THE UNIVERSITY OF
MELBOURNE

COMP30006

Software Modelling and Design

Project 1:
Image Majick Design

Author:
Meghan Mann
663657

Proffesor:
Dr. Ed Kazmierczak

Semester 2, 2014

Contents

1 Use Cases

1.1 Use Case 1: Registering with Image Majick

Use Case: Registering with Image Majick
Actors: User, Server
Trigger: User visits Image Majick site
Sequence of Events:

1. User visits the Image Majick site without being signed in.
2. Image Majick requests that user must be a member to enter site.
3. User selects to register with ImageMajick.
4. User is prompted to enter their email address, username, and to choose a password.
5. Browser sends user's input to the server.
6. The server validates that user does not already have an account with that email address, that their requested username is not taken, and that their password contains the required characters.
7. An account is created for the user
8. The user is redirected to the Image Majick homepage

Alternative Paths:

6a. If the users information is not validated, go back to step 4.

1.2 Use Case 2: Login to Image Majick

Use Case: Log in to Image Majick
Actors: User, Server
Trigger: User selects to login to Image Majick
Sequence of Events:

1. User selects to login to Image Majick.
2. User is prompted to enter their email address and password.
3. Browser sends user's input to the server.
4. The server validates that the user's information is correct.
5. The user is redirected to the Image Majick homepage

Alternative Paths:

4a. If the users information is not validated, go back to step 2.

1.3 Use Case 3: Uploading an Image to Image Majick

Use Case: Uploading an Image to Image Majick
Actors: User, Server
Trigger: User selects to upload an image **Sequence of Events:**

1. User selects to upload an image.
2. A form is displayed prompting the user to choose a photo to upload and to enter a name for the photo.
3. The browser sends this form to the server.
4. The server validates that the user has chosen an image, and that a name has been chosen that is different from any image names previously used by the user.
5. The image is uploaded to the server.
6. The user is informed that their photo has been uploaded successfully.

Alternative Paths:

4a. If the server does not validate that the user chose a photo, go back to step 2.

4a. If the server does not validate that the user chose a unique name for their photo, go back to step 2.

1.4 Use Case 4: Sharing a Photo with Other Users

Use Case: Sharing a Photo with Other Users
Actors: User, Server
Trigger: User selects to share images
Sequence of Events:

1. User selects to share images.
2. The user is presented with all the image versions and users.
3. The user selects the image versions that they want to share and the users they they want to share the images with.
4. This information is sent to the server.
5. The server validates that at least one image version and one user were selected.
6. The server shares the selected photos with the selected users.
7. The user is informed that their photos have been shared successfully.

Alternative Paths:

5a. If the user did not select any photos to share, then they are prompted to select at least one photo, then go back to step 2.

5a. If the user did not choose any users to share the photo with, they are prompted to choose at least one user, then go back to step 2.

1.5 Use Case 5: Editing an Image

Use Case: Editing an Image
Actors: User, Server **Trigger:** User selects the editor
Sequence of Events:

1. The user selects the editor.
2. The user is prompted to select an image to edit.
3. The user selects an image to edit.
4. The server validates that an image was selected.
5. The selected image is displayed along with the available filters and options to change attributes of the filter.
6. If the user selects a filter, then the image edited with the selected filter is displayed.
7. If the user changes an attribute of the filter, the the image is displayed with the selected setting.
   Alternative Paths:

   2a. If an image is already selected, go directly to step 5.
   2a. If an image is already selected and the user wants to select another image, then the user selects "Edit Another Image". Then go to step 3.
   4a. If the server determines no images was selected, go to step 2.

1.6 Use Case 6: Saving an Image Version

Use Case: Saving an Image Version
Actors: User, Server
Trigger: User selects to save an image version
Sequence of Events:

1. The user selects to save an image version.
2. The user is prompted to select a unique name for that image version.
3. The user selects a name for the image version.
4. The server validates that a name was selected that is unique.

5. The server creates a new image version.
6. The user is informed that the image version has been created successfully.

   Alternative Paths:

   4a. If no name was chosen or it was not unique, fo to step 2.

1.7 Use Case 7: Saving a Filter

Use Case: Saving a Filter
Actors: User, Server
Trigger: User selects to save a filter
Sequence of Events:

1. The user selects to save a filter
2. The user is prompted to select a unique name for the filter.
3. The user selects a name for the filter.
4. The server validates that a name was selected that is unique.
5. The server creates a new filter.
6. The user is informed that the filter has been created successfully.

   Alternative Paths:

   4a. If no name was chosen or it was not unique, go to step 2.

1.8 Use Case 8: Deleting an Image

Use Case: Deleting an Image
Actors: User, Server
Trigger: User selects to delete an image/s
Sequence of Events:

1. The user selects an image/s that they want to delete.
2. The user confirms they want to delete these images.
3. The images are moved to the trash.
4. If the user wants to permanently delete the photos in the trash, then they select to empty the trash.

