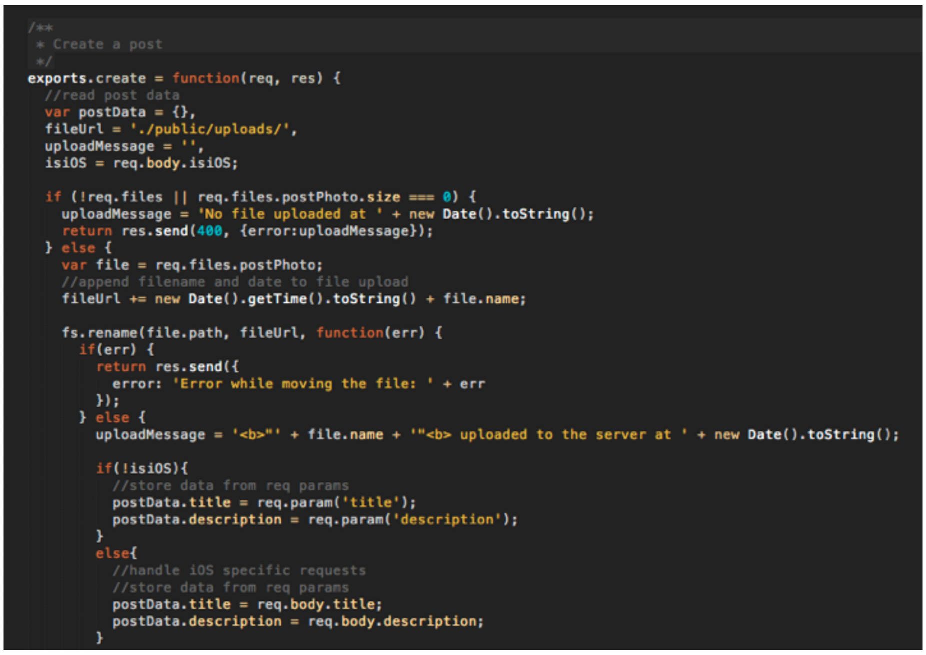


```
/**
 * Create a post
 */
exports.create = function(req, res) {
  //read post data
  var postData = {},
  fileUrl = './public/uploads/',
  uploadMessage = '',
  isiOS = req.body.isiOS;

  if (!req.files || req.files.postPhoto.size === 0) {
    uploadMessage = 'No file uploaded at ' + new Date().toString();
    return res.send(400, {error:uploadMessage});
  } else {
    var file = req.files.postPhoto;
    //append filename and date to file upload
    fileUrl += new Date().getTime().toString() + file.name;

    fs.rename(file.path, fileUrl, function(err) {
      if(err) {
        return res.send({
          error: 'Error while moving the file: ' + err
        });
      } else {
        uploadMessage = '<b>"' + file.name + '"<b> uploaded to the server at ' + new Date().toString();

        if(!isiOS){
          //store data from req params
          postData.title = req.param('title');
          postData.description = req.param('description');
        }
        else{
          //handle iOS specific requests
          //store data from req params
          postData.title = req.body.title;
          postData.description = req.body.description;
        }
```

Photo campaign platform: frameworks report

Prepared for: Frameworks elective final mandatory assignment

Prepared by: Dan Mindru

21 May 2014

Notice: the screenshots presented in this document represent the current state of the web & iOS application and may have been improved or extended in the meantime. (Differences may be visible on photocampaign.net)

TABLE OF CONTENTS

INTRODUCTION

The following document contains diagrams and screen dumps describing the functionality of the web platform. A separate document is provided with a full description of the iOS Application. Diagrams in this document are available in the provided '**resources**' folder.

The web application is deployed at photocampaign.net and a demo account can be created by going to the signup page.

A **video** demo of the entire platform is available at youtube.com/watch?v=xlu2rq1LZaY

The video begins with the presentation of the web platform, then continues with describing technical information (code explanations for the web back-end and front-end). Following, a short iOS code explanation is provided and finally the iOS app is demoed in relation to the web platform.

