# FrameItIn

Rich Internet Application Project

Jenny Cooper – x12101303

Lucjan Stepien – x12118150

**Github Repository:**

https://github.com/menuitem/FrameItIn

**Deployed On:**

http://frameitin.herokuapp.com/

# Contents

## Summary

We developed a Rich Internet Application, which allows users to capture the video stream of their devices' camera, and convert it into pictures, which they can then edit and tweak, before either saving to their computer or uploading to cloud.

We did this using a variety of technologies and APIs, and making use of features available in HTML5.

We used **Ruby on Rails** to develop a backend server, which also takes care of security and user login using Devise. The Ruby on Rails backend also takes care of the connection to the database for the uploading and retrieval of user's pictures.

The rest of the functionality is all front end, using JavaScript, JQuery, HTML5 video and canvas elements, and using the following APIs:

**getUserMedia**

**localStorage**

**Easeljs** library for editing pictures (part of the createjs library)

**screenFull.js** – a cross browser wrapper for the FullScreen API

**ngAnimate** – from AngularJS (note: we did not use the full AngularJS framework for our project, but were able to implement some animation for displaying our pictures for 2 of our pages. This approach is discussed in further detail below).

The application is responsive for different sizes of browser/device. This was achieved using the **Bootstrap** fluid layout and Bootstrap grid layout.

Background themes are used from Bootstrap's **Bootswatch**, and the user is able to change themes dynamically.

The application is tested using **Jasmine** and **Sinon** test libraries.

We also used **Bower** package manager, which made it easy for us to manage our JavaScript libraries, and to install new ones.

Our application is cross-browser compatible for Chrome, Firefox and Opera. It is unfortunately not compatible for mobile or for IE at present, however as our project brief is to design an application in the year 2020, we hope that by then cross browser support and use with mobile applications will not be an issue!

Our application performs well, and has been tested on Chrome and Firefox. Automated manual testing of our JavaScript code proved difficult in places, especially since our code is very dom-heavy. At present, our testing coverage is at approx. 75%, so this would need to be improved on.

Looking back, some of our JavaScript functions could have been designed better in smaller modules, which would have facilitated testing better. Using a framework like Angular or Backbone.js would have helped greatly with this also. Since we were able to add some

animation for displaying pictures using ngAnimate from Angular, this would also have been a big reason to have used Angular for our entire project. However we were too far into our project to change over, but have definitely learned of the benefits of using Angular, which we can now apply to future projects.

## Scope

The scope of this project is to design a Rich Internet Application within the recreation environment. Our project allows a user to capture their camera's output and convert it into pictures, which can then be edited, and either saved or uploaded.

Although it is outside the scope of this project, due to time constraints, to be able to upload these photos to facebook, youtube and other social media, this is where we would see the future development and final use of this type of application.

The application can also be used to take suitable headshots and profile shots for linkedIn and other professional social medias.

Research has been put into which technologies are the most suitable for this application, and details on these technologies are given later in this document.

## Problem Domain

At present, having suitable pictures, for either fun or professional use, means having to download a picture from a device, edit it using editing software, save the picture again, and then upload to an online site. This would be much faster and hassle-free if it could all be achieved in the one application.

With the ability of HTML5's getUserMedia, this opens possibilities for taking a camera feed into a RIA application, where it could be transformed directly into pictures, and these pictures could then be modified and uploaded. In order for this work, all the functionality would need to be implemented through HTML and JavaScript, along with the use of external JavaScript libraries and APIs.

## Description of FrameItIn Application

FrameItIn is an application where users can use their device's camera to convert a video stream into a series of pictures.

Users can choose to save any of these pictures as a .png file on their computer, upload the picture to their own storage space in the FrameItIn database for later access. They can also edit the picture before saving or uploading.

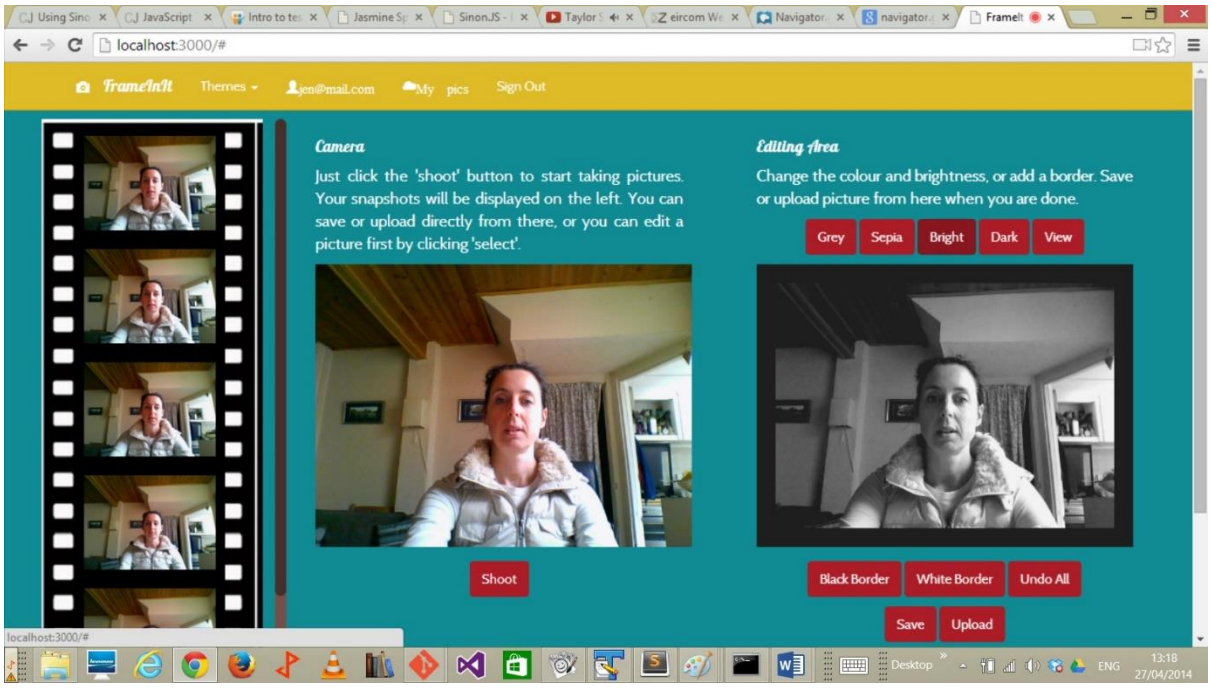
Editing options are:

