##### Senior Project CIS 4911

SaaS Integration Library

Adam Merille

Professor Masoud Sadjadi

©Copyright SaaS Integration Library Team

Executive Summary

This document is intended to be an overview of the entire project. The document starts with an introduction to the current problem and lays out the current system in place. A feasibility study is conducted following that to determine what resources to use in order to solve the problem. The project plan in chapter 3 is meant to give an overview of what the team is doing and when. After that the system requirements are itemized before doing an analysis on them. The system design is presented after in order to show the subsystem decomposition along with persistent data management, hardware and software mappings, and a security discussion. This is followed up by a detailed design that shows the various static and dynamic models that were used for the system.

The final chapter discusses validation of the system and how that validation was conducted. At the end of the document there is a glossary followed by many appendices with more detail. Finally, there are some references at the end.

Table of Contents

1. Introduction……………………………………………………………………………...……... 5
   1. Problem definition…………………………………………………………………...……. 5
   2. Scope of system…………………………………………………………………………… 5
   3. Development Methodology……………………………………………………………….. 5
   4. Definitions, acronyms, and abbreviations …………………………………………...….... 5
   5. Overview of document …………………………………………………………………… 6
2. Feasibility Study……………………………………………………………………………….. 7
   1. Current System……………………………………………………………………………. 7
   2. Alternative Solutions……………………………………………………………………… 7
   3. Recommendations……………………………………………………………………….... 8
3. Project Plan……………………………………………………………………………………... 10
   1. Project Organization………………………………………………………………………. 10
      1. Project Personnel…………………………………………………………………….. 10
      2. Hardware and Software Resources ………………………………………………….. 10
   2. Identification of Tasks, Milestones and Deliverables…………………………………….. 10
   3. Cost of the Project………………………………………………………………..……….. 11
4. System Requirements…………………………………………………………………………... 12
   1. Functional and Nonfunctional Requirements……………………………………………... 12
   2. Requirements Analysis …………………………………………………………………….13
5. System Design …………………………………………………………………………………. 15
   1. Overview …………………………………………………………………………………. 15
   2. Subsystem Decomposition ……………………………………………………………….. 16
   3. Hardware and Software Mapping …………………………………………………………17
   4. Persistent Data Management ……………………………………………………………... 18
   5. Security/Privacy……………………………………………………………………………19
6. Detailed Design………………………………………………………………………………… 20
   1. Overview………………………………………………………………………………….. 20
   2. Static model ………………………………………………………………………………. 20
   3. Dynamic model ……………………………………………………………………………21
   4. Code Specification ………………………………………………………………………...23
7. System Validation……………………………………………………………………………….25
   1. Subsystem Tests. …………………………………………………………………………. 25
   2. System Tests ……………………………………………………………………………… 25
   3. Evaluation of Tests ……………………………………………………………………….. 28
8. Glossary ………………………………………………………………………………………... 29
9. Appendix……………………………………………………………………………………….. 30
   1. Appendix A ……………………………………………………………………………… 30
   2. Appendix B……………………………………………..…………………………………. 30
   3. Appendix C ……………………………………………………………………………….. 32
   4. Appendix D ………………………………………………………………………………. 36
   5. Appendix E ……………………………………………………………………………….. 38
   6. Appendix F ……………………………………………………………………………….. 40
   7. Appendix G ………………………………………………………………………………. 41
   8. Appendix H ……………………………………………………………………………….. 42
10. References…………………………………………………………………………………….....46
11. Introduction

This chapter gives a brief introduction to the problem and development methodology used to address it. There is also an overview for the rest of the document and some definitions.

* 1. Problem definition

The software company Brighgauge would like a platform to test certain datasources. Namely they want the data to come back in a formatted way which would enable them to dive into the details and decide the viability of including that datasource on their platform.

* 1. Scope of system

The scope of the system is limited for this semester. The main goal of the product owner is to enable data retrieval from the Trello API. As such, there is not outset intention to have the system integrate multiple APIs at this time. Care will be taken to design the system in such a way that will enable future APIs to be added in a simple manner, but those may come secondary to accomplishing the goal at hand.

* 1. Development methodology

The software process follows an agile methodology. This allows the developers to get the requirements from the product owner, formulate those into user stories, create use cases and sequence diagrams, append to class diagrams, write the code, and test the code. It enables the process to iterate throughout the development cycle. By using the sequence, class, use case, package model diagrams from the UML format going back when something isn't working or needs to be changed is relatively simple. The difficulty is making sure the overarching design is one that can be modified throughout for the big picture of the project.

* 1. Definitions, acronyms, and abbreviations

**API**: Application Programming Interface. Gives the ability to access code written by someone else in a formatted manner

**SaaS:** Software-as-a-Service is defined as a software licensing model that is subscription based. It enables a user to have access to some business application that they can use for their own unique purposes.

**RDBMS:** Relational Database Management System.

* 1. Overview of document

From here this document goes on to a feasibility study in chapter 2. After that is the project plan discussed in chapter 3. That chapter goes through the personnel breakdown, the hardware and software resources, identification of tasks and the cost of the project. Chapter 4 lists the system requirements and gives an analysis on those requirements. Chapter 5 introduces the system design intended to address the requirements. That chapter goes into the subsystem decomposition, hardware and software mapping, security and privacy issues, and persistent data management. Chapter 6 is the detailed design analysis which overs the static and dynamic modeling, as well as code specification. Chapter 7 wraps up with the validation of the system including subsystem tests, system tests, and the evaluation of those tests. At the end there is a glossary followed by supplementary appendices.

1. Feasibility Study
   1. Current System

As of right now there is now current system in place for this project. This is version zero and so is a foray into the possibility of this testing ground and the features it might contain.

* 1. Alternative Solutions

Alternatives for website construction:

1. Java

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Lots of experience | Little experience with Java websites |
| Lots of resources | Will probably have to build certain things |
| Highly scalable for web |  |

Table 2.2.1

1. PHP

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Probably tons of examples | Very little experience |
| Lots of resources | Extremely verbose |
|  | Slightly steeper learning curve |

Table 2.2.2

1. Python

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Experience with language | Little experience building website with Python |
| Django gets going fast | Generally considered not highly scalable |
| Shallow learning curve |  |
| Easy to follow tutorials |  |

Table 2.2.3

Alternatives for databases:

1. PostgreSQL

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Strong Community | Performance can be slow |
| Lots of third-party support | Not as popular |
| Stored procedures extensible |  |

Table 2.2.4

1. MySQL

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| Many third party tools | Reliability issues |
| Many Features | Integration with some languages might be difficult |
| High security |  |
| Scalable |  |
| Fast |  |

Table 2.2.5

1. SQLite

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| File based | Limited abilities |
| Easy to develop with | No user management |
| Comes preloaded with Django |  |

Table 2.2.6

* 1. Recommendations

Based on the above analysis and through discussion with members of Brightgauge there is a certain path that is being recommended. For the web application construction the choice is Python using the Django framework. For the database, PostgreSQL would like to be used but SQLite will be utilized for this first version.

Python with Django has been chosen due to its ease of use, team familiarity, and shallow learning curve. It will enable the project to move forward quickly without getting stuck on the details of the website.

SQLite was mainly chosen because it comes built into Django. Support for PostgreSQL is available for Django but requires some set up. The way Django works changing the database later on will not affect the current code as writing and reading are handled by the framework rather than by the programmer through his/her development code.

1. Project Plan
   1. Project Organization

The following section is a breakdown for who is doing what throughout the project. It also lays out the hardware and software requirements, the milestones, and the cost of the project.

* + 1. Project Personnel

Adam Merille will be responsible for everything for this first version. He will be writing all the documentation, creating all the diagrams and charts, setting up and creating the Django project and web app. He will also be writing the code for the api calls on the backend. He will be working on the frontend development, including HTML, CSS, and Javascript. He will be helping with the user interface design and responsible for managing the database. Lastly, he will also be writing the test cases to make sure the system functions as set out.

* + 1. Hardware and Software Resources

The following is necessary to the development of this project:

**Hardware:**

* Machine for hosting, provided by FIU
* Personal computer with at least 2.0 GHz processor
* 4 GB of RAM
* At least 4 GB of free hard disk space

**Software:**

* Linux or Mac Operating System
* PyCharm or similar IDE
* Python 2.7
* Django 1.7
* Text Editor (OpenOffice, Gedit, etc.)
  1. Identification of Tasks, Milestones and Deliverables

|  |  |  |
| --- | --- | --- |
| **Milestone** | **Description** | **Date** |
| 1 | Feasibility and Project Plan, Requirements Document | February 1 |
| 2 | Design Document | April 22 |
| 3 | Final Deliverable | April 30 |

Table 3.2.1

* 1. Cost of the Project

|  |  |
| --- | --- |
| **Criteria** | **Python** |
| Infrastructure | $0 |
| Development | $0 |
| Operation | $0 |
| Total | $0 |

Table 3.3.1

|  |  |
| --- | --- |
| **Criteria** | **SQLite** |
| Infrastructure | $0 |
| Development | $0 |
| Operation | $0 |
| Total | $0 |

Table 3.3.2

The website is hosted on FIU servers, so that cost is assumed to be included in tuition and fees. Beyond the above technologies a personal laptop may be need (about $500 - 600 should suffice), although there are great computers in the School of Computing and Information Sciences lab. As for the labor, it is a student's joy to work for free now in the hopes of earning it back later.

1. System Requirements

This chapter discusses the requirements asked for by the product owner and performs an analysis of those requirements. The chapter dives into the different scenarios that may arise and provides a use case model for the various requirements, as well as a set of sequence diagrams.

* 1. Functional and Nonfunctional Requirements
* The system shall enable a user to add a datasource, namely via Trello.
* The system shall allow the user to add multiple accounts from the same resource.
* The system shall make a calls to get labels, get cards, get lists, get boards, and get members from Trello. The system shall clean the data before displaying the results in a table format.
* The system shall allow for specific filtering with the get cards request into getting the users cards, cards due in 7 days, and cards past due.
* The system shall secure the credential information in a storage format that requires password protection.

The following are various scenarios that may take place on the project.

