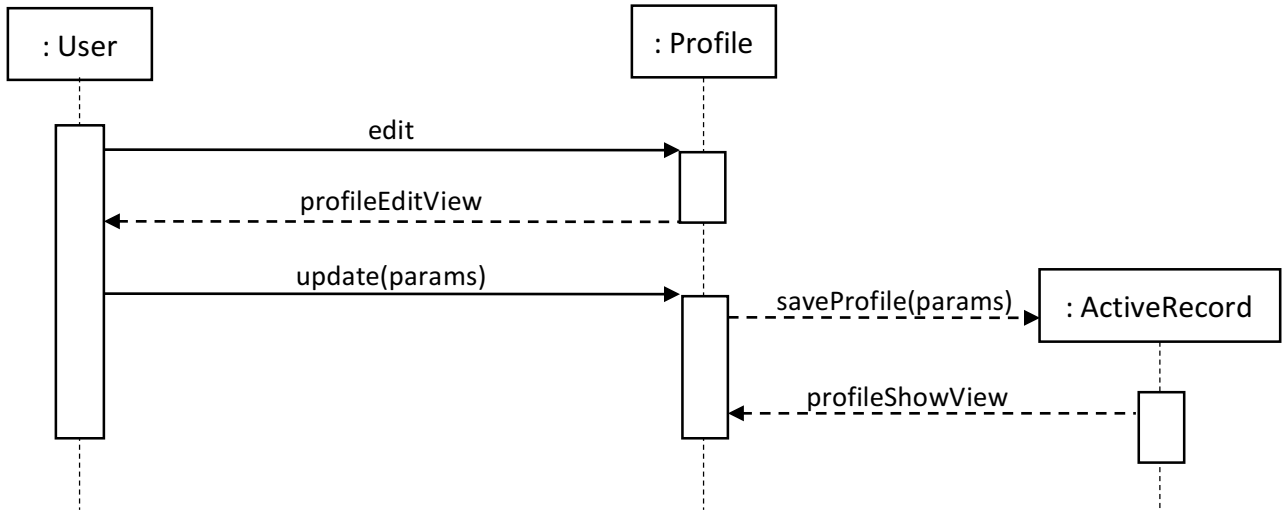
Iteration 2 Design Document

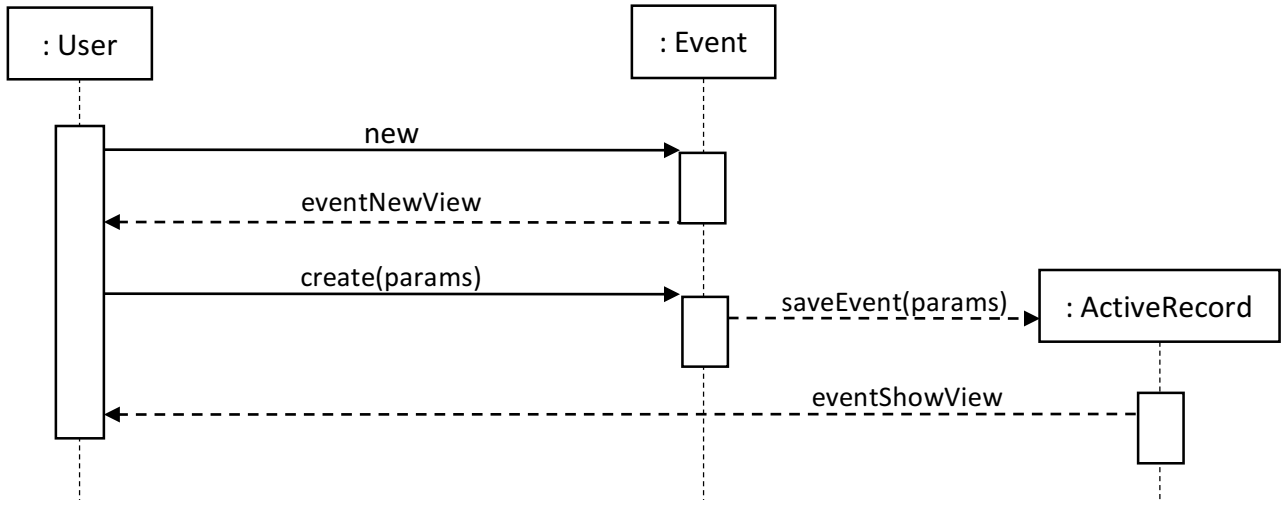
Edit Profile Scenario (Basic Flow)