- Turn colour to greyscale
- Turn colour to sepia
- Brighten or darken the image
- Draw a border

When the user is happy with their editing, they can now save the picture, or upload it to the database.

Users can view all their photos that are stored in the database on their home page.

Users can also choose what background theme they prefer to use for the application. Their preference will be stored, but they can change it at any time.




A public page is also accessible by anyone, which will display all the pictures that user's have designated as 'public'. This is a place where pictures can be shared and viewed by anyone.

## Research into Available Technologies

We chose the following technologies to best achieve the following functionality:

### Capturing video

In the past, third party plugins like Flash and Silverlight were necessary to be able to capture video and audio from a device's webcam. HTML5 has made it possible to do this without using a plugin. Initially, APIs like HTML Media Capture were developed. This API allows a video to be captured and changed into snapshot pictures, and is compatible with mobile devices. However the main downside of this API is that it can't be used to take the video and convert it to canvas elements for modification.

getUserMedia is an API that is suitable for being able to convert the video output into canvas elements so they can then be modified and adjusted. This API is unfortunately not supported by mobile devices and IE at present, but we decided that we would use this in our project instead of HTML Media Capture, with the hopes that mobile devices will soon be able to support getUserMedia in the near future. getUserMedia also has a feature to detect if a

camera exists, and if the API is supported, so this also means that the rest of our application will still work on non-supported browsers and mobile.

### Changing video to images

We wanted to be able to work with the pictures once we had captured them from the camera. HTML5's canvas element is cross browser compatible for all modern browsers, and allows rendering of images by placing it in a drawing area. The drawing area is then accessible and modifiable through JavaScript functions. We could also see that using canvas would make it easy to convert between images and canvas elements, so this would enable us to be able to download and upload as images to the database. The canvas element is also easy to use for things like drawing borders around our pictures.

### Editing pictures

There are many graphics libraries built on canvas that are available, like KineticJS, Fabric.js, three.js and Easeljs, to name but a few. We chose to use Easeljs, as it is straightforward to use, and we only needed to do a small amount of modification with our pictures (changing to black and white, lightening or darkening an image etc), compared to what some of the large and powerful libraries offer. Easeljs is one part of the createjs library, and only this part needs to be included for the types of modifications that we wanted for our pictures, so is lighter weight that some of the larger libraries.

### Uploading pictures to database

The simplest and neatest way to do this is using AJAX post requests. We wanted to validate what the user enters as a filename before the file gets sent to the server, so we have some validation that we wrote in JavaScript in the application.

### Retrieving pictures from database to display to users

We wanted to be able to display a user's pictures on their personal page, as well as displaying all public pictures from the database on the public page. We knew we would be using AJAX to retrieve and display the pictures in groups (eg 9 per page), so a user could choose to view more pictures and they would be dynamically loaded. In the end, we added some animation using ngAnimate, which is used in conjunction with AJAX to retrieve and display the pictures. This is mentioned in further detail in the next paragraph.

### Displaying the pictures using some animation – ngAnimate

This was not necessary as part of the core functionality of the application, but we felt we had enough time at the end to add some extra nice-to-have functionality into the project. This was done using the ngAnimate directive from AngularJS. We did not use Angular as a framework in general, but were able to use this directive to give an appearance that the pictures fade into the screen when they are loading.

### Viewing in Fullscreen Mode

We wanted to be able to display a picture to the user in full screen mode when the user has finished editing it, so they can see the picture clearly before they decide to save it. This is possible through JavaScript, however there are different ways for doing this for different browsers. So instead we used the screenfull.js API, which wraps the JavaScript API but takes care of cross browser compatibility, meaning only one straightforward call is required. It can also let he application check if fullscreen mode is enabled on the browser, so this makes it easy to handle the code if fullscreen mode is not enabled.

### User preferences, look-and-feel and styling

We wanted the user to be able to have some type of user preferences that would be stored by the application, so as to make the look and feel of the application more fun and more personal. We did this by using Bootswatch themes from Bootstrap, which the user can change dynamically. This also supplied some really nice css styling to our application so we did not need to write as much css. In order to store the user's choice of theme, we decided to make use of the local storage of the browser. This saves unnecessary calls to the server to retrieve the information, and HTML5's local storage is supported by all modern browsers. It is also very simple to use the localStorage APIs – only a single line of code is needed to add and retrieve items from local storage.

### Responsive Layout

To achieve a responsive layout we needed to choose a good grid layout. Twitter Bootstrap makes it easy to implement a grid layout, and can make the whole page fluid, which puts all widths as percentages instead of a fixed width. Using Bootstrap also enabled us to be able to use the Bootswatch themes mentioned above.

### Testing Frameworks/Libraries chosen:

Jasmine is a stand-alone test framework for testing JavaScript code. There are other test frameworks available, like QUnit from jQuery and YUI from Yahoo. But using a standalone framework that is not tied to a coding framework is beneficial as it makes the tests more impartial.

