Software Design Description for Mashbot

George D'Andrea Andrew Gall Josiah Kiehl Cody Ray Vito Salerno

January 31, 2010

Revision History

Name	Date	Reason for Changes	Version
George D'Andrea, Andrew Gall, Josiah Kiehl, Cody Ray, Vito Salerno	17 January 2010	Initial Version	1.0

Contents

Chapter 1

Introduction

1.1 Purpose

This document serves to expand upon the requirement into implementation details and technology choices. This document should be referenced when specific features are being implemented.

1.2 Scope

This covers the general architecture of Mashbot, as well as the design decisions used to apply that architecture via various appropriate technologies, libraries and frameworks.

1.3 Definitions, Acronyms, and Abbreviations

- MVC — Model View Controller
- SASS — Syntactically Awesome Style Sheets
- HAML — XHTML Abstraction Markup Language
- XHTML — Extensible Hypertext Markup Language
- CSS — Cascading Style Sheets
- API — Application Programmming Interface

1.4 Context Diagram

Chapter 2

Architecture

2.1 Overview

Generally, Mashbot will be implemented using a strict Model-View-Controller architecture. This is augmented by the inclusion of the Presentation and Aggregation Platform, the purpose of which is to abstract the interaction with external service APIs from the application as a whole, thus allowing a pure MVC architecture to be implemented, increasing maintainability, extensibility and accessibility.

2.2 Four-Tier Architecture

- Data Layer / Model
- Presentation Layer / View
- Business Layer / Controller
- Publishing and Aggregation Platform

2.3 Service-Oriented Architecture

Mashbot will be implemented as two distinct yet related services. The Campaign Manager will handle the interaction between the user and the data the Campaign Manager is concerned with, where the Presentation and Aggregation Platform will handle the interaction between external service APIs and the Campaign Manager.

2.4 Survey of Technologies Used

2.4.1 Campaign Manager

- Presentation Layer
  - HAML — HTML replacement markup language, for building web layout structure.
  - SASS — CSS replacement stylesheets, for applying visual styles to the layout built in HAML.
  - jQuery — JavaScript library which provides cross-browser compatibility as well as streamlined Ajax request handling.
  - Google Chart API — Public service provided by Google which generates many different kinds of charts and graphs.

- Business Layer
  - Ruby — Dynamic programming language.
  - Rails — Web application framework written in Ruby which provides a concise Model-View-Controller architecture.
  - Heroku — Rails engine which provides enhanced production deployment via Rails compilation, a fast readonly filesystem, and horizontal scaling.
- Data Layer
  - ActiveRecord — Component of Rails which provides the Active Record pattern of data access, creating data model objects and relationships for interacting with resources in a database.
  - MySQL — Fast and free relational database which plugs into Rails without effort.

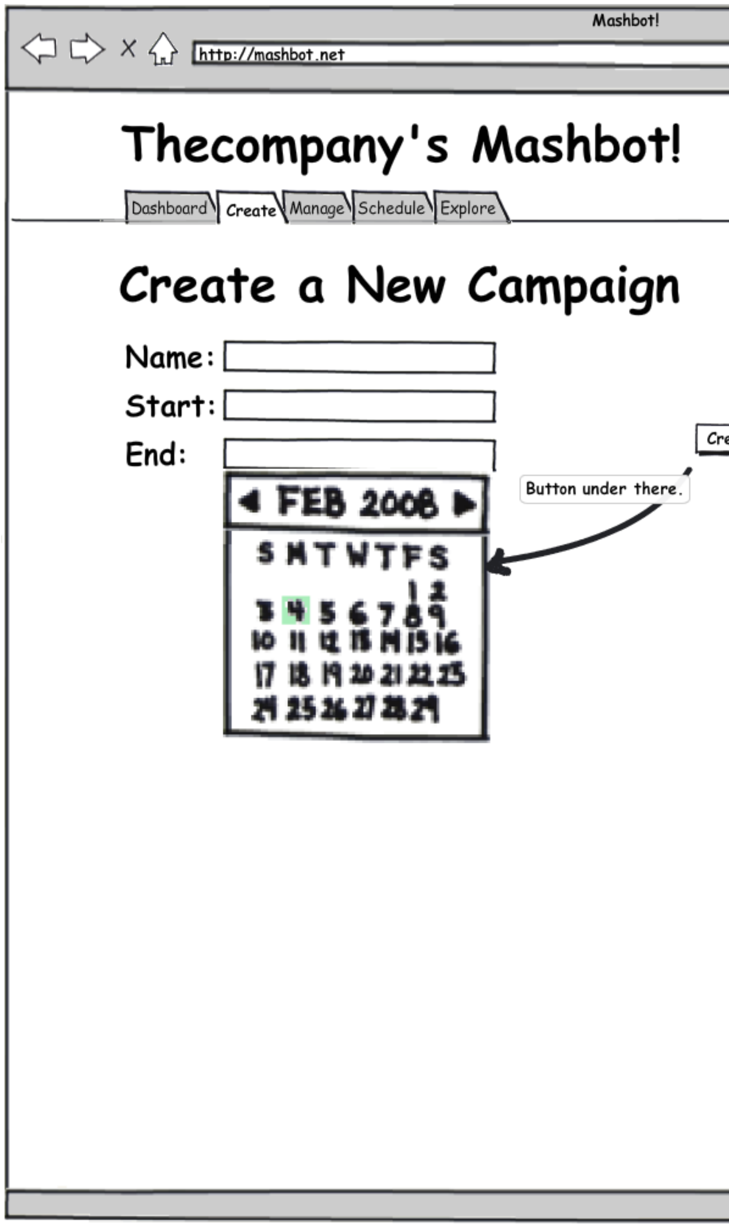
2.4.2 Publishing and Aggregation Platform

- TODO:TODO:TODO:TODO:TODO:TODO:TODO:TODO:TODO:TODO:TODO:TODO:TODO:TODO:TODO:TOD
  Put tech stuff here.

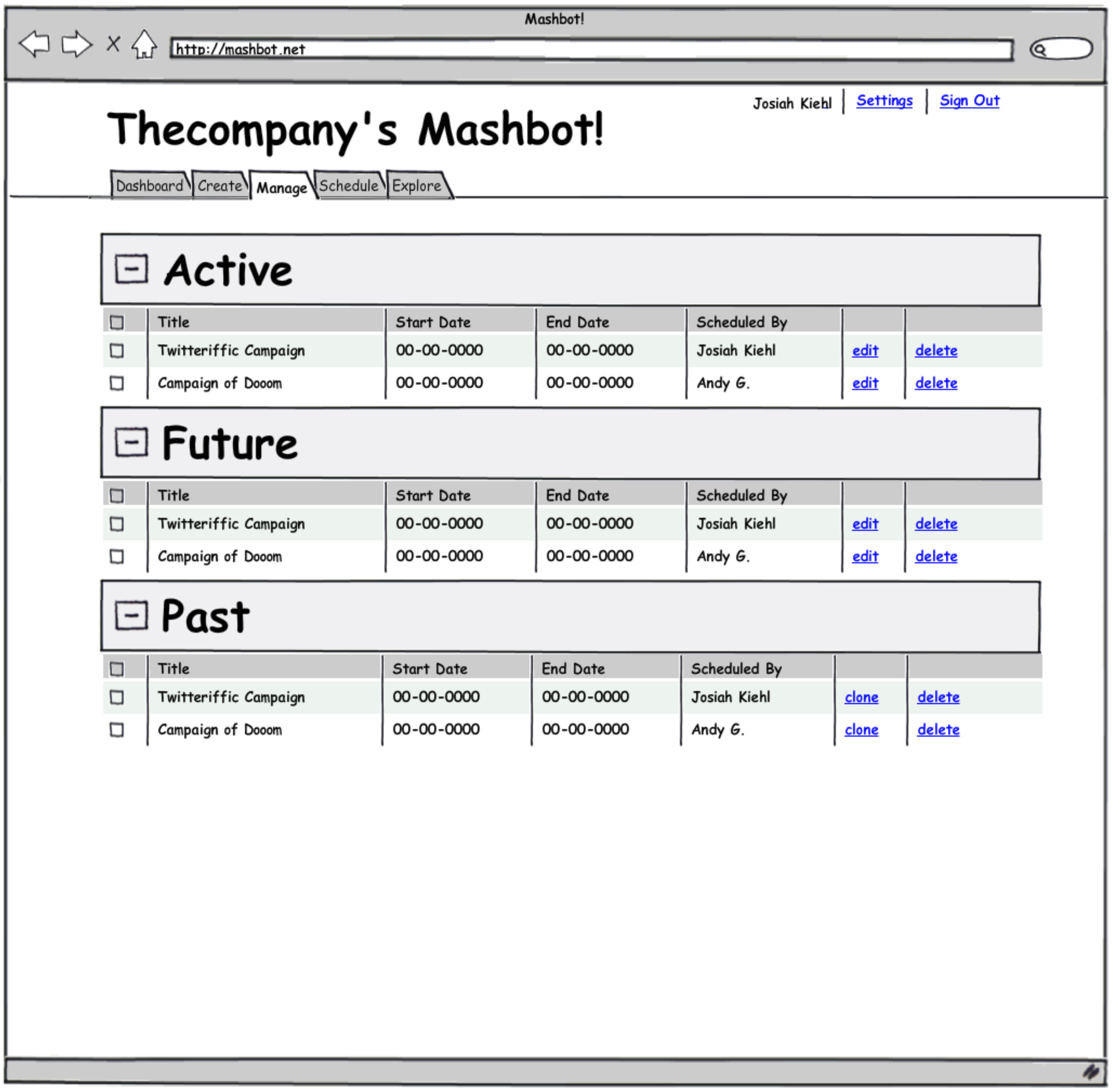
2.5 Presentation Layer Components

2.5.1 Campaign Views

Campaigns are accessed via the Create and Manage tabs on the primary navigation tabs. Create is for the Create view, Manage is for List, Show and Edit.



- Create — This is where users can create new campaigns.
- List — This is where users can view, update or delete existing campaigns.



Thecompany's Mashbot!

