Smart-Gantt

by

Hind Abdulwahab Almushigih

This project is submitted to the Gannon University graduate faculty in
partial fulfillment for the degree Master of Science in Computer and Information Science.
Option: Software Engineering

Approved:

Stephen T. Frezza , Ph.D., P.S.E.M..	*Theresa M. Vitolo, Ph.D.*
Advising Professor in Charge of Research	Committee Member
Barry J. Brinkman, Ph.D.	*Mei-Huei . Tang , Ph.D.*
Committee Member	***Chair, Computer and Information Science Department***

Gannon University
Erie, Pennsylvania 16541

May 2016

Acknowledgements

I would like first and foremost to thank God for being my strength and guide in working in this project. Without Him, I would not have had the power or the physical ability to do so.

Special thank to my parents (my father Abdulwahab Almushigih, my mother Laila Alomari) for always being supportive of my education, for standing by my side and allowing me to study aboard to get my master degree, for making everything I dreamt for possible. I also take this opportunity to acknowledge everyone in my family, thank you for asking and calling me from time to time I appreciate your support and caring. I would also like to thank all of my friends for hearing me out and trying to help me with my problems.

My gratitude goes to The Ministry of Education of Saudi Arabia for enabling me to earn the best education that I possibly could have earned at Gannon University.

I would like to thank the following professors for their help at various stages of my project and my study:

Mei-Huei . Tang , Ph.D.

Mark C. Blair, MS

Yunkai . Liu, Ph.D.

Theresa M. Vitolo, Ph.D.

Barry J. Brinkman, Ph.D.

Jeremy C. Cannell, MS

John H. Coffman

Special thank to Dr. Stephen T. Frezza for all the lessons that I have learned in and out of his classes.

Table of Contents

Abstract

This document is a final report for my master degree project. It is mainly for my academic committee in Gannon University and for developers of iOS platform. This report will describe the work for developing the iOS app for Redmine™ and for implementing the Smart-Gantt as a native representation for iOS platform.

The app will allow the user to log in to Redmine™ to manage a project by adding and viewing projects and issues. View the Gantt chart and have the ability to create new issues from the Gantt chart view. This report will show how Scrum methodology have been used for building the app as it was planned for and how tough decision have been made and end up by changing the project name from Auto-Scheduler to Smart-Gantt.

List of Figures

List of Tables

1. Introduction

1.1 Overview

With the rise of using mobile smartphones to access the Internet, it is becoming essential to make a useful alternative to the website or desktop application. The aim of this project is to build an iOS app [1] that is an extension of an existing project management tool named Redmine™ [2], to provide a mobile front end and add mobile features that go beyond the web interface for Redmine™. The fundamental need driving this project is having the app on a smartphone as a project management tool that help to find mutual times for a team members to have some event. This feature is labeled as Dynamic GANNT, as the App view will allow interaction with the GANNT chart data directly, rather than through web forms as is supported by the current release of Redmine™ (version 2.3).

1.2 Background

Redmine™: it is a flexible project management web application. The website is cross-platform and cross-database, it is public open-source code. http://www.redmine.org/ [2]

Figure 1 Redmine Logo

Redmine™ supports many feature like:

- Multiple projects support
- Issue tracking
- Gantt chart
- Multilanguage support

- Multiple databases support
- Feeds & email notifications

Redmine™ provide a shared online demo for users to use can [3].

Redmine™ has a forum to offer help for users and developer or to submit and discuss bugs report, feature request for Redmine™ releases.

If you ever want to experience Redmine, they offer some books for mastering Redmine™ Redmine™ Cookbook is one of the books listed on Redmine™ wiki page. I personally have purchased this book; it helps me with the process of Redmine™ installation on Windows Server 2012 R2 [4].

1.2.1 OBJECTIVE

- Build an iOS app to extend the Redmine™ web application.
  - Dynamic GANTT
  - Mobile UI for to choose the range of a dates for task
  - Support access to any Redmine™ URL server that the user provides.

1.2.2 DYNAMIC GANTT Design (Modal View Controller)

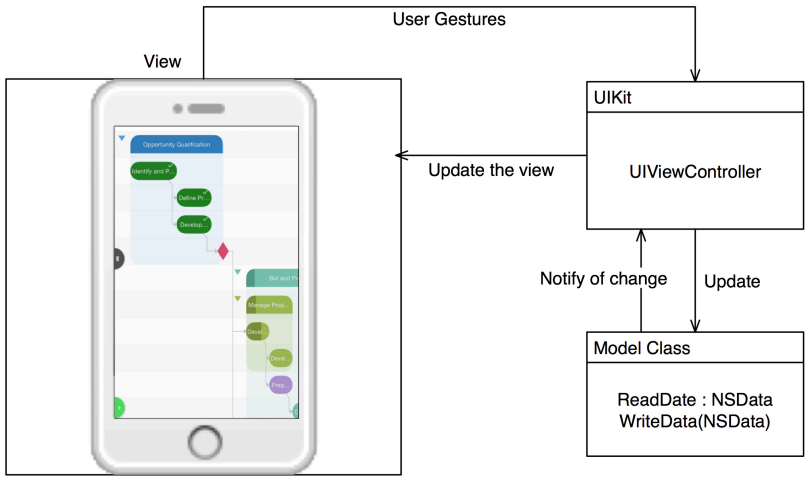


Figure 2 Dynamic Gantt

- The view is a native iOS presentation of the Redmine™ Data
- Controller get the user gestures and communicate to the model
- The model will manage the interaction with the Redmine™ server

1.3 Then it will send the data to the Controller so it can show it to the view

1.3.1 Product Perspective

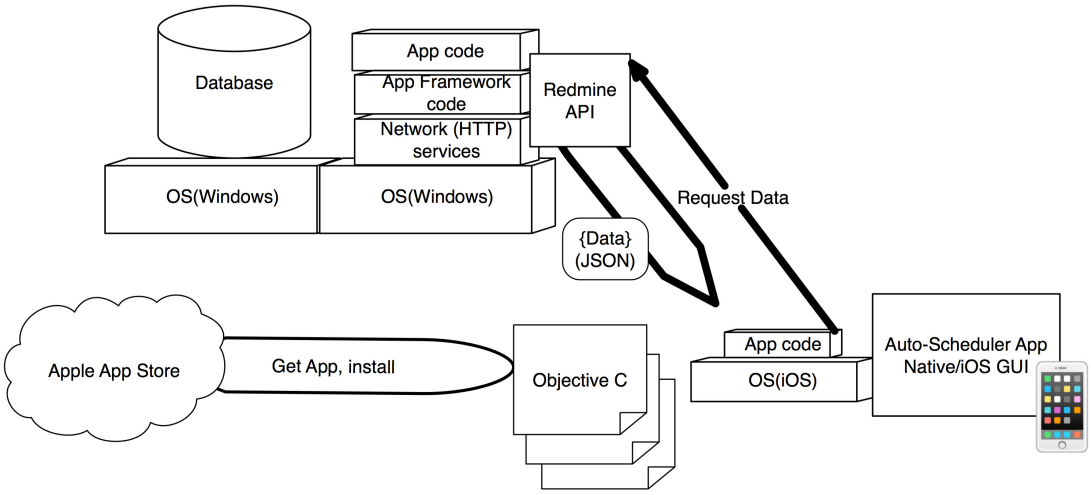


Figure 3 Product Perspective

The app can't work alone. The app require a web-server to be installed the web-app. The app will be used to support some of the functionality of Redmine™. The mobile application will need to communicate to the web-server, which in turn send a JSON data to show it on the app user interface, see Figure 3. Figure 3 presents an overall view of the application architecture.

Smart-Gantt is being marketed as a project management app, to allow users to 'manage' their projects. Smart-Gantt will be commercially distributed via the Apple app-store. Smart-Gantt will be available free for any project use like educational or business purpose. It is also useful for iOS developers to implement the Gantt-Chart *[5]*.

1.4 Summary of Capabilities

(Smart-Gantt)

Benefit	**Supporting Features**
Add project	The app allows the user to add project to Redmine™ web-app.
View project	The app allows the user to view project to Redmine™ web-app.
Add issue	The app allows the user to add issue to Redmine™ web-app.
View issue	The app allows the user to view issue to Redmine™ web-app.
Gantt Chart	The app allows the user to view Gantt-Chart and add new data without leaving the view of the Gantt-Chart.

TABLE 1: Benefits and Supporting Features

1.5 Project Management Plan

The approach that was used for the development of this project is Scrum *[6]*. The reasons for using Scrum are because it is a management framework. Second, It has a weekly meeting with my Scrum Master "Dr. Frezza" for reviewing and planning for each sprint. Third, Each sprint has its own backlog that has a task list and burn down chart that shows the estimated time for a task and the actual time and that's encourage me when seeing how much I've burned. Forth, Each sprint has its own test cases. For sure I have faced some difficulties and challenges. The next list is the difficulties for scrum itself, followed by the difficulties for each sprint.

Here are the difficulties for the scrum itself:

1. The estimated time.
    - When I estimate the task duration, I estimated it based on my understanding of the expected code, but I always face some problem because Apple keep changing its libraries and then I will have to find an equivalent to the old codes.
2. Size of the scrum team

The scrum team is supposed to be at least 2 people, to split the work between them. And each one work independently. But for this project, I have to do all of the work and I have to manage my time between being a designer, developer and tester.

Difficulties for each sprint:

First Sprint:

Creating the first backlog: Although that I know what am I going to build but I didn't know how to start maybe it's because of me panicking of new beginning so I forgot to add things that I should be doing. I manage that by doing a backlog refinement.

Second Sprint:

I actually have learned how to make HTTPs request for the first time in this sprint. I can't say that it was difficult to learn but it was a challenge, especially by looking for the hidden request on the pages. I explore some new ways to find HTTPs request on the page using some tools.

Third Sprint:

By the end of this sprint, I found that the Gantt-Chart was not a good looking plus it was not a dynamic as I planned for. So I requested an approval to make a change for my project. The time was not enough to implement the rest of the objectives. I needed more time to focuses only on the dynamic Gantt.

Forth Sprint:

In this sprint my objective was to make the Gantt-Chart a good looking. To do so, I have to make some changes in the library code. But the library uses some of Apple library that I have never used, it was for drawing the chart using points on the view. I had to learn few things in order to make some of these changes.

Fifth Sprint:

Releasing the beta version for the app, Apple requested to have a full access to the app just like one of the user. To do so, I have to provide them with an account for Gannon University to set up a VPN configuration. That makes me late for releasing the Sprint in time.

