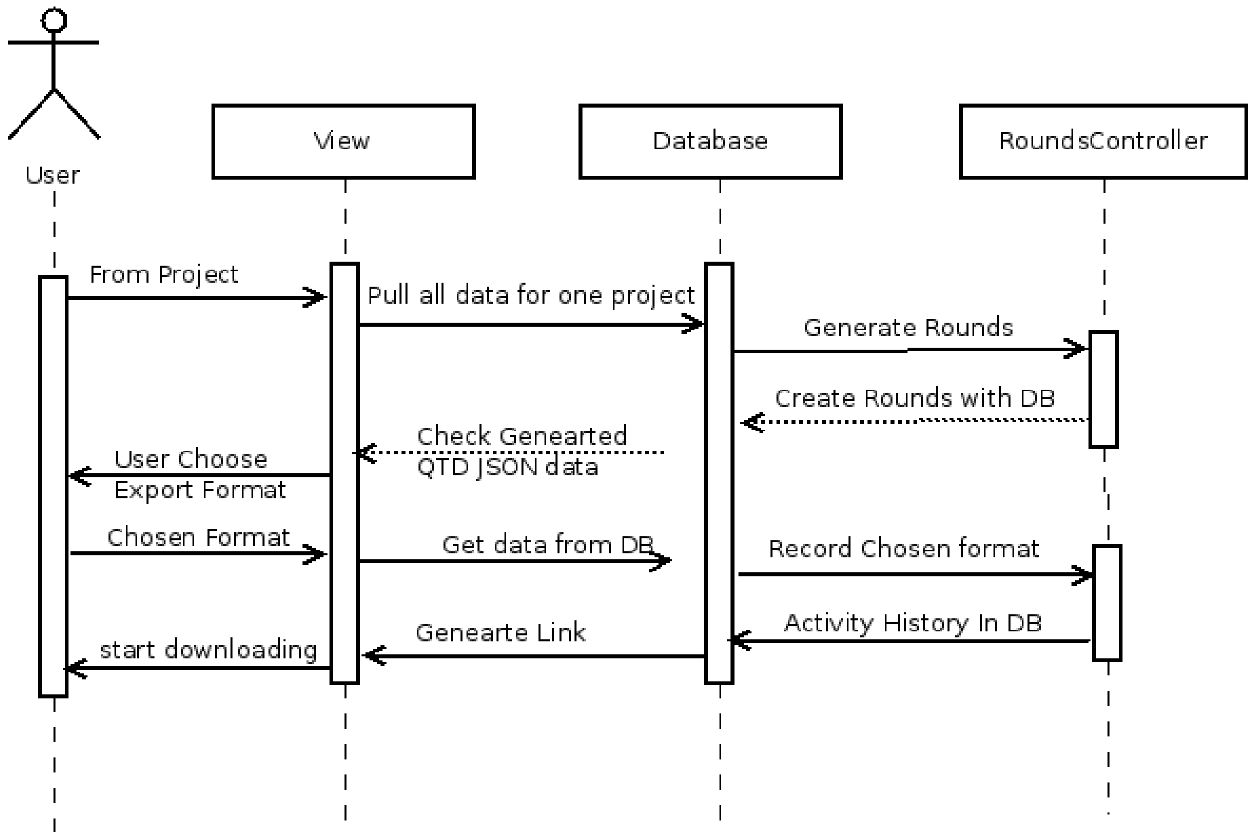


Figure 8 Export Project Sequence Diagram

The user exports an existing project through the export option. Once selected, the user will be prompted with the choice to export the project as a PNG, JPG or a QTD formatted file. A downloadable file will be generated for the user based on the format selected.

3.2.3.6 Task management

3.2.3.6.1 Add Task

The user adds a task in a project page through the add task option. Similar to the project creation page, user enters the details for the task and once it is validated, a new task will be added to the project.

3.2.3.6.2 Modify Task

The user modifies an existing task in a project through the edit task option. Once the changes are validated successfully, the new changes will be saved to the database.

3.2.3.6.3 Delete Task

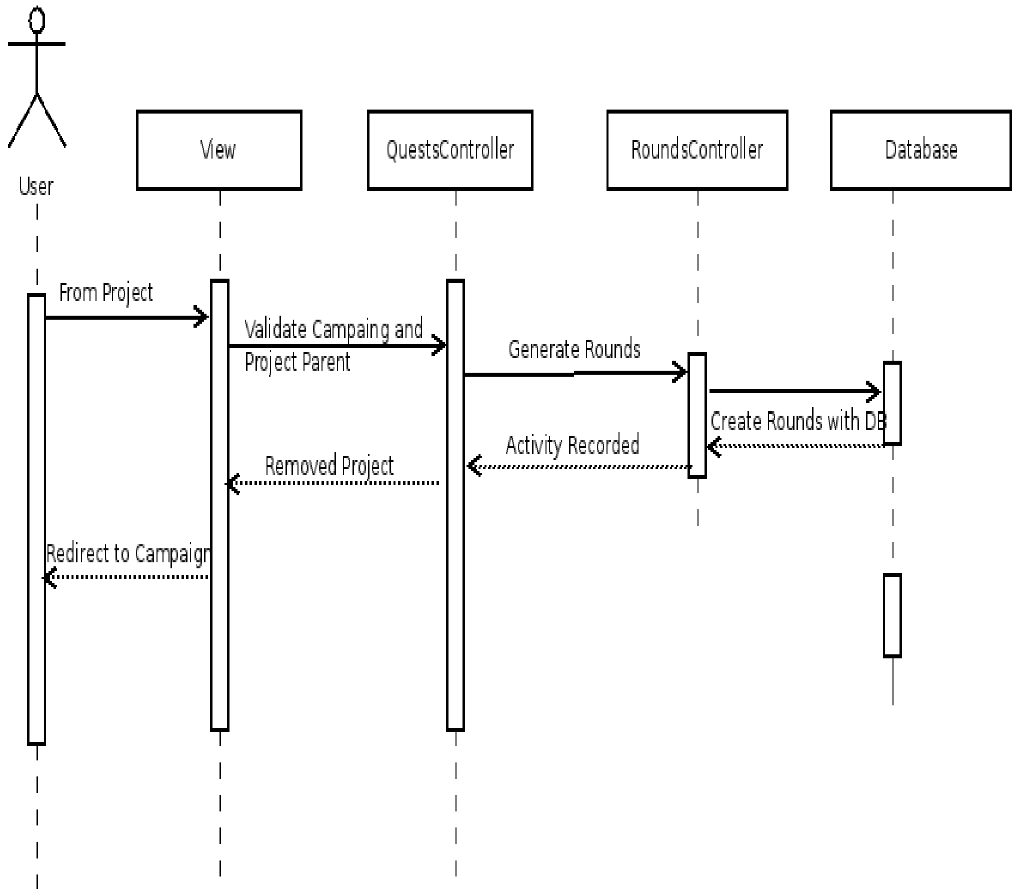


Figure 9 Delete Task Sequence Diagram

The user deletes an existing task in a project through the delete task option. The user will be prompted for confirmation before deletion. Once confirmed, the system will delete any record associated with the task, and then delete the task from the database, before returning the user the project page.

3.2.4 Group Definitions

The group model will serve as the primary permission management tool within QTD, restricting access to view, edit, or otherwise manipulate data in the system. From a data level, this means using the Consul gem to define permissions for each action on each model, as determined by group membership, which in turn will be determined by the database tables for group members and group admins. On a user level, this means that users will be able to quickly share every aspect of their development process by making a shared group containing shared projects, notes, and other data. A single action adds a user to the group member list, and gives them basic permissions for all associated data, and thus simplifies permissions management to a simple question of checking the QTD group member type. Full details of what permissions are available to users and admins are documented below in the Power model definitions.

3.2.4.1 Group Creation, Deletion

This feature is defined in Group model. The model links the user to group relationship.

3.2.4.2 Group Information

The group information is provided in the social tab which described in 3.2.4.2 in SRS. The user interface is defined in section 5.2.4 in SRS.

3.2.4.3 Group Membership

A user can join a group by accepting the invitation via notification in their profile. Once accepted, the user will be added to the group as a QTD Member. Similarly, a user can leave a group by using the leave group option in their profile page. Once confirmed the user will be removed from the group and no longer has access to the group’s page.

3.2.4.4.1 Group Administrator Add Member

The group administrator invites other QTD users to the group via the add member option in the group page. The invite will be sent to the user once a valid user name is entered in the search box via storing the invite in the database intended for the targeted user.

3.2.4.4.2 Group Administrator Remove Member

The group administrator can remove an existing QTD member from the group via the remove member option. Confirmation box will be prompted to the user. Once user confirms the action, the targeted QTD Member will be removed from the group by removing the user in the group’s list of user in the database.

3.2.4.4.3 Assign and Remove Group Administrator Status

The group administrator assigns or removes other QTD Members as a group administrator through the promote and demote options. The user will be prompted for confirmation. Once confirmed, the QTD Member’s role will be updated in the database. The changes will be reflected in the page through a refresh. If it is removing a group administrator, similarly the QTD Group Administrator’s role in the group will be updated in the database.

3.2.4.4.4 QTD Group without Group Administrator

To prevent orphan groups, when a QTD Group Administrator leaves a group, the system will check for the number of remaining members and admins. If there are no members left in the group, the system will delete the group from the database. If there are members, but no admins left in the group, then the oldest member in the group will be automatically promoted to administrator status.

3.2.4.4.5 QTD Group Administrator Project Management

Group administrator can create project for the group. New project created in the database will belong to the group, which is accessible to all members via the permission defined in the group’s users table in the database.

3.2.4.5 User permissions

As defined in the power model in rails, the user will have the following abilities:

A user will be able to create, update and view any campaign in their groups.
A user will be able to destroy any campaign they are an administrator over.
A user will be able to create, update and view any quest in their groups.
A user will be able to destroy any quest they are an administrator over.
A user will be able to create and view records related to any group they are part of.
A user will be able to destroy records related to any group they are an administrator over.
A user will be able to update records they create.
A user will be able to view, create, and leave groups they are part of.
A user will be able to update and destroy any campaign they are an administrator over.