Active

☐	Title	Start Date	End Date	Scheduled By		
☐	Twitteriffic Campaign	00-00-0000	00-00-0000	Josiah Kiehl	edit	delete
☐	Campaign of Dooom	00-00-0000	00-00-0000	Andy G.	edit	delete

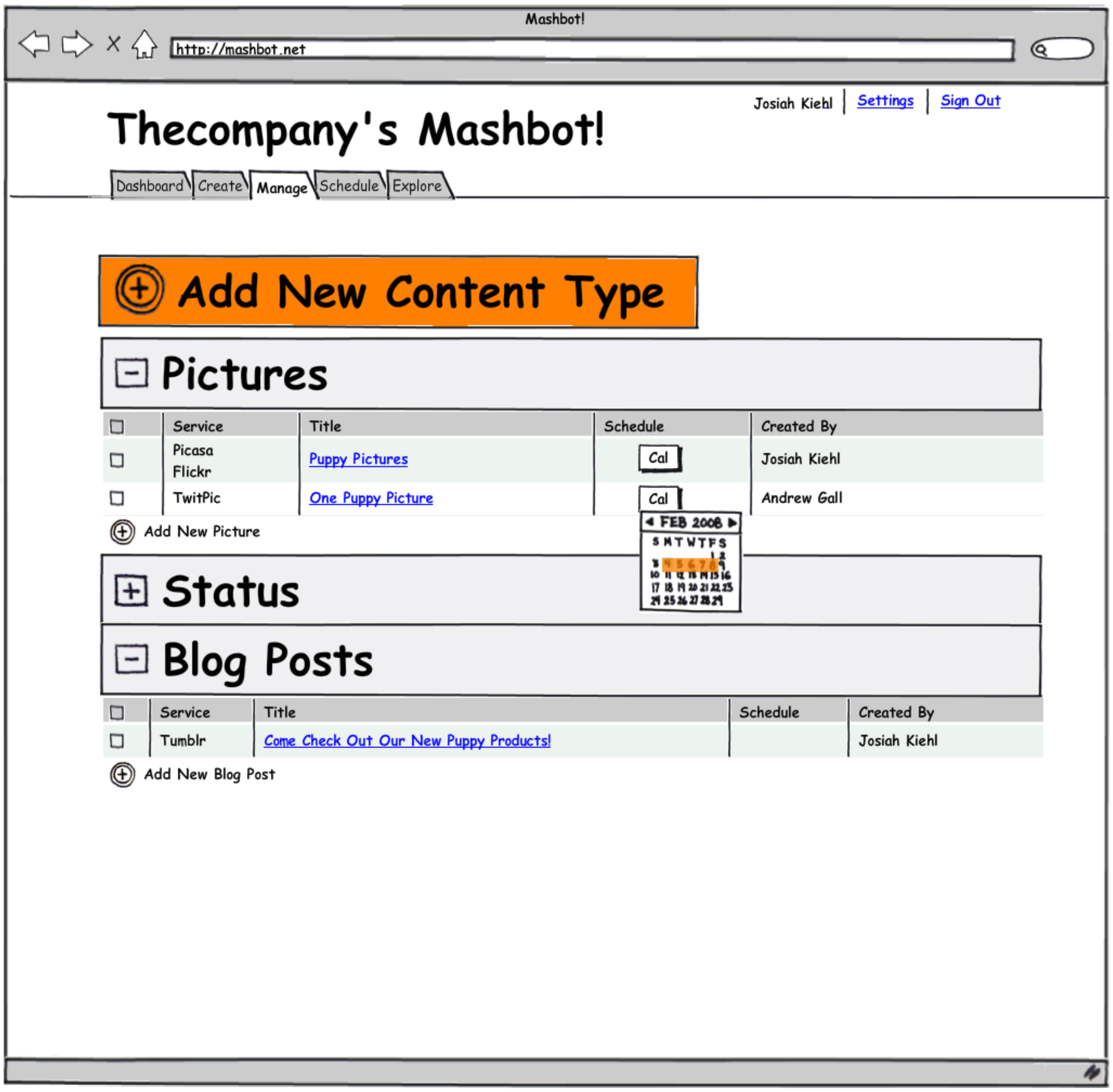
Future

☐	Title	Start Date	End Date	Scheduled By		
☐	Twitteriffic Campaign	00-00-0000	00-00-0000	Josiah Kiehl	edit	delete
☐	Campaign of Dooom	00-00-0000	00-00-0000	Andy G.	edit	delete

Past

☐	Title	Start Date	End Date	Scheduled By		
☐	Twitteriffic Campaign	00-00-0000	00-00-0000	Josiah Kiehl	clone	delete
☐	Campaign of Dooom	00-00-0000	00-00-0000	Andy G.	clone	delete

- Show — This view is what is shown when the user wants to view an existing campaign via the Show view. This is also where the Content pieces will be listed.



	Service	Title	Schedule	Created By
☐	Picasa Flickr	Puppy Pictures	Cal	Josiah Kiehl
☐	TwitPic	One Puppy Picture	Cal	Andrew Gall

Add New Picture

	Service	Title	Schedule	Created By
☐	Tumblr	Come Check Out Our New Puppy Products!		Josiah Kiehl

Add New Blog Post

- Edit — This is virtually the same view as Create, however this will be prepopulated with the existing content of the given Campaign.

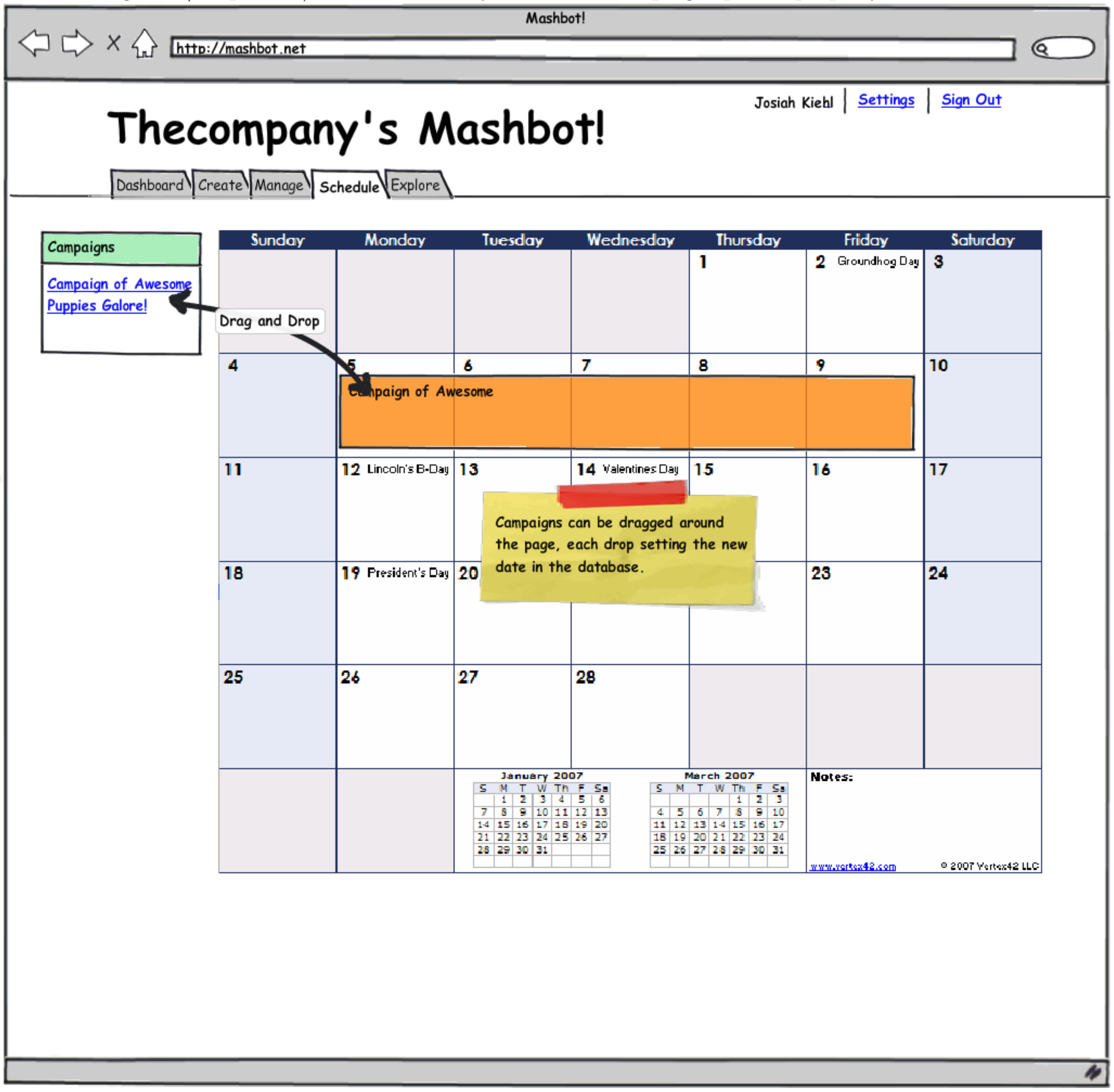
2.5.2 Content Views

Content pieces are included inside Campaigns. These views are accessible via the Show view of a Campaign for the corresponding Campaign id.

- Create — When on the Show view of a given campaign, the user can enter the Create view for Content.
- Show — This is how the user previews the Content they have created.
- Edit — This is virtually the same view as Create, however this will be prepopulated with the existing content of the given Content.