1.) User navigates to the Edit Profile page by clicking the Edit Profile button.
2.) User edits any information that they want to change, including first name, last name, profile picture, occupation, and about me.
3.) User clicks Update Profile button to save changes to their profile and view them on their profile page.



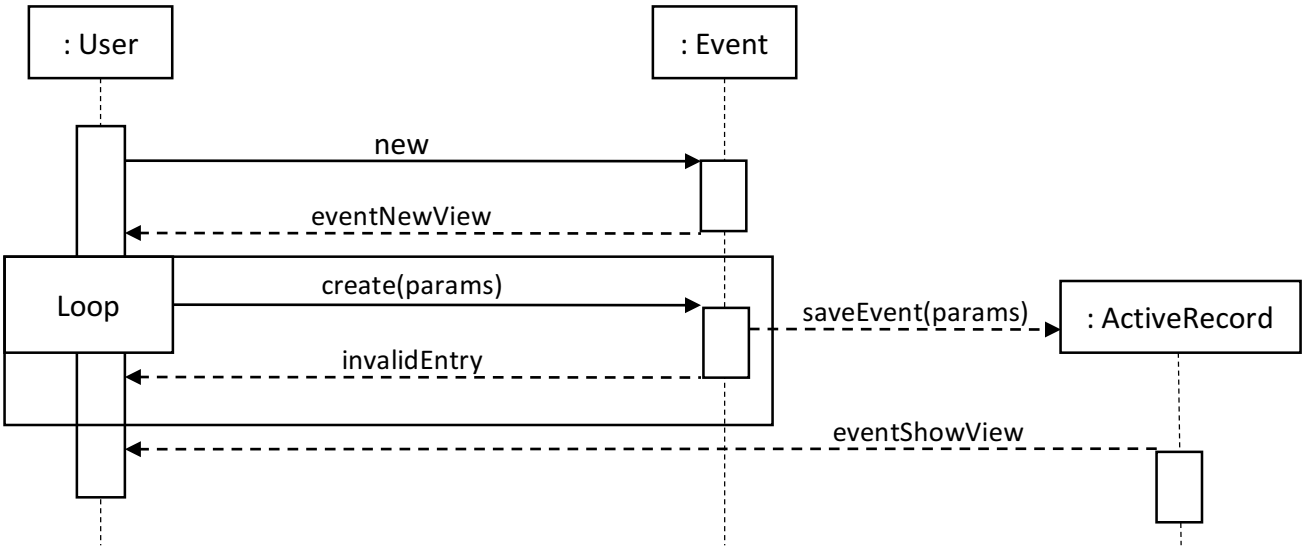
Create New Event Scenario (Basic Flow)