3.2.5 Timer

3.2.5.1 Start Timer

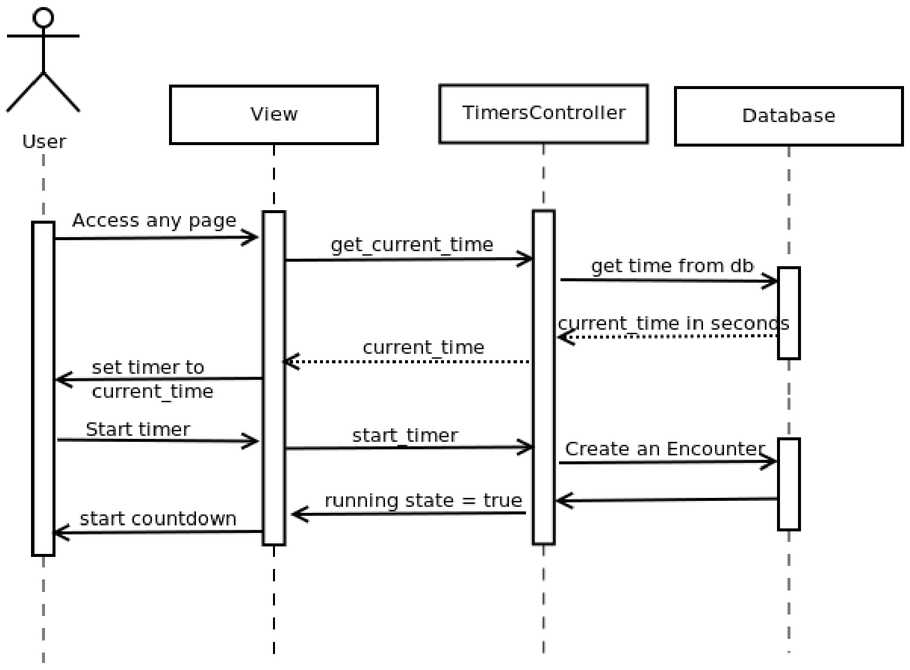


Figure 10 Sequence Diagram for Starting Timer

The user starts the timer by using the start timer option in the timer interface. The page will get the current remaining time (if any) or the default time length for the user if there is no remaining time from the last countdown, and return to the view. The view will generate a countdown timer based on that value and present to the user. Once the option is selected, the server will record the current state of the timer, and notify the view to start the countdown.

3.2.5.2 Pause Timer

The user pauses the timer by using the pause timer option in the timer section. Similar to the sequence in start timer, the view will notify the controller to stop the timer, where the server will record the current remaining time and the current state of the timer.

3.2.5.3 Stop Timer

The user stops the timer by using the stop timer option in the timer section. Once the user selects the stop option, the view will notify the controller to stop and reset the timer, which resets the timer to the default time length configured by the user. The controller will close previously created encounter and update it in the database.

3.2.5.4 Configure Timer

The user configures the timer settings in the timer configuration page. The user can set the default time length for an encounter, enable or disable the timer for auto and manual mode, and configure the break duration in the setting page. The view will notify the server about any changes and save them in the database. See 5.2.2 for more details.

3.2.5.4.1 Auto or Manual Mode

In Auto Mode, the timer is automatically determined from the user_config table as long as the user’s session is still valid. When the timer runs out, a Stop timer and a Start timer operation will be automatically performed for the user, and hence the old encounter will be closed and a new encounter will be generated.

In Manual Mode, the timer will not refresh when the timer is up. The timer will be reset and any encounter previously created is closed. This allows for more custom control over the user’s workflow, and allows the user to utilize the break system, defined in 3.2.5.6-7.

3.2.5.5 Extend Encounter

The user extends an ongoing encounter by using the extend option in the timer section. A time length specified by the user in the time configuration section is then added to the current timer to extend the countdown. This is defined in the user_configs table.

3.2.5.6 Short Rest

The user enters rest period when a timer is up, when the rest functionality is enabled in the timer configuration. A new timer will be started with an encounter containing a ‘break flag’, and that period is counted as a rest period for the user. User_configs table contains this information.

3.2.5.7 Extended Rest

As per a short rest, a user can separately define an extended rest duration for longer breaks throughout the working day, as a reward for a series of successful encounters.

3.2.6 Record

3.2.6.1 Add Record

The user adds a new record to a specific task by using the “Add Record” button in the header bar to add for the current active task, or add record option for each individual task. The user selects the appropriate record type (note, link or image) and the system will prompt for input as shown in the diagram. Upon successful validation, the new record will be saved to the database for the specified task. See Figure 40

3.2.6.2 Modify Record

The user modifies an existing record through the modify option. For a note, the description could be changed. For a link, both description and URL can be changed. For an image type record, the image file can be re-uploaded to replace the existing and the description can be changed. Upon successful validation, the new changes will be saved to the database. See Figure 41

3.2.6.3 Delete Record

The user deletes an existing record through the delete option. When the delete option is selected, the page sends a delete request to the server where the record will be deleted from the database and updated in the view. See Figure 41

3.2.7 Tagging and Search

3.2.7.1 Taggable Item

The Acts-as-taggable-on gem handles all of our tagging functionality allowing tasks and records to be tagged both with integration into the search power of Searchkick, and for integration with the skill points system for assigning skill points and achievements to each user as they interact with the system. There few built-in tags defined in tag model, and tagging controller in section 3.3. Built in tags are defined in 3.2.7.2 of SRS. The user can also add self-defined tags. See class documentation section for tagging definition.

3.2.7.2 Filtration

Filtration is based on tagging and is used for delivering a few interfaces; these include Search, Deadline and Importance View. Section 5.6 shows this interface in detail. Filtration will be used in view layer and displaying interfaces by adding constraints defined by tags. See class documentation for full detail.

3.2.7.3 Search

As our project is powered by Rails and the Searchkick full text engine, 95% of this functionality has been offloaded to either Rails or the Searchkick environment. Each Rails model has all relevant fields mapped to the Searchkick full text search as either a limiting index (match this id, only this status, etc), or a text body. As required in SRS 3.2.7.1.4.

3.3 Class Documentation

This section contains the Class documentation for all Models and Module and Controllers of the Rails app. The models are core data entity classes and modules are complex helper functions that help to ensure cooperation of views and controllers. Controllers are simply functions that directly controlling the view of certain data types. The classes are listed in alphabetical order.

3.3.1 Model

Class: Campaign

Inherits:	Quest
Defined in:	app/models/campaign.rb

Overview

single table inheritance with the Quest model. That is, a Campaign is largely a scope on quest, a Quest.where('campaign_id = NULL')

Instance Method Details

```
- (Object) progress
```

Will generate a float value representing the completion percentage of the given campaign, for display on the campaign show page.

```
- (Object) search_data
```

Will define additional relational data for the purpose of deep searching, as specified through the searchkick gem

```
- (Object) to_link
```

Will define the link generated in the timeline when interacting with this model.

Class: Commit

Inherits:	Record
Defined in:	app/models/commit.rb

Method Summary

*Methods inherited from **Record***

`#assign_encounter`, `child_classes`, `inherited`, `#normalize_friendly_id`, `#to_link`, `#to_s`

Class: Encounter

Inherits:	ActiveRecord::Base
Defined in:	app/models/encounter.rb

Overview

The Encounter model is the central reference point for all activity within a given time block, as a tool for displaying timeline information, as well as for chunking the work-day into discreet units.

Instance Method Details

```
- (Object) before_save
```

Will allow the determination of whether the encounter is empty or not.

```
- (Object) clean
```

Will remove encounter if empty of data

```
- (Object) close
```

Will set the end time for the encounter when called, using the current time on the server as the end point.

Class: GithubRepo

Inherits:	ActiveRecord::Base
Defined in:	app/models/github_repo.rb

Instance Method Details

```
- (Object) to_link
```

Will define the link generated in the timeline when interacting with this model.

Class: Group

Inherits:	ActiveRecord::Base
Defined in:	app/models/group.rb

Overview

timelines

Instance Method Details

```
- (Object) demote(user)

- (Object) leave(user)
```

Will leave the group while preventing orphan groups. If the user is the last admin, the function will automatically promote the oldest member to group admin before leaving. If the user is the last member in the group, the function will delete the group and associated campaigns from the database

Class: GroupRound

Inherits:	Round
Defined in:	app/models/group_round.rb

Overview

An event related to manipulating a group

Method Summary

Methods inherited from Round

`create_event`, `#related_link`, `#related_obj`

Class: Image

Inherits:	Record
Defined in:	app/models/image.rb

Overview

A record that will represent a file uploaded by the user. The path field will store the location of the file on the server to allow retrieval

Instance Attribute Summary

Attributes inherited from Record

`#encounter`, `#quest`, `#questname`

Method Summary

Methods inherited from Record

`#assign_encounter`, `child_classes`, `inherited`, `#normalize_friendly_id`, `#to_link`, `#to_s`

Class: Issue

Inherits:	Quest
Defined in:	app/models/issue.rb

Class Method Summary (collapse)

- `+ (Object) model_name`

Methods inherited from Quest

`#campaign?`, `#descendants`, `#get_campaign`, `#is_ancestor`, `meta_search`, `#search_data`, `#set_status`, `#to_link`, `#to_s`

Class: Link

Inherits:	Record
Defined in:	app/models/link.rb

Overview

A record that will represent a web reference. The path field will hold the url of the website in question

Instance Attribute Summary

Attributes inherited from Record

`#encounter`, `#quest`, `#questname`

Method Summary

Methods inherited from Record

`#assign_encounter`, `child_classes`, `inherited`, `#normalize_friendly_id`, `#to_link`, `#to_s`

Class: Note

Inherits:	Record
Defined in:	app/models/note.rb

Overview

A record that will contain a simple text note

Instance Attribute Summary

Attributes inherited from Record

`#encounter`, `#quest`, `#questname`

Method Summary

Methods inherited from Record

`#assign_encounter`, `child_classes`, `inherited`, `#normalize_friendly_id`, `#to_link`, `#to_s`

Class: Notification

Inherits:	ActiveRecord::Base
Defined in:	app/models/notification.rb

Instance Method Details

```
- (Object) admin
```

This function will return the user who authorized the action listed in the notification, for the purpose of record keeping for group management.

```
- (Object) link_models(source, target)
```

This function will store any 2 arbitrary models as source and target on a newly created notification

Parameters:

- **`source`** (`Object`) —

  originating side of request, often a user or group

- **target** (Object) —

  receiving side of request, will always be a user

`- (Object) source`

Source will be any arbitrary rails Model, stored by source_type and source_id in the database. this function will reconstruct the actual source object from those fields.