2.5.3 Scheduling Views

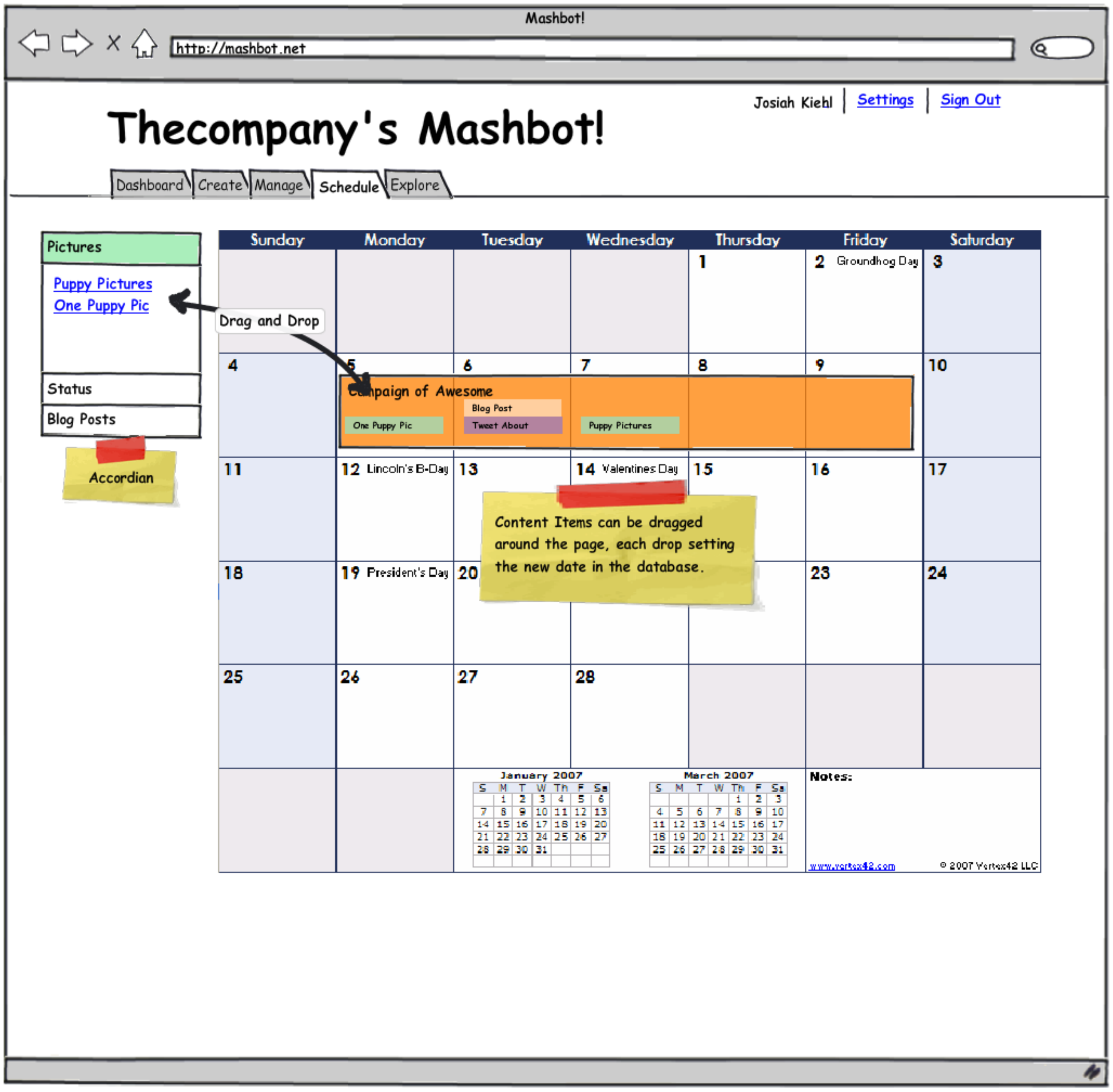
- Primary Scheduling View — consists of a list of Campaigns available to be scheduled (ie: they do not have existing start/stop dates) as well as already scheduled Campaigns placed properly on the calendar.



Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
				1	2 Groundhog Day	3
4	5	6	7	8	9	10
11	12 Lincoln's B-Day	13	14 Valentines Day	15	16	17
18	19 President's Day	20			23	24
25	26	27	28			

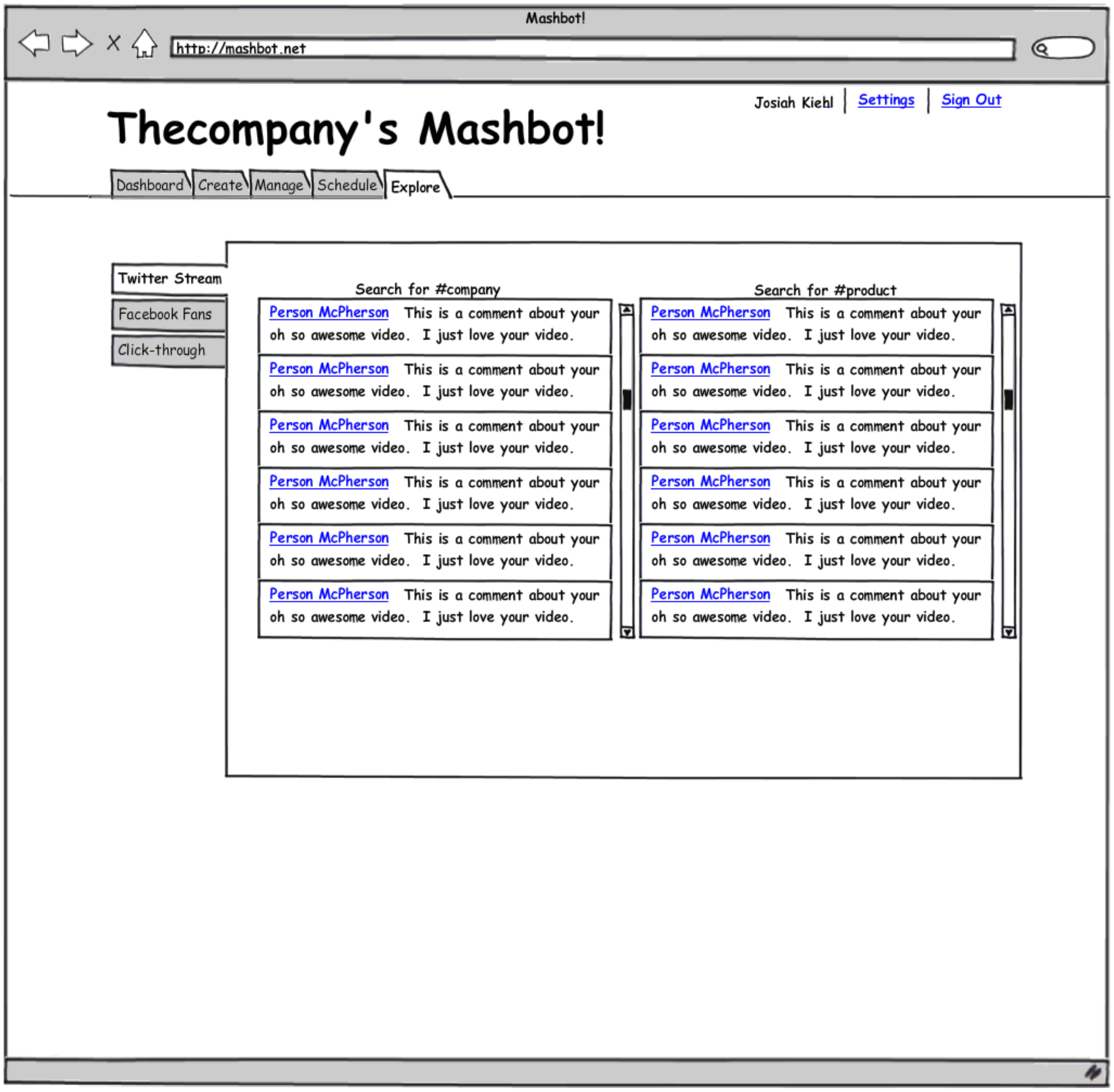
Campaigns can be dragged around the page, each drop setting the new date in the database.

- Content Scheduling View — similar to the Primary Scheduling View, however the items available to be scheduled here are the individual content pieces of the Campaign. This is accessed via selecting the Campaign from the calendar, or via the List Campaign or Show Campaign views.

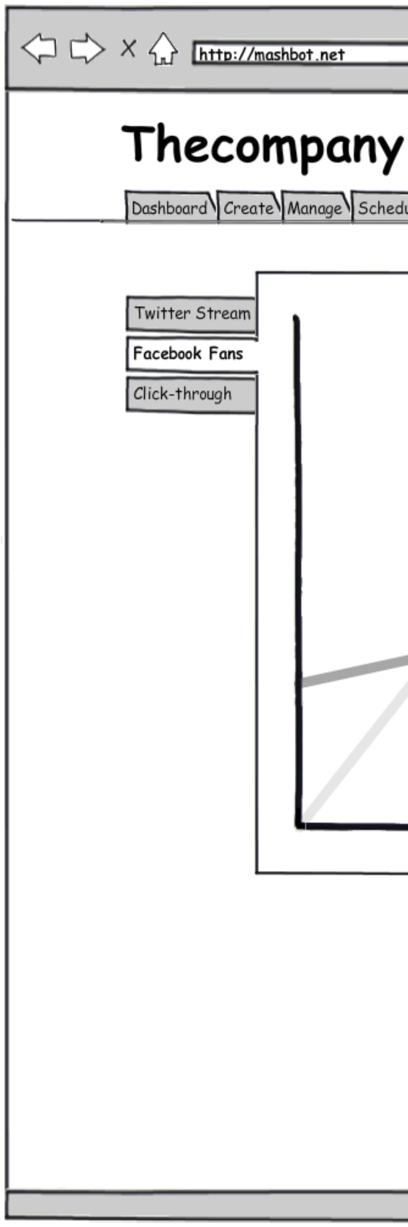


2.5.4 Explore View

Here the user will have several available "Insight Views." These are dependent on which plugins exist in the Publishing and Aggregation Platform, however there will be some provided by the Campaign Manager alone. These will provide charts that are layerable, such that multiple charts can be seen on the same graph.



- Plugin Independent
  - Clickthrough tracking — Any time a link is generated via Mashbot, it is given a special redirecting URL that will allow Mashbot to track how many times the link has been clicked.
  - Rate of publishing — How often does the user tweet/blog/etc. This will most likely be used to correlate frequency with user engagement.
- Plugin Dependent



– Facebook Fan tracking — A line chart of how many fans the user's fan page has.
– Twitter Follower tracking — A line chart of the number of twitter followers the user's twitter account has.
– Number of times retweeted — A line chart of the number of times a tweet of the user's has been retweeted.

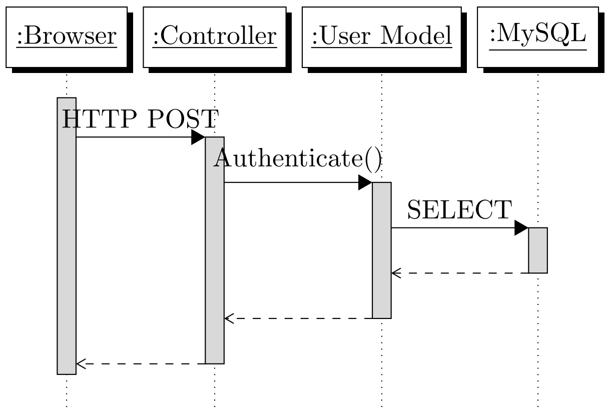


Figure 2.1: Log in to authenticate the user.