1.6 Curriculum Scope

Applying software engineering techniques in the development of this project was coming from learning these techniques from courses that I took during my study of MSc of Software Engineering. Starting from GCIS 514 Requirement & Project Management, GCIS 521 Advanced Programming iOS, GCIS 516 Data Centric Concept and methodology, GCIS 533 Software Patterns and Architect, GCIS 634 Software Maintenance & Deployment and ending with GCIS 639 Interactive Software Development.

2. Requirements

2.1 Requirements Development Perspective

Smart-Gantt is a new app. The app required Redmine™ web app installation on a server. The app requires server-side services to provide the functionality that the Redmine™ web app runs on.

2.2 Use Characteristics

2.2.1 User Classes and Characteristics

The various users that will use Smart-Gantt app are as follows:

- Gannon University Redmine™ users:

Redmine™ users are user or group of users who use Redmine™ web app to manage their project.

Redmine™ users have an understanding of what a Redmine™ web app is.

- iOS Developer:

iOS Developer is a person who have learned how to program an iOS apps on Objective c. This person should also have known how to use source control like GitHub, Inc.

2.2.2 User Documentation

- README file and HTML file for code documentation.
    - For developer: a person who wants to contribute back to the code.
    - For IT department for CIS in Gannon University: User who wants to administrate Windows server that the Redmine™ was installed on.

2.3 Key System Features

The key system features were determined to match the most important functionality feature of Redmine™ web app. Each feature has its own description and requirements. Each requirement has a unique ID (e.g., REQ-N) where the N represents a number.

2.3.1 Add RedmineServer URL

The Smart-Gantt app allows Redmine™ web app users to configure their own URL. This allows the App to be useable with any Redmine™server available to the device. This includes the following key functional requirements:

REQ-1: User must provide a valid URL to Redmine server.

REQ-2: URL fields are a mandatory to fill in order to use the app.

REQ-3: Loading indicator after posting the URL to check its reachability.

2.3.2 Services for Redmine™ users

The Smart-Gantt app allows Redmine™ web app to have some of the Redmine™ functionality that the app provide.

REQ-1: User must be registered to Redmine™ for Gannon University.

2.3.3 Log in

The Smart-Gantt app allows Redmine™ web app users to log in. Accessable to the selected Redmine server.
Key Functional Requirements

REQ-1: User must be registered to Redmine™ for Gannon University.

2.3.4 Add Project

The Smart-Gantt app allows Redmine™ web app users to create project.

REQ-1: User must be registered to Redmine™ for Gannon University.

REQ-2: User must be logged in to Smart-Gantt app.

REQ-3: All fields are a mandatory to fill to create a project.

REQ-4: Loading indicator after saving to confirm the creation.

2.3.5 View Project list and details

The Smart-Gantt app allows Redmine™ web app users to view list of projects and project details.

REQ-1: User must be registered to Redmine™ for Gannon University.

REQ-2: User must be logged in to Smart-Gantt app.

REQ-3: Loading indicator until the project list and data details are loaded.

2.3.6 Add Issue

The Smart-Gantt app allows Redmine™ web app users to create issue for a project.

REQ-1: User must be registered to Redmine™ for Gannon University.

REQ-2: User must be logged in to Smart-Gantt app.

REQ-3: All fields are a mandatory to fill to create an issue for a project.

REQ-4: Loading indicator after saving to confirm the creation.

2.3.7 View Issue list and details

The Smart-Gantt app allows Redmine™ web app users to view list of issues and issue details.

REQ-1: User must be registered to Redmine™ for Gannon University.

REQ-2: User must be logged in to Smart-Gantt app.

REQ-3: Loading indicator until the project list and data details are loaded.

2.3.8 Dynamic GanttChart

The Smart-Gantt app allows Redmine™ web app users to view GanttChart with the ability to view details for Gantt elements and the ability to add new elements.

REQ-1: User must be registered to Redmine™ for Gannon University.

REQ-2: User must be logged in to Smart-Gantt app.

REQ-3: Loading indicator until the Gantt elements are loaded.

REQ-4: All fields are a mandatory to fill to add new elements for Gantt chart.

REQ-5: Loading indicator after adding new elements to Gantt chart to confirm the creation.

2.3.9 Operating Environment

Smart-Gantt is designed to be compatible with iOS 9 and iPhone devices that support iOS 9 and later. Redmine™API is a REST service, which enables access and basic operations (create, update, delete) that require installing webserver.

2.4 Interface Requirements

2.4.1 User Interfaces

The app must have a great graphical user interfaces; these are basically the views that manage what the user will see when the app is open. The user can navigate through these views. This include choosing appropriate colors, font, signs that can be easily seen, readable and understandable:

- **Add Redmine™ server URL view**
    - A user who has never used the app should see this view when launch the application.
- **Login view**
    - A user who have never used the app should see the log-in page when opens the application.
    - If the user is already logged in, then the home page will be shown.
    - Through the home page, the user will have two options
        - Logout from the account.
        - Swipe right to view the menu that will include (Home - Project - Issue - Gantt-Chart) views.
- **Home view**

- The Home view will present the current user information.

- **Project view**
  - The project view will present the list of the projects on Redmine™.
  - From the list on the project view, the user can view details of the project by tapping on one of cell's table.
  - The project view will have (+) button to add new project to Redmine™.
    - On new project view the user will have two options, save or cancel adding new project.

- **Issue view**
  - The Issue view will present the list of the issues on Redmine™.
  - From the list on the issue view, the user can view details of the issue by tapping on one of cell's table.
  - The Issue view will have (+) button to add new issue to Redmine™.
    - On new issue view the user will have tow option either to cancel or to chose the rang of the issue duration dates. If the user decided to chose the second option and move on to the next view, there will be two options, save or cancel adding new issue.

- **Gantt-Chart view**
  - The Gantt-Chart view will present a label for the project's issue.
  - The Gantt-Chart will have (+) button to add new issue to Redmine™.
    - The Issue view will have (+) button to add new issue to Redmine™.
    - On new issue view the user will have the option either to cancel or to chose the range of the issue duration dates. If the user decided to chose the second option and move on to the next view, there will be two options, save or cancel adding new

issue.

The Gantt-Chart will have a gesture recognition to support viewing the issue details by tapping on the labels.

2.4.2 Hardware Interfaces

The app required installation and configuration of a webserver in order to run.

2.4.3 Software Interfaces

- **Redmine™ API:**

  This API provides access and basic CRUD operations (create, update, delete) for some of the sources that have been used for Smart-Gantt app. The API supports JSON formats *[7]*.

2.4.4 Communications Interfaces

Smart app communicates with Redmine™ server through API's call with HTTP's request.

2.5 Nonfunctional Requirements

2.5.1 Software Quality Attributes

There are two groups for the nonfunctional requirement, the users who are going to use the app and the developers who are going to build the app or pull the source code for maintenance. Availability and usability concerned the users while maintainability, testability and reusability concern the developer.

2.5.2 Availability

The Smart-Gantt app shall be available when it is used.

N-REQ-1: The app must be available for more than 98% of usage time.

N-REQ-2: The app must be connected to Gannon University network in order to connect to the server.

N-REQ-3: Show an alert to the user if the app is not available.

2.5.3 Usability

The Smart-Gantt app shall be easy to use and learn.

2.5.3.1 Key Nonfunctional Requirements

N-REQ-1: The app must have a great graphical user interfaces.

N-REQ-2: The app must follow the iOS human interface guidelines.

2.5.4 Maintainability

The Smart-Gantt app should be flexible to modify and easy to extend.

N-REQ-1: The app must be flexible.

N-REQ-2: The app must be written in a way that it could be easy to change and extends.

2.5.5 Testability

2.5.5.1 Description

The Smart-Gantt app should be testable by some of GU students using TestFlight.

N-REQ-1: The app code must be flexible.

N-REQ-2: The app must be written in a way that it could be easy to change and extends.

2.5.6 Reusability

The Smart-Gantt app source code should be reusable for iOS developer.

N-REQ-1: The app code must be readable.

N-REQ-2: The app code must be written in a way that it could be easy to reuse in other project.

N-REQ-3: The app code must avoid having any code smells

3. Design

This chapter provides an overview of the design of Smart-Gantt app. This section will document how I will implement the app by following the requirement that was specified on the previous section. The document will describe the design of the app, its views and its model view controllers. As a native representation for Redmine™, the design needs to achieve availability, maintainability, testability, usability and reusability. The app will help the users to view and manage changes on the Gantt chart while managing their project. The project data should remain and be updated on the Redmine server, so the software will graphically assist the user with User Created Objects (e.g., projects, Issues).and viewing project and issue details.Data Design This section provides a description of all major data structures including internal, global, and temporary data structures.

3.1 Major Internal Software Data Structure

The app followed the Model-View-Controller (MVC) design pattern [8].

- The view is a native iOS presentation of the Redmine™ Data.
- A controller gets the user gestures and then communicates to the model.
- The model will manage the interaction with the Redmine™ server.
- Then it will send the data to the Controller so it can show it to the view.

3.2 Global Data Structure

Data of the app is stored on the webserver. This limits the data shown to the user if there is no Internet connection.

3.3 Temporary Data Structure

When a user made an action that asked for a data, the server is called to fetch the appropriate data to show it on the screen.

3.4 Database Description

The database is the same as the database that the Redmine™ will be using. While this is SQLite [9], this is accessed via the Redmine™ API.

3.5 Architectural and Component-Level Design

The app structure contain of a 3 things that works together which are the Model, the View and the Controller. The reason for choosing to the MVC pattern to design the app is because Cocoa technologies and architectures are based on MVC and require that a custom objects play one of the MVC roles *[8]*. Designing with MVC design will make it easily to extend the app or maintain the code for iOS applications.

The view is basically displaying the user interface elements that the user will interact with, like a button or any kind of gesture. The controller will take these interactions and pass them to the model to get a response. Since the model can't communicate to the view directly, the model will pass the data to the controller to show it on the view.

3.5.1 Program Structure

Architecture diagram

A pictorial representation of the architecture is presented.