* A User arrives at the homepage and wants to get started. The user should click on the link which directs him/her to add a datasource.
* A user wants to add his/her Trello datasource to the application and is on the datasource page. S/he clicks the link for Trello and follows Trello's instructions to allow the application to have access. After, the user is redirected to a confirmation pages showing him/her where to find the dataset.
* A user wants to make a Trello API call. The user navigates to his/her dataset and selects one of the links to make a call (Get all cards, Get all Boards, Get Labels, Get Members, Get Lists). After selecting the call, the system displays the output from the call in a table.
* A user wants to filter the get all cards call. Along the top of the page the user may select "My Cards", "Due in Seven", "Past Due", or "Total Cards". The metrics remain at the top so navigation is possible through each.
  1. Requirements Analysis

The following use cases resulted from the above requirements:

**Title:** View Data Source

**Description:** After authentication, User has the ability to make API calls on that account in order to retrieve data (Card #110)

**Actor:** SaaS-Integration-Library User (SIL user) and third party API

**Preconditions:** Data source for API calls has been verified and SIL user is on API call page.

**Steps:**

Step 1: User clicks link for an API call.

Step 2: System gets the Api model object for the call.

Step 3: System makes call to API behind the scenes, creating an API object to make the call.

Step 4: API returns data from call.

Step 5: System loads new page with data returned from API call.

**Postconditions:** User has been passed to a page containing the returned data from specific API call.

**Title:** Create Data Source

**Description:** After authentication, User has the ability to create a data source (Card #160)

**Actor:** SaaS-Integration-Library user (SIL user) and third party API

**Preconditions:** User has an account with third party and has navigated to Create Data Source page

**Steps:**

Step 1: User clicks link for an API.

Step 2: System requests a token on the users behalf from the API.

Step 3: After user accepts SIL to have access to his/her account, system creates ApiCredential and Api object with token information.

Step 4: System redirects to confirmation page.

**Postconditions:** User has been passed to a page displaying successful creation.

**Title:** Get My Cards

**Description:** Filter to get cards only for user from the get\_all\_cards method (which returns all cards user can see)

**Actor:** SaaS-Integration-Library user (SIL user)

**Preconditions:** The user has selected the call "Get all cards" from the dataset and is on the "Get all cards" page

**Steps:**

Step 1: The user clicks the metric which says "My Cards"

Step 2: The system filters the cards

Step 3: The system reloads the table

**Postconditions:** The table updates and displays only the cards associated with the user

**Title:** Get Due in Seven

**Description:** Filter to get cards that are due in seven days from the get\_all\_cards method (which returns all cards user can see)

**Actor:** SaaS-Integration-Library user (SIL user)

**Preconditions:** The user has selected the call "Get all cards" from the dataset and is on the "Get all cards" page

**Steps:**

Step 1: The user clicks the metric which says "Due in Seven"

Step 2: The system filters the cards

Step 3: The system reloads the table

**Postconditions:** The table updates and displays only the cards due in seven days

**Title:** Get Past Due

**Description:** Filter to get cards which are past due from the get\_all\_cards method (which returns all cards user can see)

**Actor:** SaaS-Integration-Library user (SIL user)

**Preconditions:** The user has selected the call "Get all cards" from the dataset and is on the "Get all cards" page

**Steps:**

Step 1: The user clicks the metric which says "Past Due"

Step 2: The system filters the cards

Step 3: The system reloads the table

**Postconditions:** The table updates and displays only the cards which are past due

1. System Design

The following sections give an overview of the system design by showing the subsystem decomposition, the hardware and software mappings, the persistent data management, and the security and privacy designs. The system will implement all the use cases that were discussed above.

* 1. Overview

The main architecture for the system will be coming from the Django framework which uses a variation on Model-View-Controller. Namely it does away with controller and uses a Model-View-Template architecture. There is some discussion as to whether the framework itself acts as a controller with url mappings and such. Behind the scenes there will be some pipe and filter architecture for the various APIs and the calls they allow. As well as the client portion of the server-client architecture with the third party APIs.

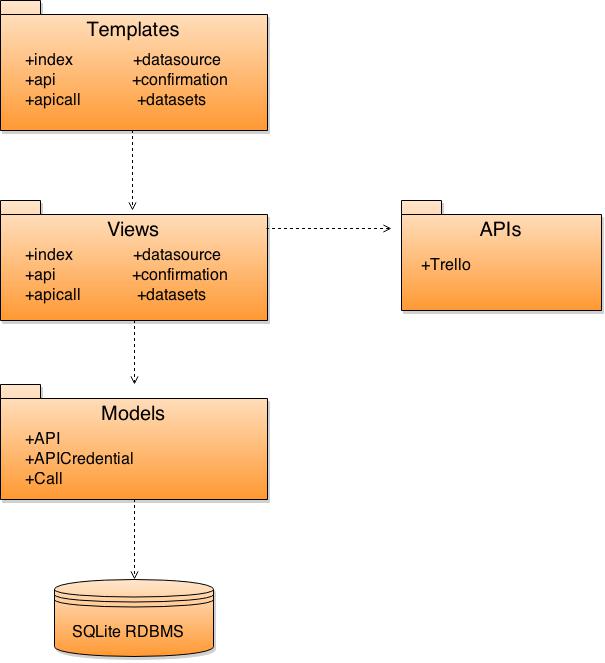


Figure 5.1-1- Package Diagram

* 1. Subsystem Decomposition

The Templates subsystem will deal with how the website looks and reacts. This will be handled by the various template html documents. They will handle how the pages look and if any data is passed to the page, whether or not to show it. The use cases associated with this subsystem are all of them since the initiation comes from the user selecting something from one of the templates.

The Views subsystem is the behind the scenes operator for being able to deal with multiple third-party APIs. This decides what information should be passed to the Templates subsystem by receiving queries and reaching out to the APIs subsystem and SQLite RDBMS subsystems. The use cases associated with this subsystem deal with the filtering as well as navigation around the site.

The APIs subsystem contains the logic for a given third-party API. The main focus for this semester was the Trello API. This will handle the details of making the call, cleaning the response, and deciding what will be available to the Views subsystem. The main use case associate with this use case is the viewing of a dataset since this subsystem makes the calls.

The SQLite RDBMS subsystem will contain the data that is retrieved from the APIs subsystem. The Views subsystem will be relaying the data retrieved from the APIs subsystem to the SQLite RDBMS as well as reading from the storage when needed from the Templates subsystem. The main use cases for this subsystem are the creation of a new datasource and the calling of the API.

Refer to use cases in appendix B for more detail.

* 1. Hardware and Software Mapping

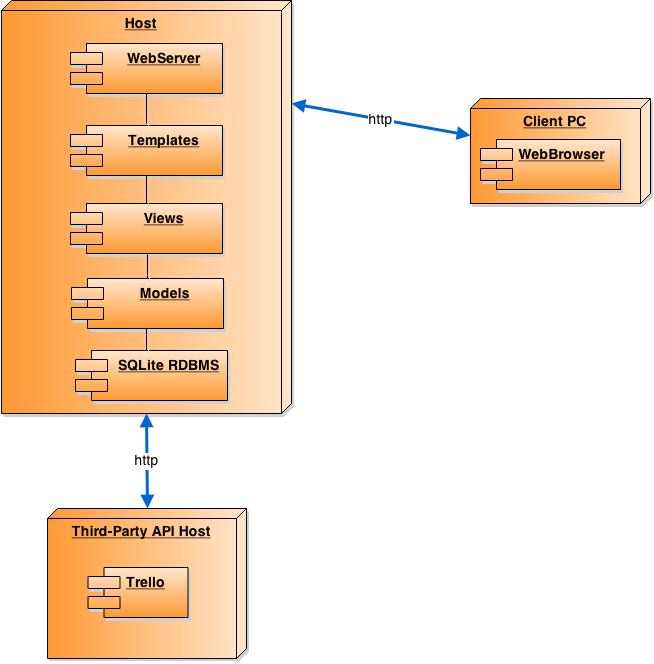


Figure 5.3-2 Deployment Diagram

The system has three main components. The host which contains the webserver, templates, views, models, and database, the client pc which reaches out to the host, and the third party API host that our host reaches out to for data retrieval.

* 1. Persistent Data Management



Figure 5.4-3- Code for Data Models

This data is stored on the SQLite RDBMS within Django. The above code is the model that is used to create the tables and attributes. As can be seen the Api class contains only a name and calls field. The ApiCredential class has a name, settings, and Api foreign key. Finally the Call class has a foreign key to an Api, a name field, and a response field.

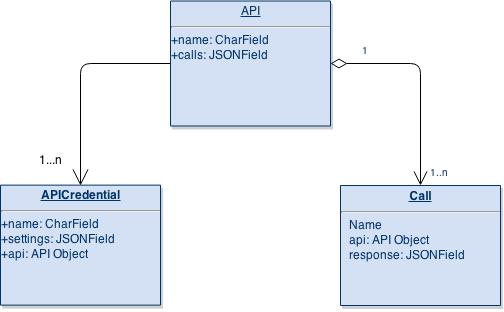


Figure 5.4-4- Data Model

* 1. Security/Privacy

Django comes with great security features available out of the box. This includes cross site scripting protection, cross site request forgery protection, SQL injection protection, session security, and others.

In terms of privacy, on a basic level the project will have only one user that has access to create data sources. The authentication is done by third party APIs, so that privacy is outside the system. Access to the database will be restricted to the single user.

1. Detailed Design

Introduce the detailed design chapter (one or two paragraphs).