`- (Object) target`

Target will be any arbitrary rails Model, stored by source_type and source_id in the database. this function will reconstruct the actual target object from those fields.

Class: Power

Inherits:	Object
Includes:	Consul::Power
Defined in:	app/models/power.rb

Constructor Details

`- (Power) initialize(user)`

Will return a new instance of Power

Class: Quest

Inherits:	ActiveRecord::Base
Extended by:	FriendlyId
Defined in:	app/models/quest.rb

Overview

A specific, actionable task to complete in given project. Shares a table through STI with Campaign

Direct Known Subclasses

Campaign, Issue

Class Method Details

`+ (Object) meta_search(query)`

Will generate a list of campaigns related to quests found for the given query

Instance Method Details

`- (Boolean) campaign?`

Will report whether this quest is also a campaign

Returns:

- (`Boolean`)

`- (Object) descendants`

`- (Object) get_campaign`

Will report whether this quest is also a campaign

`- (Object) is_ancestor(quest)`

Will report whether the provided quest is an ancestor of the current quest

Parameters:

- **quest** (`Quest`)

`- (Object) search_data`

Will define additional relational data for the purpose of deep searching, as specified through the searchkick gem

`- (Object) set_status`

Will ensure that a new quest has a status

`- (Object) to_link`

Will define the link generated in the timeline when interacting with this model.

`- (Object) to_s`

Class: QuestRound

Inherits:	Round
Defined in:	app/models/quest_round.rb

Overview

An event related to manipulating a quest

Method Summary

Methods inherited from ***Round***

`create_event`, `#related_link`, `#related_obj`

Class: Record

Inherits:	ActiveRecord::Base
Extended by:	FriendlyId
Defined in:	app/models/record.rb

Overview

Record base model for Link, Note and Image

Direct Known Subclasses

Commit, Image, Link, Note

Instance Attribute Details

`- (Object) encounter`

Returns the value of attribute encounter

`- (Object) quest`

Returns the value of attribute quest

`- (Object) questname`

Returns the value of attribute questname

Class Method Details

`+ (Object) child_classes`

Will report the list of the child classes of Record

`+ (Object) inherited(child)`

Will generate a list of child classes based on Rails inheritance

Instance Method Details

`- (Object) assign_encounter(user)`

Will assign the last encounter the user created to the current record being created, to ensure it passes validation

`- (Object) normalize_friendly_id(string)`

Will generate a truncacted friendly_id string

`- (Object) to_link`

Will define the link generated in the timeline when interacting with this model.

`- (Object) to_s`

Class: Round

Inherits:	ActiveRecord::Base
Defined in:	app/models/round.rb

Overview

Round stores a single action in an encounter, for exp tracking and timeline display purposes. Using STI, the model stores a related quest, campaign, skill_point, or group to store the related ActiveRecord object

Direct Known Subclasses

GroupRound, QuestRound, UserRound

Class Method Details

`+ (Object) create_event(model, operation, campaign)`

Will record the event processed by the controller as a round, for later display in a timeline.

Parameters:

- **`model`** (`ActiveRecord::Base`) —

  the object being operated on

- **`operation`** (`String`) —

  the controller operation being performed

- **`campaign`** (`Campaign`) —

  the campaign related to the event, if any

Raises:

- (`ArgumentError`)

Instance Method Details

- (`Object`) **`related_link`**

Will return the link for the original object

- (`Object`) **`related_obj`**

Will restore the original object passed in as model in create event

Class: Tag

Inherits:	ActiveRecord::Base
Defined in:	app/models/tag.rb

Overview

Tags as managed through the acts-as-taggable-on gem

Class: Timer

Inherits:	ActiveRecord::Base
Defined in:	app/models/timer.rb

Overview

Timer class for a timer configuration on a per user basis

Instance Method Details

- (`Boolean`) **`get_state`**

Will return the current state of the timer

Returns:

- (`Boolean`) —

  state of the timer, true for enabled

- (`Object`) **`init`**

Will set default value for each value in the setting table

- (`Object`) **`set_state`**`(state)`

Will set the current state of the timer

Parameters:

- **state** (Boolean) —

  state to set to for the timer

Class: User

Inherits:	ActiveRecord::Base
Defined in:	app/models/user.rb

Instance Attribute Details

```
- (Object) login
```

Returns the value of attribute login

Class Method Details

```
+ (Object) addGroup(groupName, isAdmin)

+ (Object) find_first_by_auth_conditions(warden_conditions)
```

Will check for authentication based on the Devise library

Instance Method Details

```
- (Object) add_group_as_admin(group)
```

Will simultaneously add a user to a group and promote them to admin

```
- (Object) add_group_as_member(group)
```

Will add a user to a group with member privileges

```
- (void) deleteRequest
```

This method returns an undefined value.

Will request account deletion

```
- (bool) expired?
```

Will check if user session is expired

Returns:

- (bool) —

Returns true if user session is expired

```
- (Object) github
```

Will return an authentication token for github access

```
- (Object) groups_less_wrapper
```

Will return all groups less the wrapper group

```
- (Object) groups_where_admin
```

Will return only groups where the user is admin, excluding their private wrapper group, preventing manipulation of that permanant group

```
- (Object) groups_where_member
```

Will return only groups where the user does not have admin privileges

```
- (encounter) last_encounter
```

Will return the last encounter for the user

Returns:

- (`encounter`) —

  last encounter

```
- (Object) new_user_setup
```

Will ensure a user has all associated models created Will create a timer model to save the state of their workflow Will create a wrapper group to manage privacy of their campaigns Will create a user_config to manage their settings Will create a default campaign as a catch-all to-do list

```
- (ActiveRecord::Relation) pending_deadlines(days_in_future = 7)
```

Will generate a list of quests from all campaigns in a user's groups, with the deadline closer to the current time than the specified number of days.

Parameters:

- **`days_in_future`** (`integer`) *(defaults to: `7`)* —

  Max days out to get deadlines

Returns:

- (`ActiveRecord::Relation`) —

  List of relevant quests

`- (Object) promote_in_group(group)`

Will make a current group member into an administrator

`- (Object) remove_group(group)`

Will remove a group from the user's membership list

`- (collection) timeline(end_time = Time.now)`

Will generate a paginated collection encounters for the user end_time will default to the current time.

Parameters:

- **`end_time`** (`datetime`) *(defaults to: `Time.now`)* —

  last time included in list of encounters

Returns:

- (`collection`) —

  first page of encounters preceeding end_time

Class: UserConfig

Inherits:	ActiveRecord::Base
Defined in:	app/models/user_config.rb

Class: UserMailer

Inherits:	ActionMailer::Base
Defined in:	app/mailers/user_mailer.rb

Overview

Class responsible for generating email for registered users

Instance Method Details

`- (Object) welcome_email(user)`

Sends the default welcome email for email validation of new users

Class: UserRound

Inherits:	Round
Defined in:	app/models/user_round.rb

Overview

An event related to manipulating a user

Method Summary

Methods inherited from ***Round***

`create_event`, `#related_link`, `#related_obj`

3.3.2 Module

Module: ApplicationHelper

Defined in:	app/helpers/application_helper.rb

Instance Method Details

```
- (Object) flash_class(level)
```

when generating user alerts, sets the alert dialog to the right value

```
- (Object) new_record_link(active_quest)
```

Will generate a modal dialog to add a record to a quest, disabling the button if there is no active quest for the user as an error handling setting

Parameters:

- **`active_quest`** (`Quest`) —

  the current active quest

```
- (Object) render_timer_button
```

Will display the timer to the headerbar, as configured based on the user's user_config values

```
- (Object) render_timer_mode
```

Will display the timer mode toggle modal dialog, to switch current timer mode without effecting permanant user settings

```
- (Object) trunc(string, length = 25)
```

Will truncate a given string to the specified length, visually displaying the effect with an elipsis representing the area removed

Parameters:

- **`string`** (`String`) —

  the string to be truncated

- **`length`** (`Integer`) *(defaults to: `25`)* —

  the size to reduce the string to, defaults to 25

Module: GithubHelper

Included in:	QuestsController, RecordsController, UsersController
Defined in:	app/helpers/github_helper.rb

Instance Method Details

```
- (Object) close_issue(username, projectname, issue_no)
```

Will close Issue matching a locally closed quest

Parameters:

- **`username`** (`String`) —

  username of the project

- **`projectname`** (`String`) —

  project name on github

- **`issue_no`** (`integer`) —

  number of issue on Github

```
- (Object) del_project(username, projectname)
```

Will delete Project From QTD

Parameters:

- **`username`** (`String`) —

  username of the project

- **`projectname`** (`String`) —

  project name on github

```
- (Object) github_init(username, projectname)
```

Will initiate a connection to github based on a user's security token

```
- (Object) github_update_all_projects
```

Will synchronize accross all projects associated with the current user

```
- (Object) initial_import(username, projectname)
```

Will import a project to QTD Note: this should be run only when first time import is initiated

Parameters:

- **`username`** (`String`) —

  username of the project

- **`projectname`** (`String`) —

  project name on github

```
- (Object) list_branches(username, projectname)
```

Will list all branches for the specified project under a specified user using the permissions of the active user's github authentication

Parameters:

- **`username`** (`String`) —

  username of the project

- **`projectname`** (`String`) —

  project name on github

```
- (Commits) list_commits(username, projectname, encounter, campaign)
```

Will get commits from a project using the permissions of the active user's github authentication

Parameters:

- **`username`** (`String`) —

  username of the project

- **`projectname`** (`String`) —

project name on github

- **encounter** (Encounter) —

  user's currently active encounter

- **campaign** (Campaign) —

  the local campaign linked to the remote project

Returns:

- (Commits) —

  A list of commits found for the specified project

```
- (Hash) list_issues(username, projectname, encounter, campaign)
```

Will list all issues for the specified project using the permissions of the active user's github authentication