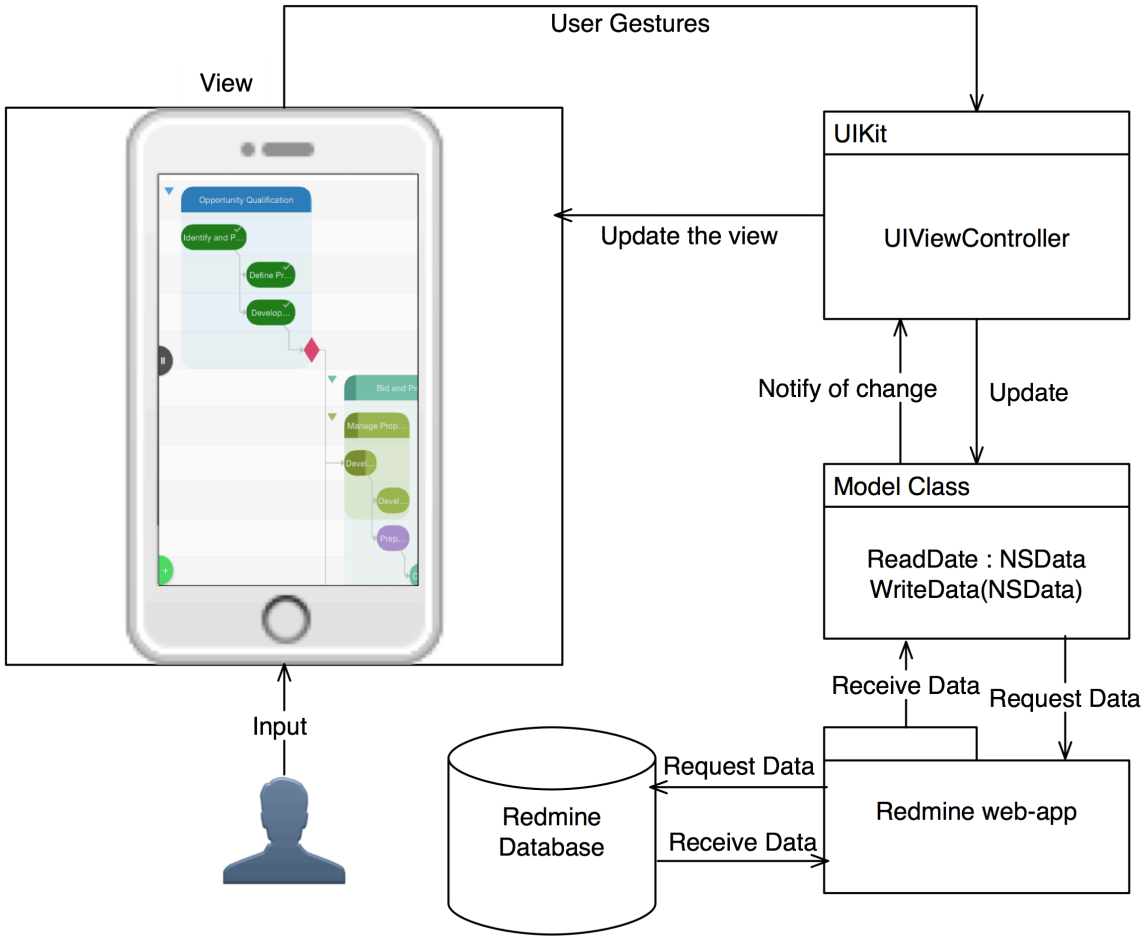


Figure 4 Architecture diagram

3.5.2 Key Software Components

3.5.2.1 Processing Narrative for "View"

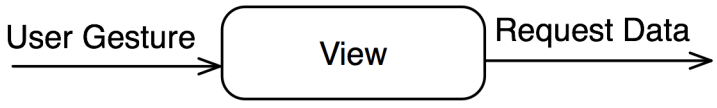


Figure 5 View Discretion

- **Component View interface description.**

  The view is basically displaying the user interface elements that the user see and will interact with. So the view input is the user interaction and its output is a request data that will be send to the Controller.

- **Component View processing detail**

  - **Interface description:**

    The view is the user interface elements that the user will see and interact with.

  - **Algorithmic model:**

    A user touches the screen

    The view gets the gesture (trigger an event)

    The controller receives the event call (IBAction).

  - **Restrictions/limitations:** No direct connection between the Model and the view component. The only way is to have a mediator, which is the controller.
  - **Local data structures:** User interface elements.
  - **Performance issues:** If the user interface elements are not connected as an IBOutlet to the view, then the event may not be send to the controller at all.

3.5.2.2 Processing Narrative for "Controller"

Figure 6 Controller Discretion

- **Component controller interface description.**

  The controller is getting the view request of data. So the controller input is as an IBAction from the view component and its output is a request data that will be send to the Model. The model then will send back to the controller a response to send it to the view.

- **Component Controller processing detail**

  - **Interface description:** Controller is (view controllers) on iOS. The view controller manages connections and updates to subviews of its view.
  - **Algorithmic model:**
    - If the input is coming from the view:

      The controller receives the event call (IBAction).

      The controller sends a request to the model.
    - If the input is coming from the model:

      The controller receives the response from the server.

      The controller sends the response from the model.

3.5.2.3 Processing Narrative for "Model"

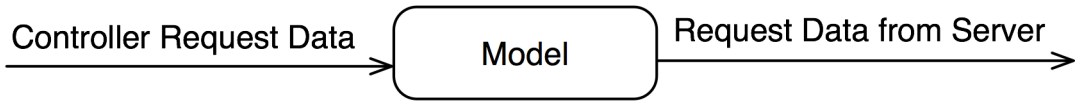


Figure 7 Model Discretion

- **Component Model interface description.**

  The Model is getting the Controller request of data. So the Model input is as a function call from the Controller component and its output is a response to the call as a return value. Which is the data that will be send to the View component.

- **Component Model processing detail**
    - **Interface description:** The Model input is as a function call from the Controller component and its output is a response to the call as a return value.
    - **Algorithmic model:**

      The Model receives the request from the Controller.

      The Model send a request to the server.

      The Model receives the data from the server.

      The Model sends the response to the Controller.

    - Local data structures: **Model stores any return data from the API calls to pass it to the controller.**

- **External machine interfaces**

  The app is sending and receiving data from Redmine™ and the app is connected to its database through an API.

- **External system interfaces**

  In order for the app to send or receive request for the data, the app interfaces with Gannon University WIFI or to use a VPN to access Gannon University resources including the app server.

3.6 User Interface Design

The following section will provide screenshots of the following views:

1. Server Link view.
2. Login view.
3. Home view.

4. Menu view.
5. Gantt-Chart View.

3.6.1 Description of the User Interface

A detailed description of user interface including screen images of the app views.

Screen images

The Server Link view

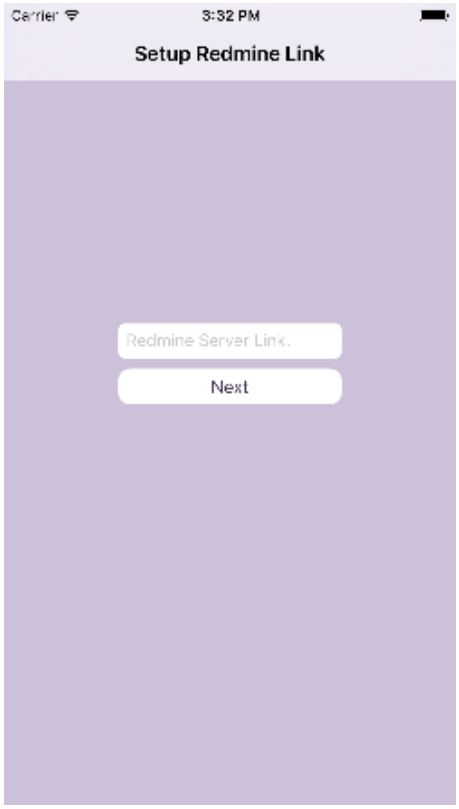


Figure 8 Server Link View

Objects and actions

The Server Link view present when launching the app. The Server Link view will have a text field that will ask for a link to the Redmine™ server. It will also contain of a next button. Once the user inters the URL, the link will be saved inside the app. The button will push the view to the Login view.

The Login View

Figure 9 Login View

Objects and actions

If the user is not already logged in to the app, then the Login view will present when launching the app. The Login view will have a login form that contains of two user interface text fields. One of them will be used to input the username and the other for the password. It will also contain of a login button. Once the user inters all the information, there will be a verification process and loading view will be presented until the informations are verified. If the login is successful, the login view will transit to the next view, which is the home view. If the login is not successful then an alert view will present to the user with the error details. The login view also contains a button that represents an info button. The button will push the view to the About view.

The Home view

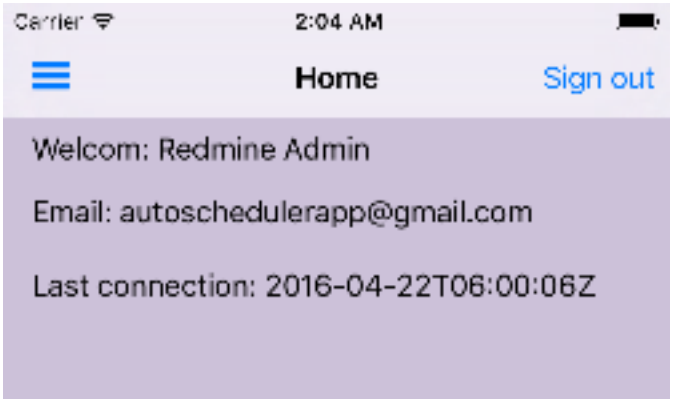


Figure 10 Home View

Objects and actions

The home view is the first view that appears when the user opens the app if the user already logged in. The Home view will have UI label that will present the current user information. The Home view contains two buttons. Logout button, which is going to take the user to the Login view, will be on the right side of the navigation bar. Menu button, which can be functioning by swiping to the left or by tabbing the button, will be on the left side of the navigation bar.

The Menu view

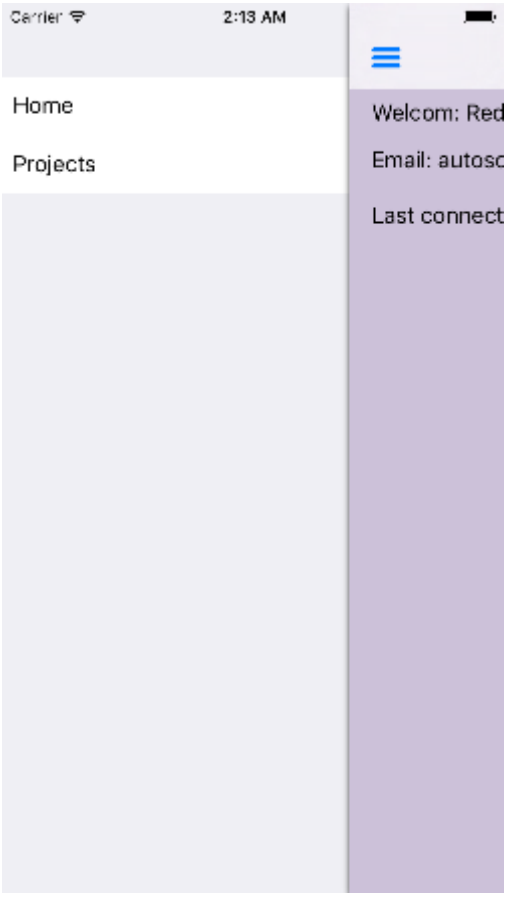


Figure 11 Menu View

Objects and actions

The menu will display all of the elements on the menu table. Each element has its own view.