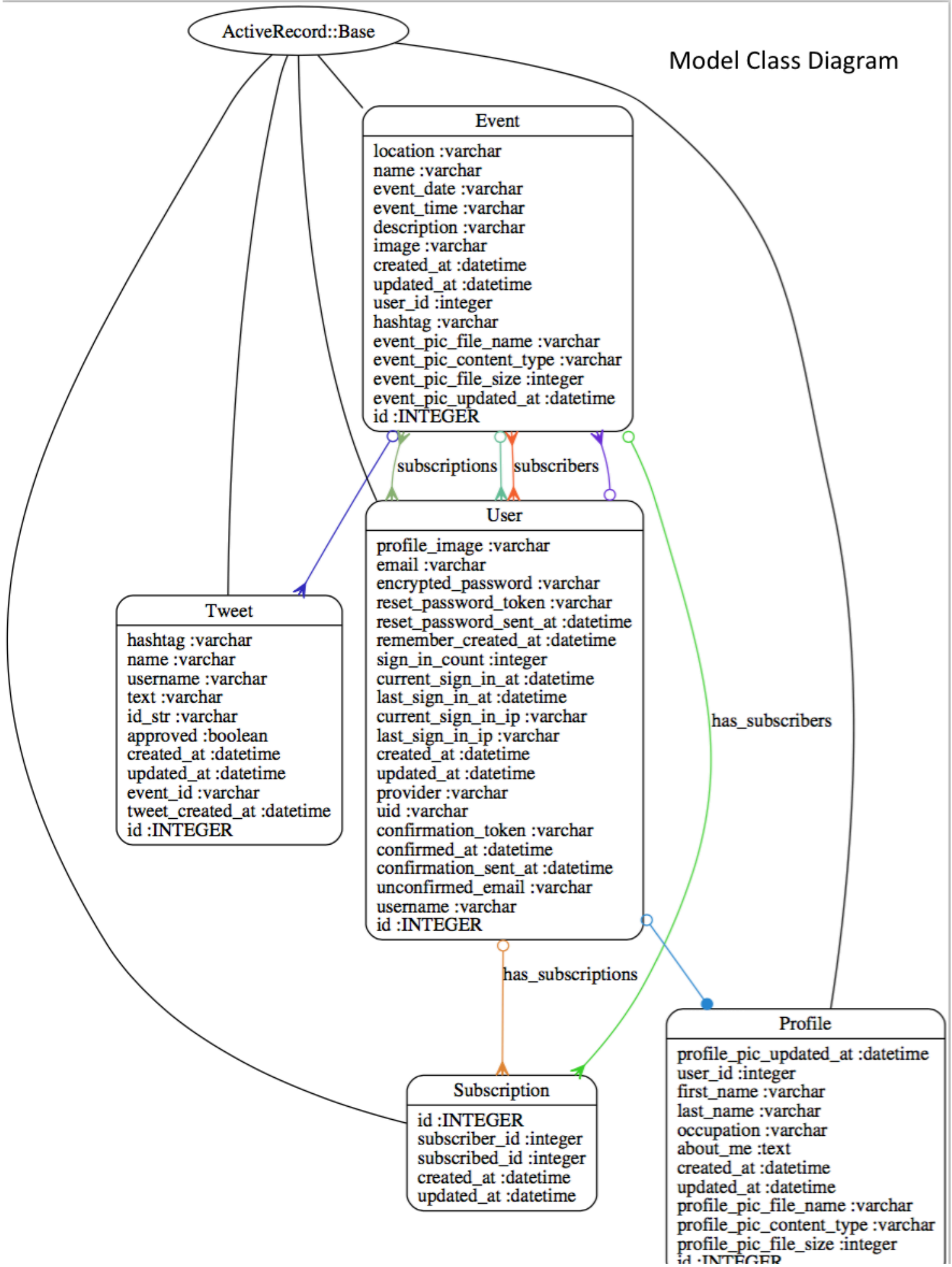
1.) User navigates to the Create New Event page by clicking one of the Create New Event Buttons.
2.) User enters in the information for the event, including the name, hashtag, header image, date, start time, and description.
3.) User clicks the Create Event button to create the event and view it on the event page.