2 Component Diagram

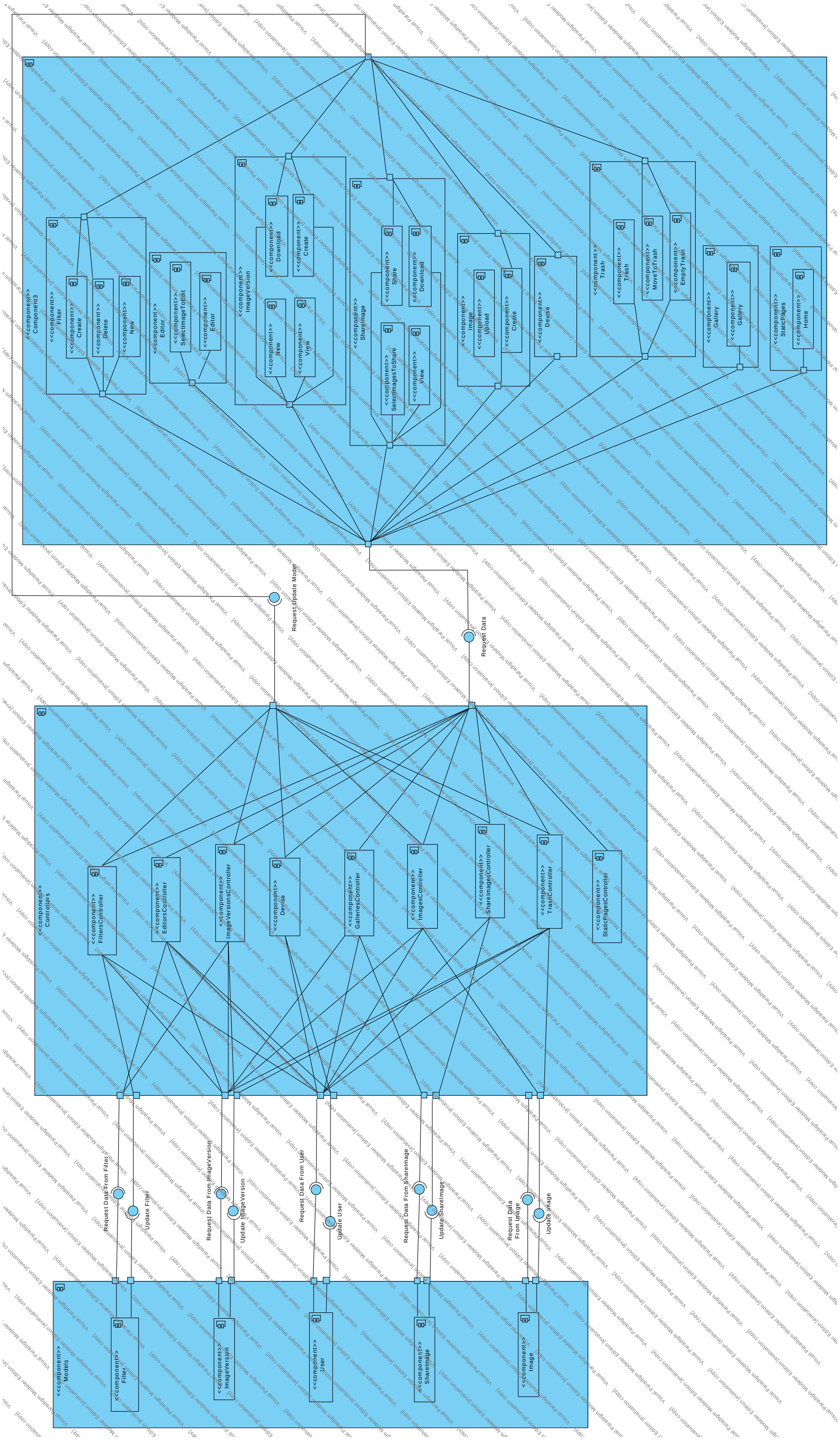

Figure 1: Component Diagram

3 Class Diagram

- Model = YELLOW
- Controller = GREEN

**Relationships between classes are displayed in component diagram

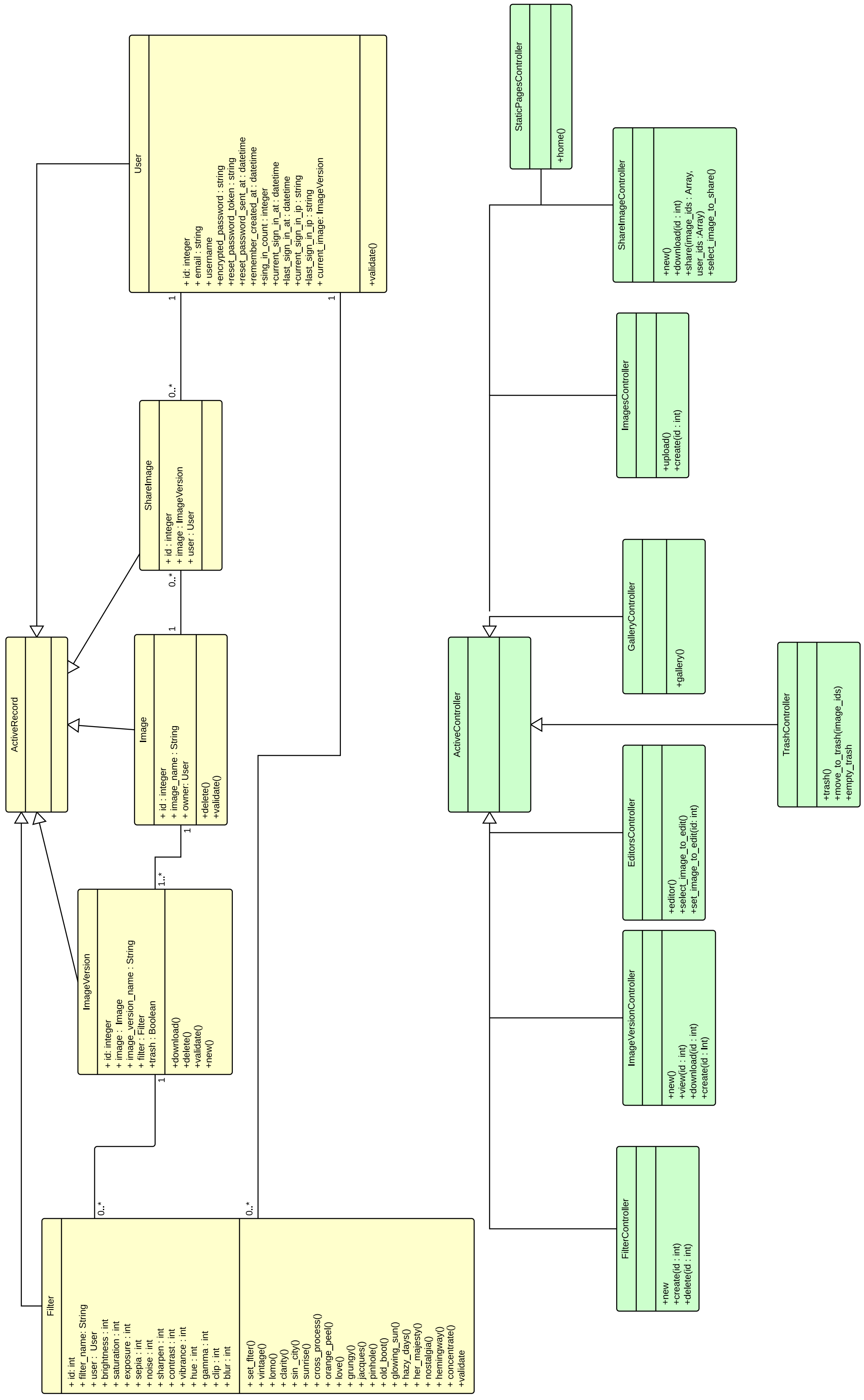

Figure 2: Class Diagram

4 Sequence Diagrams

4.1 Registering with Image Majick

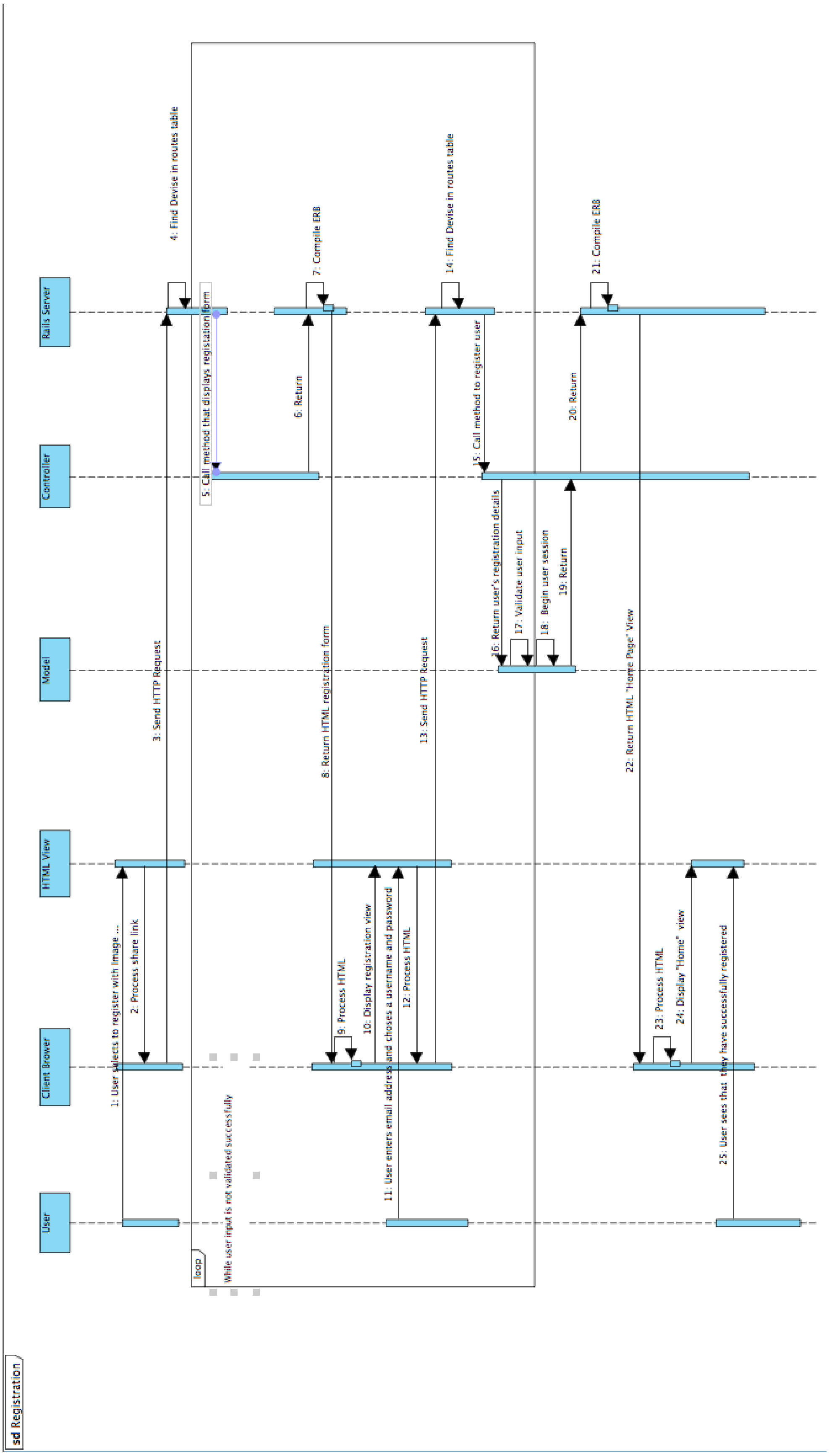
sd Registration
User
Client Brower
HTML View
Model
Controller
Rails Server
1: User selects to register with Image ...
2: Process share link
3: Send HTTP Request
4: Find Devise in routes table
5: Call method that displays registration form
6: Return
7: Compile ERB
8: Return HTML registration form
9: Process HTML
10: Display registration view
11: User enters email address and choses a username and password
12: Process HTML
13: Send HTTP Request
14: Find Devise in routes table
15: Call method to register user
16: Return user's registration details
17: Validate user input
18: Begin user session
19: Return
20: Return
21: Compile ERB
22: Return HTML "Home Page" View
23: Process HTML
24: Display "Home" view
25: User sees that they have successfully registered
loop
While user input is not validated successfully