Parameters:

- **username** (String) —

  username of the project

- **projectname** (String) —

  project name on github

- **encounter** (Encounter) —

  user's currently active encounter

- **campaign** (Campaign) —

  the local campaign linked to the remote project

Returns:

- (Hash) —

  The full list of issues

```
- (Hash) list_projects
```

Will list all projects attached to the user's github credentials

Returns:

- (`Hash`) —

  The full list of projects

`- (Github) login`

Will generate the authentication token used for manipulating API activity

Returns:

- (`Github`) —

  Github Session

`- (bool) login?`

Will check if login is sucessful

Returns:

- (`bool`) —

  Logged in or not

`- (Object) open_issue(username, projectname, quest)`

Will open an Issue on GitHub from a locally created quest

Parameters:

- **`username`** —

  Github User Name

- **`projectname`** —

  Github Project Name

`- (Object) push_comment(username, projectname, issue_no, comment)`

Will push local Notes as comments to Github Issue

Parameters:

- **`username`** (`String`) —

  username of the project

- **projectname** (`String`) —

  project name on github

`- (Object) update_project(username, projectname)`

Will synchronize Issues and Commits to the Github repo associated with it

Parameters:

- **username** (`String`) —

  username of the project

- **projectname** (`String`) —

  project name on github

Module: GroupHelper

Defined in:	app/helpers/group_helper.rb

Instance Method Details

`- (String) invite_user_path`

Will return a path to the current group, with the invite_user action

Returns:

- (`String`) —

  url to invite user

`- (Html) render_add_member`

Will generate button to add a user to the group, presuming they have permissions to do so, given the current user's permissions and the permissions of the target user

Returns:

- (`Html`) —

  add member button's html

`- (Html) render_demote(user_id)`

Will generate button to demote specified user for the group, presuming they have permissions to do so, given the current user's permissions and the permissions of the target user

Parameters:

- **`user_id`** (`Integer`) —

  the id of the target user

Returns:

- (`Html`) —

  demote button's html

`- (Html) render_promote(user_id)`

Will generate button to promote specified user for the group, presuming they have permissions to do so, given the current user's permissions and the permissions of the target user

Parameters:

- **`user_id`** (`Integer`) —

  the id of the target user

Returns:

- (`Html`) —

  promote member button's html

`- (Html) render_quest_count(campaign, status)`

Will generate an icon listing the number of quests matching a current status within the specified campaign

Parameters:

- **`campaign`** (`Campaign`) —

  the campaign to search

- **`status`** (`String`) —

  the string to search for

Returns:

- (`Html`) —

quest count html

```
- (Html) render_remove(user_id)
```

Will generate button to remove specified user for the group, presuming they have permissions to do so, given the current user's permissions and the permissions of the target user

Parameters:

- **`user_id`** (`Integer`) —

  the id of the target user

Returns:

- (`Html`) —

  remove member button's html

```
- (Html) render_role(group, user)
```

Will generate the role text based on the QTD group membership type for the user

Parameters:

- **`group`** (`Group`) —

  the group in question

- **`use`** (`User`) —

  the user in question

Returns:

- (`Html`) —

  render member role html

Module: JsonGenerator

Defined in:	app/concerns/json_generator.rb

Overview

Module for Generating JSON for displaying with Javascript

Defined Under Namespace

Modules: EncounterModule, QuestModule

Module: JsonGenerator::EncounterModule

Includes:	ActionView::Helpers::DateHelper
Included in:	EncountersController
Defined in:	app/concerns/json_generator.rb

Overview

JsonGenerator Module for Encounter

Instance Method Details

```
- (JSON) generateTree(rounds, campaign_id)
```

Will generate a tree JSON for a user's encounter This will allow a proper timeline display

Parameters:

- `user` (`User`) —

  User to generate encounter JSON for

Returns:

- (`JSON`) —

  JSON formatted data

Module: JsonGenerator::QuestModule

Included in:	CampaignsController, QuestsController, RecordsController
Defined in:	app/concerns/json_generator.rb

Overview

Module for Quest and Campaigns

Instance Method Details

```
- (JSON) generateCampaignTree(campaign)
```

Will generate a Campaign tree JSON for a campaign This will allow a proper timeline display

Parameters:

- **`campaign`** (`Campaign`) —

  Campaign to generate JSON for

Returns:

- (`JSON`) —

  JSON formatted tree data

`- (JSON) generateChildTree(quest)`

Will recursively generate json for all quests underneath the specified quest

Parameters:

- **`quest`** (`Quest`) —

  Quest to generate JSON

Returns:

- (`JSON`) —

  JSON formatted tree data

`- (JSON) generateQuestTree(quest)`

Will generate a Quest tree JSON for a quest This will allow a proper timeline display

Parameters:

- **`quest`** (`Quest`) —

  Quest to generate JSON

Returns:

- (`JSON`) —

  JSON formatted tree data

Module: RoundHelper

Includes:	TimerHelper
Included in:	CampaignsController, QuestsController, RecordsController, UsersController
Defined in:	app/helpers/round_helper.rb

Instance Method Details

```
- (Object) create_round(model, operation, campaign)
```

Will create a round linking the specified model, campaign, and recording the operation the round represents

Parameters:

- **`model`** (`ActiveRecord::Base`) —

  A rails object reference by the activity

- **`campaign`** (`Campaign`) —

  The campaign the round is related to

- **`operation`** (`String`) —

  The controller action performed on the model

Class: NotificationService

Inherits:	Object
Defined in:	app/services/notification_service.rb

Overview

NotificationService class

Used for notifying users with emails about different events

Ex.

```
NotificationService.new.new_issue(issue, current_user)
```

Instance Method Details

```
- (Object) mailer (protected)
```

Will define how to send email notifications

```
- (Object) new_group_member(users_group)
```

Will notify user being a new member of a group

```
- (Object) new_user(user)
```

Will notify new user with email after creation

```
- (Object) project_was_moved(project)
```

Will notify user if project was moved

```
- (Object) update_group_member(users_group)
```

Will notify user if there is an update in the members of a group

Module: SearchesHelper

Defined in:	app/helpers/searches_helper.rb

Defined Under Namespace

Classes: LinkRenderer

Instance Method Details

```
- (String) render_result_count(results)
```

Will return count for search result of the result

Parameters:

- **`result`** (`String`) —

  class item

Returns:

- (`String`) —

  count of search result

```
- (String) render_row_class(result)
```

Will return css class for the result

Parameters:

- **`result`** (`String`) —

  class name

Returns:

- (`String`) —

  css class

`- (Html) render_status_tag(result)`

Will format the status for results of the quest and campaign classes

Parameters:

- **`result`** (`String`) —

  class object

Returns:

- (`Html`) —

  status tag for quest/campaigns

Class: SearchesHelper::LinkRenderer

Inherits:	WillPaginate::ActionView::LinkRenderer
Defined in:	app/helpers/searches_helper.rb

Instance Method Details

`- (Object) gap (protected)`

Will define a list item based on the pagination

`- (Object) html_container(html) (protected)`

Will create a paginated container for specified html

Parameters:

- **`html`** (`String`) —

an html-formatted string

```
- (Object) link(text, target, attributes = {}) (private)

- (Object) next_page (protected)
```

Will generate a button for the next page based on current pagination data

```
- (Object) page_number(page) (protected)
```

Will link to the specified page, unless the page is the same as the current page

Parameters:

- **page** (`Integer`) —

  the page to find the link

```
- (Object) previous_or_next_page(page, text, classname) (protected)
```

Module: TimerHelper

Included in:	RoundHelper, TimersController
Defined in:	app/helpers/timer_helper.rb

Overview

Helper for handling timer

Instance Method Details

```
- (Hash) extend_timer
```

Will extend the current time on the timer

Returns:

- (`Hash`) —

  hash containing extended time

```
- (Hash) get_break_time
```

Will get the default time for a short break from user configuration

Returns:

- (`Hash`) —

  hash containing short break time

```
- (hash) get_current_time
```

Will get remaining time on the timer

Returns:

- (`hash`) —

  data containing current remaining time and timer mode

```
- (Hash) get_setting_time
```

Will get default time from user configuration settings

Returns:

- (`Hash`) —

  hash containing setting time

```
- (Hash) get_timer_state
```

Will get current state of the timer

Returns:

- (`Hash`) —

  hash containing timer mode

```
- (Object) reset_timer
```

Will reset the timer

```
- (Hash) restart_timer
```

Will try to reset time if in auto mode

Returns:

- (`Hash`) —

Default time for timer and timer mode

`- (Object) start_break`

Will start a short break

`- (Object) start_timer`

Will start timer and create an encounter if neccessary

`- (Object) stop_timer`

Will pause the timer

Parameters:

- **`current_time`** (`Integer`) —

  Current countdown remaining time in seconds

3.3.3 Controllers

Class: ApplicationController

Inherits:	ActionController::Base
Includes:	Consul::Controller
Defined in:	app/controllers/application_controller.rb

Overview

Default controller in Rails, from which all other users inherit

Direct Known Subclasses

CampaignsController, EncountersController, GroupsController, NotificationsController, PrioritiesController, QuestsController, RecordsController, SearchesController, SkillPointsController, SkillsController, TimersController, UsersController, WelcomeController

Instance Method Details

`- (Object) check_password_expiration (protected)`

Will ensure that password timeouts are followed up with prompts for logging back into the program

`- (Object) configure_permitted_parameters (protected)`

Will limit parameters to those valid for devise user control, to prevent parameter injection from influencing the security of the site

`- (Object) full_search(query) (protected)`

Will generate aggregate search results from recor and quest searches against a specified query by the Searchkick gem

Parameters:

- **`query`** (`String`) —

  search string specified in SQL-like format specified

`- (Object) load_user (protected)`

Once a user is signed in, the application will use this to ensure that the header displays the correct information for the active task of the current user

`- (Object) record_autocomplete (protected)`

Will generate JSON results for a record search for use in generating autocomplete forms

Class: CampaignsController

Inherits:	ApplicationController
Includes:	JsonGenerator::QuestModule, JsonGenerator::TimelineModule, RoundHelper
Defined in:	app/controllers/campaigns_controller.rb