All of the views will have the menu button on the left side of the view.

The Gantt-Chart view

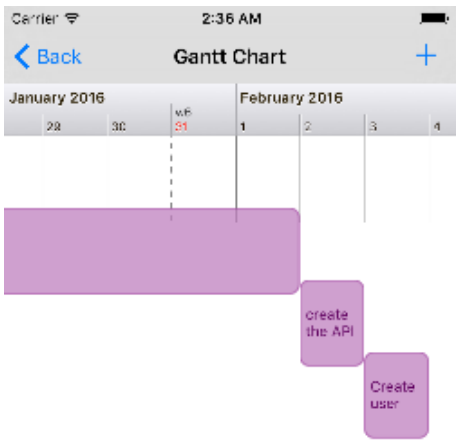


Figure 12 Gantt-Chart "Portrait Mode"

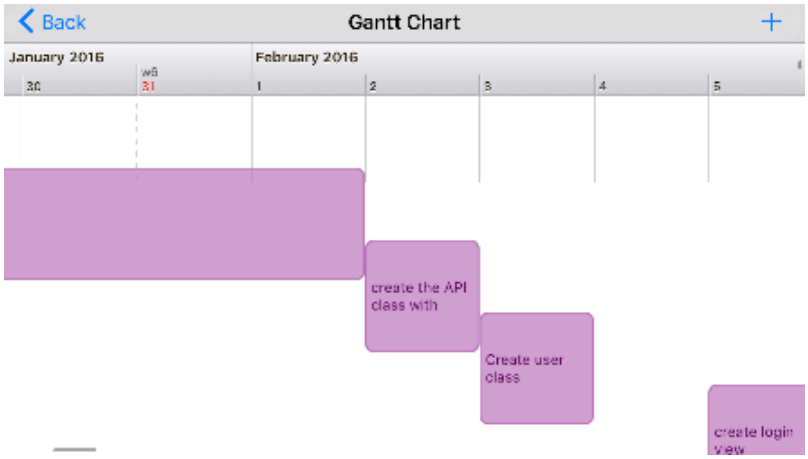


Figure 13 Gantt-Chart "Landscape Mode"

Objects and actions

The Gantt-Chart view will display all of Gantt-chart data. Each element of the chart will have the ability to response to a user gesture. The gesture will transit to a view that will represent the details of the element. The Gantt-Chart views will have a plus button (+) on its right side of the navigation bar. The function for this button will be to add new data to the server. The Gantt-Chart view does not only support Portrait mode but also support the Landscape mode for the view orientation.

3.6.2 Interface Design Rules

User interface for this app must follow the iOS human interface guidelines for design [10]. Apple's recommended to follow these guidelines for designing good user interfaces for iOS apps. The HIG explains everything about the UI elements and how, where, when to use them. As an example:

1. Ease of Navigation Divide categories clearly (Such as the menu list).
2. Make all navigation elements clickable links (The buttons and elements on the UI).
3. Use accurate navigation titles (Each view has its own title to tell the user where they are now).

3.6.3 Components Available

All the GUI components are available from Apple framework. Manage the app user interface with the UIKit framework [11]. This Objective-C framework provides an application object, event handling support, drawing support, windows, views, and controls designed specifically for the Multi-Touch interface.

3.6.4 User Interface Design Description

The user interface development system used storyboard inside Xcode apple app for building apps [12]. It allows adding views and then adding UI elements by drag and drop to the view. The views can be linked together by dragging segues from a view to another. Next figure is the Storyboard of the Xcode project for Smart-Gantt app.

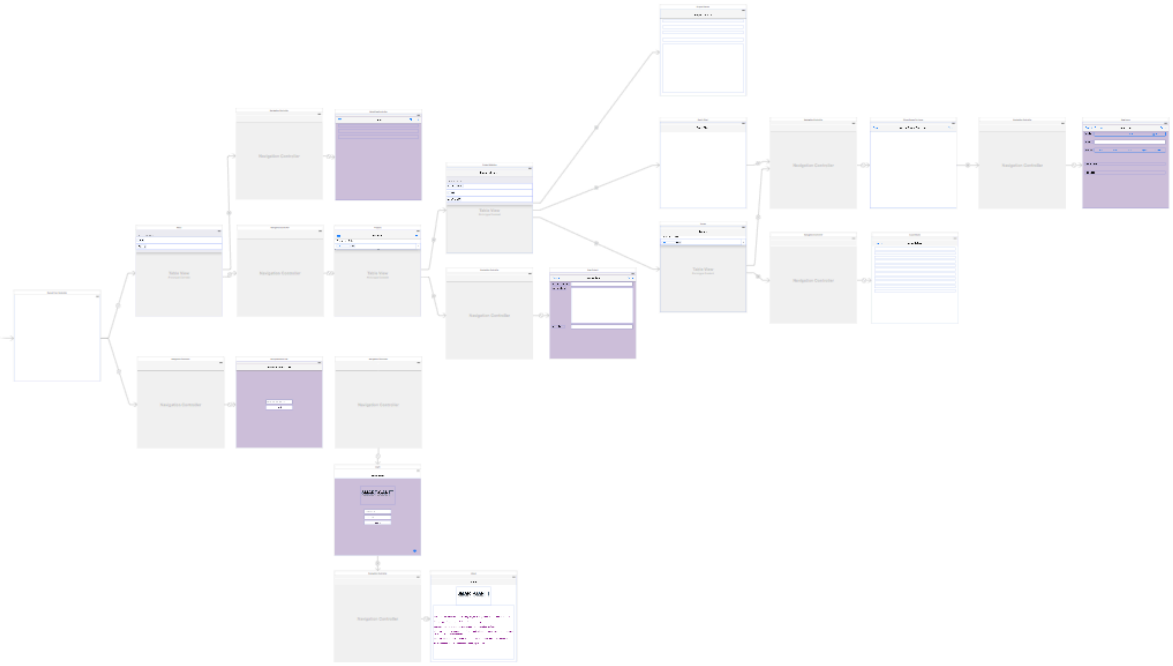


Figure 14 Xcode Storyboard For Smart-Gantt App

4. Smart-Gantt Software Design

This section will list the design patterns that have been used for developing Smart-Gantt app, where they have been used and what for. It will also show how the source code was managed and controlled as well as the process of generating HTML files that act as code documentation for a future iOS developer.

4.1 Design Patterns

Since Redmine™ is running on Ruby on Rails environment, I looked up for how to make an iOS app for Ruby Ruby on Rails environment [13]. And I found a guide to build and iOS app for Rails environment [14]. It required building an API on the web-app and then makes an HTTP's request to the server. So I did not have to build an API because Redmine™ already provide that in its version that was installed on the server which is 2.3. So this is the design based on what I found. The Guide link can be found in the bibliography.

4.1.1 Singleton Pattern:

a. **Where:**

   i. ASRESTAPI

   ii. ASUserSingleton

b. **Why?**

   Inside the app, there has to be a way that only one instance of exist for API's calls. That can happen by creating a singleton class to manage all the API's calls and data.

4.1.2 Factory Method Pattern:

a. **Where:**

   i. All the views that need to have a Loading View when waiting for a response.

b. **Why?**

Inside the app, the loading view is needed to let the user that the app is processing your call or waiting for a response by showing an alert view and a loading indicator. So instead of implementing the loading view inside all the views, a class for a loading view has been created and then adding the factory method that will return a new loading view object inside the views that need the loading view.

4.1.3 Observer Pattern:

a. Where:

ii. After the login succeeded.

iii. After log out of the app.

b. Why?

Notify the app of the current user status, logged in or logged out to check which view to present when the app launch.

4.1.4 Memento Pattern:

a. Where:

iv. ASRESTAPI

v. ASUserSingleton

b. Why?

Using NSDefault from Apple to store the current user username and password and to store the URL of Redmine™ server to use it for the API's calls.

On the Appendix B and C there are a class diagram for the model classes of the app and the data discreption of the MVC for the app.

4.2 External Library

4.2.1 IQWidgets for iOS:

Reuseable GUI component library for devices running iOS *[15]*. The library has been used for implementing the Gantt-Chart view. The library is available on GitHub and it is a public resource.

4.2.2 GLCalendarView:

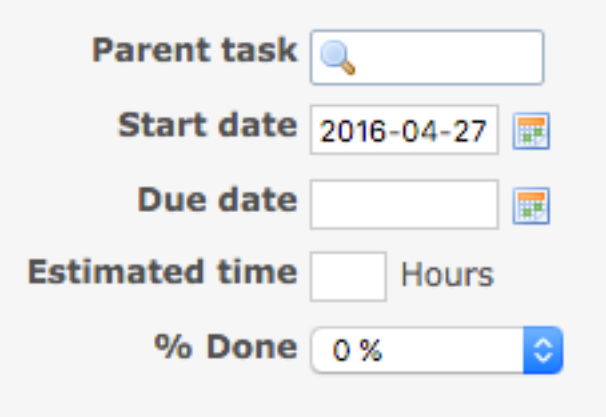


Figure 15 Redmine™ server Date selection for Issue

A fully customizable calendar view acting as a date range picker *[16]*. I used this library for choosing the range of dates for the task duration. In Redmine™ server, the end date of an issue can be a blank. So I had to look up for something to solve leaving the end date undecided.

4.2.3 SWRevealViewController:

A UIViewController subclass for revealing a rear (left and/or right) view controller behind a front controller *[17].*

4.3 Results of Reusing External Libraries:

This section I will show what were the results of the library that I planned to use for implementing Smart-Gant app and then describe things that I had to changed in those libraries to show the results that I want.