Figure 3: Registering with Image Majick Sequence Diagram

4.2 Login To Image Majick

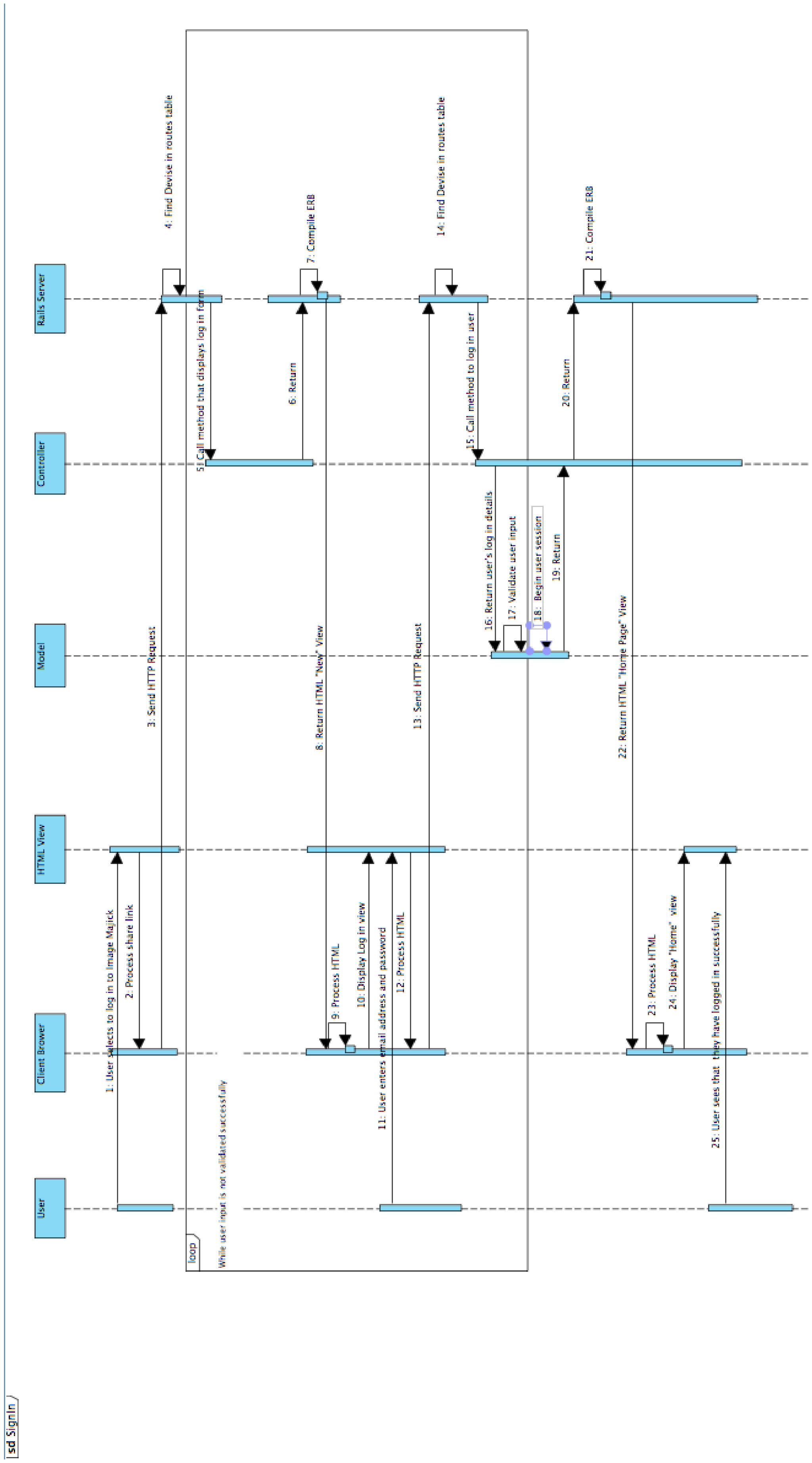
sd SignIn
User
Client Brower
HTML View
Model
Controller
Rails Server
loop
While user input is not validated successfully
1: User selects to log in to Image Majick
2: Process share link
3: Send HTTP Request
4: Find Devise in routes table
5: Call method that displays log in form
6: Return
7: Compile ERB
8: Return HTML "New" View
9: Process HTML
10: Display Log in view
11: User enters email address and password
12: Process HTML
13: Send HTTP Request
14: Find Devise in routes table
15: Call method to log in user
16: Return user's log in details
17: Validate user input
18: Begin user session
19: Return
20: Return
21: Compile ERB
22: Return HTML "Home Page" View
23: Process HTML
24: Display "Home" view
25: User sees that they have logged in successfully