Instance Method Details

`- (Object) campaign_params`

Will restrict parameters to those formally specified

Parameters:

- **`description`** (`String`) —

  Campaign's description

- **`name`** (`String`) —

  Campaign's name

`- (String) campaign_timeline_path(campaign)`

Will generate a custom helper path for timeline

Parameters:

- **`id`** (`Integer`) —

  Campaign

Returns:

- (`String`)

- (`Html`) **`create`**

Will save a new campaign as generated by the new action

Parameters:

- **`campaign_params`** (`campaign_params`) —

  field input from creation page

Returns:

- (`Html`) —

  redirect back to the new campaign page

- (`Html`) **`destroy`**

Will delete a campaign and all the quests and records that are associated

Parameters:

- **`id`** (`Integer`) —

  Campaign's id

Returns:

- (`Html`) —

  redirect back to campaigns index page

- (`Html`) **`edit`**

Will edit an existing campaign

Parameters:

- **`id`** (`Integer`) —

  Campaign's id

Returns:

- (`Html`) —

  Campaign editing page

- (`File`) **export**

Will export a Campaign to a QTD specific format

Parameters:

- **`id`** (`Integer`) —

  Campaign's id

- **`type`** (`String`) —

  File format to be exported

Returns:

- (`File`) —

  downloadable file

- (`JSON`) **get_campaign_timeline**

Will get the current timeline for the campaign

Parameters:

- **`campaign`** (`Campaign`) —

  Campaign

Returns:

- (`JSON`) —

  JSON of the timeline details

- (`JSON`) **getTree**

Will Generate JSON for tree view

Parameters:

- **`id`** (`Integer`) —

  Campaign's id

Returns:

- (JSON) —

  campaign's information in JSON format

- (Object) import

Will import a QTD specific format Campaign to generate a campaign

Parameters:

- **path** (String) —

  file path

- (Html) index

Will show all of user's perosonal campaigns

Returns:

- (Html) —

  the index page for all campaign

- (Html) new

Will create a new campaign

Returns:

- (Html) —

  New campaign page

- (Html) show

Will show the details of a campaign

Parameters:

- **id** (Integer) —

  Campaign's id

Returns:

- (Html) —

the campaign detail with that id

- (Html) **timeline**

Will generate a modal display containing a timeline for the specified campaign

Parameters:

- **`id`** (`Integer`) —

  Campaign'id to be viewed

Returns:

- (`Html`) —

  partial view of the timeline

- (Html) **update**

Will update a campaign and save the changes

Parameters:

- **`campaign_params`** (`campaign_params`) —

  field input from creation page

Returns:

- (`Html`) —

  redirect back to campaigns index page

Class: EncountersController

Inherits:	ApplicationController
Includes:	JsonGenerator::EncounterModule
Defined in:	app/controllers/encounters_controller.rb

Instance Method Details

- (Object) **get_user_timeline**

Will gather the data and render the timeline for the user

- (JSON) **getTree**

Will show the tree view data of an encounter

Parameters:

- **`id`** (`Integer`) —

  Encounter's id

Returns:

- (`JSON`) —

  the encounter's tree data in JSON format

- (`Html`) **index**

Will show a timeline generated from a user's encounters

Returns:

- (`Html`) —

  the index page for all encounter

- (`Html`) **new**

Will create a new encounter

Returns:

- (`Html`) —

  New encounter page

- (`Object`) **not_found**

Will forcibly raise an "object not found" error

Raises:

- (`ActionController::RoutingError`)

- (`Html`) **show**

Will show the details of an encounter

Parameters:

- **`id`** (`Integer`) —

  Encounter's id

Returns:

- (`Html`) —

  the encounter detail with that id

`- (Object) start (private)`

Will start a new encounter for the user

Parameters:

- **`id`** (`Integer`) —

  Encounter's id

`- (Object) stop (private)`

Will stop an encounter and clear the currently active encounter from the user's session

Parameters:

- **`id`** (`Integer`) —

  Encounter's id

Class: GroupsController

Inherits:	ApplicationController
Includes:	JsonGenerator::TimelineModule
Defined in:	app/controllers/groups_controller.rb

Overview

Controller for the group page

Instance Method Details

`- (Object) accept_user`

Will accept a user to the group

`- (Object) create`

Will take the specified parameters and save a new group

`- (Object) demote`

Will remove a user from admin group

`- (Object) group (private)`

Will set the current group, if not already set

Parameters:

- `id` (`Integer`)

`- (Object) group_params`

Will restrict parameters to those formally specified

`- (Object) index`

Will display a list of all groups for the current user

`- (Object) invite_user`

Will generate a notification for a user, inviting them to the group. Only admins will be able to access this function

`- (Object) join`

Will generate a notification for admins of the group. Any admin will then be able to approve the request, adding the requesting user to the group

`- (Object) kick`

Will remove another user from the group specified, presuming the current user has admin privileges for the group

Parameters:

- `id` (`Integer`) —

  The id of the group

- `user_id` (`Integer`) —

  The id of the group

`- (Object) leave`

Will remove the user from the group specified

Parameters:

- `id` (`Integer`) —

The id of the group

`- (Object) new`

Will allow the user to create a new group

`- (Object) promote`

Will promote a user to admin group

`- (Object) show`

Will display the group page for the specified group, presuming the user has access rights granted to them.

`- (Object) timeline`

Class: NotificationsController

Inherits:	ApplicationController
Defined in:	app/controllers/notifications_controller.rb

Instance Method Details

`- (Object) group_invite`

Will allow group to invite new user and notify the user

`- (Object) group_kick`

Will allow group to kick an user and notify the user

`- (Object) group_promote`

Will promote a user to group admin

Generated on Tue Apr 29 20:35:43 2014 by yard 0.8.7.3 (ruby-2.1.1).

Class: QuestsController

Inherits:	ApplicationController
Includes:	GithubHelper, JsonGenerator::QuestModule, RoundHelper
Defined in:	app/controllers/quests_controller.r

Instance Method Details

`- (Html) create`

Will save new quest

Parameters:

- `quest_params` (`quest_params`) —

  field input from creation page

Returns:

- (`Html`) —

  redirect back to the new quest page

`- (Html) destroy`

Will delete quest and all the records it associated with

Parameters:

- `id` (`Integer`) —

  Quest's id

Returns:

- (`Html`) —

  redirect back to quest's campaign page

`- (Html) edit`

Will allow user to edit existing quest

Parameters:

- `id` (`Integer`) —

  Quest's id

Returns:

- (`Html`) —

  Quest's editing page

`- (JSON) getTree`

Will generate JSON to display tree relations for a quest

Parameters:

- **id** (`Integer`) —

  Quest's id

Returns:

- (`JSON`) —

  quest's information in JSON format

- (`Html`) index

Will show all of user's quests

Returns:

- (`Html`) —

  A list of quests of the user

- (`Html`) new

Will allow input for a new quest

Returns:

- (`Html`) —

  New quest page

- (`Object`) quest_params

Will restrict parameters to those formally specified

Parameters:

- **id** (`Integer`) —

  Quest's id

- **description** (`String`) —

  Quest's description

- **parent_id** (`Integer`) —

  Quest's parent quest id

- **campaign_id** (`Integer`) —

  Quest's campaign id

- **`user_id`** (`Integer`) —

  Owner's user_id

- **`status`** (`String`) —

  Quest's status

- **`importance`** (`Boolean`) —

  Quest importance check

- (`Object`) **set_active**

Will set a specified quest as user's current active quest

Parameters:

- **`id`** (`Integer`) —

  Quest's id

- (`Html`) **show**

Will show the detail of a quest

Parameters:

- **`id`** (`Integer`) —

  Quest's id

Returns:

- (`Html`) —

  Quest detail page with that id

- (`Html`) **update**

Will update quest and save the changes

Parameters:

- **`quest_params`** (`quest_params`) —

  field input from creation page

Returns:

- (`Html`) —

redirect back to quest's campaign page

Class: RecordsController

Inherits:	ApplicationController
Includes:	GithubHelper, JsonGenerator::QuestModule, RoundHelper
Defined in:	app/controllers/records_controller.rb

Overview

Controller for Record

Instance Method Details

- `(Html)` **`create`**

Will save a new record as specified by the params

Parameters:

- **`record_params`** (`record_params`) —

  field input from creation page

Returns:

- (`Html`) —

  redirect back to records index page

- `(Html)` **`destroy`**

Will delete the specified record

Parameters:

- **`id`** (`Integer`) —

  record id

Returns:

- (`Html`) —

  redirect back to record index page

- `(Html)` **`index`**

Will show all records belongs to a user

Returns:

- (Html) —

  All records belong to a user

- (Html) **modify**

Will modify a record

Parameters:

- **id** (Integer) —

  record id

Returns:

- (Html) —

  redirect back to record index page

- (Html) **new**

Will allow a user to define a new record for a quest

Parameters:

- **id** (Integer) —

  Record id

Returns:

- (Html) —

  New record creation page

- (JSON) **quest_autocomplete**

Will render auto complete for quests

Parameters:

- **query** (String) —

  Query to search for auto complete

Returns:

- (JSON) —

Search result in JSON

- (Object) **record_params**

Will restrict parameters to those formally specified

Parameters:

- **description** (String) —

  Record's description

- **encounter_id** (Integer) —

  Record's encounter_id

- **encounter** (Encounter) —

  Record's encounter

- (Html) **show**

Will show all details for a record

Parameters:

- **id** (Integer) —

  record id

Returns:

- (Html) —

  Record detail page based on the id

Class: TimersController

Inherits:	ApplicationController
Includes:	TimerHelper
Defined in:	app/controllers/timers_controller.rb

Overview

Controller for Timer related functions

Instance Method Details

- (JSON) **break_countdown**

Pause the timer and create a new timer for break

Returns:

- (JSON) —

  break_timer

`- (Object) change_mode`

Ajax set timer to auto/manual

Parameters:

- **mode** (String) —

  Mode to change to

`- (JSON) extend_break`

Extend the current break

Returns:

- (JSON) —

  Updated break_timer

`- (Object) extend_countdown`

Extend the timer duration for manual mode

Parameters:

- **current_time** (Integer) —

  current time in seconds

`- (JSON) get_time_current`

Get current remaining time on the time counter

Returns:

- (JSON) —

  the total remaining time or setting time

`- (JSON) get_time_setting`

Get user's default time length for an encounter

Returns:

- (`JSON`) —

  the default time length in seconds

- (`Object`) **`pause_countdown`**

Stop/Pause the timer and record the current remaining time

Parameters:

- **`current_time`** (`String`) —

  current remaining time on the timer in seconds

- (`String`) **`reset_countdown`**

Stop and reset the timer to the user's default time length Closes any last opened encounter

Returns:

- (`String`) —

  the default time length in seconds

- (`Object`) **`restart_countdown`**

Restart the timer when then timer reaches 0 Close any previous encounter, and start the timer again

- (`Object`) **`start_countdown`**

Start the timer countdown Open an encounter if there is not one currently active

Class: UsersController

Inherits:	ApplicationController
Includes:	GithubHelper, RoundHelper
Defined in:	app/controllers/users_controller.rb

Instance Method Details

- (`Object`) **`authorize`**

Will start authorization process for Github

```
- (Object) callback
```

Will Get Github Access Token from auhtorize

```
- (Object) destroy_avatar
```

Will destroy avatar

```
- (Object) getFriends
```

Will Get a list of user related friends

```
- (Object) getGroups
```

Will Get a list of user related groups

```
- (Object) github_authorize
```

Will try to authenticate to Github via github_access token

```
- (Object) github_background_jobs
```

Will load Github related background jobs

```
- (Object) github_callback
```

Will return github oauth token

```
- (Object) github_list
```

Will get a list of projects from authenticated user

```
- (Object) github_project_del
```

Will delete a project from listing of projects by the authenticated github user

```
- (Object) github_project_import
```

Will import a select github project

```
- (Object) github_update
```

Will update github repositories information

`- (Object) index`

Will show the user index page

`- (Object) settings`

Will show the user settings for timer, email, password and more

`- (Object) show`

Will display github project choices

`- (Binary) show_avatar`

Will define avatar by default value

Returns:

- (`Binary`) —

  image file

`- (Html) update_config`

Update timer config for the user

Parameters:

- **`id`** (`Integer`) —

  User config id

Returns:

- (`Html`) —

  User index page

`- (Object) user_config_params`

Will define allowed parameters for the users controller

`- (Object) user_config_path`

Will define the user configuration path

Class: WelcomeController

Inherits:	ApplicationController
Defined in:	app/controllers/welcome_controller.rb

Instance Method Details

`- (Object) index`

Will load the home page of the website

Class: SearchesController

Inherits:	ApplicationController
Defined in:	app/controllers/searches_controller.rb

Overview

Controller for Record

Instance Method Details

`- (Object) all_autocomplete`

auto complete search for record, quest and campaign

`- (Object) get_search_result(model, query) (private)`

Will get search result from defined query

`- (Html) index`

Will all search result for the spefied query

Parameters:

- `query` (`String`) —

  SQL like as specified by Searchkick

Returns:

- (`Html`) —

  All results

`- (Object) is_valid_model(model) (private)`

Will check if the parsed in model is valid

```
- (Object) quest_autocomplete
```

auto complete search for quest

```
- (Object) search_json(value) (private)
```

Class: SkillPointsController

Inherits:	ApplicationController
Defined in:	app/controllers/skill_points_controller.rb

Overview

Controller for Skill Points

Instance Method Details

```
- (Object) add(skillName, @user, skillpoint)
```

Add skill points to a user for a skill

Parameters:

- **`skillName`** (`String`)
- **`@user`** (`user`)
- **`skillpoint`** (`skill_point`)

```
- (Object) subtract(skillName, @user, skillpoint)
```

Subtract skill points to a user for a skill

Parameters:

- **`skillName`** (`String`)
- **`@user`** (`user`)
- **`skillpoint`** (`skill_point`)

4 Data Design

4.1 Database Schema and Rails Model Relationship

4.1.1 Rails Model Relationship

The Rails framework models the application into MVC, the classes are not able to be shown as traditional UML diagram, however, a similar relationship (semi-UML) can be draw as Figure 11 below.

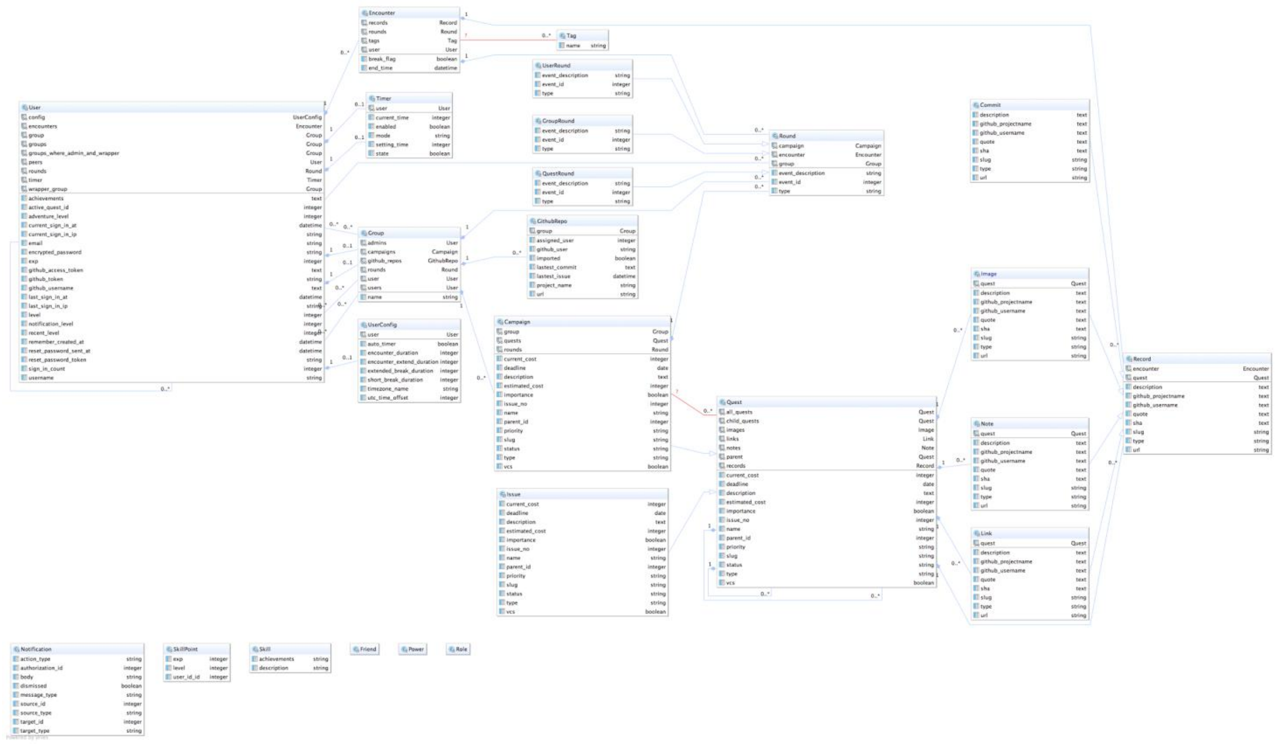


Figure 11 Rails Data Model

Due to the limitation of the document formatting, a vector image is stored at http://162.243.247.94/diagram.svg for better viewing purpose.

4.1.2 Database Table Listing

In addition to the Rails data model relationship another straightforward view for the database is the straight table listing. Database is stored in PostgreSQL and the tables will be standalone objects containing the data. Table relationship is not enforced by the database, thus there will be seemingly no relationship at the pure database design level. This relationship is managed and enforced by Rails and is shown in section 4.1.1. Real table design is shown in section 4.2 on a per model/table basis.

4.2 Data Model

4.2.1 User

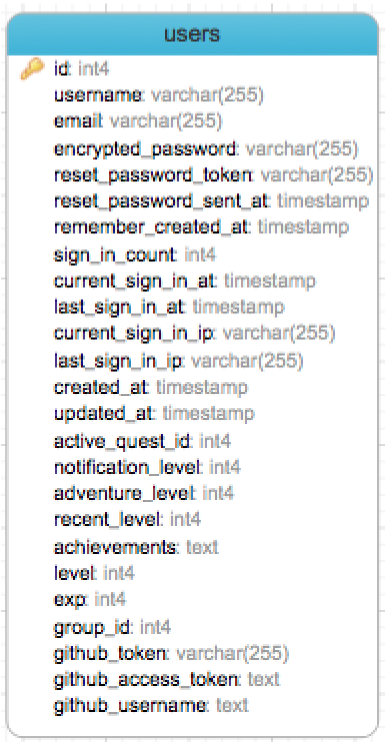
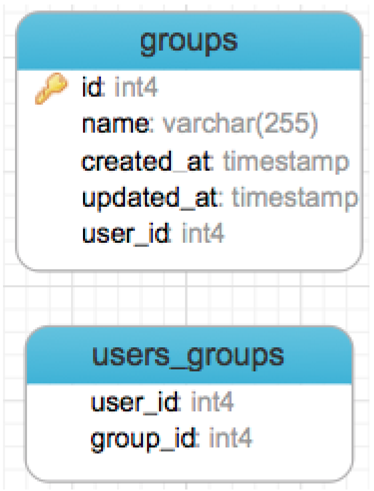

users
id: int4
username: varchar(255)
email: varchar(255)
encrypted_password: varchar(255)
reset_password_token: varchar(255)
reset_password_sent_at: timestamp
remember_created_at: timestamp
sign_in_count: int4
current_sign_in_at: timestamp
last_sign_in_at: timestamp
current_sign_in_ip: varchar(255)
last_sign_in_ip: varchar(255)
created_at: timestamp
updated_at: timestamp
active_quest_id: int4
notification_level: int4
adventure_level: int4
recent_level: int4
achievements: text
level: int4
exp: int4
group_id: int4
github_token: varchar(255)
github_access_token: text
github_username: text

Figure 12 User Table

User table will define a single user entity. group_id field is present because user will always belongs to a default group, to ensure privacy of their personal projects. This table also contains standard information for login session, login time, email, achievements, experiences, level and github information etc.