Jasmine is simple to set up and very easy to use. It has a good selection of assertions for matching the results of the tests, and can divide tests into suites to make them easier to organise. It also supplies a setup and teardown function (beforeEach and afterEach) for setting up test data.

Jasmine also has the ability to run tests using spys, stubs and mocks. However we chose to use a further library, sinon.js, to run these types of tests.

Sinon is a JavaScript library that helps make testing easier. Like Jasmine, it is also standalone, so has no dependencies on other frameworks. It works well in conjunction with Jasmine, and is cross browser compatible.

Spies can be used to mimic a function and check its behaviour, or how often it was called. Stubs and mocks can be used to mimic functionality, and are useful when testing calls to libraries like jQuery or other APIs.

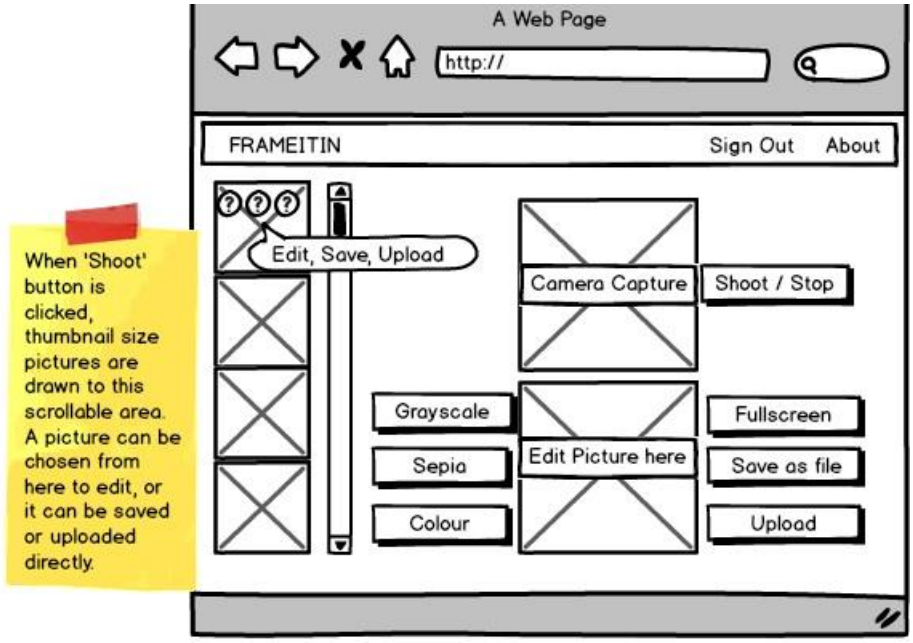
## User Interface Design

### Wireframes

#### Home Page: Camera and Editing Page - Publicly available to anyone.

This page allows any user to take pictures with the camera and edit them. They can also save them to their own computer.

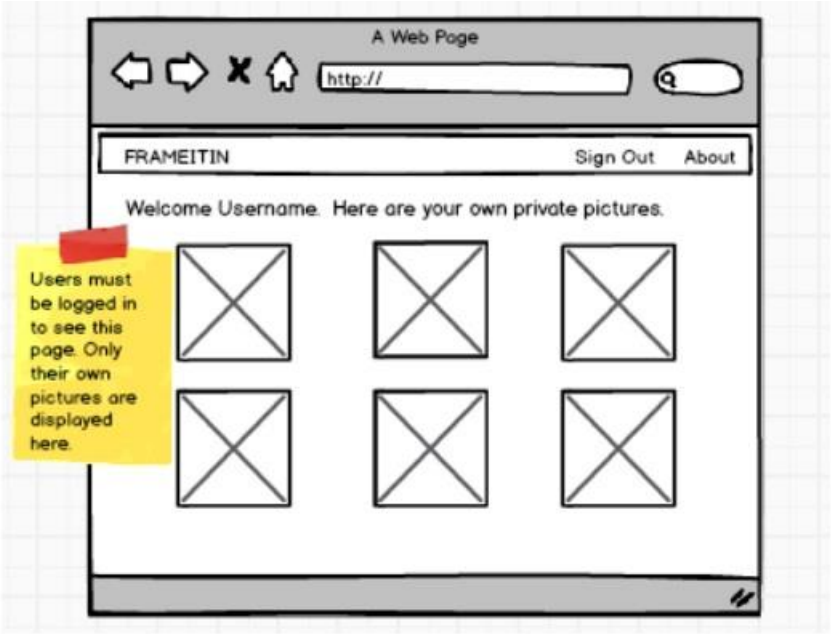
However, in order for the user to upload a picture to the FrameItIn database, they must be logged in.




They must also be logged in to continue on to the next page, which displays all their pictures that they have stored on the database.

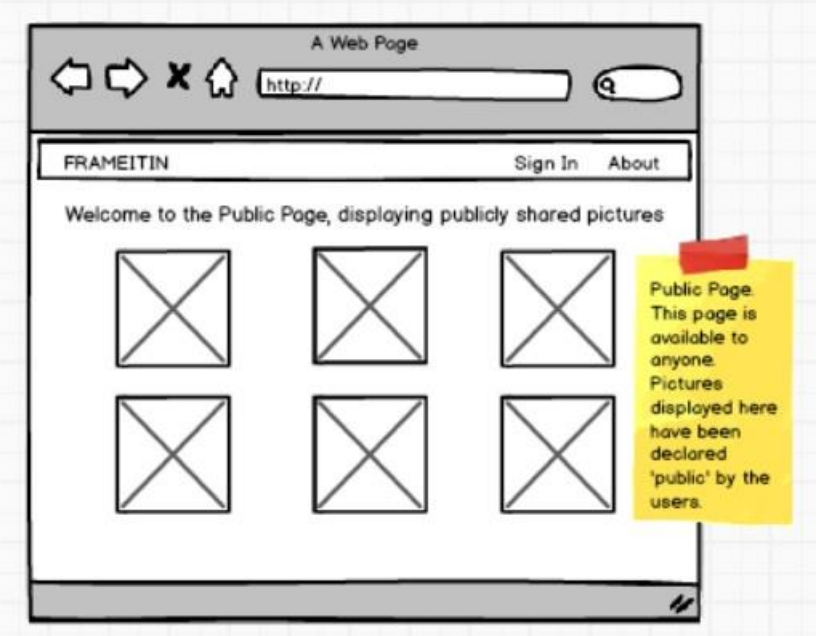
### Users' Pictures Page: User must be logged in to access this page

The pictures they have uploaded to the database are displayed here.