Figure 4: Login To Image Majick Sequence Diagram

4.3 Uploading an Image to Image Majick

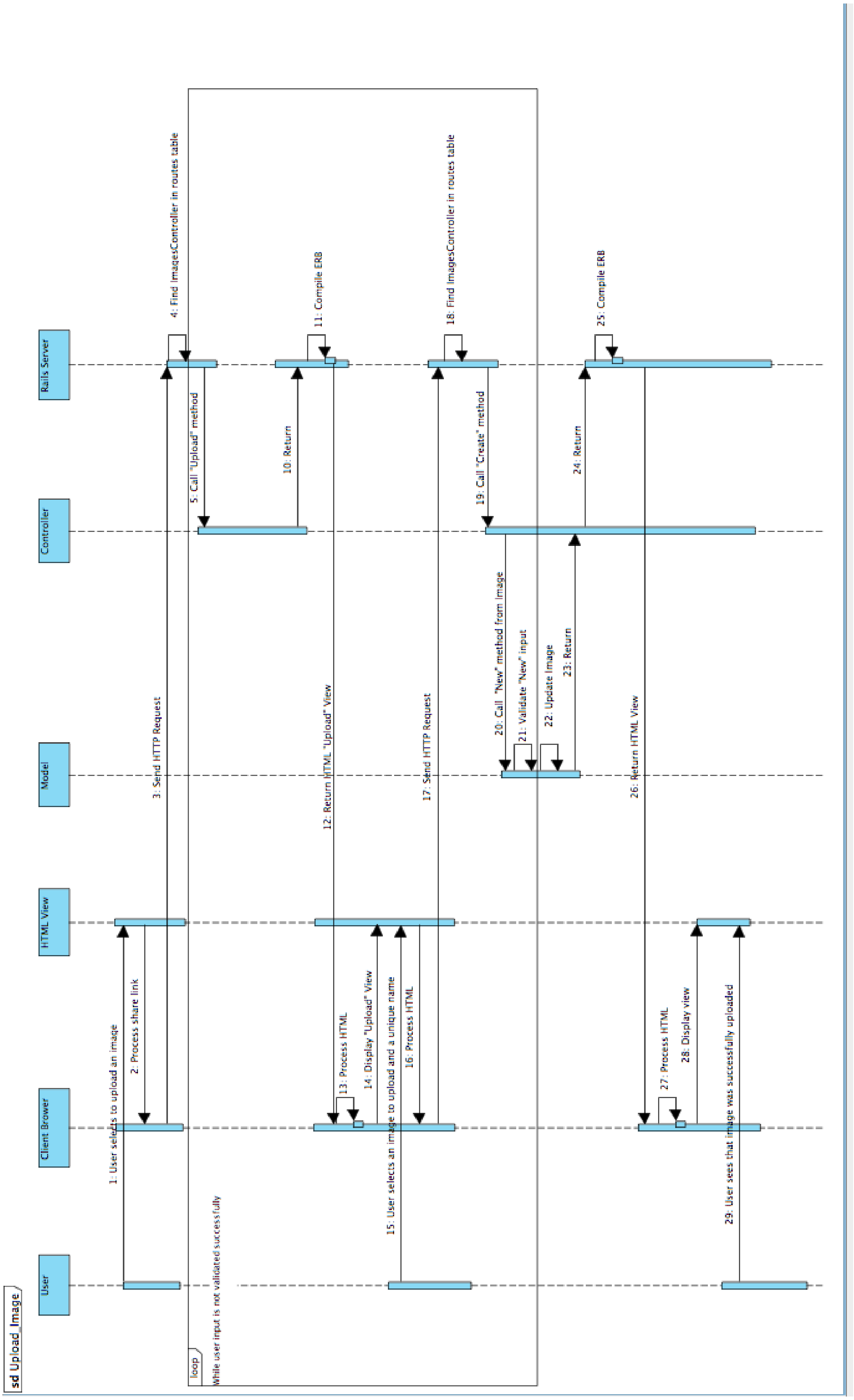

Figure 5: Uploading an Image to Image Majick Sequence Diagram

4.4 Sharing an Image with Other Users

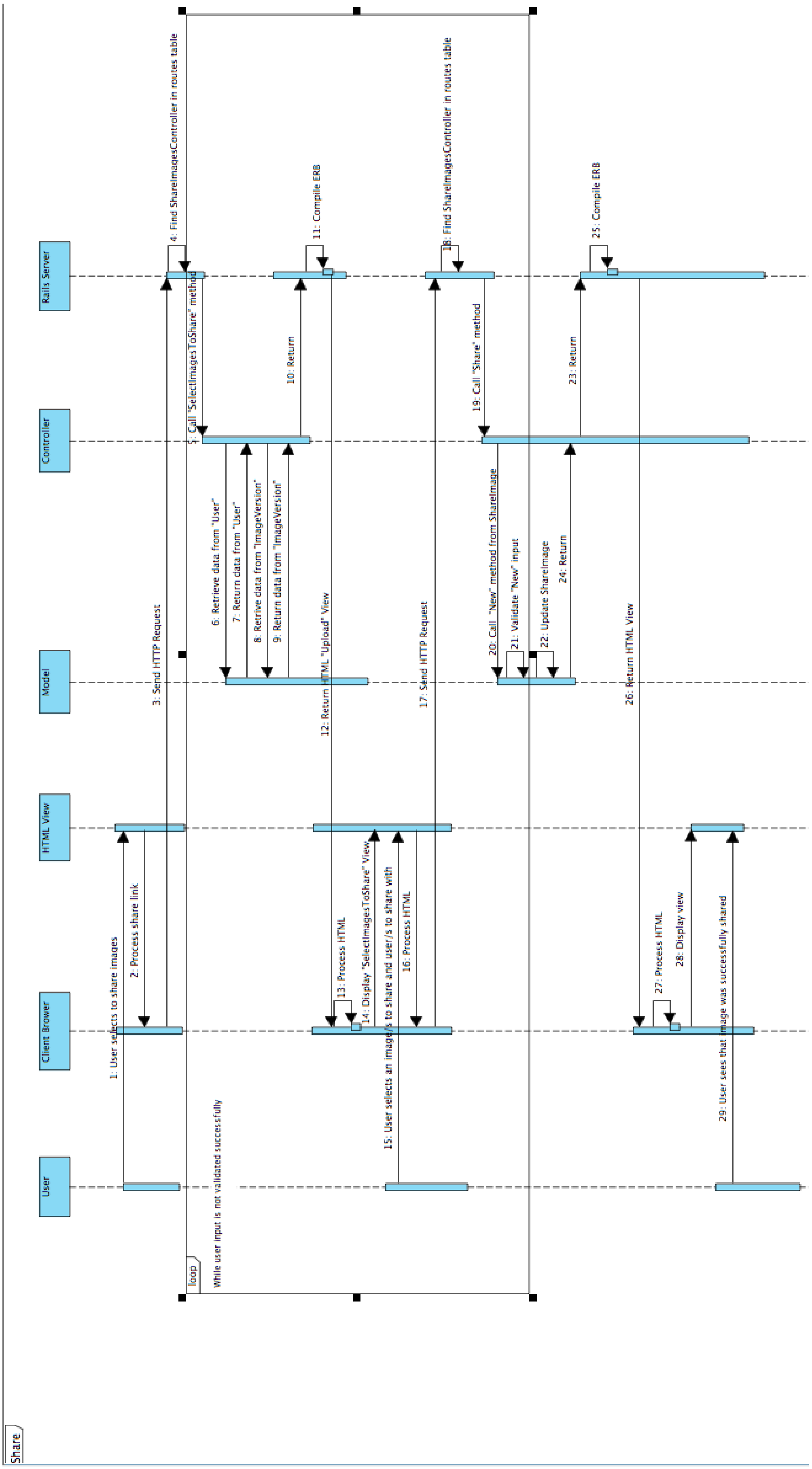
Share
User
Client Brower
HTML View
Model
Controller
Rails Server
loop
while user input is not validated successfully
1: User selects to share images
2: Process share link
3: Send HTTP Request
4: Find ShareImagesController in routes table
5: Call "SelectImagesToShare" method
6: Retrieve data from "User"
7: Return data from "User"
8: Retrive data from "ImageVersion"
9: Return data from "ImageVersion"
10: Return
11: Compile ERB
12: Return HTML "Upload" View
13: Process HTML
14: Display "SelectImagesToShare" View
15: User selects an image/s to share and user/s to share with
16: Process HTML
17: Send HTTP Request
18: Find ShareImagesController in routes table
19: Call "Share" method
20: Call "New" method from ShareImage
21: Validate "New" input
22: Update ShareImage
23: Return
24: Return
25: Compile ERB
26: Return HTML View
27: Process HTML
28: Display view
29: User sees that image was successfully shared