* 1. Overview

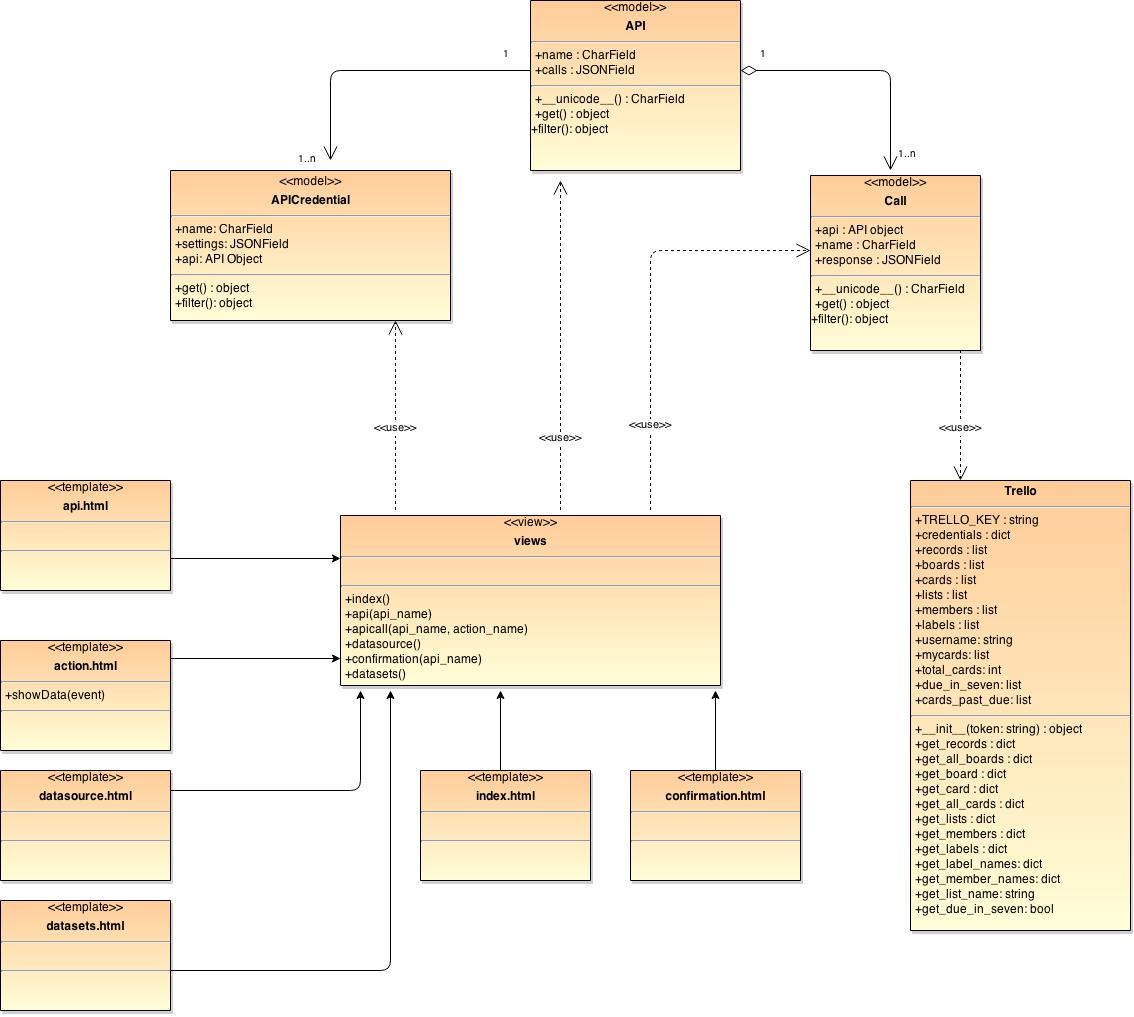
The Templates subsystem will deal with how the website looks and reacts. This will be handled by the various template html documents. They will handle how the pages look and if any data is passed to the page, whether or not to show it.

The Views subsystem is the behind the scenes operator for being able to deal with multiple third-party APIs. This decides what information should be passed to the Templates subsystem by receiving queries and reaching out to the APIs subsystem and SQLite RDBMS subsystems.

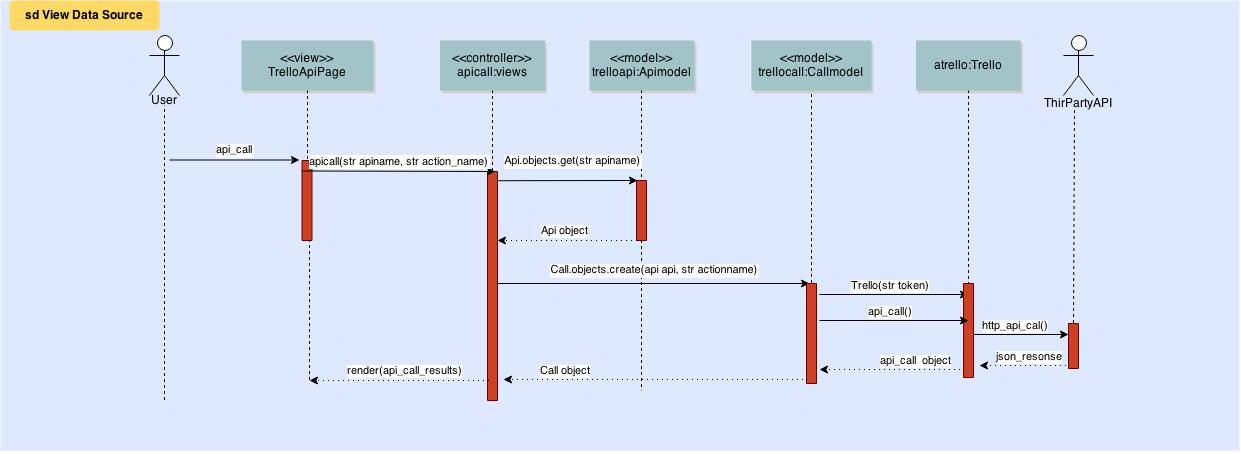
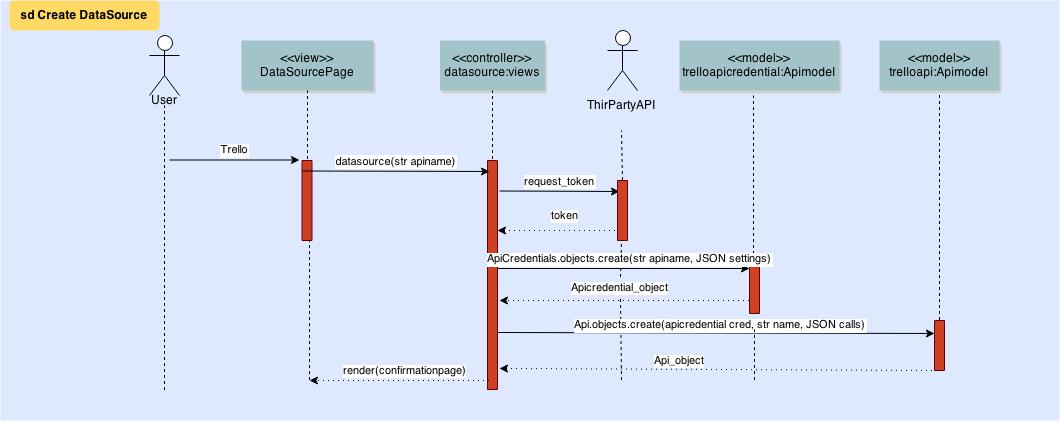
The APIs subsystem contains the logic for a given third-party API. The main focus for this semester was the Trello API. This will handle the details of making the call, cleaning the response, and deciding what will be available to the Views subsystem.

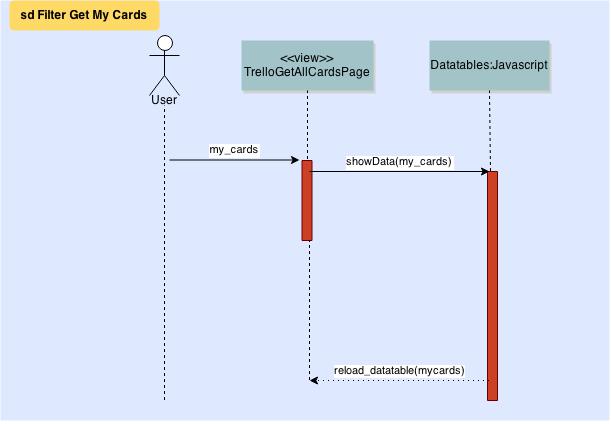
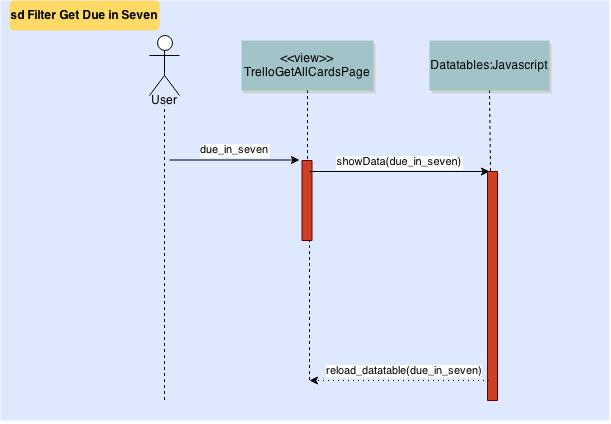
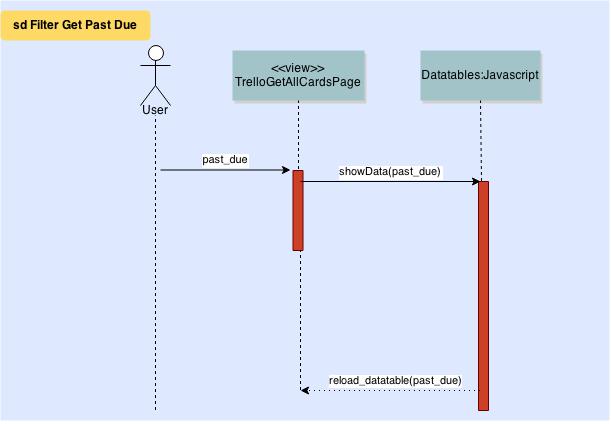
The SQLite RDBMS subsystem will contain the data that is retrieved from the APIs subsystem. The Views subsystem will be relaying the data retrieved from the APIs subsystem to the SQLite RDBMS as well as reading from the storage when needed from the Templates subsystem.

* 1. Static model



* 1. Dynamic model – state machine diagram for the main control object in each subsystem. Include the design of the algorithms used in the problem solution. Refinement of the sequence diagram from the analysis model. Appendix D.





* 1. Code Specification

The Templates subsystem only has a few actual methods, most of it is formatting. The methods it does have are in JavaScript to render the datatables or to re-render those tables on a click of an element on the page.

Most of the methods for the views subsystem deal with make queries to the API subsystem and the database subsystem and then routing that data to the templates subsystem.

The API subsystem contains code for the Trello class. A series of methods are for making the http request to the Trello API. Some of the methods are for cleaning that data and then storing it in a fashion that can be used easily by the views and templates subsystems.

The code references can be seen in appendix F.

1. System Validation

This chapter discusses the validation that was done on the project through various test systems. It starts with subsystem tests and then moves onto system tests. Finally the chapter concludes by giving an evaluation of how those tests were successful or not.

* 1. Subsystem Tests

To test the subsystems the Python Mock library was utilized which allowed subsystems to be created without dealing with the actual code. This allowed certain features, such as making an API call with the Trello class, to be tested before even testing that the third party was returning any data at all. See Appendix G for the code containing test drivers and stubs.