### Public Pictures Page: Available to view by anyone.

This page will display pictures that user's have uploaded and tagged as 'public'.

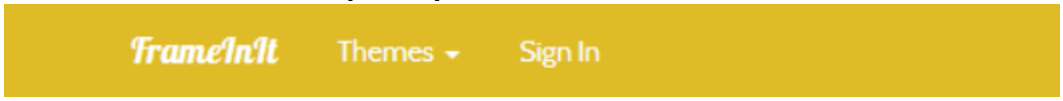



## User Interface Interaction

We have designed the application to be as straightforward and simple to use as possible.

Since this is a Rich Internet Application, there is a lot of functionality and therefore a lot of controls, so a lot of design consideration was put into them.

**The main menu bar is kept simple:**

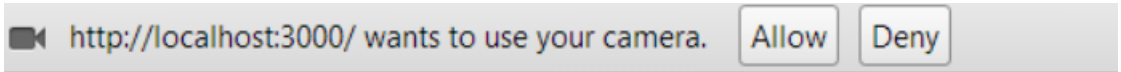



This menu bar scrolls with the page, so is always visible no matter how far down the page the user has scrolled to.

The FrameItIn logo has a link to this home page behind it, so the user can always easily return here.

The Themes dropdown list allows users to change the background theme of the application at any time. Their theme preference is stored in the browser's local storage.

**Allowing the application to access the camera:**




The rest of the page does not load until the user chooses to allow access to their camera.

**Starting and stopping the camera:**

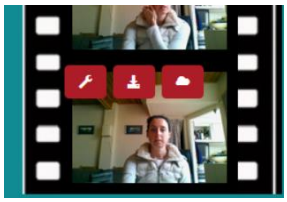
There are 2 ways to do this.

There is a large ‘Shoot’ button directly below the camera feed on the screen.


A camera icon also appears in the main menu bar when the user has allowed access to the camera. Clicking this icon starts and stops the camera. This icon turns red when the camera is running, and returns to white when the camera stops. Since the menu bar scrolls with the page, this icon is always available even if the user has scrolled down lower that the ‘shoot’ button.



**Selecting from the choice of small snapshots:**



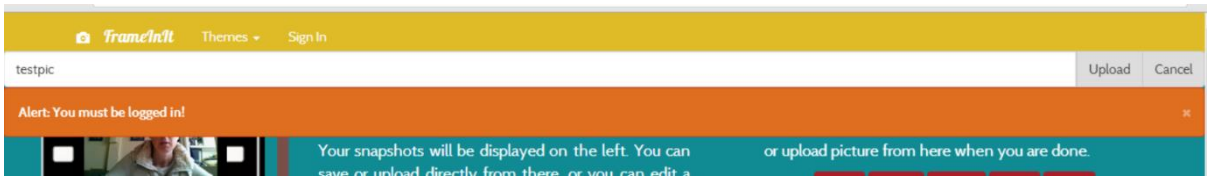
Hovering over any of these pictures will bring up 3 buttons, for edit, save or upload.

Hovering over any of the buttons displays a tooltip for what the button does.

**Uploading a picture to the database or saving to computer:**


When the user clicks on the ‘upload’ or ‘save’ link, an input box will appear so they can enter the name for their picture. The input box is tied to the menu bar, so is always visible no matter how far down the page the user has scrolled.

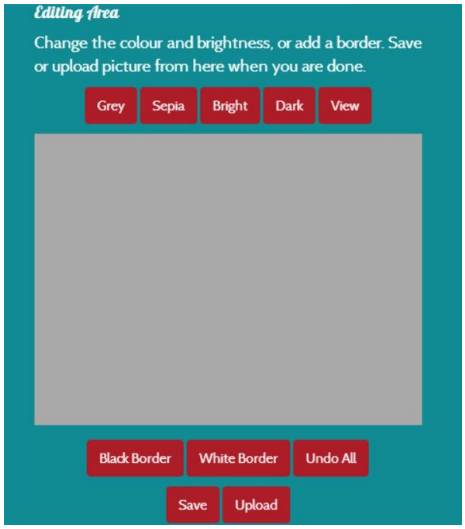
**Trying to upload a picture without being logged in:**




A large alert message is displayed if the user tries to upload without logging in.

This alert message is tied to the menu bar, so is always visible no matter how far down the page user has scrolled.

**Editing a picture:**




Large buttons make editing and saving simple for the user.

Clicking ‘Save’ or ‘Upload’ has the same controls as mentioned above.