4.3.1 Gantt-Chart view

a. View Before:

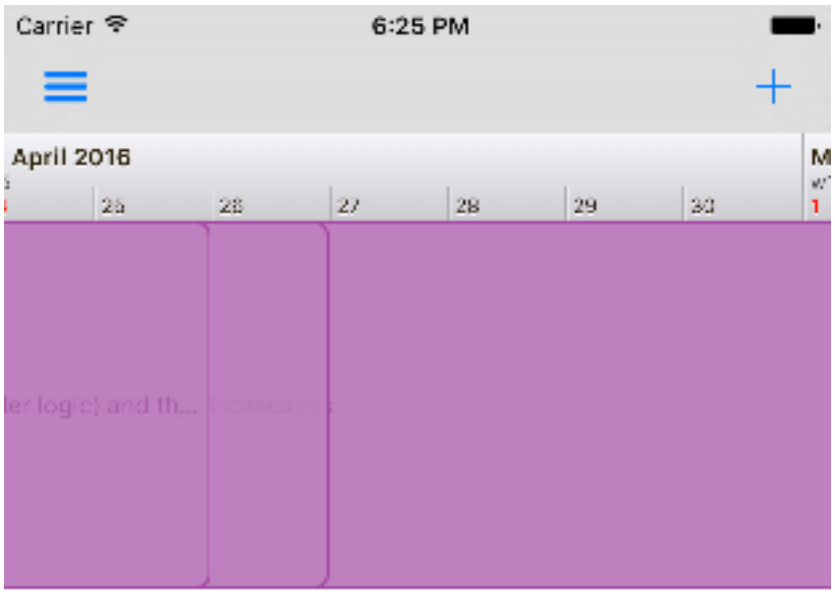


Figure 16 Old Gantt-Chart UI Data

Figure 15 showing that the IQWidgets library was drawing the data of Gantt-Chart as labels but it was not organized. One label was placed on top of the others and the label text was not supported. It also did not support user interactions. So it was just a set of blocks and the time range was not showing a full year.

a. View After modifying the code:

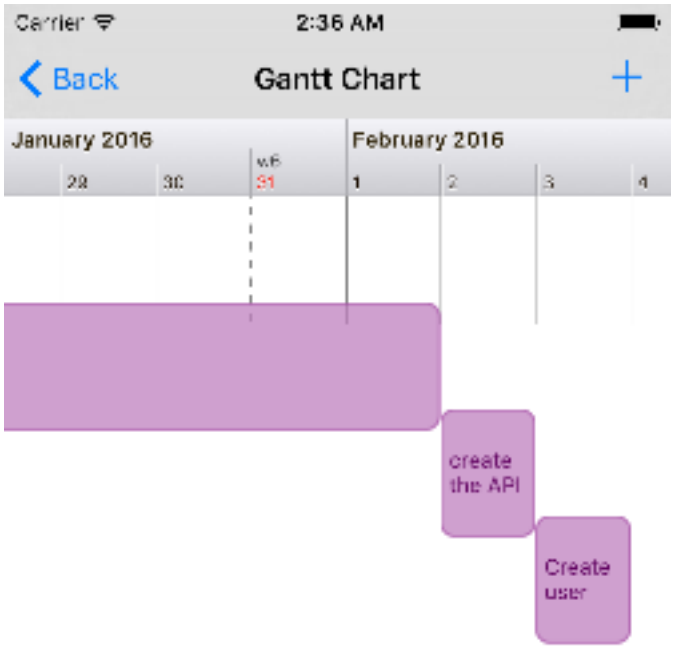


Figure 17 New Gantt-Chart UI Data Portrait

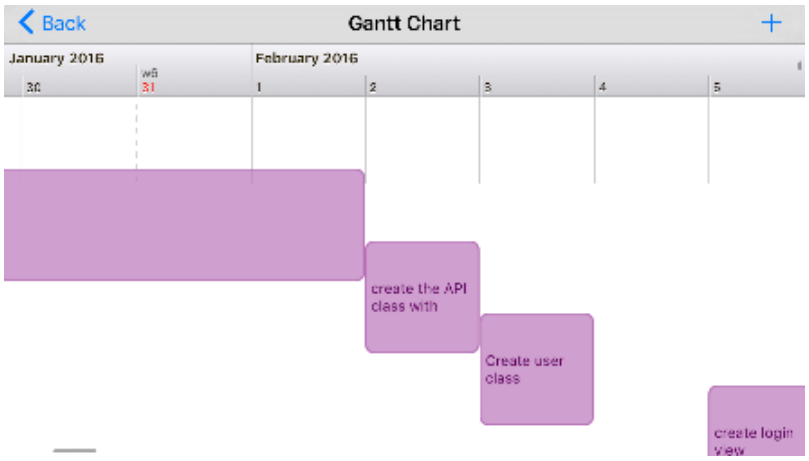


Figure 18 New Gantt-Chart UI Data Landscape

Figures 17 and 18 show how the IQWidgets library depicts the Gantt-Chart data with labels and a better organization. One label is placed next to the other and the label text is supported with multiple line of text. The labels is now support user interactions after adding a gesture recognizer

to the label. The view also supports orientation, so when the user rotates the device to the right it will still keep the layout of the UI elements organized as Figure 18 showing.

4.3.2 Date range selection view

a. View Before:

Figure 19 Old Date range selection view

Figure 19 showing that the GLCalendarView library was supporting multiple range selection.

b. View After modifying the code:

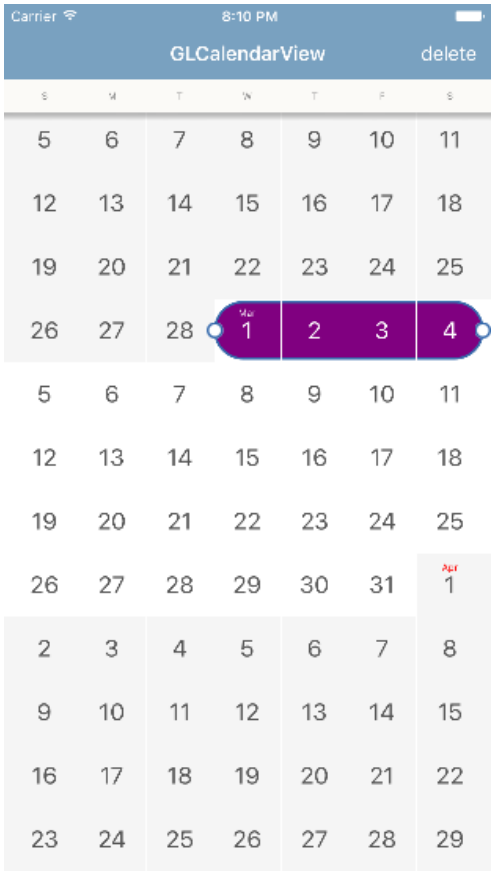


Figure 20 New Date range selection view

Figure 20 showing the result of editing the code. The user can select only one range of date at a time now.

The view won't let the user transit to the next view until the user chooses a range.

4.4 Source Code Controlling

4.4.1 Github.com

Figure 21 Github Logo

Using Github to control the source code. I had an experience working with github to store and control source code so I decided to keep controlling my code on github.

1- Creating a new Repository.

2- Creating a separated branch for each sprint.

3- Adding the task title for each commit.

4- Pushing each commit to its branch.

4.5 Source Code Documenting

4.5.1 How to Document Objective-C code

HeaderDoc is a command line tool that can be use to automatically generate an HTML documentation file for the source code of Xcode project [18]. The code needs to have a structure comment so the compiler can understand its comments to generate a HeaderDoc. There are keywords or as Apple calls them “Tags” that can be used inside these structure comments. Next I will list tags that I have used for my code and example of my code.

4.5.2 Smart-Gantt Cod Tags Documentation

```
/*!
 * @discussion A login method for the registered user
 * @param username NSString to be used.
 * @param password The password the user account.
 */

+(void)loginToASWithusername:(NSString*)username andPassword:(NSString*)password
    completionBlock:(void(^)(BOOL response))completion
{
    NSString *post = [NSString stringWithFormat:@"username=%@&password=%@",username
        , password];
    NSData *postData = [post dataUsingEncoding:NSASCIIStringEncoding
        allowLossyConversion:YES];
```

Figure 22 Example of Smart-Gantt Code Documentation

Figure 22 is showing the Tags that I used inside the comment block

1- @discussion represent a discreption of the code function or a description of a variable role inside a class.

2- @param represent a method parameter (argument).

3- Ther's another one, called @return, it represent the return value of a function call.

As a result of having this comment, the documentation can be displayed in quick help inspector for the Developer in to help the developer understand the code functionality without needing to build and run the code.

Quick Help

Declaration
```
+ (void)loginToASWithusername:(NSString *)username
                  andPassword:(NSString *)password
              completionBlock:(void (^__strong)(BOOL))completion;
```

A login method for the registered user to login to redmine

Parameters

`username`
NSString to be used.

`password`
The password the user account.

`completion`
A completionBlock:(void(^)(BOOL response))completion The API will try to login for the user to redmine server and the block will return a Boolean value if the login is successful or not.

Declared In ASRESTAPI.m

Figure 23 Result of Smart-Gantt Code Documentation

5. Verification and Validation

This section will walk through number of test cases on the app. These test cases have been done not only to check for whether the app is doing its functionality or not, but also to check how the app can handle error when it's happened. Next list is showing types of test:

1. in Xcode: UITesting can act as a user [19]. UITesting Test the app by recording the UI while interaction with the UI elements. Meanwhile the Xcode can generate a code and add this code as testCode method to a UITest implementation file. UITesting was applied by the end of each sprint's iteration and before releasing the sprint.
2. Beta testing: Using TestFlight, Apple app for testing before submitting the app to Apple App Store, to invite users to test the app and get their feedback [20].
3. Dynamic testing: Building and running the app in order to perform the dynamic test. List of test cases to be checked with its functions, inputs/output and the results.

5.1 Test items

5.1.1 Features tested

Here are the features to be tested for Smart-Gantt app requirment:

1. Invalid Redmine™ server URL
2. Incorrect UserName or Password
3. Posting Empty Data
4. URL loading system errors

Here are the test cases to be tested for interface and design:

1. Gantt-Chart Orientation
2. Date range selection for adding new issue

5.2 Environmental needs

5.2.1 Hardware:

1. An iPhone/iPad device with an iOS 9.
2. Internet connection.

5.2.2 Software:

1. Xcode user interface testing bundle.

6. Conclusion

The Smart-Gantt app was developed successfully as a Redmine™ client. Some of the features that were planned for have been implemented. Smart-Gantt now supports some of Redmine™ feature. Dynamic Gantt was not one of the Redmine™ feature. Dynamic Gantt was one of the objectives for developing the app, and now it's available.

When Smart-Gantt was proposed, there were 3 features to be implemented which are: Dynamic Gantt, Auto-Scheduler and in app-Notification. Because of the time, only the Dynamic Gantt was implemented. But there was a new feature, which is allowing the user of Smart-Gantt app to configure other Redmine™ URL. The app can be extended to support the other two features that were not implemented.

Smart-Gantt app, require a validation process when the user inter a URL for Redmine™ server. The URL must be checked for reachability in order to use the app feature.