* 1. System Tests

**Create New Datasource:**

|  |  |
| --- | --- |
| **Test Case ID:** | **SIL\_CreateDatasource\_Test1\_Sunny\_Day** |
| Purpose: | To validate that the addition of a new account from Trello redirects the user to the proper page and adds the required information to the database. |
| Preconditions: | The user is on the Datasource page of the application. |
| Inputs: | 1. Select **Trello** datasource button. 2. Select **Login** 3. Username: siluser 4. Password: 123456 5. Select **Allow** |
| Expected Output: | The system redirects the user to a confirmation page, lets the user know that the account has been added, displays the username “siluser”, and contains a link to the datasets. The credential information for this account has been added to the database. |
| Actual Output: | Same as expected. |

|  |  |
| --- | --- |
| **Test Case ID:** | **SIL\_CreateDataSource\_Test2\_Sunny\_Day** |
| Purpose: | To validate that the addition of a new account from Trello redirects the user to the proper page and adds the required information to the database even if the user is logged into Trello already. |
| Preconditions: | The user is on the Datasource page of the application. |
| Inputs: | 1. Select **Trello** datasource button. 2. Select **Allow** |
| Expected Output: | The system redirects the user to a confirmation page, lets the user know that the account has been added, displays the username “siluser”, and contains a link to the datasets. The credential information for this account has been added to the database. |
| Actual Output: | Same as expected. |

|  |  |
| --- | --- |
| **Test Case ID:** | **SIL\_CreateDataSource\_Test3\_Rainy\_Day** |
| Purpose: | To validate that if a user changes his/her mind mid addition of a datasource the system handles it appropriately. |
| Preconditions: | The user is on the Datasource page of the application. |
| Inputs: | 1. Select Trello datasource button. 2. Select Login 3. Username: siluser 4. Password: 123456 5. Select Deny |
| Expected Output: | The system should not redirect the user to a confirmation page nor should it save any information in the database |
| Actual Output: | Same as expected. |

**View Dataset:**

|  |  |
| --- | --- |
| **Test Case ID:** | **SIL\_ViewDataset\_Test1\_Sunny\_Day** |
| Purpose: | To validate that a given dataset is viewable. |
| Preconditions: | The user is on the Datasets page and the Trello siluser dataset has been added. |
| Inputs: | 1. Select get all labels. |
| Expected Output: | The system redirects the user to a page with the output from the get all labels Trello call. The data is displayed in a table format |
| Actual Output: | Same as expected. |

|  |  |
| --- | --- |
| **Test Case ID:** | **SIL\_ViewDataset\_Test2\_Sunny\_Day** |
| Purpose: | To validate that a given dataset is viewable. |
| Preconditions: | The user is on the Datasets page and the Trello siluser dataset has been added. |
| Inputs: | 1. Select get all cards. |
| Expected Output: | The system redirects the user to a page with the output from the get all cards Trello call. The data is displayed in a table format. |
| Actual Output: | Same as expected. |

|  |  |
| --- | --- |
| **Test Case ID:** | **SIL\_ViewDataset\_Test3\_Rainy\_Day** |
| Purpose: | To validate that a given dataset is not viewable if it does not exist. |
| Preconditions: | The user is on the Datasets page and the Trello siluser dataset has been added and all the cards have been deleted. |
| Inputs: | 1. Select get all cards |
| Expected Output: | The system should redirect the user to a get all cards page, but no output will be displayed in the table. |
| Actual Output: | Same as expected. |

* 1. Evaluation of Tests – evaluate how successful the tests were. Use a tabular form.

|  |  |
| --- | --- |
| **Test** | **Evaluation** |
| SIL\_CreateDatasource\_Test1\_Sunny\_Day | Successful |
| SIL\_CreateDatasource\_Test2\_Sunny\_Day | Successful |
| SIL\_CreateDatasource\_Test3\_Rainy\_Day | Successful |
| SIL\_ViewDataset\_Test1\_Sunny\_Day | Successful |
| SIL\_ViewDataset\_Test2\_Sunny\_Day | Successful |
| SIL\_ViewDataset\_Test3\_Rainy\_Day | Successful |

1. Glossary - define terms used in document, especially domain specific terms.

**API**: Application Programming Interface. Gives the ability to access code written by someone else in a formatted manner

**Django**: Web framework that is open source. Follows similar style to model-view-controller, except the framework become the controller. Architectural pattern is called model-view-template, with template relating to view in MVC.

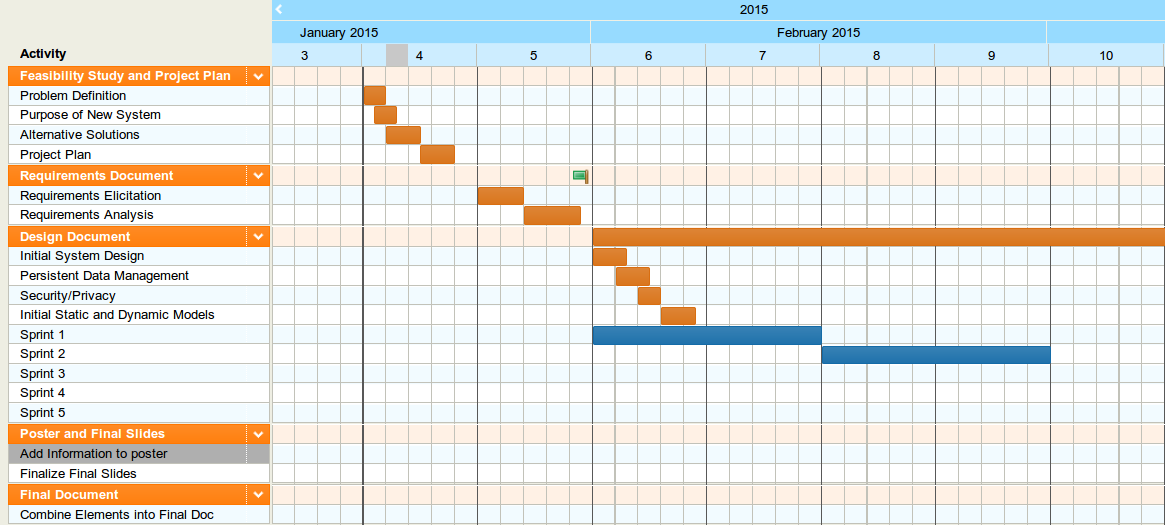
**Python**: General purpose language. Highly readable code that is not very verbose. Has dynamic typing.

**RDBMS:** Relational Database Management System.

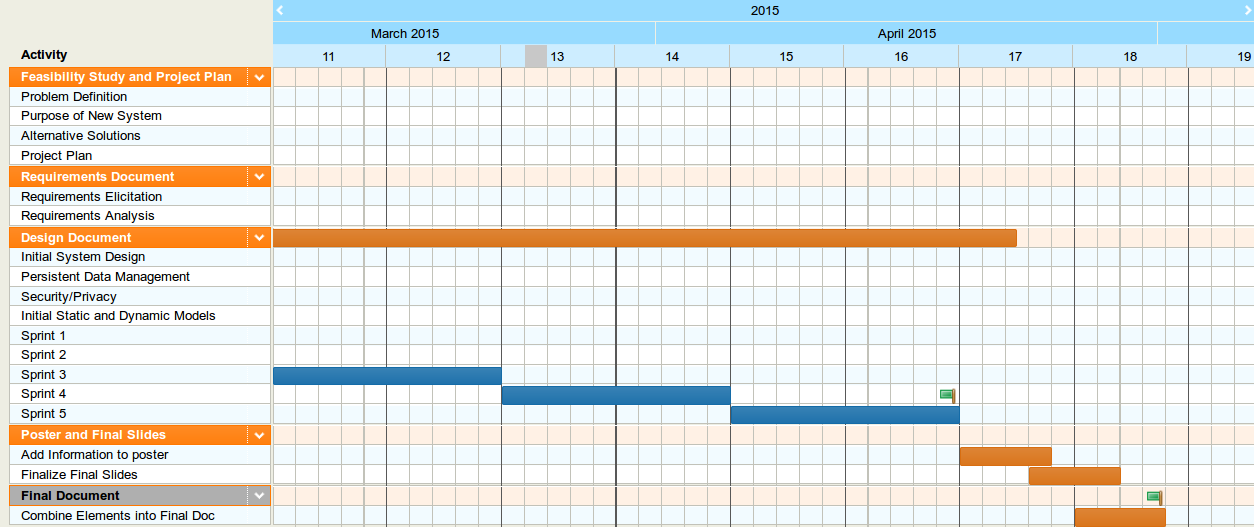
**SaaS:** Software-as-a-Service is defined as a software licensing model that is subscription based. It enables a user to have access to some business application that they can use for their own unique purposes.

**Trello**: Popular online project management tool. Allows for creation of boards with lists that can be assigned cards. The cards can be assigned to members. Labels can be created for categorization.

1. Appendix
   1. Appendix A - Project schedule



Gantt Chart 1



Gantt Chart 1

* 1. Appendix B – Use Cases

**Title:** View Data Source

**Description:** After authentication, User has the ability to make API calls on that account in order to retrieve data (Card #110)

**Actor:** SaaS-Integration-Library User (SIL user) and third party API

**Preconditions:** Data source for API calls has been verified and SIL user is on API call page.

**Steps:**

Step 1: User clicks link for an API call.

Step 2: System gets the Api model object for the call.

Step 3: System makes call to API behind the scenes, creating an API object to make the call.

Step 4: API returns data from call.

Step 5: System loads new page with data returned from API call.

**Postconditions:** User has been passed to a page containing the returned data from specific API call.

**Title:** Create Data Source

**Description:** After authentication, User has the ability to create a data source (Card #160)

**Actor:** SaaS-Integration-Library user (SIL user) and third party API

**Preconditions:** User has an account with third party and has navigated to Create Data Source page

**Steps:**

Step 1: User clicks link for an API.

Step 2: System requests a token on the users behalf from the API.

Step 3: After user accepts SIL to have access to his/her account, system creates ApiCredential and Api object with token information.

Step 4: System redirects to confirmation page.