Figure 6: Sharing an Image with Other Users Sequence Diagram

4.5 Editing an Image

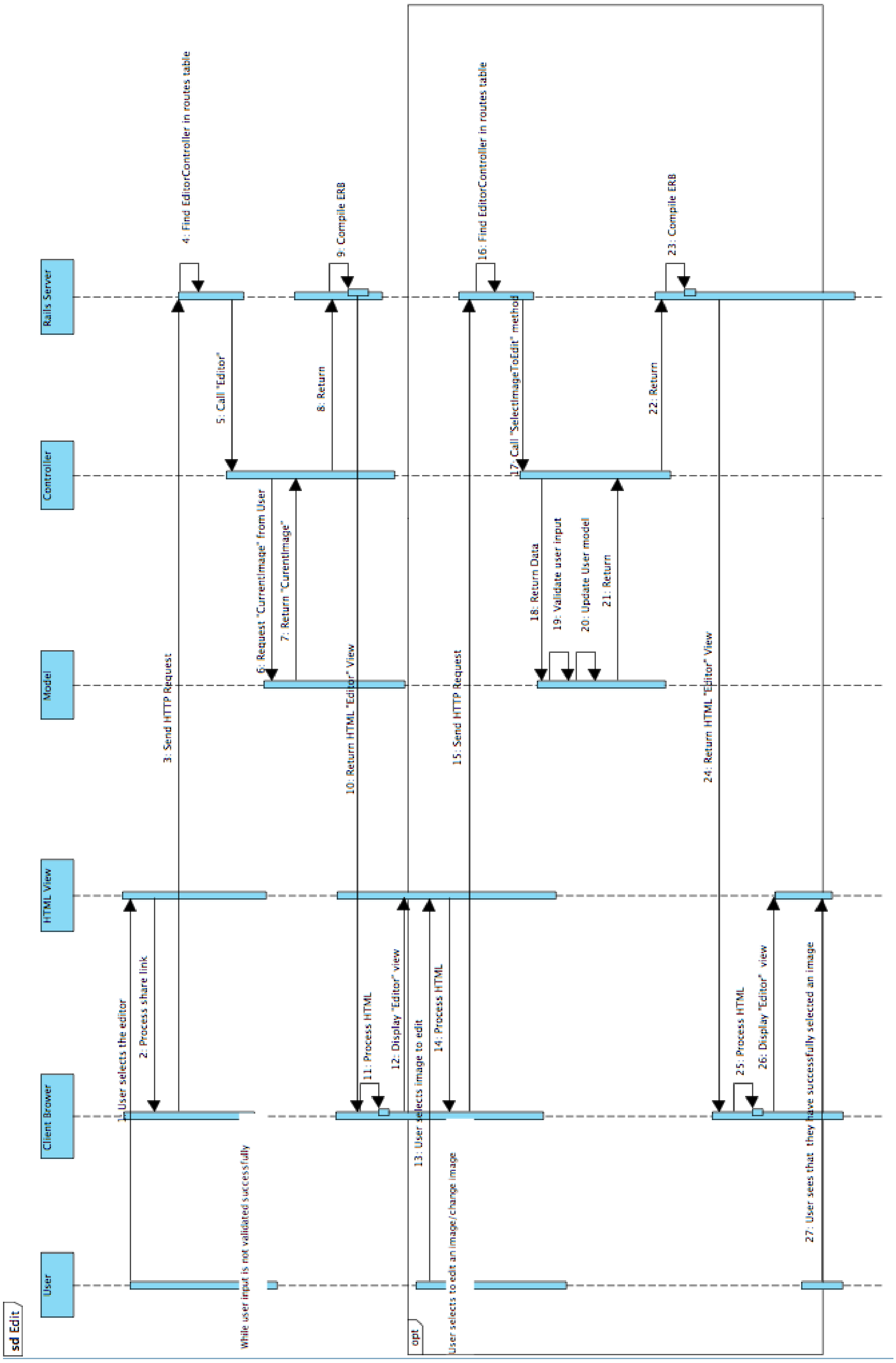
sd Edit
User
Client Brower
HTML View
Model
Controller
Rails Server
1: User selects the editor
2: Process share link
3: Send HTTP Request
4: Find EditorController in routes table
5: Call "Editor"
6: Request "CurrentImage" from User
7: Return "CurrentImage"
8: Return
9: Compile ERB
10: Return HTML "Editor" View
11: Process HTML
12: Display "Editor" view
While user input is not validated successfully
opt
User selects to edit an image/change image
13: User selects image to edit
14: Process HTML
15: Send HTTP Request
16: Find EditorController in routes table
17: Call "SelectImageToEdit" method
18: Return Data
19: Validate user input
20: Update User model
21: Return
22: Return
23: Compile ERB
24: Return HTML "Editor" View
25: Process HTML
26: Display "Editor" view
27: User sees that they have successfully selected an image