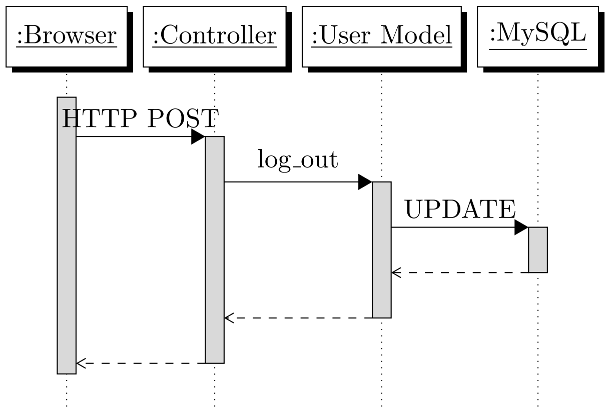


Figure 2.2: End authenticated session by logging out.

2.6 Business Layer Components

2.7 Session and Authentication

2.7.1 Log In

2.7.2 Log Out

2.7.3 OAuth

2.7.4 Openid

2.7.5 Session Handling

2.8 Data Layer Components

2.9 External Components

2.9.1 Publishing and Aggregation Targets

- Twitter

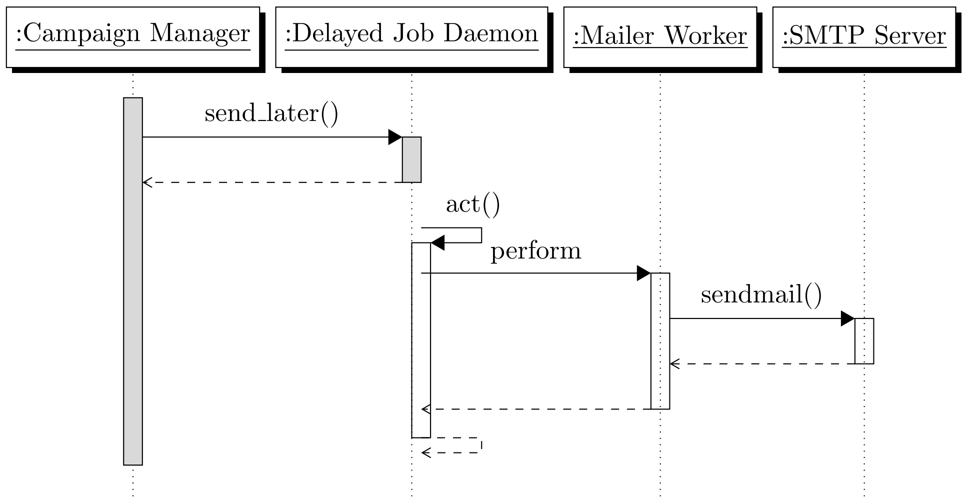


Figure 2.3: Send email (Requirements 3.1.1)

- Tumblr
- Wordpress
- TODO: etc.

2.9.2 Email/SMTP Service

Chapter 3

Design Features

3.1 External Authentication via Openid

3.2 Create User Account

This is a basic CRUD operation: Create User Account.

3.3 Update User Account

This is a basic CRUD operation: Update User Account.

3.4 Delete User Account

This is a basic CRUD operation: Delete User Account.

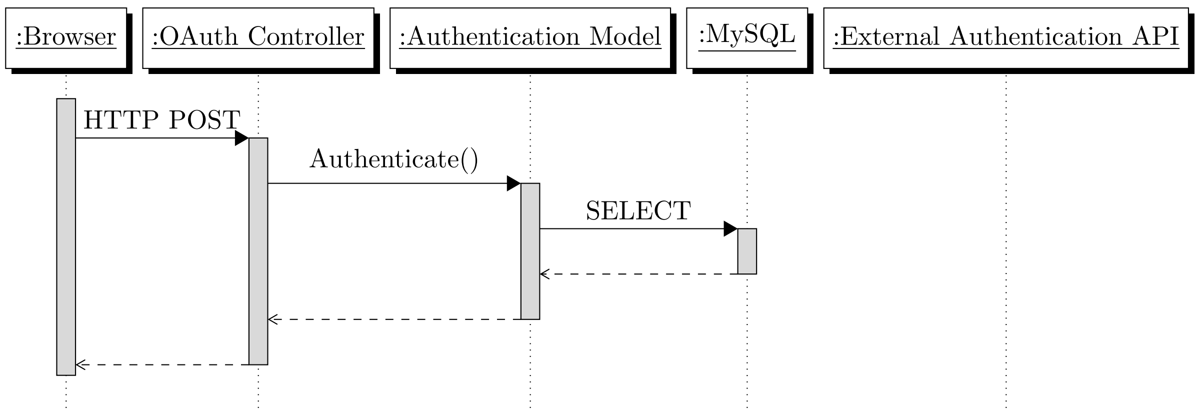


Figure 3.1: Authentication via OAuth with stored token.

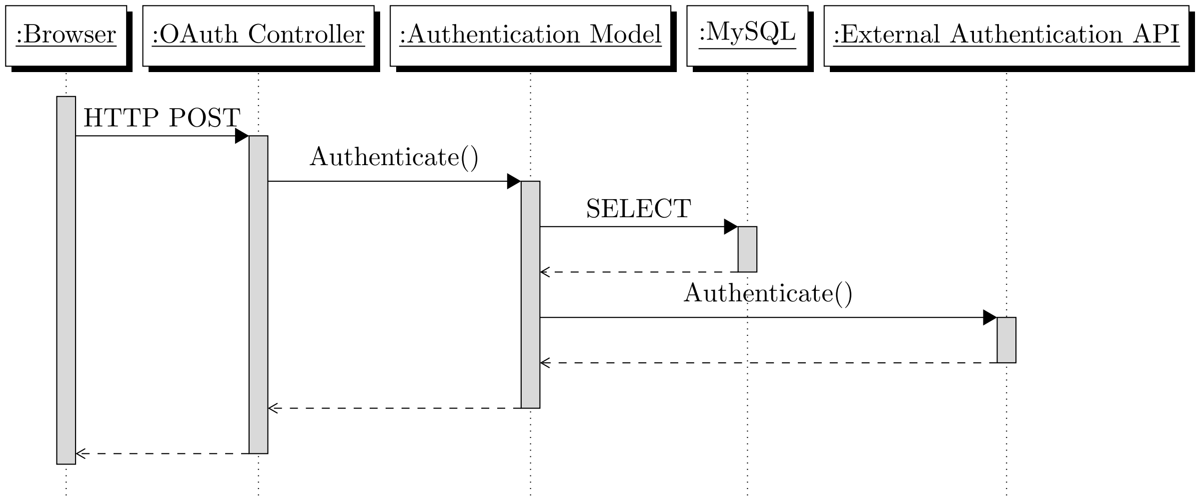


Figure 3.2: Authentication via OAuth without stored token.