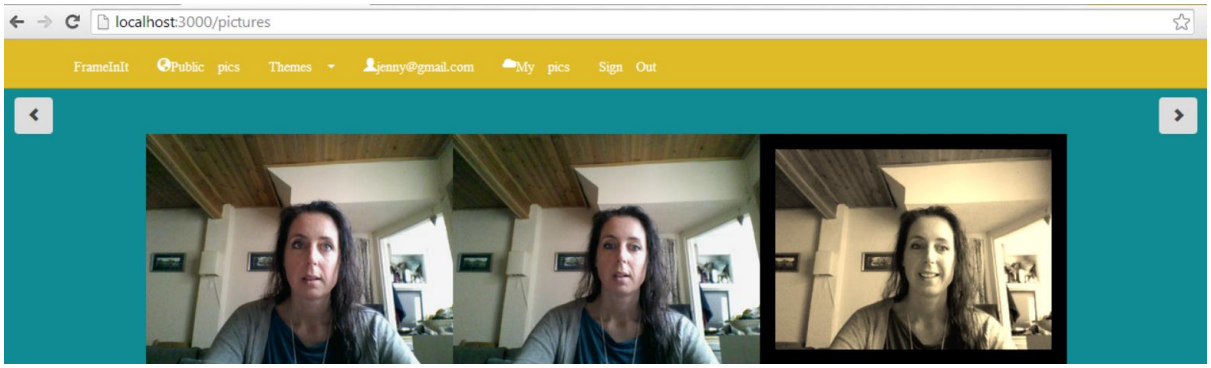
**Viewing personal page with all user’s pictures:**


A user must be logged in to upload a picture, and to view their pictures that are stored on the database.

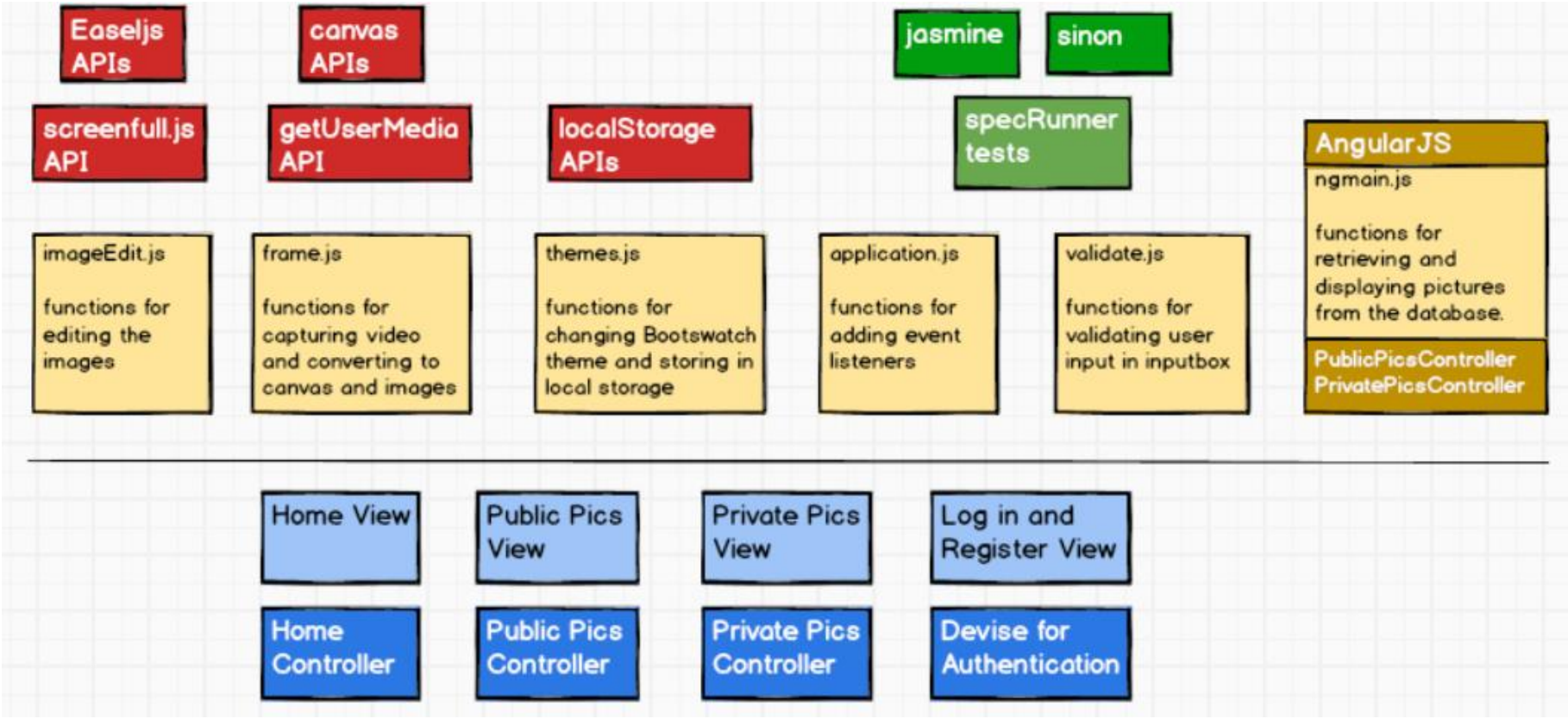
When the user logs in, the menu bar at the top of the page additionally includes their username, and a link to the user’s pictures page. Clicking the ‘My Pics’ link will bring the user to their personal pictures page.

These will not be displayed in the menu bar when the user logs out.

**Scrolling through pictures on database:**




Easy to use left and right arrows make scrolling through the pictures simple.


Easeljs APIs
canvas APIs
jasmine
sinon
screenfull.js API
getUserMedia API
localStorage APIs
specRunner tests
AngularJS
ngmain.js
functions for retrieving and displaying pictures from the database.
PublicPicsController
PrivatePicsController
imageEdit.js
functions for editing the images
frame.js
functions for capturing video and converting to canvas and images
themes.js
functions for changing Bootswatch theme and storing in local storage
application.js
functions for adding event listeners
validate.js
functions for validating user input in inputbox
Home View
Public Pics View
Private Pics View
Log in and Register View
Home Controller
Public Pics Controller
Private Pics Controller
Devise for Authentication

# Details of Technologies Used

## getUserMedia API

One of the main technologies in our application is making use of the getUserMedia API, which gives our app access the user's device camera, without having to use a third party plugin. The user can choose to allow the getUserMedia API access to their camera, which then streams this video to a html5 <video> element.