Smart-Gantt app, can be used as a Redmine™ client app. Gant-Chart code can be reused in other projects to present a Gantt-Chart data.

7. Bibliography

[1] Apple inc. (2016, April) Apple. [Online]. http://www.apple.com/ios/

[2] Redmine Organization. (2016, April) Redmine. [Online]. Redmine.org

[3] Redmine Organization. (2016, April) Redmine. [Online]. http://demo.redmine.org/

[4] Aleksandar Pavic, "Redmine cookbook," in *Redmine cookbook*. S.I.: Packt Publishing Limited. [Online]. https://www.packtpub.com/big-data-and-business-intelligence/redmine-cookbook

[5] Wikipedia Organization. (2016, April) Wikipedia. [Online]. https://en.wikipedia.org/wiki/Gantt_chart

[6] Wikipedia Organization. (2016, April) Wikipedia. [Online]. https://en.wikipedia.org/wiki/Scrum_(software_development)

[7] Redmine Organization. Redmine. [Online]. http://www.redmine.org/projects/redmine/wiki/Rest_api

[8] Apple inc. (2016, April) Apple Developer. [Online]. https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

[9] SQLite Organization. (2016, April) SQLite. [Online]. https://www.sqlite.org/about.html

[10] Apple inc. (2016, April) Apple Developer. [Online]. https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/index.html

[11] Apple inc. (2016, April) Apple Developer. [Online]. https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIKit_Framework/

[12] Apple inc. (2016, April) Apple Developer. [Online]. https://developer.apple.com/xcode/

[13] David Heinemeier Hansson. (2016, April) Ruby On Rails. [Online]. http://rubyonrails.org/

[14] inc. Thoughtbot. (2016, April) Github. [Online]. https://github.com/thoughtbot/ios-on-rails

[15] Rickard. (2016, April) Github. [Online]. https://github.com/evolvIQ/iqwidgets

[16] Rober Dimitrov. (2016, April) cocoacontrols. [Online]. https://www.cocoacontrols.com/controls/glcalendarview

[17] John Lluch. (2016, April) Github. [Online]. https://github.com/John-Lluch/SWRevealViewController

[18] Apple inc. (2016, April) Apple Developer. [Online]. https://developer.apple.com/library/mac/documentation/DeveloperTools/Conceptual/HeaderDoc/tags/tags.html

[19] Apple inc. (2016, April) Apple Developer. [Online]. https://developer.apple.com/library/ios/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/09-ui_testing.html#//apple_ref/doc/uid/TP40014132-CH13-SW1

[20] Apple inc. (2016, April) Apple Developer. [Online]. https://developer.apple.com/testflight/

Appendix A: Glossary

ACM : Association for Computing Machinery

: A professional organization of the computing field. ACM professionals provide guidance to standards-committees and to government in defining and understanding computing technology.

CIS : Computer and Information Science

: Department conferring graduate degree

HeaderDoc : It is a command line tool that can be use to automatically generate an HTML documentation file for the source code of Xcode project.

iOS : An operating system used for mobile devices manufactured by Apple Inc.

Objective-C : Objective-C is the primary programming language you use when writing software for OS X and iOS.

Redmine™ : a free and open source, web-based project management and issue tracking tool

REQ-N : A unique ID to represent a requirement of a feature. The N represents a number.

Scrum : Scrum is an agile way to manage a project development.

SE : Software Engineering

Sprint : In project development, a sprint is a set period of time, 2 weeks that can be extended but not more than a month, during which specific tasks has to be completed and made ready for review to release. Each sprint has its own iteration.

Sprint iteration : Single development cycle.

TCP/IP : Transmission Control Protocol / Internet Protocol

: Communication protocol

TestFlight : The TestFlight app allows testers to install and beta test apps on iOS devices.

VPN : A virtual private network (VPN) extends a private network across a public network, such as the Internet. It enables users to send and receive data across shared or public networks as if their computing devices were directly connected to the private network.

Appendix B: Analysis Models

The Model class diagram of Smart-Gantt App.

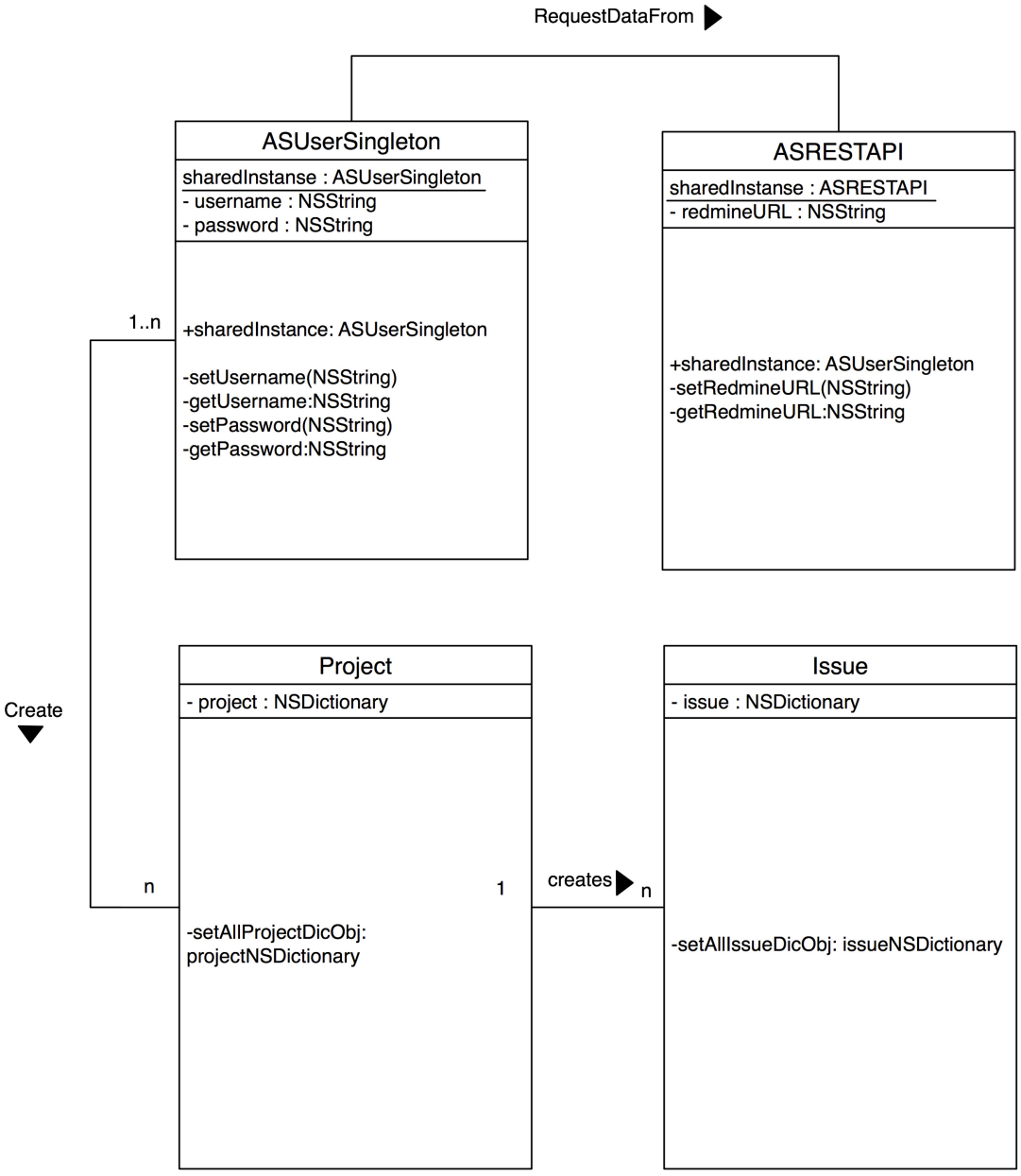


Figure C.1 The Model class diagram of Smart-Gantt App

Appendix C: Design Models

ASUserSingleton: A singleton class represents the user.

ASUserSingleton consist of the following attributes:

userName - Name of the login id of the user.

password - password of user account.

RedMine™ URL - http://www.redmine.org/

iSUserSignedIn - Whether or not the user is loged in.

sharedInstance - The only one instance of the ASUserSingleton class.

ASRESTAPI: A singleton class represents the API object for the HTTP calls.

ASRESTAPI consist of the following attributes:

RedMine™ server - The RedMine™ server URL

request - The URL request

logging - Whether or not the logging is successful.

sharedInstance - The only one instance of the ASRESTAPI class.

MenuTableViewCell : a Table View controller represent the menu table on the app.

MenuTableViewCell consist of the following attributes:

menuItems - Array list of all the view on the menu table.

ProjectsTableViewController : a Table View controller represent the project list as table on the app.

ProjectsTableViewController consist of the following attributes:

projectsItems - The response of all the project item on the RedMine™ server.

IssuesTableViewController : a Table View controller represent the issue list as table on the app.

IssuesTableViewController consist of the following attributes:

issuesItems - The response of all the issues item on the RedMine™ server.

ProjectDetailsViewController : a View controller represent the project details.

ProjectDetailsViewController consist of the following attributes:

project - the details of the project object from the selected raw from the previous table view.

IssueDetailsViewController : a View controller represent the issue details.

IssueDetailsViewController consist of the following attributes:

issue - the details of the issue object from the selected raw from the previous table view.