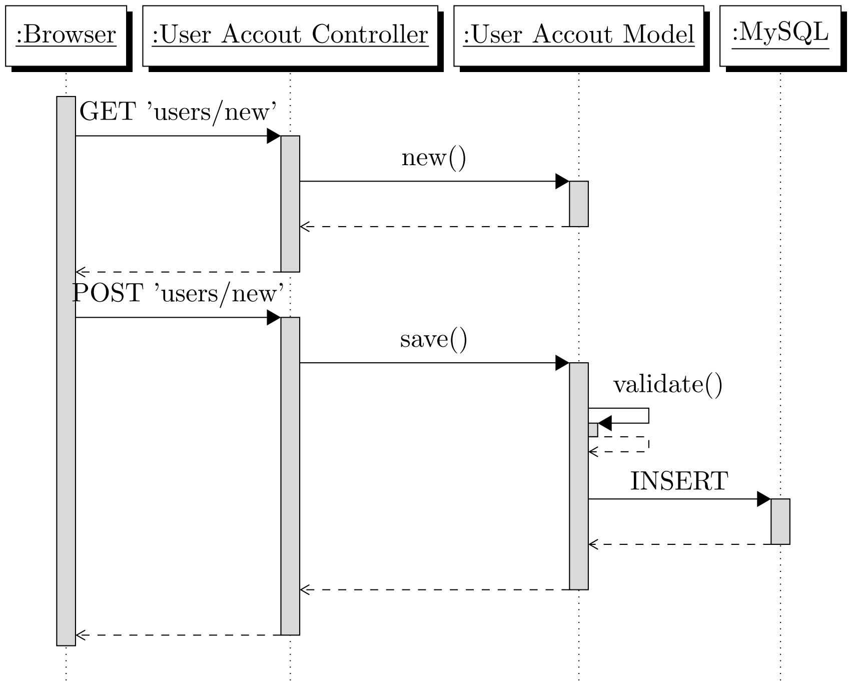


Figure 3.3: Create User Accout

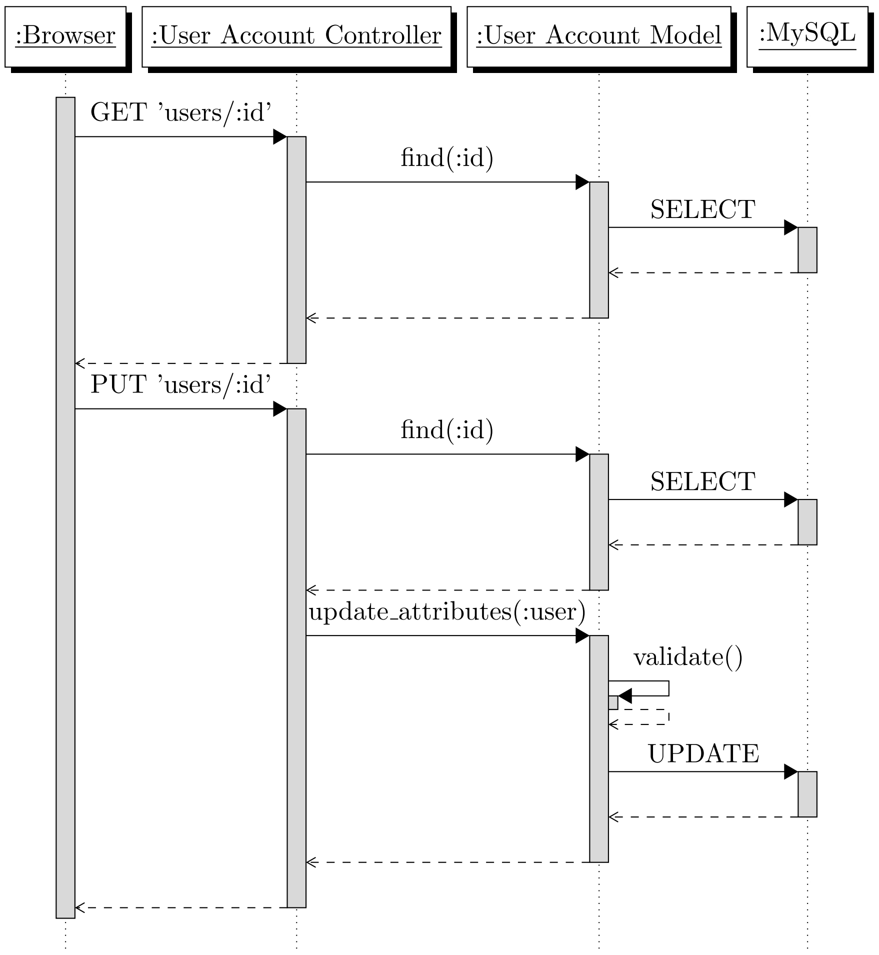


Figure 3.4: Update a User Account

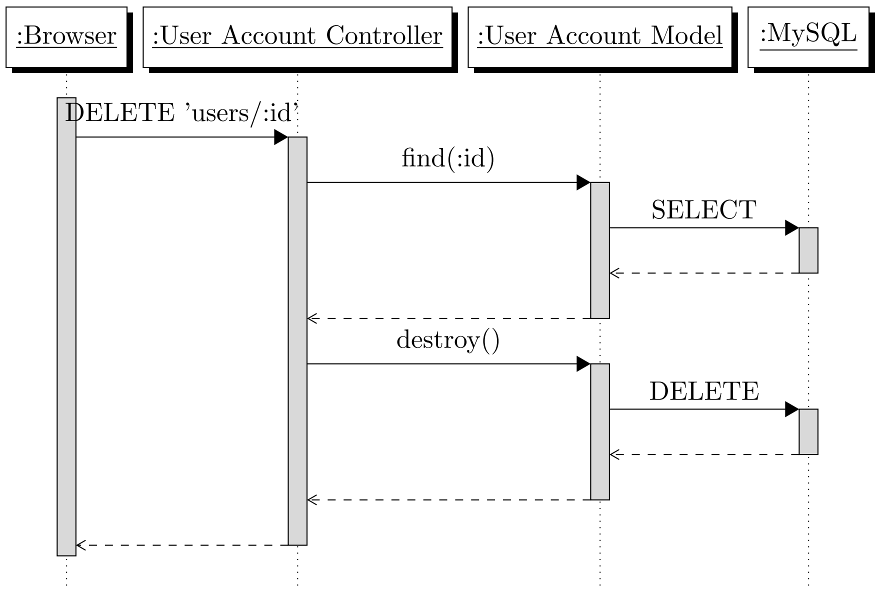


Figure 3.5: Delete User Account

3.5 Create Campaign

This is a basic CRUD operation: Create Campaign.

3.6 View Campaign

This is a basic CRUD operation: View Campaign.

This is a Read operation, similar to the above, but for one Campaign.

3.7 Update Campaign

This is a basic CRUD operation: Update Campaign.

3.8 Delete Campaign

This is a basic CRUD operation: Delete Campaign.

3.9 Schedule Campaign

This is a basic CRUD operation: Schedule Campaign.