4.2.2 Group

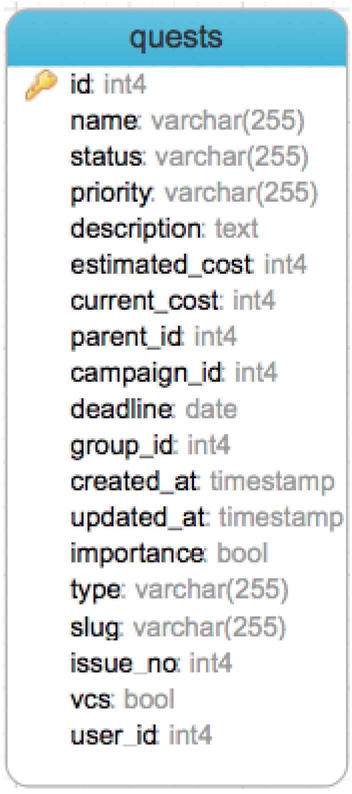

groups
id: int4
name: varchar(255)
created_at: timestamp
updated_at: timestamp
user_id: int4

users_groups
user_id: int4
group_id: int4

Figure 13 Groups and Users_groups Table

Group table will define a single group of entities. It contains the group information such as group name and which user the group belongs to. User group table defines a many to many relationship between a group and user using group_id and user_id as foreign key to both table. An identical admins_groups table stores a separate set of QTD Group Administrators for each group.

4.2.3 Campaign and Quest



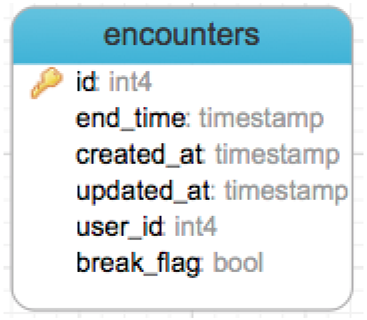
quests

id: int4
name: varchar(255)
status: varchar(255)
priority: varchar(255)
description: text
estimated_cost: int4
current_cost: int4
parent_id: int4
campaign_id: int4
deadline: date
group_id: int4
created_at: timestamp
updated_at: timestamp
importance: bool
type: varchar(255)
slug: varchar(255)
issue_no: int4
vcs: bool
user_id: int4

Figure 14 Quest Table

Quest table will define the model for either a Campaign/Project or a Quest/Task differentiated by type column in the Quest table. A quest or a campaign could have many children quest. Parent_id is the id of the immediate parent quest or campaign of the current quest. Campaign_id is the id of the campaign that the quest belongs in. User_id is the foreign key to user table, which defines the person assigned to complete that particular quest or campaign. VCS defines the VCS (whether it is a version control system quest, campaign) in which the campaign will pull data from. The estimated_cost and current_cost columns define both estimated effort and current effort for the quest, which both are positive number ranging from 0 to 100. The slug column is described later in slug table

4.2.4 Encounter and Round

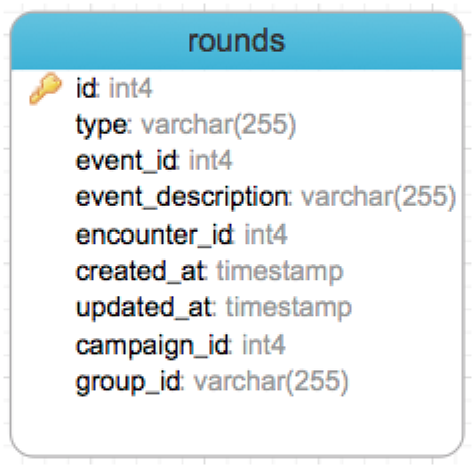


encounters

id: int4
end_time: timestamp
created_at: timestamp
updated_at: timestamp
user_id: int4
break_flag: bool

Figure 15 Encounter Table

Encounter table will define a model or entity for a single encounter. The end_time column defines the time when the encounter closed or ended, either by the system or by the user. User_id is the foreign key to user table which defines that encounter belongs to a certain user. Break_flag is the flag to indicate a certain encounter is created during a short break.



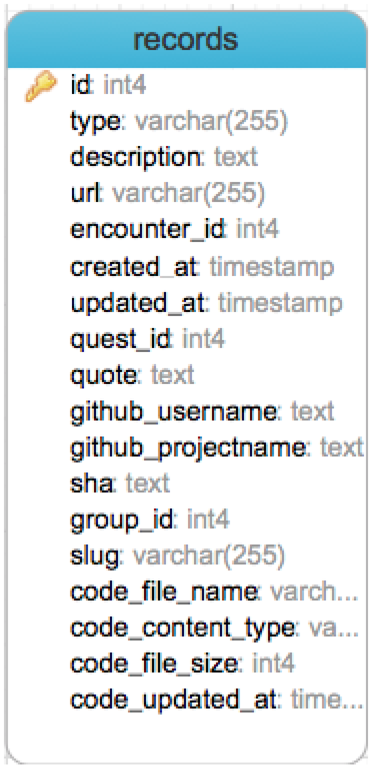
rounds

id: int4
type: varchar(255)
event_id: int4
event_description: varchar(255)
encounter_id: int4
created_at: timestamp
updated_at: timestamp
campaign_id: int4
group_id: varchar(255)

Figure 16 Round Table

Round table will define a model for round which will be used to record any user's activity regarding to quest, record, group and user's profile. The round table uses single table inheritance to capture different type of models through type column in event_id. Event_id will be the id of the model that the activity has occurred (e.g. a record has been added; hence the event_id will be the record_id). A round belongs to an encounter and also belongs to a campaign through encounter_id, campaign_id and group_id foreign keys.

4.2.5 Record



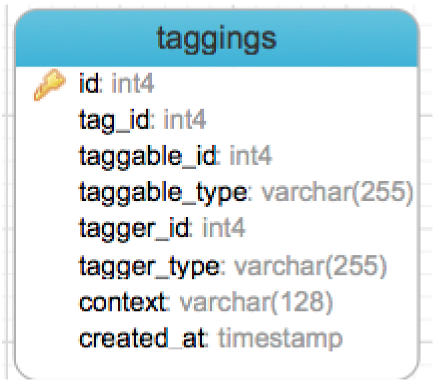
records

id: int4
type: varchar(255)
description: text
url: varchar(255)
encounter_id: int4
created_at: timestamp
updated_at: timestamp
quest_id: int4
quote: text
github_username: text
github_projectname: text
sha: text
group_id: int4
slug: varchar(255)
code_file_name: varch...
code_content_type: va...
code_file_size: int4
code_updated_at: time...

Figure 17 Record Table

Record table will define a model or entity for a single record. Record table utilizes single table inheritance to encapsulate a link, note, commit (VCS entry) and an image which are the subclasses for a record. The record_type field defines a string which let Rails knows that the type of record it is. The path column is used only by link and image, in which they are treated as url or file path respectively. The code_* columns are for image only. The slug is referenced later in slug table.

4.2.6 Tag



taggings

id: int4
tag_id: int4
taggable_id: int4
taggable_type: varchar(255)
tagger_id: int4
tagger_type: varchar(255)
context: varchar(128)
created_at: timestamp

Figure 18 Taggings Table

Tag table will define a single tag for tagging purpose. This table is generated and used by the Acts as Taggable gem for rails for tagging other models. The taggings will remember which tag is applied to which content. As Figure 19 shown, Tags table will keep all unique tags in it for overview of all tags created.

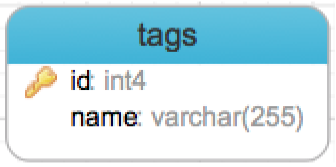

tags
id: int4
name: varchar(255)

Figure 19 Tags Table

4.2.7 Skill Point

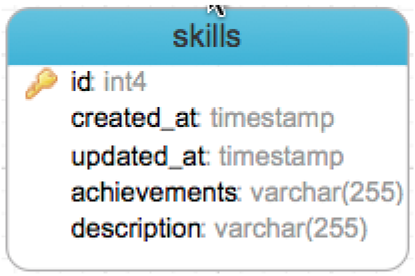

skills
id: int4
created_at: timestamp
updated_at: timestamp
achievements: varchar(255)
description: varchar(255)

Figure 20 Skill Table

Skill table will be the lookup table for skill details. It defines the detail of a particular skill, and the list of achievements of that skill in the achievement column.

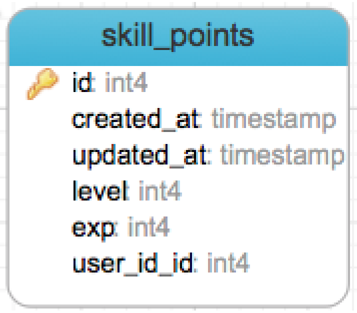
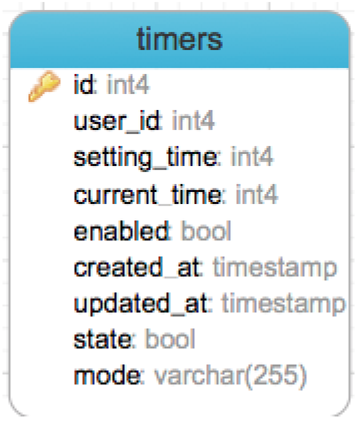

skill_points
id: int4
created_at: timestamp
updated_at: timestamp
level: int4
exp: int4
user_id_id: int4

Figure 21 Skill Point Table

Skill point table is a many to many relationship table in between skill and user table. A skill point belongs to a user, and the particular skill is referenced by the skill_id foreign key to the Skill lookup table for looking up the skills' details such as name or description. Level column is the user's current level on that particular skill and the experience column is a number from 0 to 100 defining the needed number in order to level-up.