(Similarly, this API can also be used to stream audio from the user's microphone to an <audio> tag, however we did not use audio in our application).

At present, Browser support is for Chrome, Firefox and Opera. The following code snippet shows how different browsers can be supported:

```
navigator.getMedia = ( navigator.getUserMedia || navigator.webkitGetUserMedia ||
                       navigator.mozGetUserMedia || navigator.msGetUserMedia);
```

In our code, we first check that getUserMedia is supported:

```
if (navigator.getUserMedia) {
    // Call the getUserMedia API
} else {
    // Display message to the user that camera is not supported.
}
```

The call to the getUserMedia API sends in a parameter with 'video' set to true, and also success and failure callback functions, called gotStream and noStream:

```
function (gotStream, noStream) {
                navigator.getMedia({
                  video: true
                }, gotStream, noStream);
```

The noStream failure callback will display an error message to the user.

The gotStream success callback function will send the video stream to a <video> element. From here, the stream is converted to a series of <canvas> elements, using the drawImage()

API and gives the canvas a height and width based on the video.videoHeight and video.videoWidth attributes.

## HTML5 Canvas Element

This has been used extensively throughout the application. Javascript code can access and modify the canvas area through APIs.

Some of the canvas APIs used were:

drawImage(): The video stream or images are drawn to a canvas element.

toDataUrl(): Converting the canvas to an image, as this needs happen before the picture can be downloaded to a file on the user's computer

ctx.rect() and ctx.stroke(): To draw borders around the picture.

## Easeljs library (part of the Createjs library)

This library builds on the HTML canvas element, by creating a 'stage' element, and has many APIs for modifying the canvas. In our case, we are modifiying an image, which gets converted and used as a bitmap by easeljs. The easlejs APIs made it simple to apply colour filters to change the picture to greyscale and sepia, and to brighten and darken the picture.

Examples of this are:

```
var turnGreyScale = function(canv) {
                var image = FrameItIn.canvasToImage(canv);
                var bmp = new createjs.Bitmap(image),
                        stage = new createjs.Stage(canv);
                var greyScaleFilter = new createjs.ColorMatrixFilter([
                        0.33, 0.33, 0.33, 0, 0, // red
                        0.33, 0.33, 0.33, 0, 0, // green
                        0.33, 0.33, 0.33, 0, 0, // blue
                        0, 0, 0, 1, 0  // alpha
                ]);
                bmp.filters = [greyScaleFilter];
                bmp.cache(0, 0, image.width*3, image.height*3);
                stage.addChild(bmp);
                stage.update();
        }
```

```
var brighten = function(canv) {
                    var image = FrameItIn.canvasToImage(canv);
                    var bmp = new createjs.Bitmap(image),
                            stage = new createjs.Stage(canv);

                     var matrix = new createjs.ColorMatrix().adjustBrightness(30);
                     bmp.filters = [new createjs.ColorMatrixFilter(matrix)];

                     bmp.cache(0, 0, image.width*3, image.height*3);
                     stage.addChild(bmp);
                     stage.update();
          }
```

## screenFull.js API

screenFull.js is a wrapper API for the JavaScript FullScreen API, which makes it cross browser compatible. This is used to show the edited canvas element in fullscreen mode. Because this API takes care of cross browser issues, only one simple call to the API is required. It also makes it simple to check if Fullscreen is enabled in the browser.

```
document.getElementById('fullscreen').addEventListener('click', function () {
        if (screenfull.enabled) {
                screenfull.request(elem);
                }
        });
 }
```

## localStorage API

The localStorage API is used to store the user's preference for their Bootswatch background theme in the browser's local storage. This API is very simple to use, with one method for storing and another for retrieving information:

```
localStorage.setItem("themeurl", themeurl)


var themeurl = localStorage.getItem("themeurl") || themes['default'];
```

## JQuery

JQuery was used to display the input box for users to enter the filename of the picture they wish to save. Input box does not display until the user has clicked ‘Save’ or ‘Upload’. JQuery was also used for the small buttons that slide when the user puts their mouse over the small pictures.

## JQuery with AJAX

JQuery and AJAX was used as a post request to upload a picture to the server (and the database).

## ngAnimate

This is one of the directives from Angular. Angular http AJAX GET requests are used to get the next set of 8 pics from the server, and Angular scope directive is used to display them, and give an animation effect that the pictures slowly fade in to the page as they load.

## slimScroll.js

slimScroll.js is a lightweight jquery plugin, for  displaying nicely styled scrollbars.

## Bower Package Manager

Using the Bower package manager made it really simple to install and update the JavaScript libraries and other third party tools that we used.

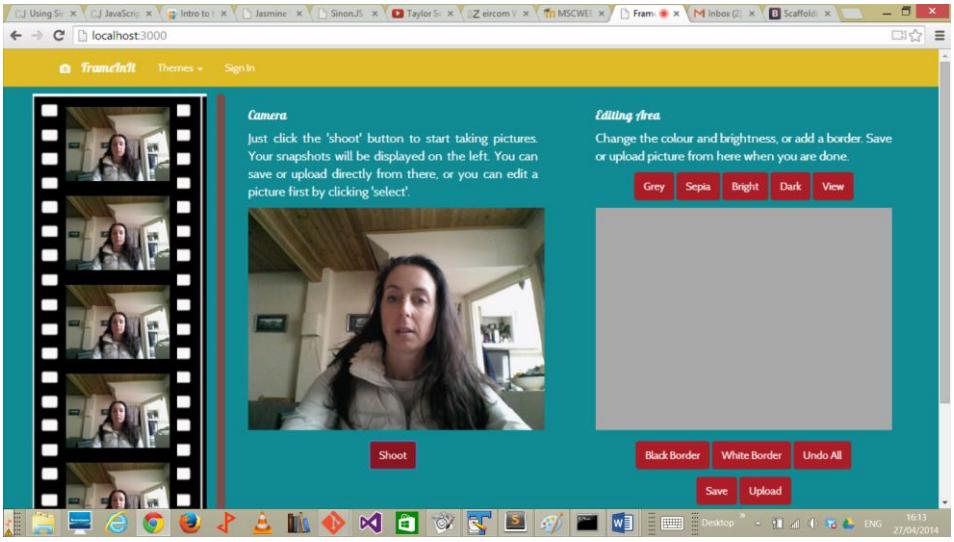
All files required by our application were then automatically stored in a Vendor\assets directory.