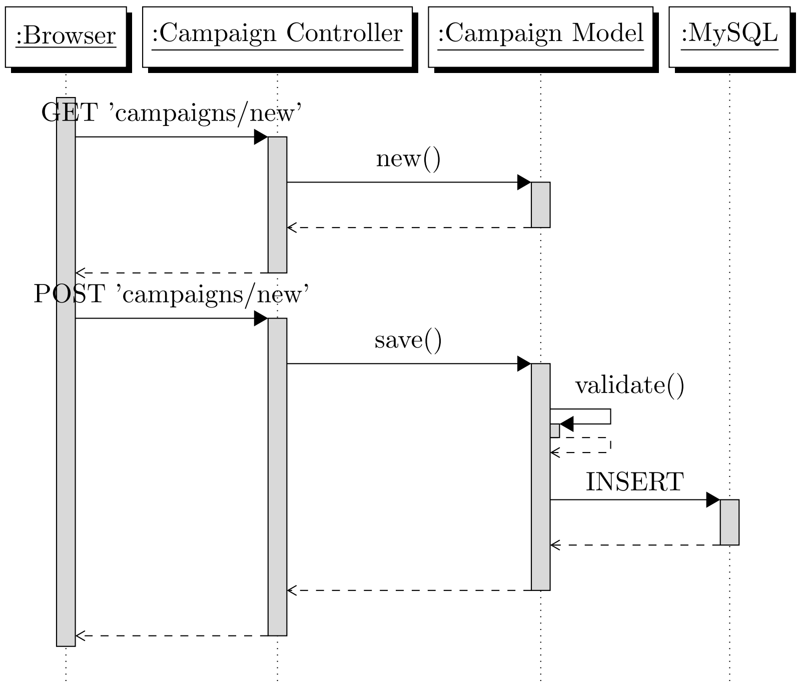


Figure 3.6: Create Campaign

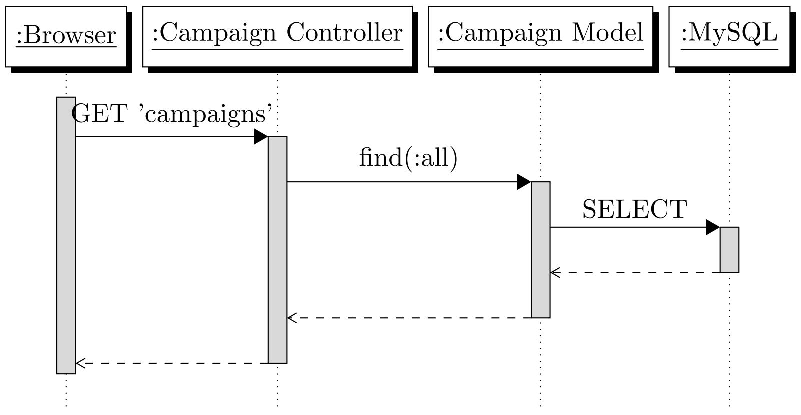


Figure 3.7: List Campaigns

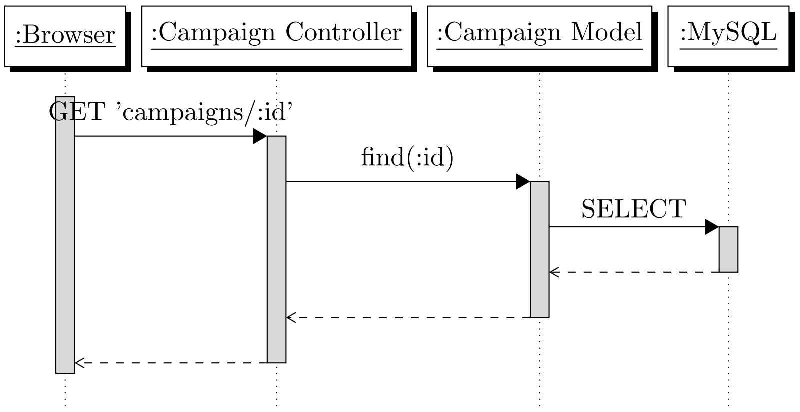


Figure 3.8: Show a Campaign

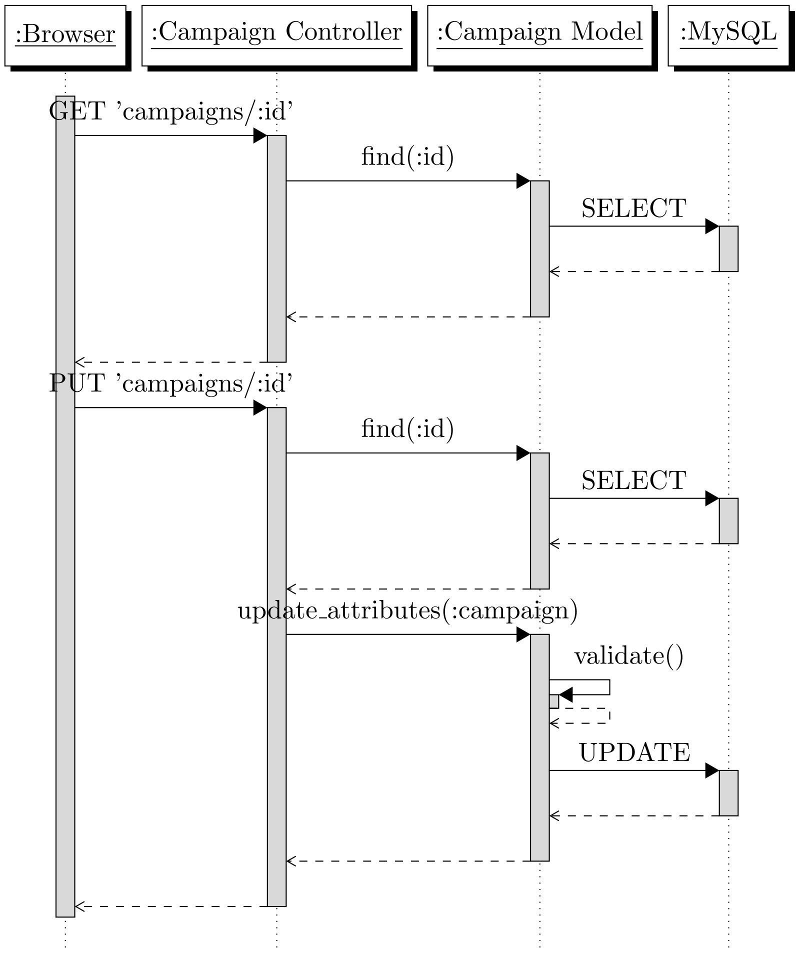


Figure 3.9: Update a Campaign

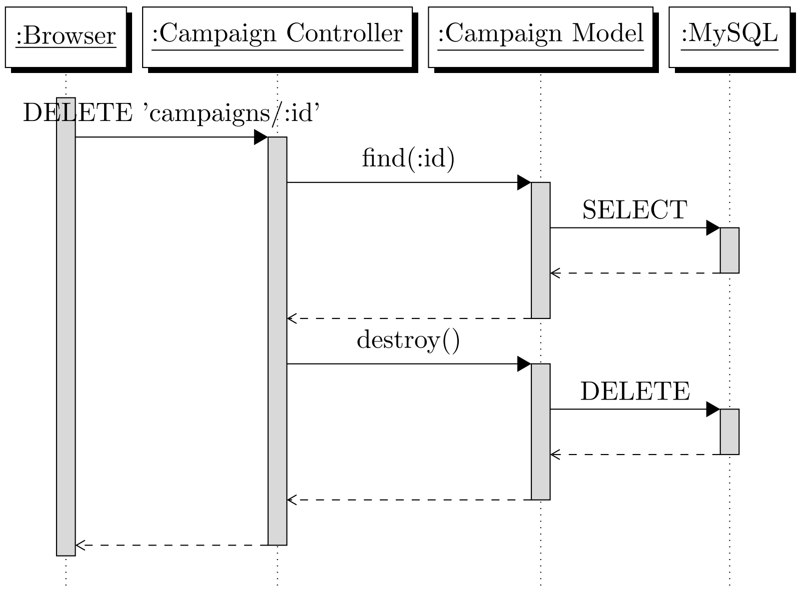


Figure 3.10: Delete Campaign

3.10 Create Content

This is a basic CRUD operation: Create Content.

3.11 View Content_Unit

This is a basic CRUD operation: View Content_Unit.

This is a Read operation, similar to the above, but for one Content Unit.

3.12 Update Content_Unit

This is a basic CRUD operation: Update Content_Unit.

3.13 Delete Content_Unit

This is a basic CRUD operation: Delete Content_Unit.

3.14 Schedule Content_Unit

In order to change the scheduled time of anything that is schedulable, the same process as an update is carried out.

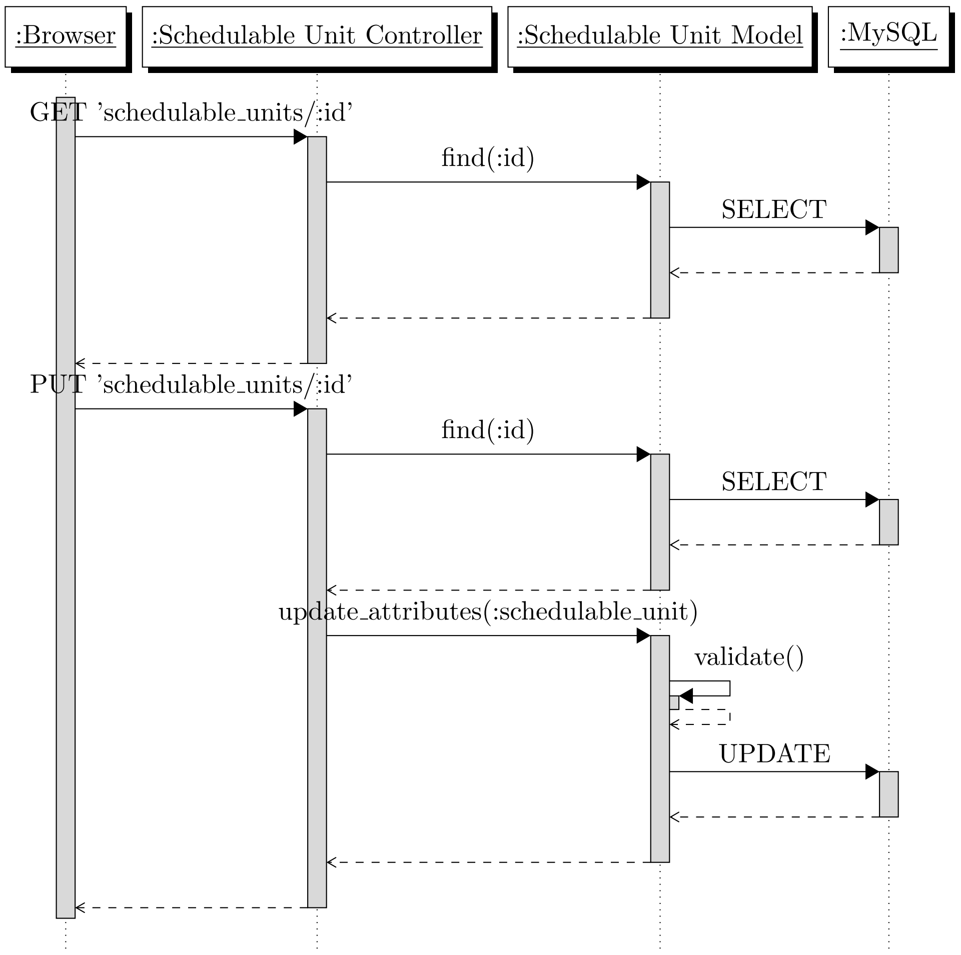

:Browser
:Schedulable Unit Controller
:Schedulable Unit Model
:MySQL
GET 'schedulable_units/:id'
find(:id)
SELECT
PUT 'schedulable_units/:id'
find(:id)
SELECT
update_attributes(:schedulable_unit)
validate()
UPDATE