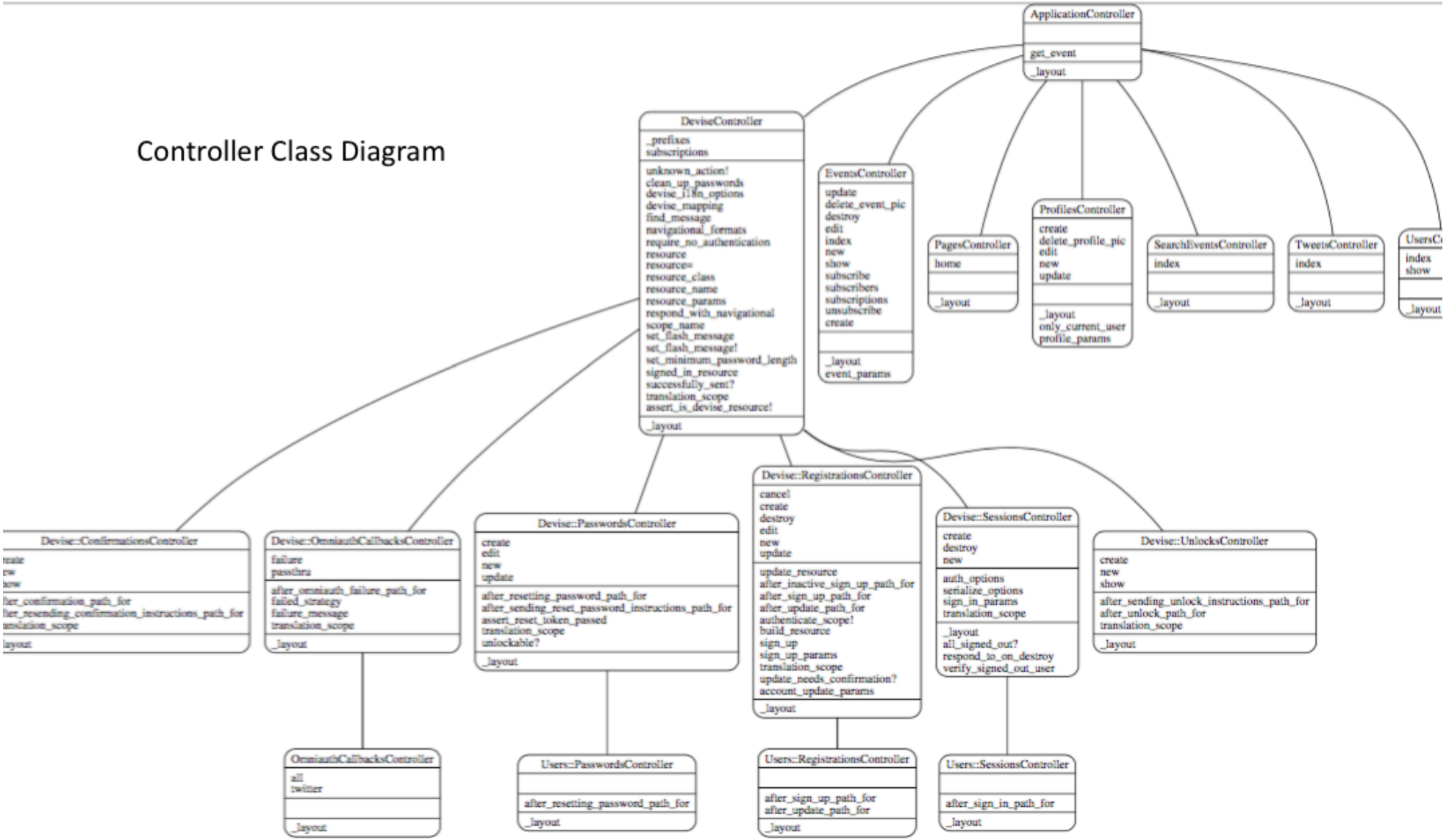
Create New Event Scenario (Alternate Flow)

1.) User navigates to the Create New Event page by clicking one of the Create New Event buttons.
2.) User enters in the information for the event, including the name, header image, date, start time, and description. They leave the hashtag field blank.
3.) User clicks the Create Event button to try to create the event.
4.) An error on the page indicates that the user must enter an event hashtag.
5.) User re-enters in an invalid event hashtag.
6.) User clicks the Create Event button to try to create the event.
7.) An error on the page indicates that the user must enter a valid hashtag.
8.) User re-enters in a valid event hashtag.
9.) User clicks the Create Event button to create the event and view it on the event page.