## Responsive Layout Design and Grid Design using Twitter Bootstrap

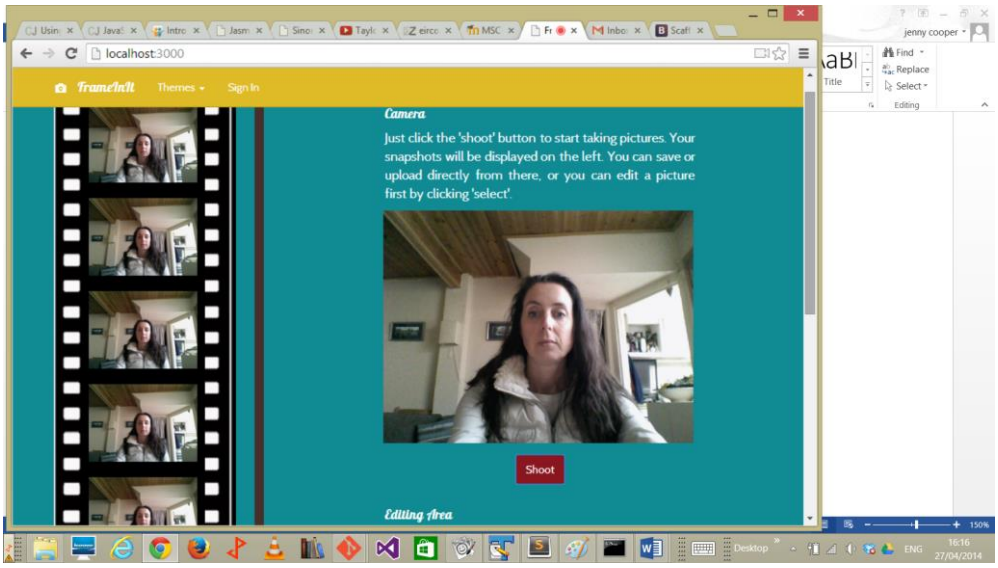
We used Twitter Bootstrap fluid grid to implement a responsive layout. The default Bootstrap layout uses a 12 column grid layout, and the fluid grid layout gives these 12 columns their width based on percentage instead of a fixed number of pixels. This grid layout also automatically adjusts the main menu bar to become a dropdown list for smaller display sizes.

Our layout is also responsive, using Bootstrap features, and will scale nicely for any size device. Bootstrap can detect when a display is either ‘large’, ‘medium’ or ‘small’, and we were able to change the grid layout based on these settings.

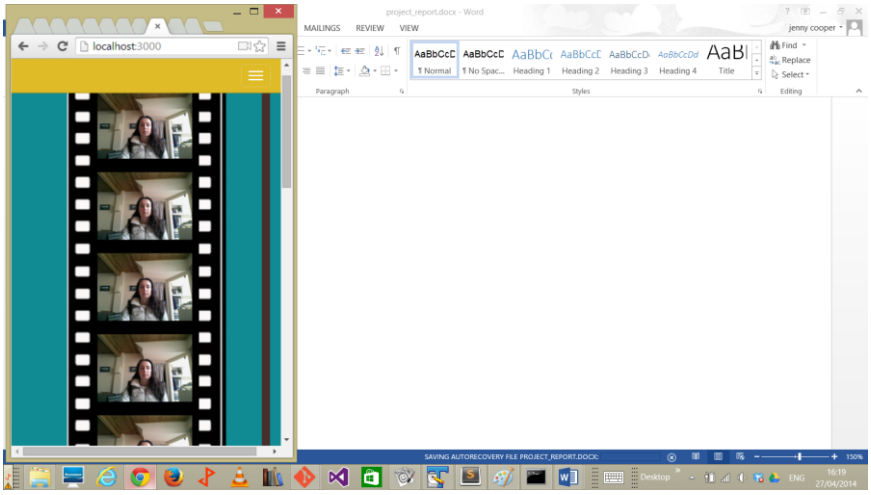
So for large displays, our main page is 3 columns for the strip of pictures div, 4 columns for the camera div and 5 columns for the editing div:




For a medium sized display (tablets & notebooks), we changed our grid layout, so the small pics on the left are now 4 columns wide and the camera div and editing div are each 8 columns wide. This forces these 2 divs to be one above the other:




Then finally for small displays (mobile phones), the small pics on the left become 12 columns wide, so take up the whole width of the screen. This forces the other 2 divs to move below this div, and can be accessed by scrolling down.




For the 2 pages that display the personal and public pictures, this was simply achieved by centering the div that the picture are in. The whole page is in a Bootstrap fluid container (ie based on percentages), so the number of pictures that can be displayed in a row gets smaller as the display size decreases. They will always remain centered however.

### Ruby on Rails

We chose Ruby on Rails as our backend server. Most of our application code is front end, however the rails application handles to upload and retrieval of pictures from the database. Rails applications are easily deployable on Heroku. The Rails application also handle the user login, which is mentioned further in the Security section. Any further details about Ruby on Rails is outside the scope of this document, as the project is a front-end project.