It is recommended that the video is watched after reading this document.

CHOICE OF FRAMEWORK

This application is built using the MEAN full stack solution (note: **MEAN.JS** is used, not MEAN.IO). One of the reasons for this choice is first of all based on the unified language that governs the entire stack: Javascript. Having just one language to code in simplifies the task at hand and introduces plenty helpful tools that help increase the quality. Except for the testing tools (which are dependencies but not extensively used in this app), the use of Javascript allows deployment of helpers such as **jshint**, **Grunt** or **Bower** over **npm.**

Another reason is founded in **Express**: mainly the ease of use when it comes to building **REST** services. To add up, **REST** becomes a lot simpler when **Mongoose** is involved with data modelling. In addition to this, **Mongo** (and the BSON format) provides a simple way to output JSON objects, without having to do any data conversions on the server side.

Moreover, **AngularJS'** abstractions allow to create a front-end logic that communicates easily with the **RESTful** architecture in the back-end. **Angular**'s routing brings up an immense improvement in speed (which is already up one notch to due node's async core).

THOUGHTS ON LARAVEL & PHALCON

Although proven to be well built frameworks and in **Laravel**'s case very pleasurable to develop in (documentation is a plus), the use of php for the back-end would result in a less harmonious communication between it and the front-end. In many aspects, both the php frameworks are better, especially because of their maturity. However, the fact that the **MEAN** stack has access to **npm**, is meticulously organised in modules, makes it easy to develop **REST** services with **NoSQL** and on top of that has a unified language that powers it narrows the advantage gap gained by php's maturity.