ActiveRecord::Base
Model Class Diagram
Event
location :varchar
name :varchar
event_date :varchar
event_time :varchar
description :varchar
image :varchar
created_at :datetime
updated_at :datetime
user_id :integer
hashtag :varchar
event_pic_file_name :varchar
event_pic_content_type :varchar
event_pic_file_size :integer
event_pic_updated_at :datetime
id :INTEGER
subscriptions
subscribers
User
profile_image :varchar
email :varchar
encrypted_password :varchar
reset_password_token :varchar
reset_password_sent_at :datetime
remember_created_at :datetime
sign_in_count :integer
current_sign_in_at :datetime
last_sign_in_at :datetime
current_sign_in_ip :varchar
last_sign_in_ip :varchar
created_at :datetime
updated_at :datetime
provider :varchar
uid :varchar
confirmation_token :varchar
confirmed_at :datetime
confirmation_sent_at :datetime
unconfirmed_email :varchar
username :varchar
id :INTEGER
Tweet
hashtag :varchar
name :varchar
username :varchar
text :varchar
id_str :varchar
approved :boolean
created_at :datetime
updated_at :datetime
event_id :varchar
tweet_created_at :datetime
id :INTEGER
has_subscribers
has_subscriptions
Subscription
id :INTEGER
subscriber_id :integer
subscribed_id :integer
created_at :datetime
updated_at :datetime
Profile
profile_pic_updated_at :datetime
user_id :integer
first_name :varchar
last_name :varchar
occupation :varchar
about_me :text
created_at :datetime
updated_at :datetime
profile_pic_file_name :varchar
profile_pic_content_type :varchar
profile_pic_file_size :integer
id :INTEGER

Controller Class Diagram



Designing object-oriented iteration diagrams and class diagrams for Ruby on Rails, which uses the Model-View-Controller paradigm instead of object-oriented, is quite the challenge. You have the Model, which essentially acts as our object that contains various attributes and some methods, you have the view, which displays the UI of the application, and you have the Controller, which acts as a bridge between the Model and the View to determine how the two will interact on certain actions.

We designed our Iteration Sequence Diagrams using our Models, and the actions between them use various methods from the Controllers and Models because they both work hand-in-hand. For example, the use case "Edit Profile" starts with the User Model who will click a button that triggers the edit action in the Profiles Controller. The Profile Model will then return the "edit profile view" where the User can edit their profile. Once the User selects the update button, the update action is triggered in the Profiles Controller. The Profile Model then receives the parameters from the edit profile view save the new values in the ActiveRecord Model, which is the database. The database is SQLite3 in development and PostgreSQL in production.

There needed to be two class diagrams for Ruby on Rails, one for the Model and one for the Controller. In the Model Class Diagram, every Model is inherited from the ActiveRecord Model. Every Model is also displayed with how it interacts with each Model. For example, an Event has many tweets, many subscribers, and a single user who created it. Each Model also displays its attributes which are stored in ActiveRecord, or the database. In the Controller Class Diagram, each Controller is inherited from the Application Controller, which controls all other controllers. Each Model has its own Controller as seen in the Controller Class Diagram. Each

Controller contains various methods, public being on top, followed by protected, and then private. For example, the Users Controller has two methods, Index and Show, which display all Users in a view, or just displays the specific User in the view.

The GRASP patterns used here are Controller and Creator. The Controller pattern is pretty much identical to how a Controller works in Ruby on Rails, so that is automatically included. The Controller pattern is used to explain how to connect the UI layer (View) with the application logic (Model). For example, the Events Controller connects the New Event button with the Event model which contains all the logic on what to do with a new event. The Creator pattern is also used because we are creating new instances of an object, such as an Event. The Creator in our case is hard to describe in an object-oriented case, but it is the ActiveRecord Model which creates a new record in the DB, representing a "new object." When we want to create a new event, we are actually creating a new record in the database, and then showing all of those instances in the database in the view. The Model in our case is represented by the Creator pattern.