## Security

User sign up and log in is handled though Ruby on Rails' Devise gem. This handles all the authentication required by the application. Device encrypts and stores users' password in the database and validate the user against this when they sign in. Further discussion about this gem is outside the scope of the document, but is mentioned here to detail how security has been handled by the application.

We have one input box in the application. Validation has been added to the input box, to help avoid cross site scripting and SQL injections. Further validation is also available at the backend, but having it at the front end also means that data does not need to be sent to the server to be validated and therefore saves a call to the server.

The user input from the input box is checked to ensure that only valid letters and numbers are entered, so this will stop a user from being able to enter malicious scripts.

## Testing

In our project we used the Jasmine and Sinon test frameworks to write unit tests for our code. Our tests are divided into 4 test suites, one for each of our JavaScript files.

Some of our code proved a bit difficult to test, as it is very DOM-heavy. We were able to assert that our functions which created new DOM elements behaved as expected. The functions we were unable to test were the ones that made use of existing DOM elements. A couple of our functions were quite large, and if they were to be re-designed into smaller functions, this would also have meant we could have tested the various parts better. In these larger functions, we have tested a lot of the functionality but did not quite cover every aspect of each if or while statement.

Using a framework like Angular would have made this easier.

We also used sinon.js with Jasmine for running tests with spys and mocks. We were able to test calls within the functions to HTML APIs (eg canvas.drawImage()) using sinon spys.

When our functions made calls to external APIs like easeljs and screenFull, we were able to use sinon to mock the APIs and assert that they were called correctly.

Our code coverage is at approx 75%.

NOTE: all tests are within the jsunittests directory in our project.

### Unit Test Results and Coverage:

| **JavaScript Function** | **Test Library** | **Result** |
|---|---|---|
| ImageEdit.turnGreyScale | Sinon mock | Pass |
| ImageEdit.turnSepia | Sinon mock | Pass |
| ImageEdit.darken | Sinon mock | Pass |
| ImageEdit.brighten | Sinon mock | Pass |
| ImageEdit.drawWhiteBorder | Sinon spy | Pass |
| ImageEdit.drawBlackBorder | Sinon spy | Pass |
| ImageEdit.returnColour | Sinon spy | Pass |
| ImageEdit.displayFullScreen | Sinon mock | Pass |

| | | |
|---|---|---|
| | | |
| FrameItIn.canvasToImage | jasmine | Pass |
| FrameItIn.getCanvasFromCamera | jasmine | Pass |
| FrameItIn.getAlertDiv | jasmine | Pass |
| FrameItIn.getLinks | jasmine | mostly covered by test |
| FrameItIn.getFileNameDiv | jasmine | mostly covered by test |
| FrameItIn.uploadToCloud | | not tested |
| FrameItIn.takeShot | | not tested |
| | | |
| Validate.fileName | jasmine | Pass |
| Validate.currentUser | jasmine | Pass |
| | | |
| themes.js - anonymous function for changing Bootswatch themes | Sinon mock | Tests the call to the localStorage, where the theme is stored. Does not test the rest of the function. |
| ng-main.js - angular animation | | not tested |

## Future Evolution

Given the chance, we would like to re-write the application using Angular. This would simplify our JavaScript code and keep it neater, as it will be difficult to maintain the code we have if the application were to get any bigger. It would also make testing a lot easier, and introduce all the additional functionality that comes with Angular.

We would like to enhance the functionality of the application to allow users to upload their pictures directly to facebook or other social media applications.

We would also like in the future, if the technologies become available, to have this app working on mobiles.

## References


Banners.aquafadas.com, (2014). *EaselJS Example: Filters test file.*. [online] Available at: http://banners.aquafadas.com/yann/htmlExperiments/easelJS/easelJS/examples/filters.html [Accessed 30 Apr. 2014].

Createjs.com, (2014). *EaselJS v0.7.1 API Documentation : ColorFilter*. [online] Available at: http://createjs.com/Docs/EaselJS/classes/ColorFilter.html [Accessed 30 Apr. 2014].

Diveintohtml5.info, (2014). *Local Storage - Dive Into HTML5*. [online] Available at: http://diveintohtml5.info/storage.html [Accessed 30 Apr. 2014].

Docs.angularjs.org, (2014). *AngularJS*. [online] Available at: https://docs.angularjs.org/api/ngAnimate [Accessed 30 Apr. 2014].

Elijahmanor.com, (2013). *Unit Test like a Secret Agent with Sinon.js*. [online] Available at: http://www.elijahmanor.com/unit-test-like-a-secret-agent-with-sinon-js/ [Accessed 30 Apr. 2014].

HTML5 Rocks - A resource for open web HTML5 developers, (2012). *Capturing Audio & Video in HTML5 - HTML5 Rocks*. [online] Available at: http://www.html5rocks.com/en/tutorials/getusermedia/intro/ [Accessed 30 Apr. 2014].

Jquery-plugins.net, (2014). [online] Available at: http://jquery-plugins.net/screenfull-js-simple-cross-browser-wrapper-for-javascript- fullscreen-api [Accessed 30 Apr. 2014].

Mozilla Developer Network, (2014). *Navigator.getUserMedia*. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/API/Navigator.getUserMedia [Accessed 30 Apr. 2014].

Rocha.la, (2014). *jQuery slimScroll | rocha.la*. [online] Available at: http://rocha.la/jQuery-slimScroll [Accessed 30 Apr. 2014].

W3resource.com, (2014). *Create full screen view with screenfull.js | w3resource*. [online] Available at: http://w3resource.com/gallery/create-full-screen-view-with-screenfull-js [Accessed 30 Apr. 2014].