Figure 3.11: Update scheduled time of Schdulable Unit

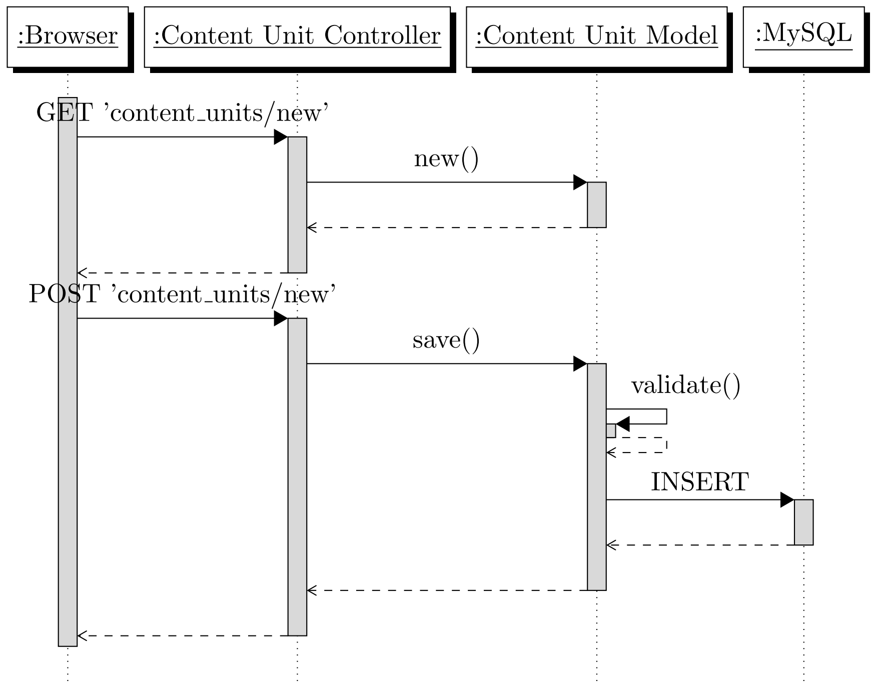


Figure 3.12: Create Content Unit

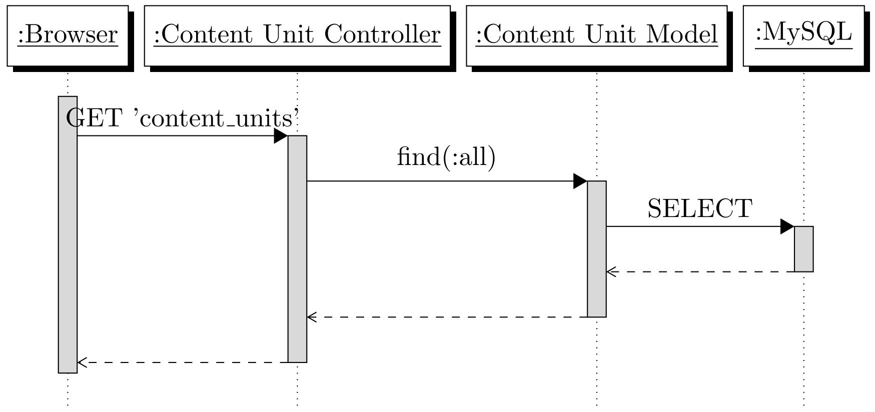


Figure 3.13: List Content Units

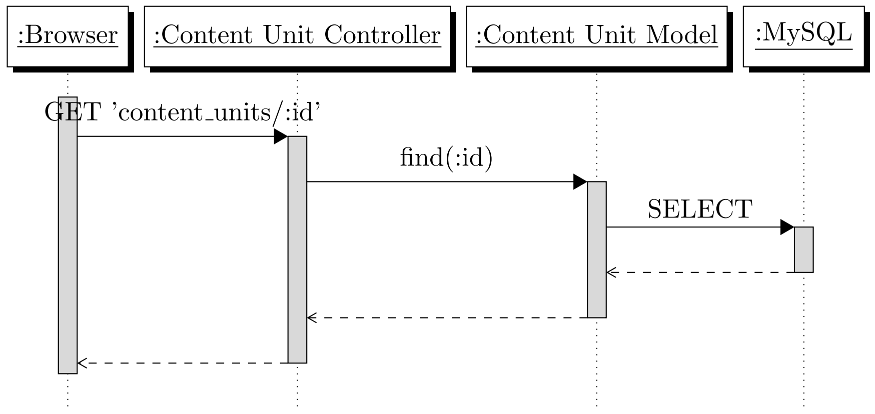


Figure 3.14: Show a Content Unit

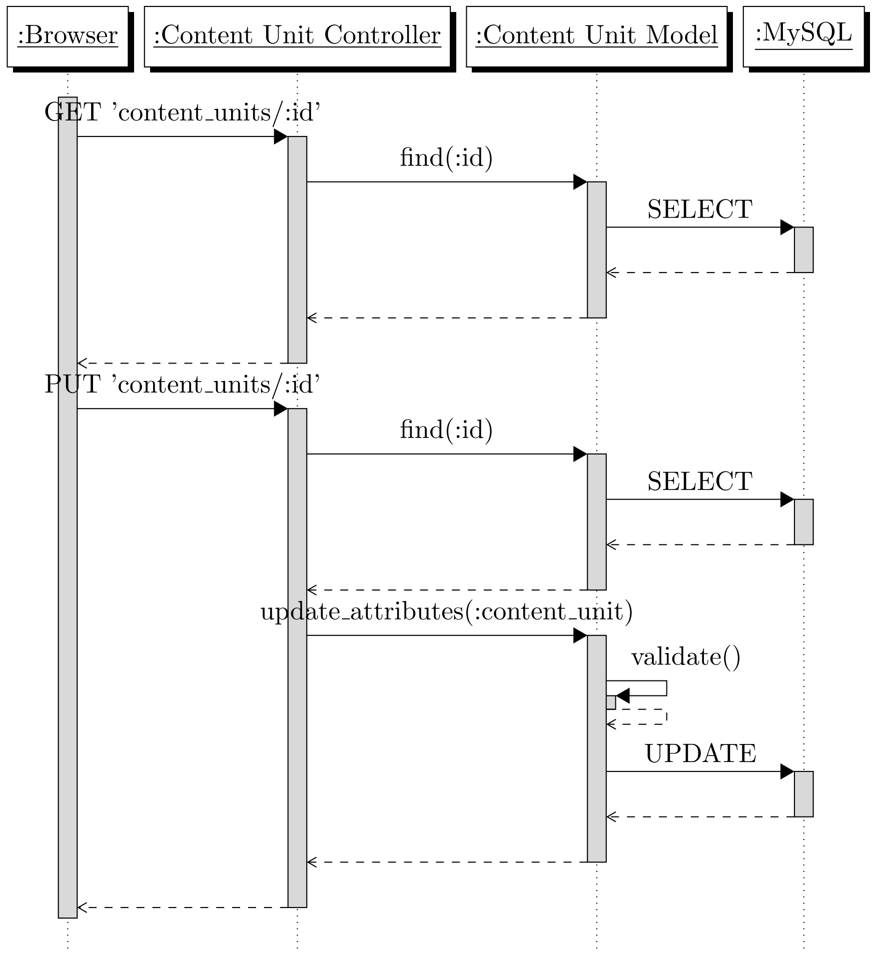


Figure 3.15: Update content_unit

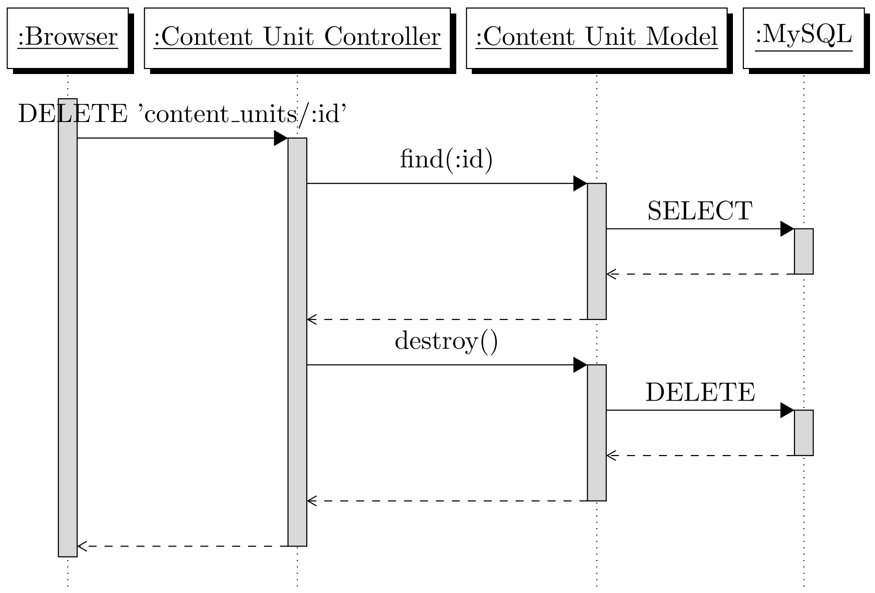


Figure 3.16: Delete Content Unit

After content is scheduled, it can be picked up by the Scheduler, triggered via Cron. Every 30 minutes, Cron will run the Scheduler, which will look in the database for active Campaigns. Within those active campaigns, each piece of content will be checked to see if the go-live time is now or past. If it's now or past, the scheduler calls the Publishing and Aggregation Platform with the content needed for a push to the given External Service.

3.15 View Metrics and Statistics via Explore Panel

3.16 Lost User Name

3.17 Lost Password

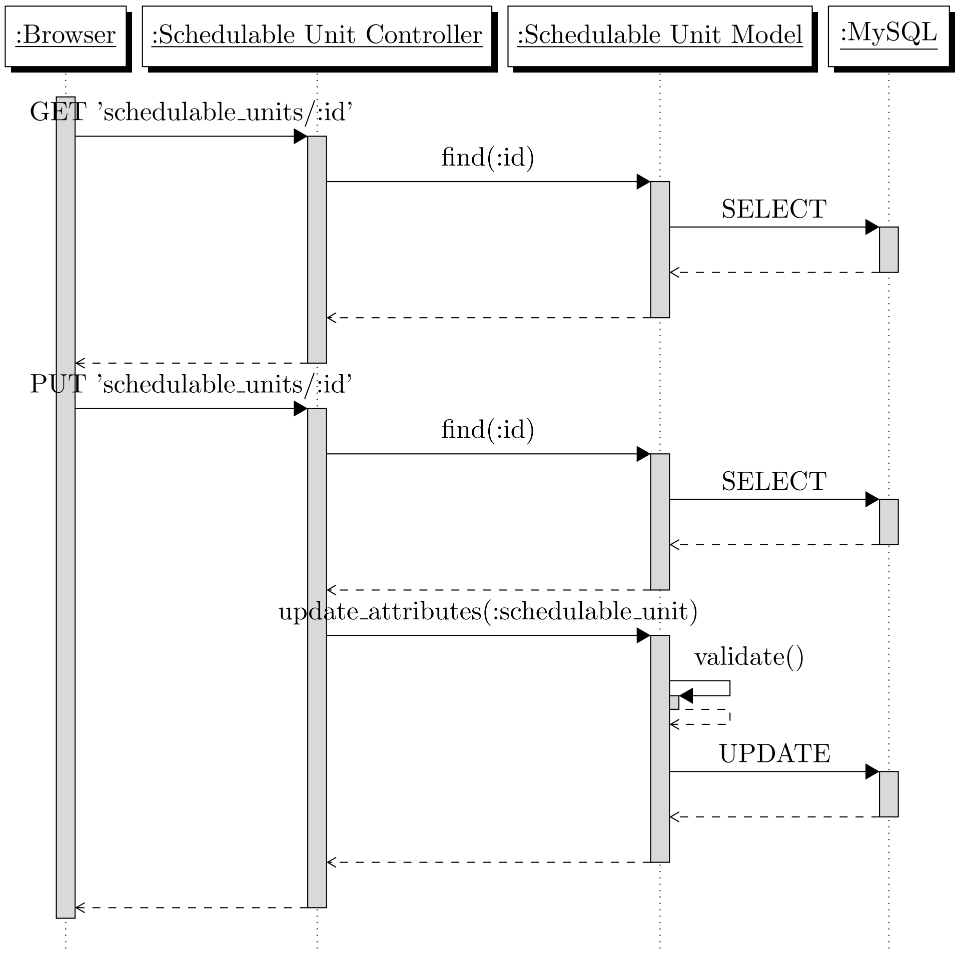

:Browser
:Schedulable Unit Controller
:Schedulable Unit Model
:MySQL
GET 'schedulable_units/:id'
find(:id)
SELECT
PUT 'schedulable_units/:id'
find(:id)
SELECT
update_attributes(:schedulable_unit)
validate()
UPDATE

Figure 3.17: Update scheduled time of a Schedulable Unit

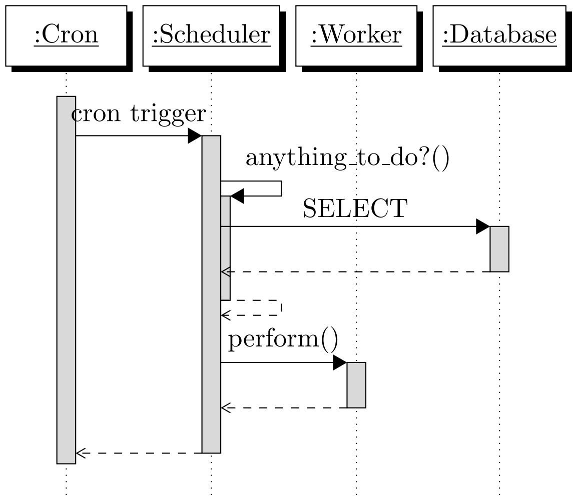


Figure 3.18: Perform a scheduled action

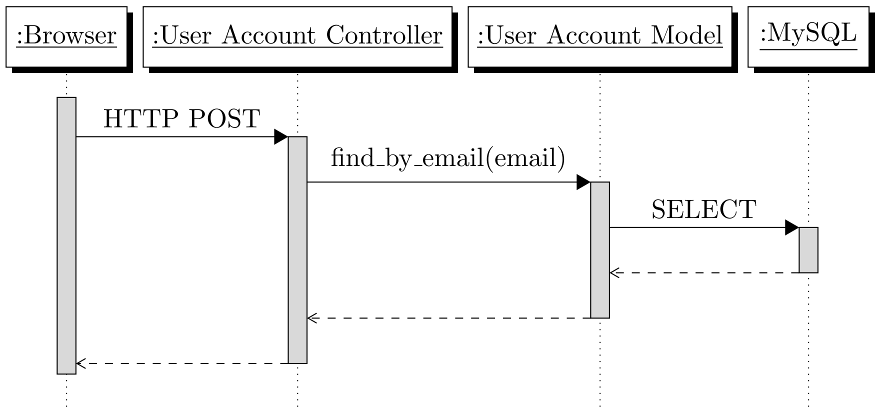


Figure 3.19: Retrieving a lost username by email id.