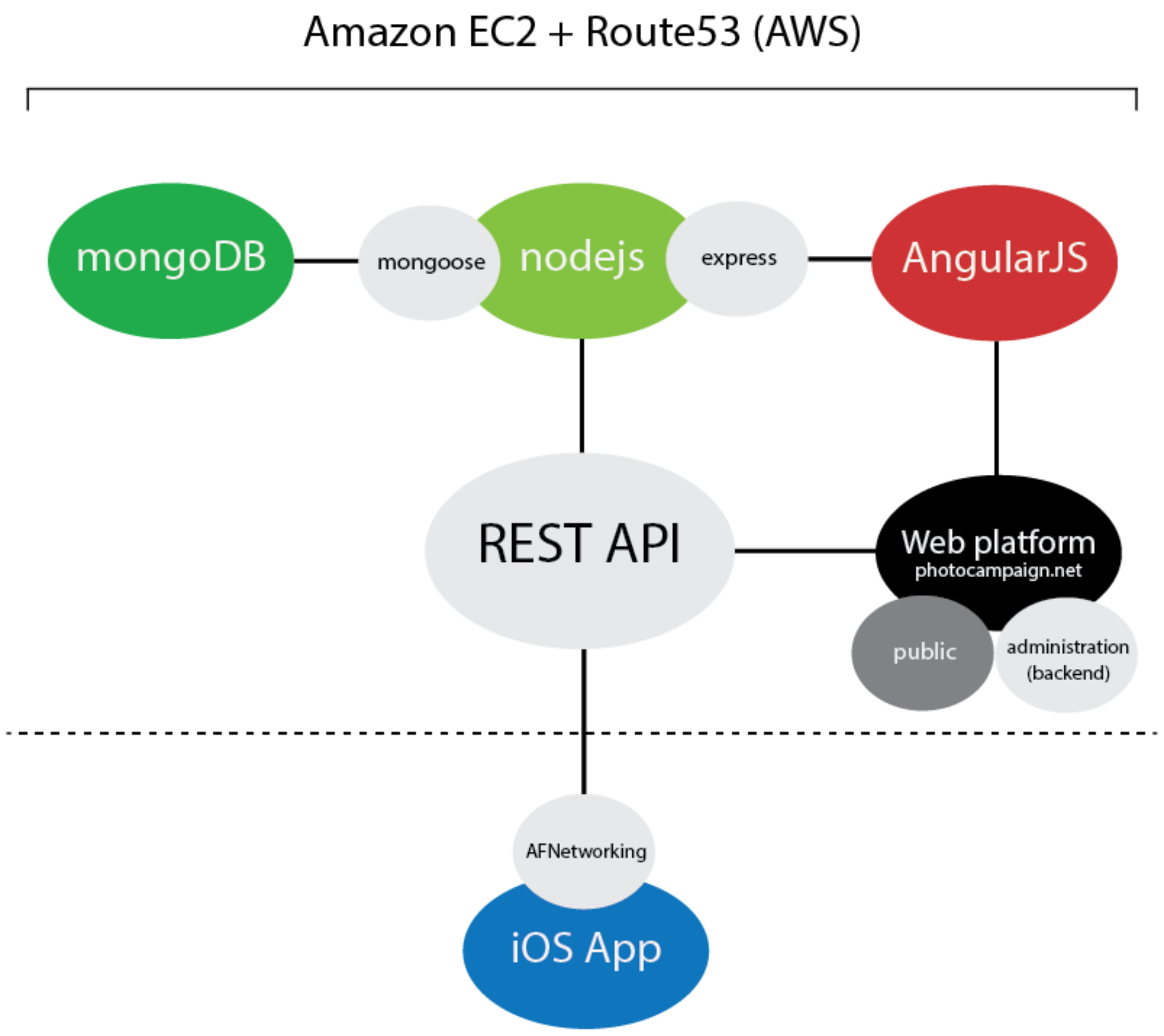
THOUGHTS ON .NET MVC 5

Microsoft's **MVC 5** is perhaps one of the most secure choices, with many built-in libraries. However, the fact that it is closed source, expensive to deploy for production and very slow to send and handle requests (even on localhost) makes it a easy choice. On the flip side, Microsoft provides great documentation, but that also comes at the cost of developing the 'Microsoft way'.

CONCLUSION

Finally, viewing the entire MEAN stack, a modular and extensible structure both for the front and back-end can be pointed out. All of the stack components seem to fit like a glove and make great use of Mongo's flexibility. Due to it's (relatively) new and different approach on web development, **MEAN** is attractive to develop in and looks like a solution which has high future potential once it will become more mature.