4.2.8 Timer



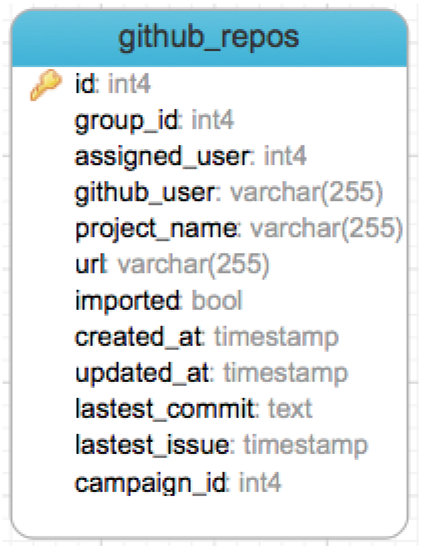
timers

```
id: int4
user_id: int4
setting_time: int4
current_time: int4
enabled: bool
created_at: timestamp
updated_at: timestamp
state: bool
mode: varchar(255)
```

Figure 22 Timer Table

Timer table will contain a single timer configuration model for a particular user. The user_id table is the foreign key which defines the one to one relationship between user and timer table. The setting_time and current_time columns are both the default time length and the current remaining time on the timer. All columns regarding unit of time is recorded and measured by seconds.

4.2.9 Github VCS Information



github_repos

```
id: int4
group_id: int4
assigned_user: int4
github_user: varchar(255)
project_name: varchar(255)
url: varchar(255)
imported: bool
created_at: timestamp
updated_at: timestamp
lastest_commit: text
lastest_issue: timestamp
campaign_id: int4
```

Figure 22 Github Repository Table

The github_repo table will contain the information of Github repositories such as github username and project name. The url is the repository web portal. The boolean imported column will indicate whether the project will be imported from scratch or updated incrementally.

Campaign_id is simply a foreign key referencing the imported project in Quests table. lates_commit and latest_issue are helpers during a update for the project.

4.2.10 Notification Table

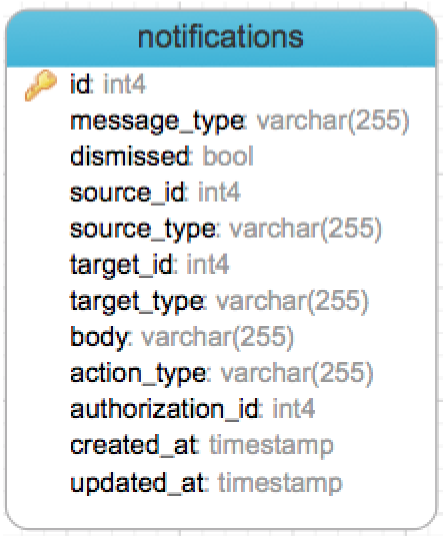


Figure 23 Notification Table

The Notification table will contain the information such as a user is invited to join a group. Message type column is the field that will determine what type of request of the notification is.

Dismissed boolean column will mark whether a message have been viewed and responded to by the target user. Source and target records from which object requested which target should reply. The body is the notification itself. Action type and authorization id ensure that each notification is unique and valid in terms of permissions.

4.2.10 Configurations and Backend jobs

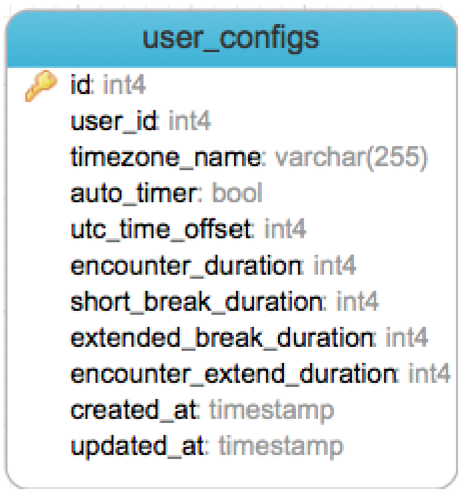
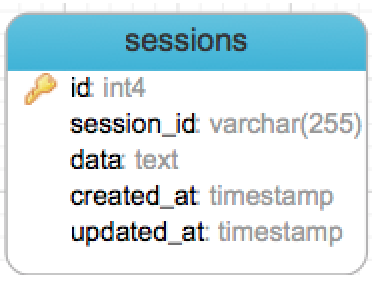


Figure 24 User Configs Table

Figure 24 shows the user_configs table, this table will contain configuration information of timer of a user. This includes time_zone setting, timer mode, and manual mode information like break time and extend options.

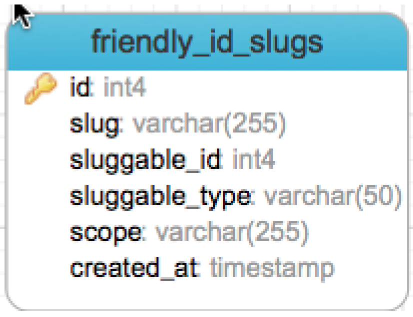


sessions

id	int4
session_id	varchar(255)
data	text
created_at	timestamp
updated_at	timestamp

Figure 25 Sessions Table

Figure 25 shows the sessions table, this will store users session information which is used to enforce session on logged in users and when should the system log them out.

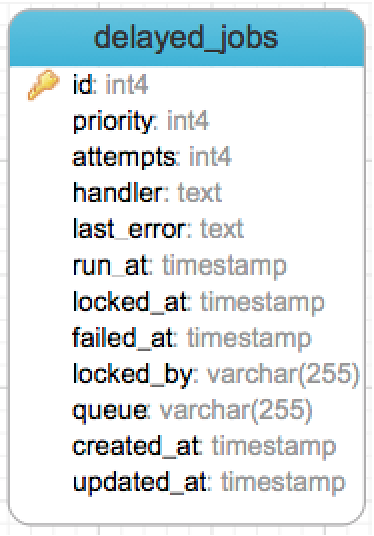


friendly_id_slugs

id	int4
slug	varchar(255)
sluggable_id	int4
sluggable_type	varchar(50)
scope	varchar(255)
created_at	timestamp

Figure 26 Friendly ID Slugs Table

The Friendly Id table will be a referenced table for all previous slugs in records and quests. This table will provide a helper for displaying user-friendly URL instead of displaying query like id number. This table is managed by the friendly-id gem.



delayed_jobs

id	int4
priority	int4
attempts	int4
handler	text
last_error	text
run_at	timestamp
locked_at	timestamp
failed_at	timestamp
locked_by	varchar(255)
queue	varchar(255)
created_at	timestamp
updated_at	timestamp

Figure 26 Delayed Jobs Table

The delayed jobs table will be the where all the background activities are scheduled. This is managed by the delayed_job gem. This table contains the working queue of all tasks need to run and the web application runs it in the background silently.

5. Design Rationale

5.1 Model-View-Controller

MVC separates the presentation, the action and the modeling of the domain. For a web-based application, it is beneficial to have clean and clearly decoupled model from the actual presentation and implementation. Because the model (Or in this case the database table) could be built and tested independently as it won't rely on both the controller and the view. On the other hand, separation of controller from the view will allow different team members to work on both view and controller at the same time with minimal conflict.

5.2 Single Table Inheritance

STI reduces the number of tables by consolidating multiple tables with similar behavior into one. Besides, STI supports polymorphism by having the type column, and Rails translate the objects very well due to Ruby's dynamic typing behavior. The performance of querying will be faster than regular segregated table approach since the related data is in one table. It generally works well for small and simple object hierarchy.

6. Requirement Matrix

Requirement ID	Section	Description
3.2.1.1.1	3.2.2.1	GitHub Integration
3.2.1.1.2	3.2.2.1	Modify / Remove GitHub Link
3.2.1.1.3	3.2.2.2	Github Synchronization
3.2.2.1.1	3.2.1.1	User Login
3.2.2.1.2	3.2.1.2	User Logout
3.2.2.2	3.2.1.3	User Registration
3.2.2.3	3.2.1.8	QTD Member Privileges
3.2.2.4	4.2.1	User Creation
3.2.2.5	3.2.1.4	User Modification
3.2.2.6	3.2.1.5	User Deletion
3.2.2.7	5.2.3	User Performance Metrics
3.2.2.8	5.2.1	User Profile
3.2.2.9.1	3.2.1.9, 5.2.4	User’s Friend
3.2.2.9.2	5.2.4	User’s Group
3.2.2.10	TC15	User security
3.2.2.11	5.2.2	User’s general settings
3.2.3.1	3.2.3.1	Add project
3.2.3.2	3.2.3.2	Modify existing project
3.2.3.3	3.2.3.3	Remove existing project
3.2.3.4	3.2.3.4	Import project
3.2.3.5	3.2.3.5	Export project
3.2.3.6.1	3.2.3.6.1	Add task
3.2.3.6.2	3.2.3.6.2	Modify existing task

3.2.3.6.3	3.2.3.6.3	Remove existing task
3.2.3.6.4	5.5	Task overview
3.2.4.1.1, 3.2.4.1.2	3.2.4.1	Group creation
3.2.4.1.3	3.2.4.4.3	Multiple Group Administrator
3.2.4.1.4	3.2.4.1	Group deletion
3.2.4.1.5	4.2.2	Group name
3.2.4.2	5.3.2	Group Information display
3.2.4.3.1	3.2.4.3	QTD Member accepts invitation
3.2.4.3.2	3.2.4.3	QTD Member Leaving Group
3.2.4.3.3	3.2.4.3	QTD Member Privilege
3.2.4.4.1	3.2.4.4.1	Member invitation
3.2.4.4.2	3.2.4.4.2	Kicking member
3.2.4.4.3	3.2.4.4.3	QTD Group Administrator assignment
3.2.4.4.4	3.2.4.4.3	Unassign QTD Group Administrator
3.2.4.4.5, 3.2.4.4.6	3.2.4.4.4	QTD Group Administrator leaving group
3.2.4.4.7	3.2.4.4.5	QTD Group Administrator project management
3.2.5.1	3.2.5.1	Start timer
3.2.5.2	3.2.5.2	Pause timer
3.2.5.3	3.2.5.3	Stop timer
3.2.5.4	3.2.5.4	Configure timer
3.2.6.1.1	3.2.6.1	Add note
3.2.6.1.2	3.2.6.1	Add link

3.2.6.1.3	3.2.6.1	Add image
3.2.6.2	3.2.6.2	Modify record
3.2.6.3	3.2.6.3	Delete record
3.2.7.1	3.2.7.3	Project search
3.2.7.2	3.2.7.1	Add tags
3.2.7.3	3.2.7.2	Filtration
3.2.8.1	5.5.1	User Timeline
3.2.8.2	5.5.2	Project Timeline
3.2.8.3	5.3.3	Group Timeline
3.2.10	5.6.1	Prioritization View