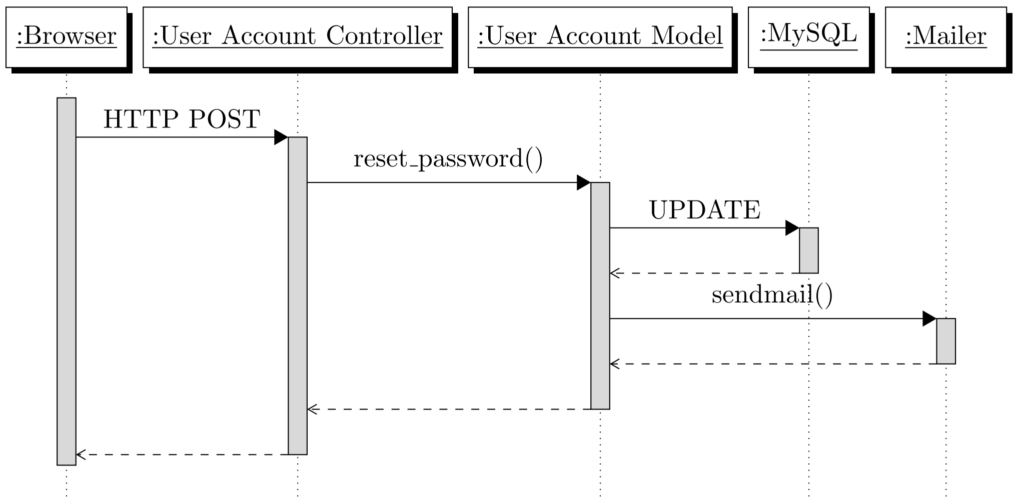


Figure 3.20: Retrieving a lost password. Refer to email figure for mailer actions.

Chapter 4

Publishing and Aggregation Platform

4.1 Object Model

4.1.1 Overview

Objects exist in Mashbot for the purposes of identification and manipulation. Objects are not stored in the Publishing and Aggregation Platform, but are merely passed through its pipeline.

Objects contain information general to that object itself and also information specific to one or more services, so as to identify that object on a service, or distinguish similar fields in different services.

Each object has a type and comprises two collections of entities: (1) a set of service associations, and (2) a list of properties in the form of a collection of objects. An object can also be considered "primitive," meaning it has a particular value assigned to it.

Semantically, speaking, an object is anything that can be manipulated by people and computers in a web context. Examples of objects are pictures, blog posts, status updates, IDs, people, et al. Not all of these are supported by Mashbot initially, but could be added through the plugin architecture.

4.1.2 The Object

The object comprises:

- Type
- Service Associations
- Properties
- *(Optionally)* Primitive value

Type

An object has type associated with it. Type is service agnostic: multiple services may support handling the same type of object, and services may support handling multiple object types.

Service Associations

Each object may be associated with one or more services. Semantically speaking, this reflects the idea that an object, as manipulated by Mashbot, exists in parallel with an object stored in some form on each associated service.

That is to say, an object associated with a particular service is "live" on that particular service, and an object not associated with any service is "not live," and does not exist on that service.

Associations tell the Publishing and Aggregation Platform where to look for an object, that is, which plugins to use to index and pull a given object.

Adding/removing an association: Adding an assocation is "pushing" an object to that service. Removing an association is "deleting" an object from that service.

These operations are defined by plugins in terms of both *behavior* and *requirements*. Requirements may be complex and hierarchical, specifying recursive requirements for values and sub-objects within a given object. Put another way, a service may require that before an object is associated with that service, it must contain certain sub-objects, some of which have a certain class of values or certain associations.

Properties

Properties represent an unordered collection of child objects belonging to the object in question.

Child objects are not mandated to be of a unique type or value, and thus cannot be addressed by type or value. Instead, objects can be queried for child objects, given criteria about the child objects themselves.

For instance, an object may be associated with two different services, and thus have two different identifiers. That object, therefore, will have one child object ID associated with the first service, and a second child object ID associated with the second service. (But the numbering here is arbitrary)

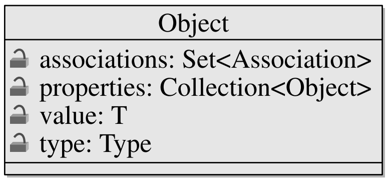
This isn't to say that child objects cannot be associated with more than one service. The demarcation again is semantic: if two entities refer to the same thing, they should be the same object. For instance, an object that has a name might have the same name on two different services, so the object would have a single child object for name, and both objects would be associated with both services.

Primitive value

A primitive value is a value with which the object can be interchanged with without losing or gaining semantic information.

For instance, IDs and names are object types that would frequently have primitive values. An ID could have a primitive value *1000*, and a name could have a primitive value *Bob's Photo*.

4.1.3 Class Diagram


Object
associations: Set<Association>
properties: Collection<Object>
value: T
type: Type

4.2 API Design

4.2.1 Request

A request to our API will have the following fields:

- OAuth Information
  - Realm - A Protection Realm
  - Consumer Key - A key for identifying the customer
  - Access Token - A token generated using the parameters and the Consumer Secret
  - Nonce - A randomly generated string given to all requests sent with the same timestamp.
  - Timestamp - The number of seconds since January 1st, 1970.

    - Signature Method - The signature method that the user used to sign the request
    - Version - Version of OAuth that you are using.

- Operation - A string containing the operation as detailed below.
- Content - The content to be passed to the appropriate Content Processors.
- Third Party Authentication Data - The third party authentication data for sending third party requests. It will contain a tree whose hierarchy will contain the following levels.
    1. Service Name
    2. User Name
    3. Authentication - Contains either the authentication token for this user or a password

4.2.2 Operation

Delete

Edit

Pull

Push

4.3 Server Design

4.4 Plugin Design

4.4.1 Content-Type Plugins

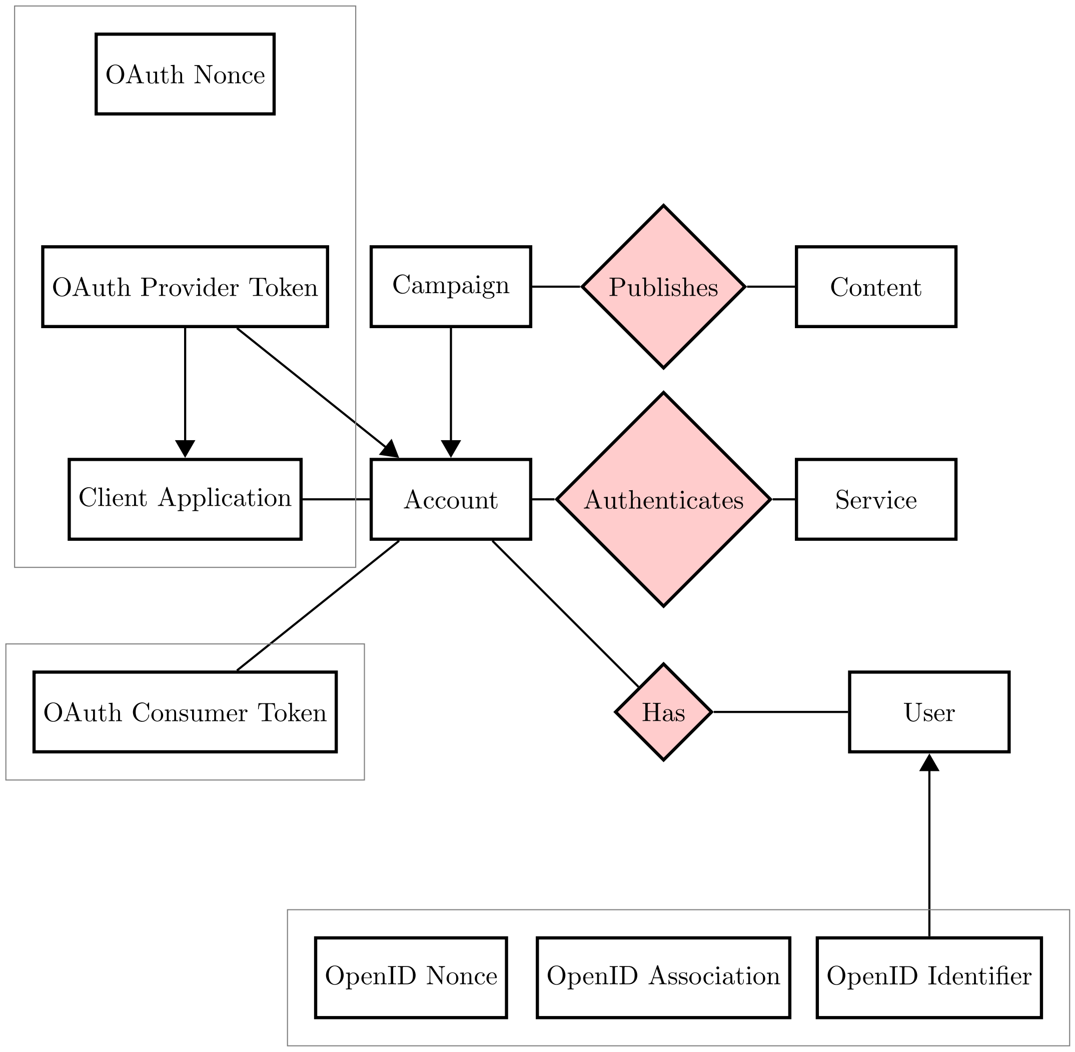
4.4.2 Service Plugins

Chapter 5

Database Design

5.1 Summary

An entity-relationship diagram depicting the data model is shown below.



5.1.1 Advantages of Design

- Decoupled Publishing Platform
  - Allows publishing capabilities to be extended without change to the Campaign Manager.
  - Provides asychronous publishing so HTTP requests do not time out when pushing a large amount of content.
  - Allows scheduled items to get fired off, even if the Campaign Manager is not fully running.

5.1.2 Disadvantages of Design

5.1.3 Design Rationale

5.2 A Requirements Traceability Matrix

5.2.1 Traceability by Requirement Numbers

5.2.2 Traceability by Design Component