Figure 7: Editing an Image Sequence Diagram

4.6 Saving an Image Version

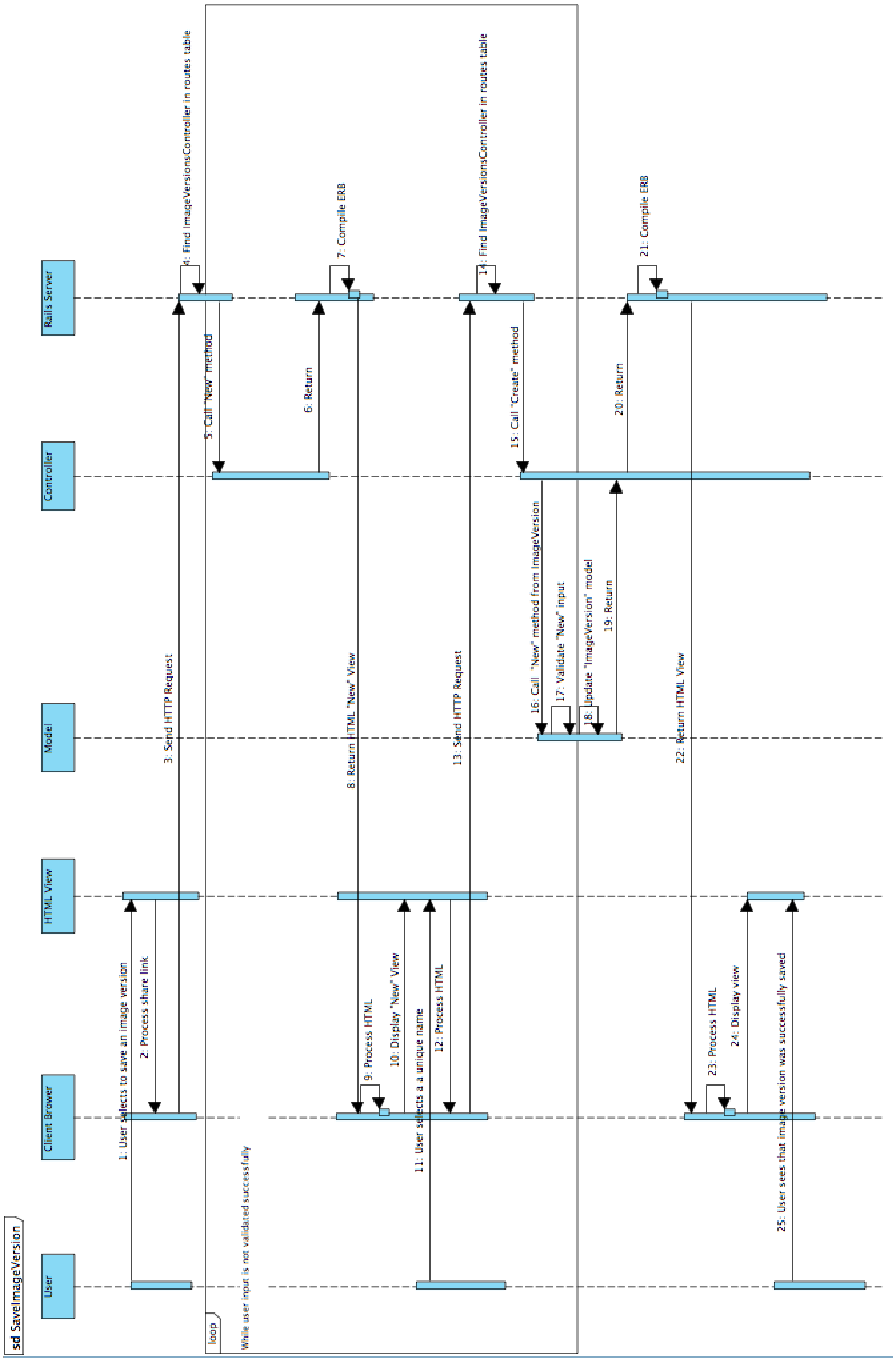

Figure 8: Saving an Image Version Sequence Diagram

4.7 Saving a Filter

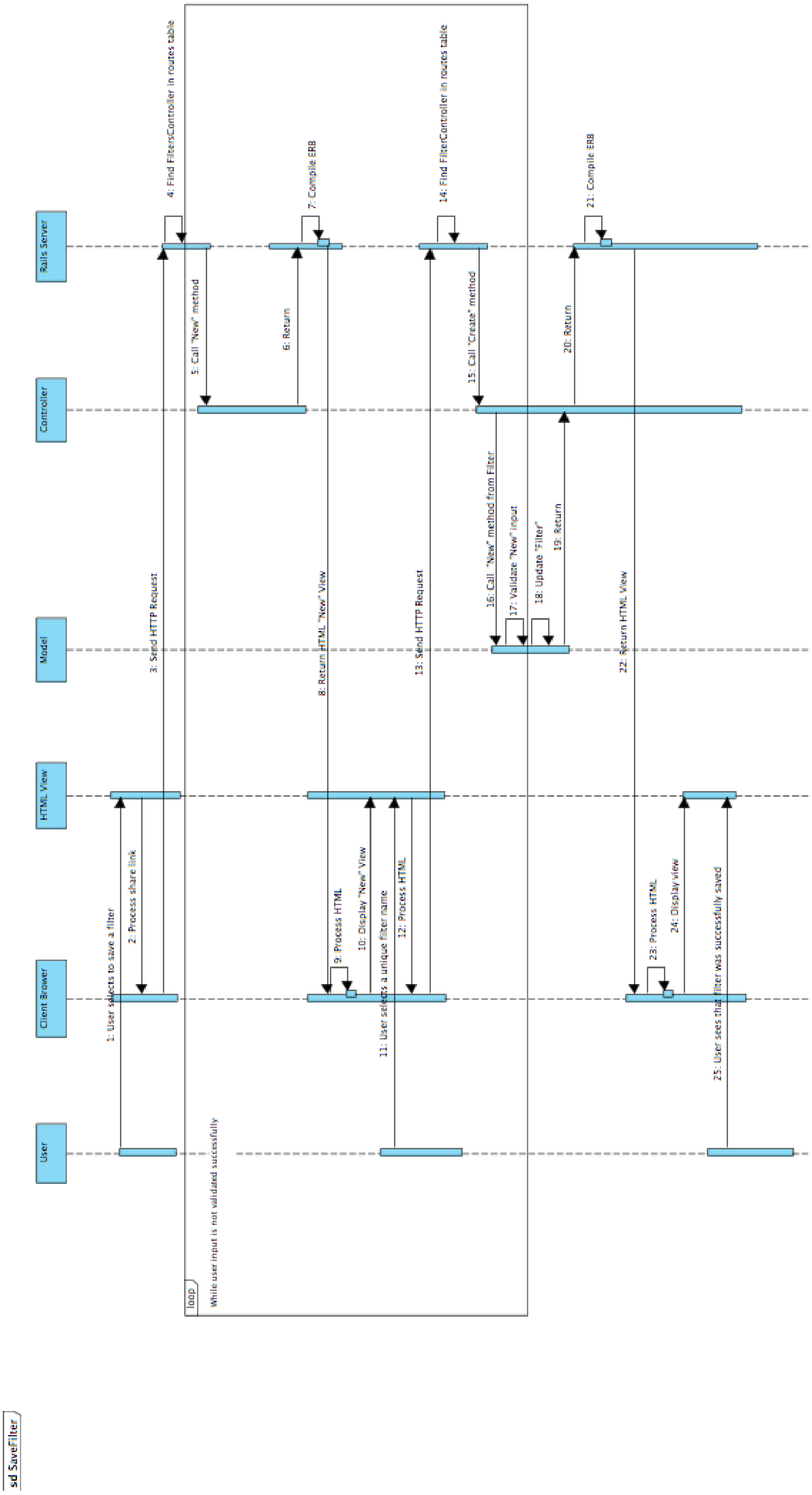

Figure 9: Saving a Filter Sequence Diagram

4.8 Deleting an Image

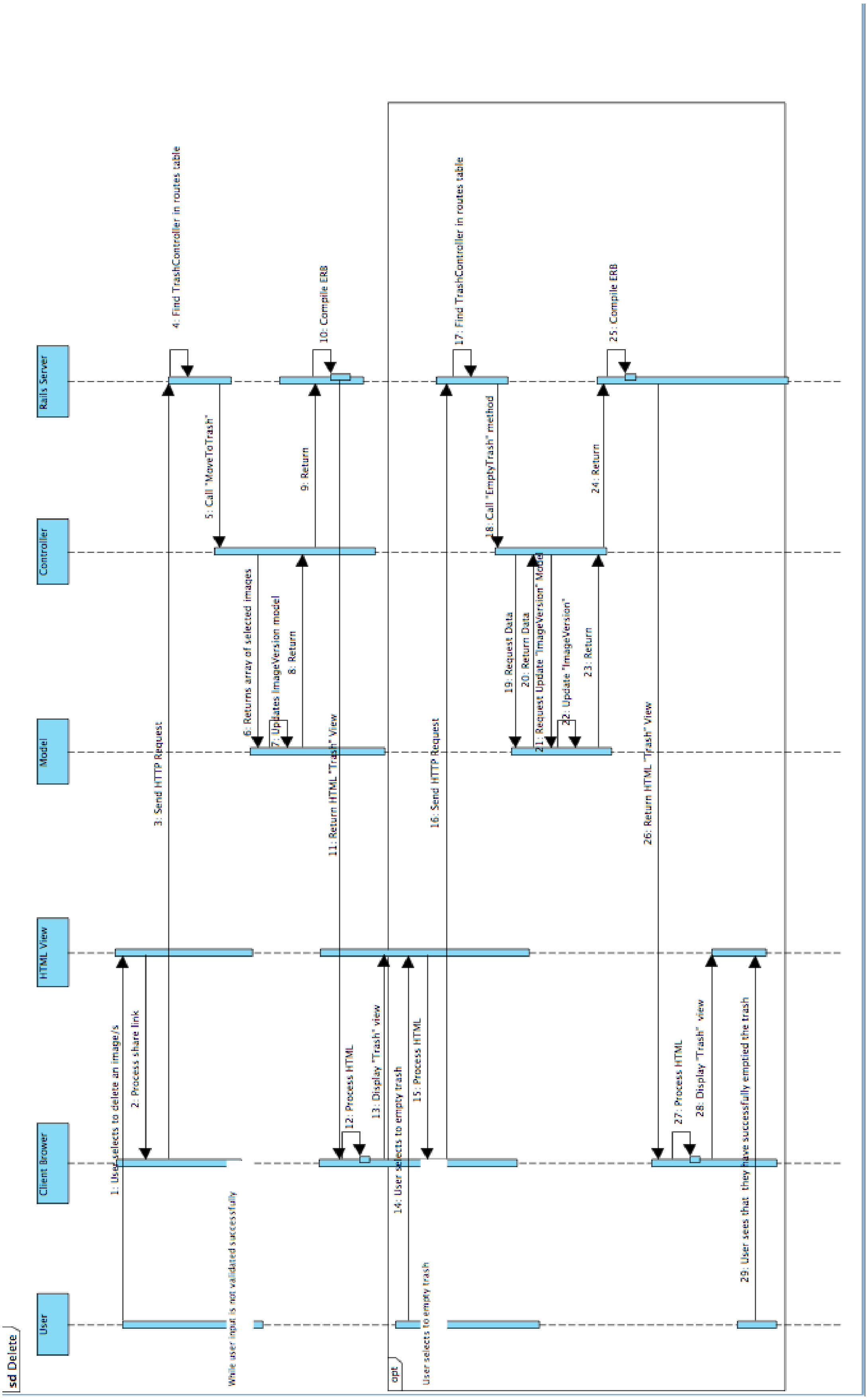
sd Delete
User
Client Brower
HTML View
Model
Controller
Rails Server
1: User selects to delete an image/s
2: Process share link
3: Send HTTP Request
4: Find TrashController in routes table
5: Call "MoveToTrash"
6: Returns array of selected images
7: Updates ImageVersion model
8: Return
9: Return
10: Compile ERB
11: Return HTML "Trash" View
While user input is not validated successfully
opt
User selects to empty trash
12: Process HTML
13: Display "Trash" view
14: User selects to empty trash
15: Process HTML
16: Send HTTP Request
17: Find TrashController in routes table
18: Call "EmptyTrash" method
19: Request Data
20: Return Data
21: Request Update "ImageVersion" Model
22: Update "ImageVersion"
23: Return
24: Return
25: Compile ERB
26: Return HTML "Trash" View
27: Process HTML
28: Display "Trash" view
29: User sees that they have successfully emptied the trash

Figure 10: Delete an Image Sequence Diagram