**Postconditions:** User has been passed to a page displaying successful creation.

**Title:** Get My Cards

**Description:** Filter to get cards only for user from the get\_all\_cards method (which returns all cards user can see)

**Actor:** SaaS-Integration-Library user (SIL user)

**Preconditions:** The user has selected the call "Get all cards" from the dataset and is on the "Get all cards" page

**Steps:**

Step 1: The user clicks the metric which says "My Cards"

Step 2: The system filters the cards

Step 3: The system reloads the table

**Postconditions:** The table updates and displays only the cards associated with the user

**Title:** Get Due in Seven

**Description:** Filter to get cards that are due in seven days from the get\_all\_cards method (which returns all cards user can see)

**Actor:** SaaS-Integration-Library user (SIL user)

**Preconditions:** The user has selected the call "Get all cards" from the dataset and is on the "Get all cards" page

**Steps:**

Step 1: The user clicks the metric which says "Due in Seven"

Step 2: The system filters the cards

Step 3: The system reloads the table

**Postconditions:** The table updates and displays only the cards due in seven days

**Title:** Get Past Due

**Description:** Filter to get cards which are past due from the get\_all\_cards method (which returns all cards user can see)

**Actor:** SaaS-Integration-Library user (SIL user)

**Preconditions:** The user has selected the call "Get all cards" from the dataset and is on the "Get all cards" page

**Steps:**

Step 1: The user clicks the metric which says "Past Due"

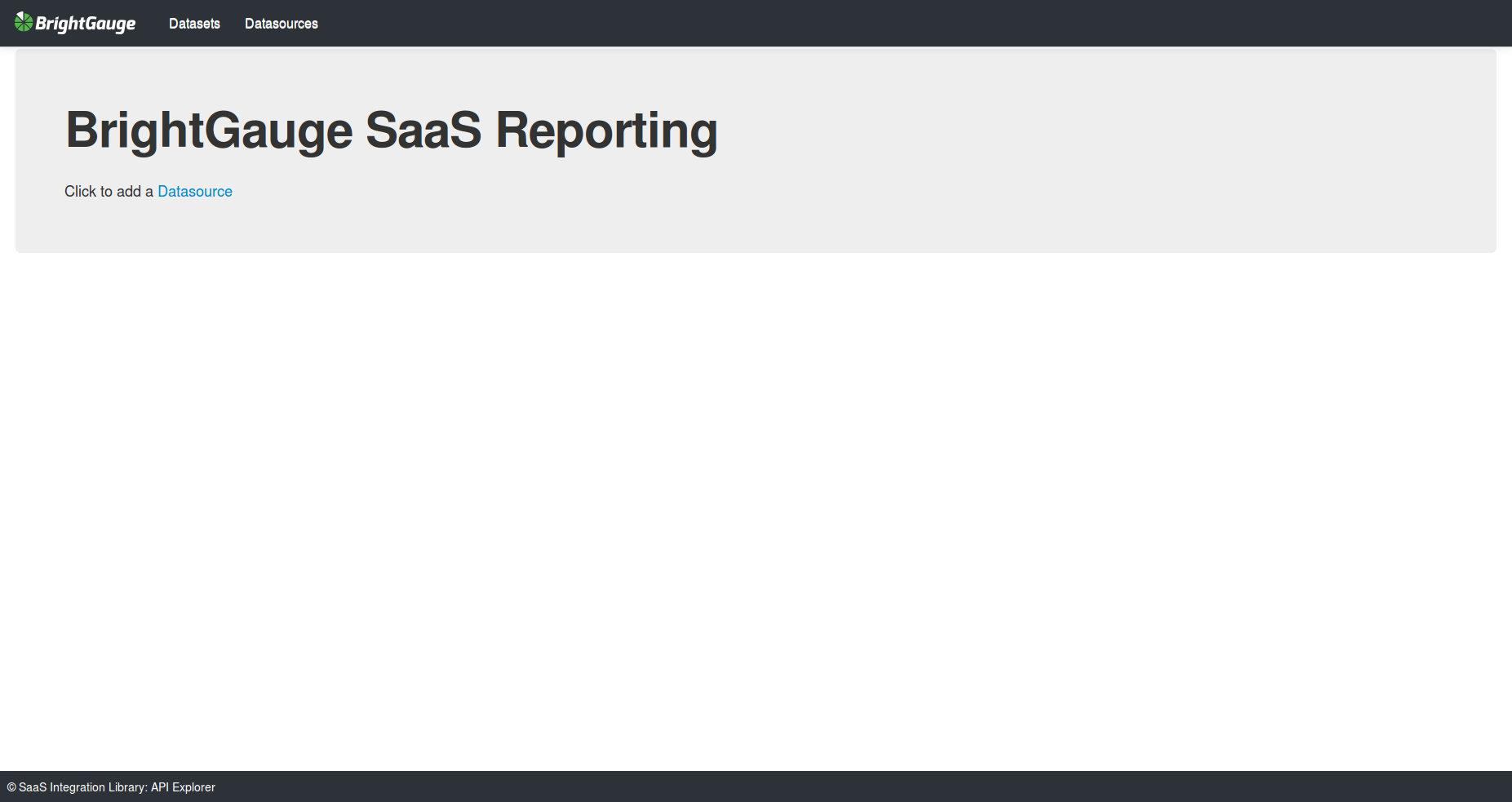
Step 2: The system filters the cards

Step 3: The system reloads the table

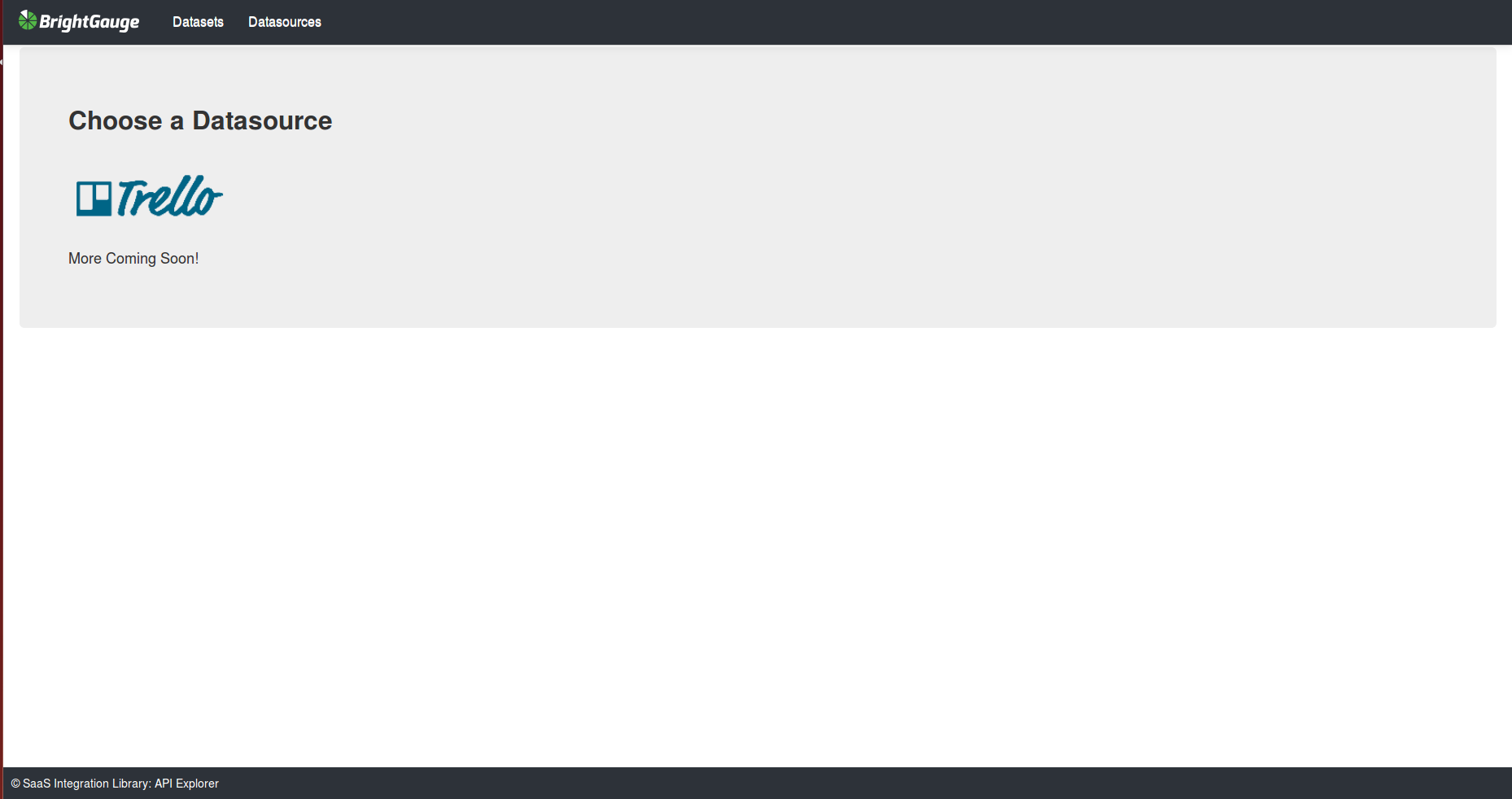
**Postconditions:** The table updates and displays only the cards which are past due