Appendix D: Testing Log and Summary Status

Project Name: "MyClassRoom"

Prepared by: Hind Almushigih

Date: 20 April,2016

Test Environment: iOS device with iOS 9

Table 1 Test Cases

#	Task	Input	Output	Task compilation (yes/No)	Notes
1	The user open Smart-Gantt app from the home screen of the iOS device	App name and icon	Launch the app	Yes	The app has an icon and a name to represent the app
2	The app will ask for Redmine server URL	URL	Error: the url is not reachable	No	Test the URL validity by entering invalid URL
3	The app will ask for Redmine server URL	URL	Login view	Yes	Test the URL validity by entering valid URL
4	In the Log on page, the app will ask for a- Username b- password	Username Password	Error: Invalid user or password	No	Test to login with randomly username and password
5	In the Log on page, the app will ask for c- Username d- password	Username Password	Home view	Yes	Test to login with valid username and password
6	In the Home view the user can tab on the menu button and tab on one of the option for the menu	Menu tab	Menu table	Yes	The user understands that there is a menu button on the navigation bar
7	The user can tab on "Project" in the menu table	Tab on Project	Project list view	Yes	The user tab on the project item on the menu list to view project list
8	The user can add new project	No input	Alert view	No	The user tab on the add button to add new project on Redmine server
9	The user can add new project	Project details	Project list	Yes	The user tab on the add button to add new project on Redmine server
10	The user can cancel adding new project	Cancel button	Project list	Yes	The user can cancel the process of adding new project on Redmine server

11	The user view details of project	Project item	Project details view	Yes	The user can view the details of the selected project
12	The user can add new issue to the project	Issue details	Issue list	Yes	The user can tab on the add button to add new issue for a project on Redmine server
13	The user can add new issue to the project	No input	Alert view	No	The user tab on the add button to add new issue for a project on Redmine server
14	The user can cancel adding new issue to project	Cancel button	Issue list	Yes	The user can cancel the process of adding new issue for a project on Redmine server
15	The user can view details of issue from Gantt-View	Tab on label	Issue details view	Yes	The user can view the details of the selected issue on Gantt-Chart view
16	The user can add new issue to the project from Gantt-View	Issue details	Issue list	Yes	The user can tab on the add button to add new issue for a project on Redmine server from Gantt-Chart View

Appendix E: Screen Captures

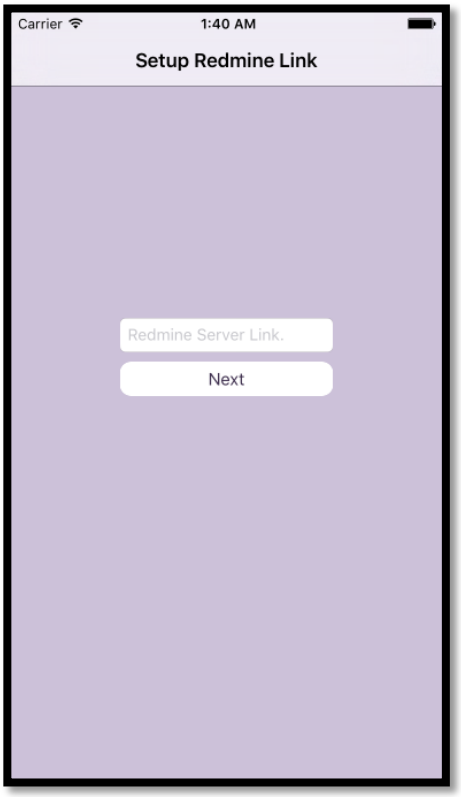


Figure E 1 Redmine Server URL view

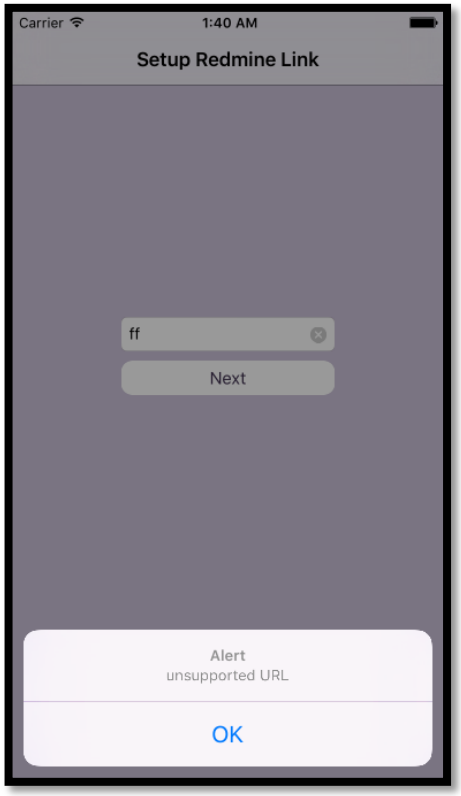


Figure E 2 URL Error handling

The app will ask for Redmine server URL, if the user provides unsupported URL, an alert view will pop up with the Error title to explain to the user what went wrong.

Figure E 3 Login view

Figure E 4 Login Error handling

The app will ask for User account information(username and password), if the user provides wrong username or password, an alert view will pop up with the Error title to explain to the user what went wrong.

Figure E 5 Home View

Figure E 6 Menu table view

Home view will be presented after the login is success. The user can navigate to the menu button to browse the projects and its modules.

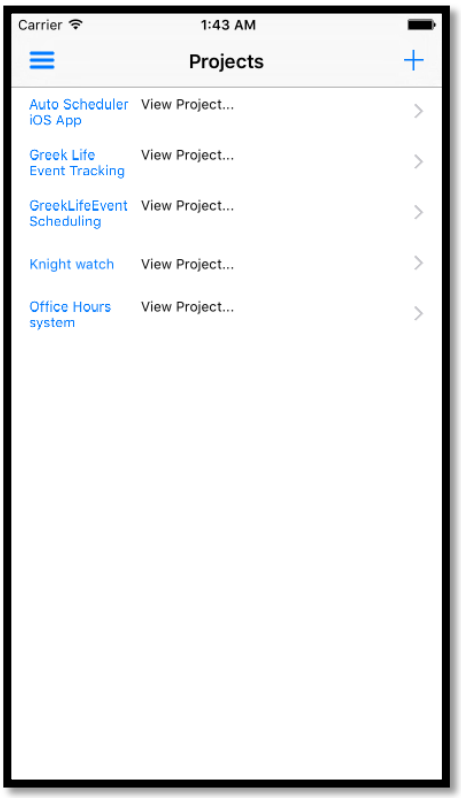


Figure E 7 Projects list view

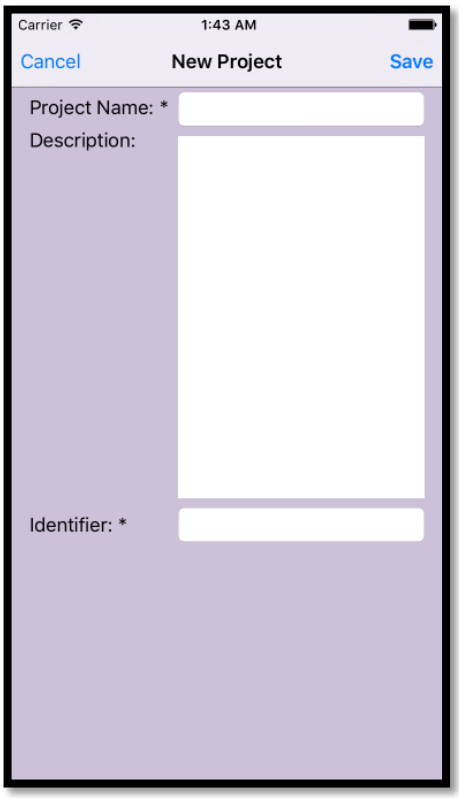


Figure E 8 New project view

The user can view the projects list and can create new project from the projects list view.

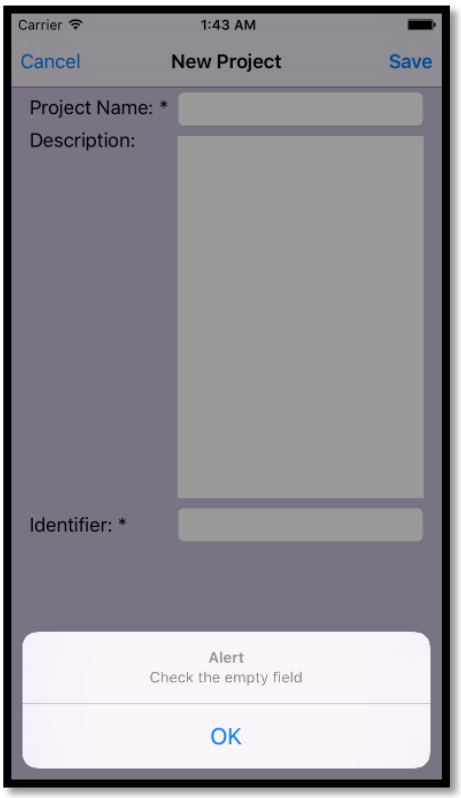


Figure E 9 New project view alert

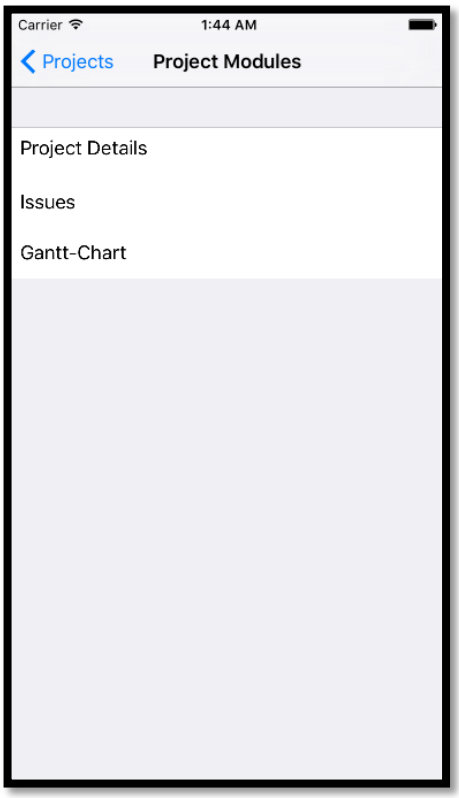


Figure E 10 Project Modules

When adding new project, if the user provides wrong input leave the fields empty, an alert view will be shown explaining what went wrong.
Each Project has 3 modules (Project details, Issues and Gantt-Chart)

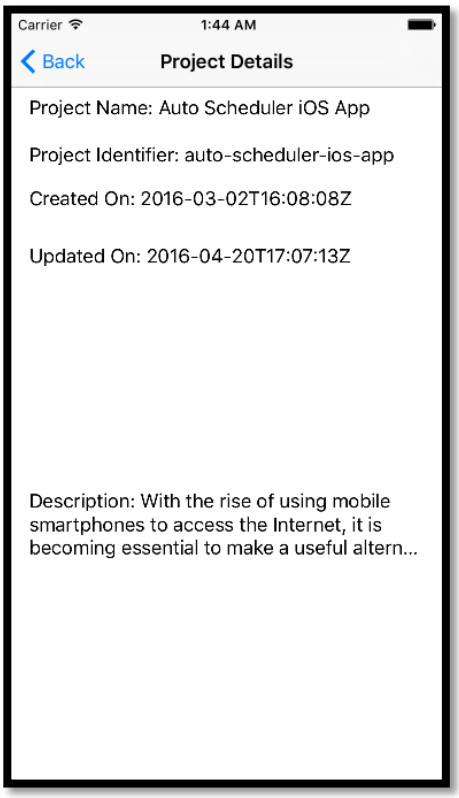


Figure E 11 project details view

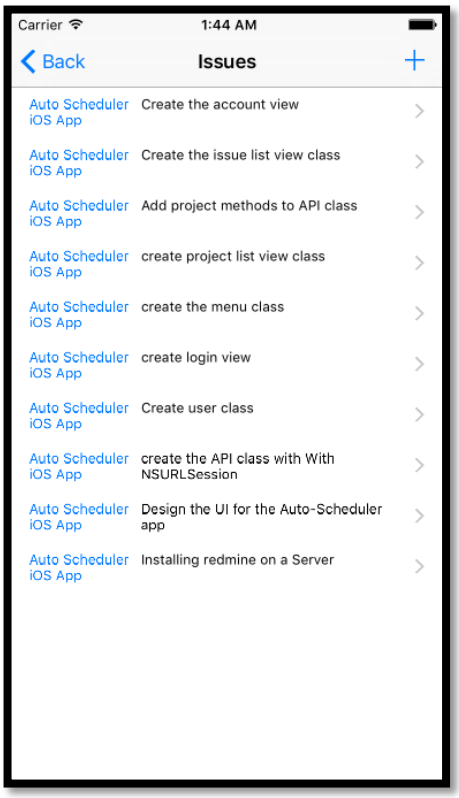


Figure E 12 Issues list view

The user can view the projects details and can view the projects' issues list

Figure E 13 Alert view for Range selection

Figure E 14 Range selection view

The user can select the date range of an Issue, if the user did not select a range, an alert view will be shown explaining what went wrong.

Figure E 19 Range selection view

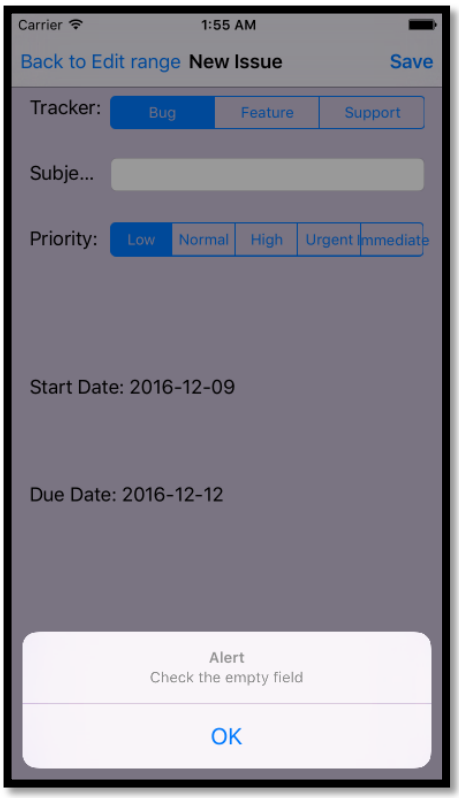
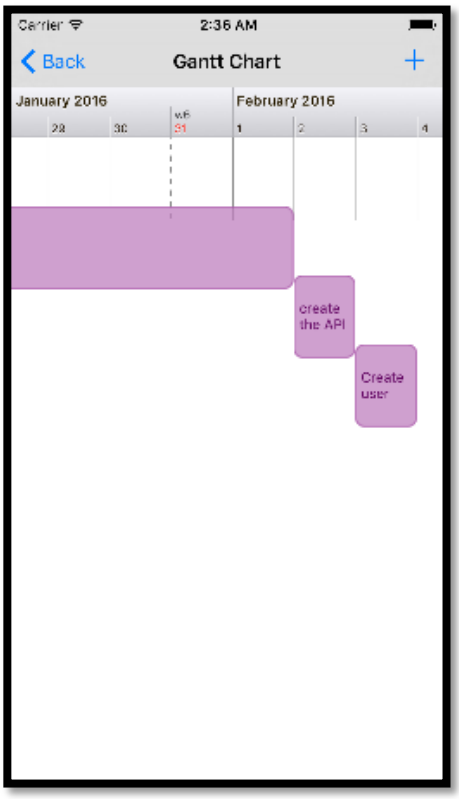


Figure E 20 New issue Alert view

If the user provides wrong input or leave the fields empty when adding new issue, an alert view will be shown explaining what went wrong.



Gantt-Chart is dynamic. User can add new issues without leaving the view. User can view the details of the issue by tabbing on the labels. Gantt-Chart supports two orientation, Portrait and landscape modes.

Figure E 21 Gantt-Chart Portrait view

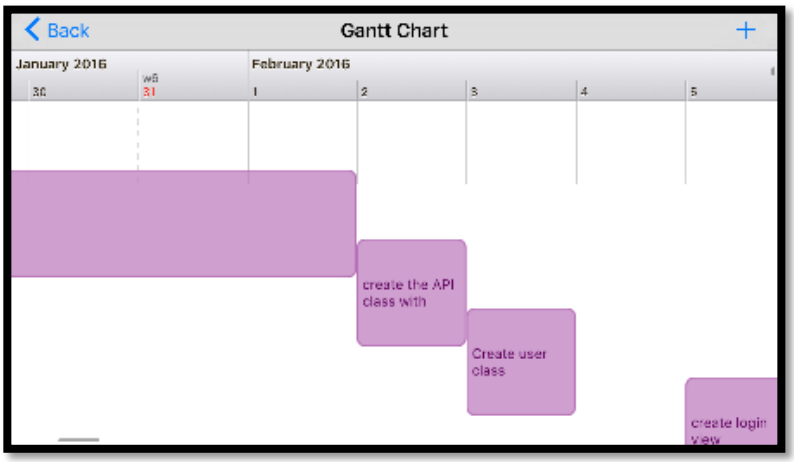


Figure E 22 Gantt-Chart Landscape view

Appendix F: Project File Repository Definitions

Using Github:

https://github.com/HindAlmushigih/AutoScheduler

There are 4 branches, each one represents sprint iteration.

Appendix G: Project Management (Scrum)

Scrum Backlog

Table 2 Scrum Backlog

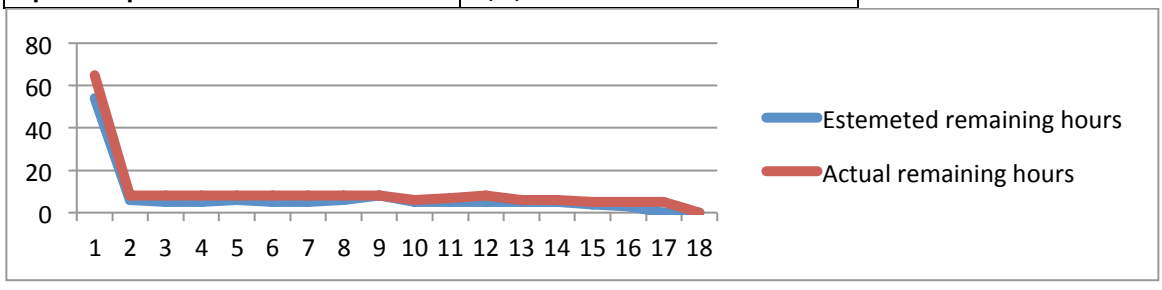
Backlog items	**Duration**
Installation for RedMine™ application	3 Days
Sprint 1: Design the UI iOS app system	2 weeks
Sprint 2: DB Connectivity and User State	2 weeks
Sprint 3: Dynamic GANTT	2 weeks
Sprint 4: Date selection for events/meetings	2 weeks
Sprint 5: Push Notifications	2 weeks
Submit the app to app store	1 weeks
Deliver all the documents of the projects.	2+ weeks

Scrum burn down charts:

Sprint 1:

Table 3 Sprint 1

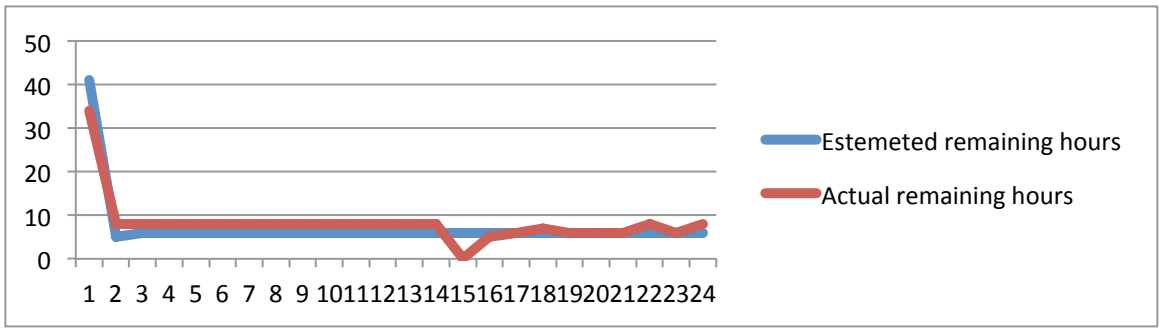
Expected duration for the sprint:	2 weeks and 3 days. (102 Hrs)
Sprint started at:	16-Jan-16
Sprint expected to finished at:	2/2/16



Sprint 2:

Table 4 Sprint 2

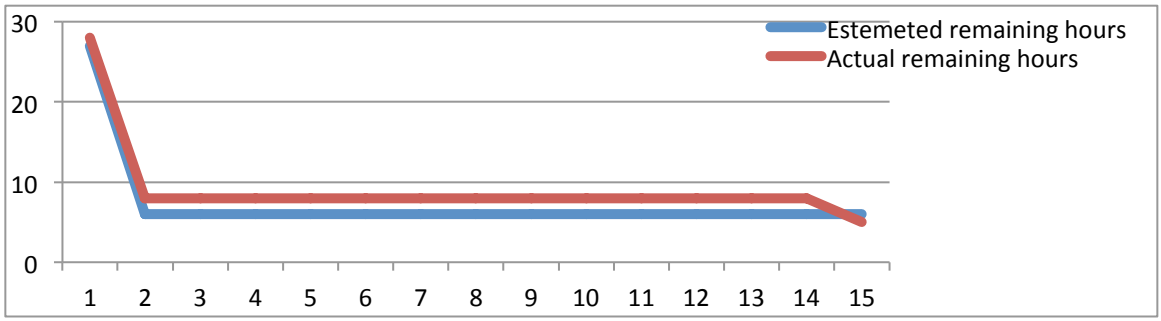
Expected duration for the sprint:	2 weeks.
Sprint started at:	2-Feb-16
Sprint expected to finished at:	16-Feb-16



Sprint 3:

Table 5 Sprint 3

Expected duration for the sprint:	2 weeks.
Sprint started at:	29-Feb-16
Sprint expected to finished at:	14-Mar-16



Sprint 4:

Table 6 Sprint 4

Expected duration for the sprint:	2 weeks and 3 days
Sprint started at:	18-Mar-16
Sprint expected to finished at:	4-Apr-16