* 1. Appendix C – User Interface designs

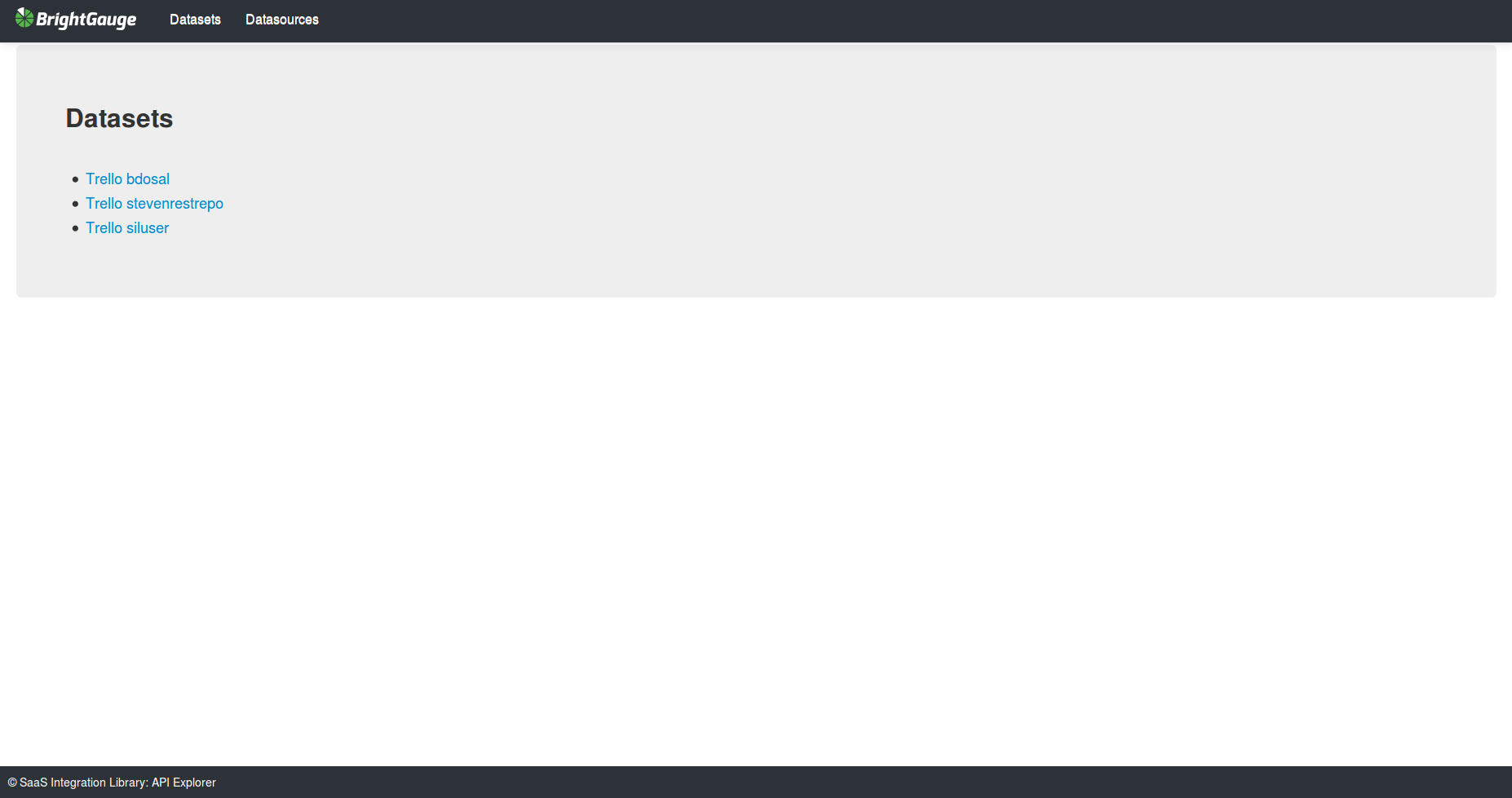
The following are screenshots from the user interface:



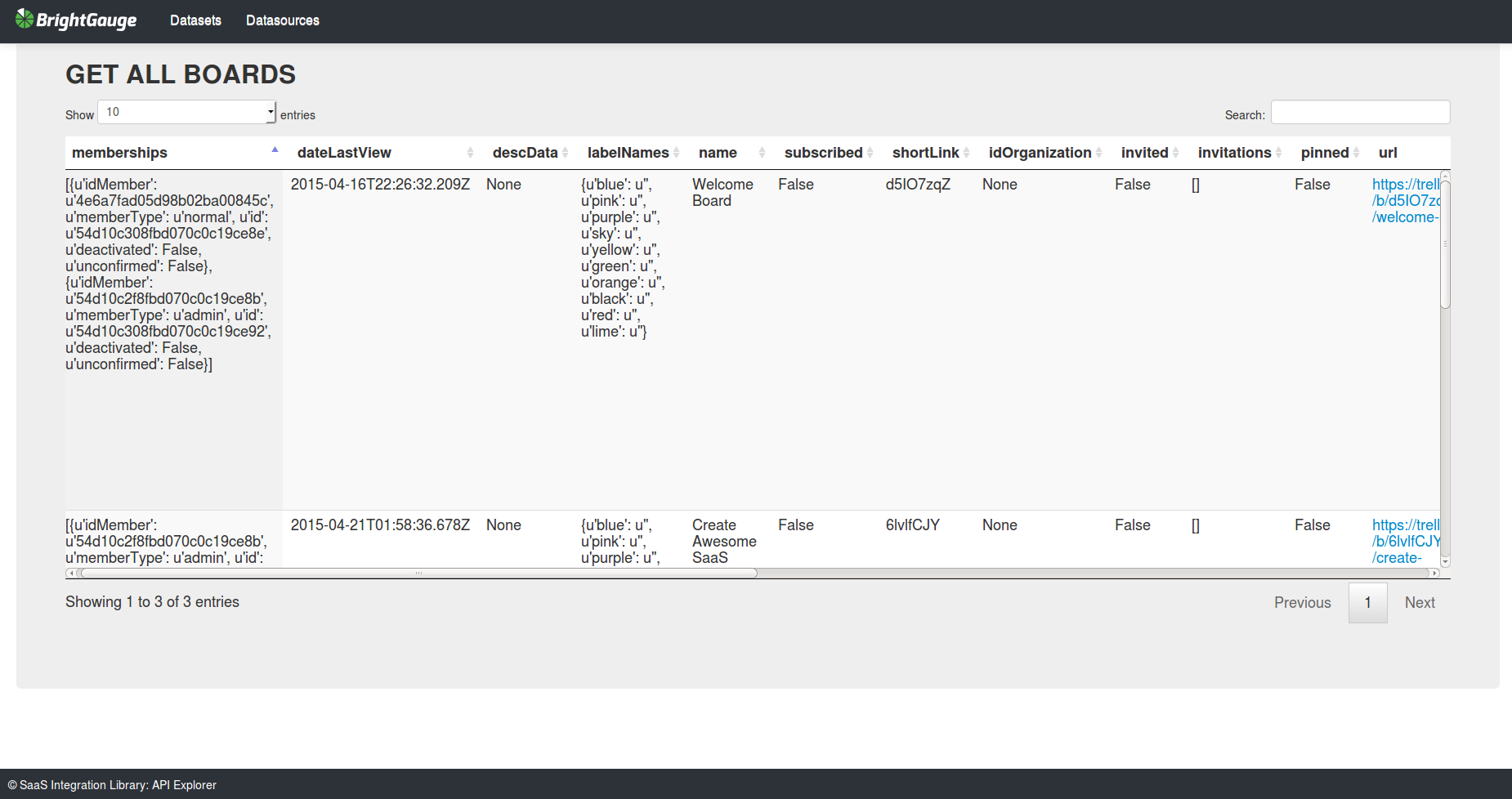
Index User Interface



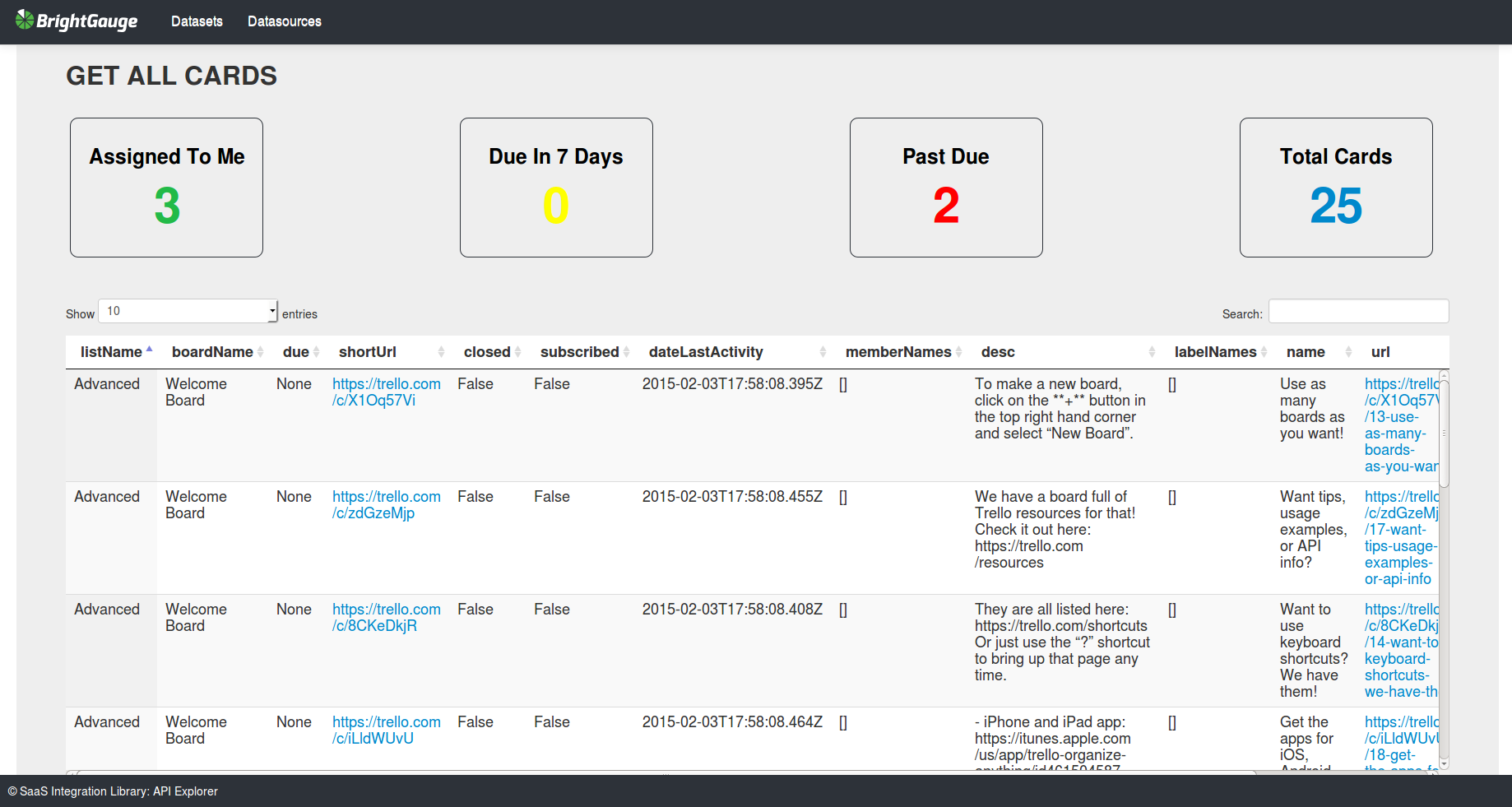
Datasource User Interface



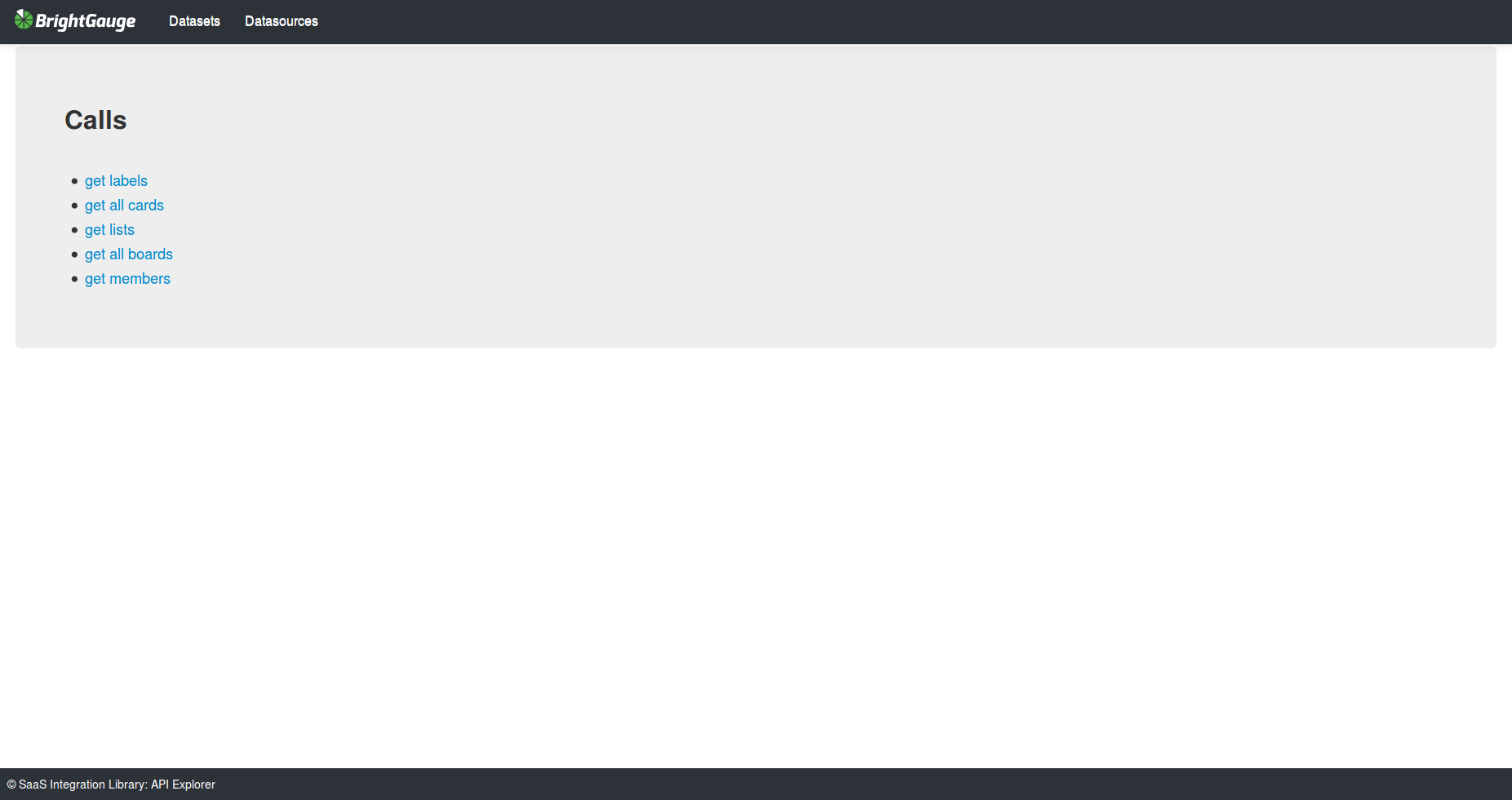
Datasets User Interface



Get All Boards Call User Interface



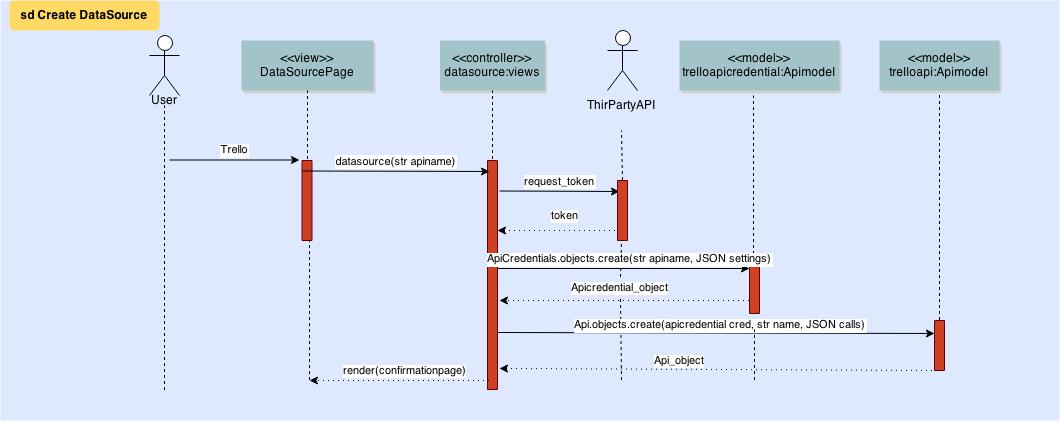
Get All Cards Call User Interface



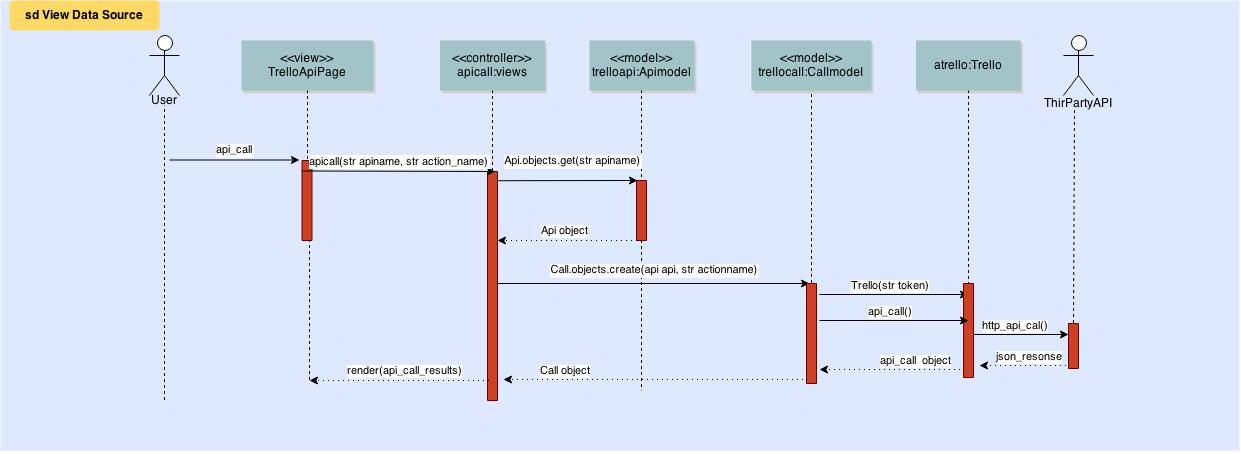
Calls User Interface

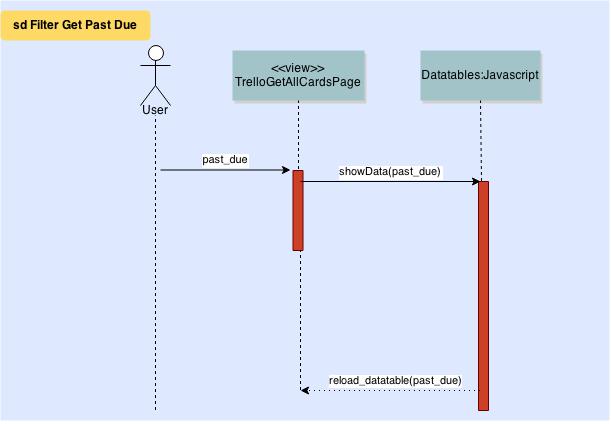
* 1. Appendix D – Analysis models

The following are sequence diagrams for the use cases:

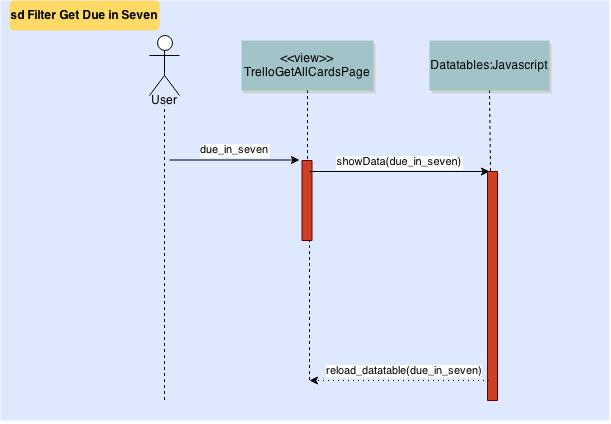


Sequence Diagram for Create Datasource

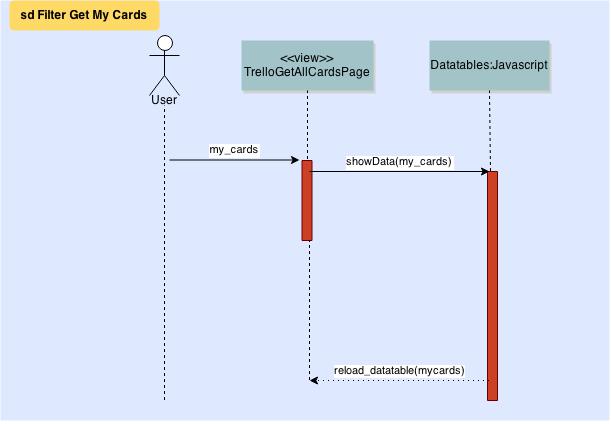
 Sequence Diagram for View Datasource



Sequence Diagram for Get Past Due

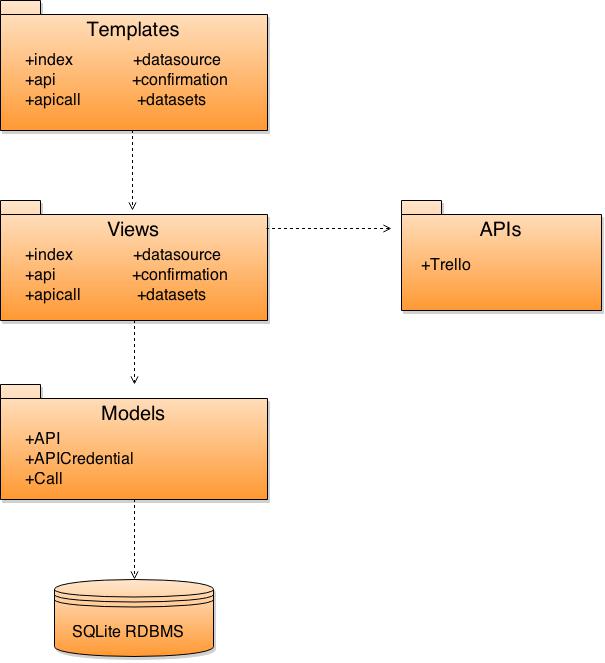


Sequence Diagram for Get Due in Seven

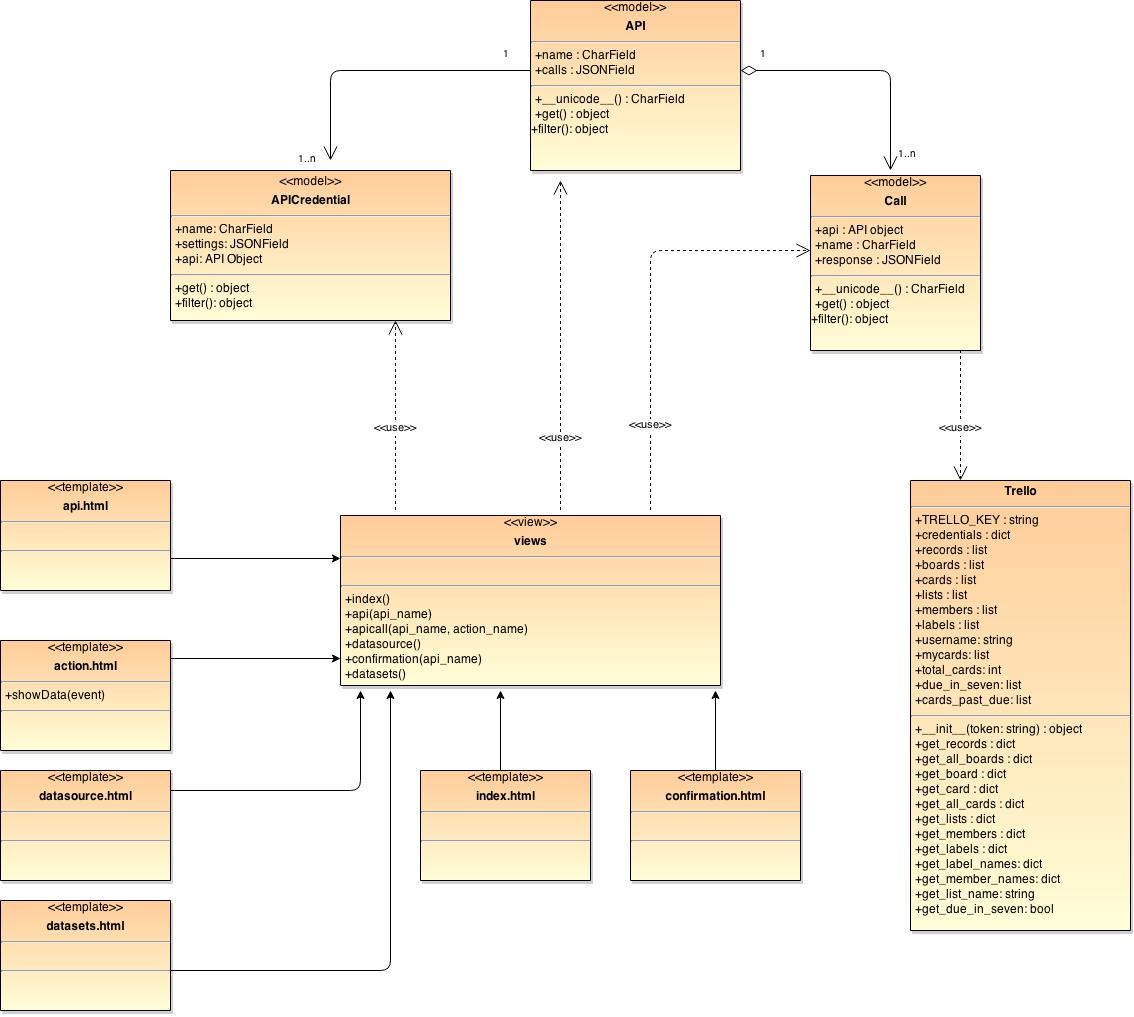


Sequence Diagram for Get My Cards

* 1. Appendix E – Design models



Package Diagram



Class Diagram

* 1. Appendix F

Code for Views subsystem:

def index(request)

def datasource(request)

def confirmation(request, api\_name)

def datasets(request)

def api(request, api\_cred)

def apicall(request, api\_cred, action\_name)

Code for Trello class:

def \_\_init\_\_(self, token)

def make\_call(self, address)

def get\_records(self)

def get\_all\_boards(self)

def get\_board(self, id)

def get\_all\_cards(self)

def get\_lists(self)

def get\_labels(self)

def get\_members(self)

* 1. Appendix G – Documented code for test drivers and stubs.

The following is python code for testing:

import unittest

import pprint

import json

from trello import Trello

from mock import Mock

from mock import patch

from mock import PropertyMock

class TrelloTestCase(unittest.TestCase):

"""Trello Api tester"""

TEST\_TOKEN = "36b68eef1b52420e5731962cdb0bef1e8f152874b10f6036ed30bd9f117dc2fe"

TEST\_BOARD\_ID = "54d10ee46bd364a1e6f063ea"

TEST\_CARD\_ID = "54d10f4c8777eefd6328ad43"

def setUp(self):

self.trello = Trello(TrelloTestCase.TEST\_TOKEN)

def test01\_getRecords\_unit(self):

self.trello.make\_call = Mock()

self.trello.make\_call.return\_value.json.return\_value = {'record':'some record'}

print("User Records:")

pprint.pprint(self.trello.get\_records())

assert self.trello.record == {'record' : 'some record'}

# def test02\_getAllBoards\_unit(self):

# print("User Boards:")

# mock = Mock()

# #self.trello.get\_all\_boards = Mock(return\_value={"boards":"Got boards!"})

# pprint.pprint(self.trello.get\_all\_boards(mock))

# mock.requests.get.assert\_called\_with()

# #assert self.trello.boards == {"boards":"Got boards!"}

#

#

# def test03\_getAllBoards\_integrate(self):

# print("User Boards:")

# pprint.pprint(self.trello.get\_all\_boards())

# def test03\_getBoard(self):

# print("User Test Board:")

# pprint.pprint(self.trello.get\_board(TrelloTestCase.TEST\_BOARD\_ID))

#

def test04\_getAllCards(self):

print ("Cards:")

pprint.pprint(self.trello.get\_all\_cards())

# def test05\_getCard(self):

# print("User Test Card:")

# pprint.pprint(self.trello.get\_card(TrelloTestCase.TEST\_CARD\_ID))

#

# def test06\_getLists(self):

# print("User Lists:")

# pprint.pprint(self.trello.get\_lists())

#

# def test07\_getMembers(self):

# print("User Members:")

# pprint.pprint(self.trello.get\_members())

#

# def test08\_getLabels(self):

# print("User Labels:")

# pprint.pprint(self.trello.get\_labels())

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

* 1. Appendix H – Diary of meeting and tasks

**Diary Entry** - January 22 2015

**Location:** BrightGauge

**Start time:** 10:00 AM

**End time:** 11:30 AM

**In attendance:** Adam, Brian, Orlando, Steve

**Agenda:** Product introduction, User stories discussion, why this product

**Summary of discussion:** Discussed use cases pertaining to SaaS Integration Library, particularly Trello data source retrieval by means of authentication and API calls.

**Assigned tasks:** Adam - Research Trello API, begin working on documentation and Mingle set up

**Diary Entry** - January 29 2015

**Location:** BrightGauge

**Start time:** 10:00 AM

**End time:** 11:30 AM

**In attendance:** Adam, Brian, Orlando, Steve

**Agenda:** Go over possible architecture, user interface envisioned by product owner

**Summary of discussion:** Mapped out basic data management system, user interface design, git branching

**Assigned tasks:** Adam - Continue documentation, learn about Django

**Diary Entry** - February 12, 2015

**Location:** BrightGauge

**Start time:** 3:30 PM

**End time:** 4:00 PM

**In attendance:** Adam, Brian, Orlando, Steve

**Agenda:** Sprint Review

**Summary of discussion:** User Stories: attribute/fields list high level view without data Highcharts

Next time: more code reviews demo at end of sprint

**Assigned tasks:** Adam - --variables in init --no camel case --format strings --requests.get takes params --checks inside of get all cards try, except move KEY to Settings

**Diary Entry** - February 25, 2015

**Location:** BrightGauge

**Start time:** 3:30 PM

**End time:** 4:00 PM

**In attendance:** Adam, Brian, Orlando, Steve

**Agenda:** Sprint Review

**Summary of discussion:** shortcut to render\_to\_response, check newer docs called render()

\_\_iexact() case insensitive searching

views - apicall - just delete query no need to check

no need to change response to json, keep as data obj

python mock library

get list of items to pass to template, then get keys() and then loop through to get items()

**Assigned tasks:** Adam - above recommendations in addition to assigned user stories

**Diary Entry** - March 19, 2015

**Location:** BrightGauge

**Start time:** 3:30 PM

**End time:** 4:00 PM

**In attendance:** Adam, Brian, Orlando, Steve

**Agenda:** Sprint Review

**Summary of discussion:** discussed sprint review and job offer

**Assigned tasks:** Adam - Overflow hidden css element Use oauth for Trello send javascript issues to Orlando and work on fixing

**Diary Entry** - April 2, 2015

**Location:** BrightGauge

**Start time:** 3:30 PM

**End time:** 4:00 PM

**In attendance:** Adam, Brian, Orlando, Steve

**Agenda:** Sprint Review

**Summary of discussion:** discussed sprint review

**Assigned tasks:** Adam - Footer, move css to dedicated folder, rm footer line, Datatables js, grab labels out of all calls (e.g. get all cards, get lists)

**Diary Entry** - April 16, 2015

**Location:** BrightGauge

**Start time:** 3:30 PM

**End time:** 4:00 PM

**In attendance:** Adam, Brian, Orlando, Steve

**Agenda:** Sprint Review

**Summary of discussion:** discussed sprint review

**Assigned tasks:** Adam - work on dynamic spacing of divs for metrics on get all cards page

1. References

* **Draw IO:** <https://www.draw.io/> used for all diagrams except Gantt.
* **Tango With Django 1**.7: <http://www.tangowithdjango.com/book17/>, great resource for learning the Django framework.
* **Tom's Planner**: <http://www.tomsplanner.com/> used for Gantt chart creation