ARCHITECTURE OVERVIEW



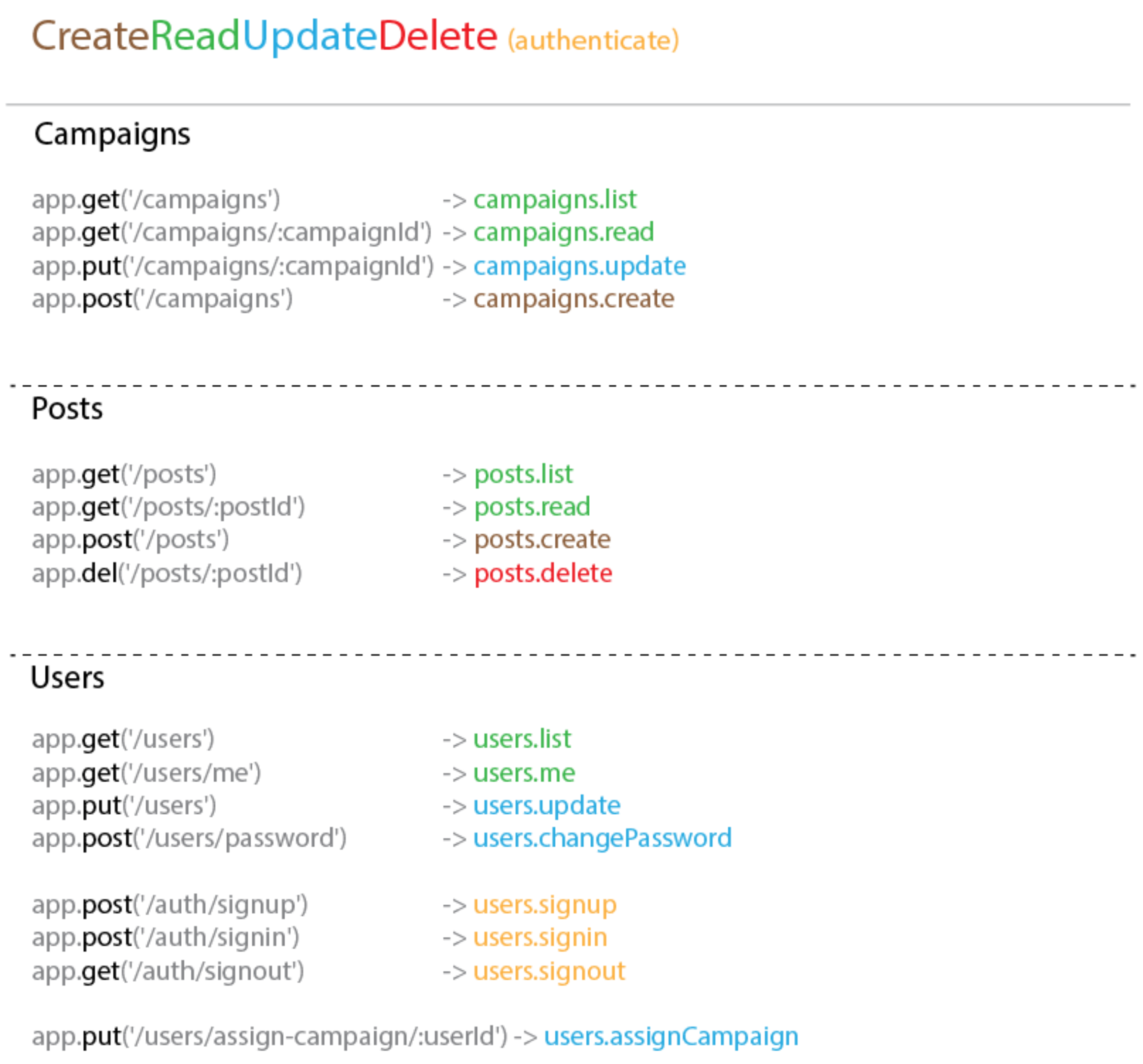
The above diagram shows how the infrastructure of the platform is organised. Web servers are hosted by Amazon on a ElasticCloud2 instance. Deployed on it are a nodejs server and a NoSQL database provided by MongoDB. The nodejs server communicates with the database through mongoose, a package that allows data modelling to be done as well as model validation.

The back-end routing and MVC logic is provided by Express, a nodejs web application framework. Finally, the web platform itself (on photocampaign.net) is built with AngularJS, a front-end framework that allows also the front-end code to perform according to a MVC pattern.

The express back-end exposes a REST API that is used by both the web and iOS application. The iOS application uses AFNetworking (a third-party library) to communicate with the REST API. The iOS app saves data received from the API in plist files.

REST OVERVIEW

In the illustration below the main methods of the REST API are shown. The diagram depicts what requests and methods are implemented for Campaigns, Posts and Users.



CreateReadUpdateDelete (authenticate)

Campaigns

```
app.get('/campaigns')                -> campaigns.list
app.get('/campaigns/:campaignId')    -> campaigns.read
app.put('/campaigns/:campaignId')    -> campaigns.update
app.post('/campaigns')               -> campaigns.create
```

Posts

```
app.get('/posts')                    -> posts.list
app.get('/posts/:postId')            -> posts.read
app.post('/posts')                   -> posts.create
app.del('/posts/:postId')            -> posts.delete
```

Users

```
app.get('/users')                    -> users.list
app.get('/users/me')                 -> users.me
app.put('/users')                    -> users.update
app.post('/users/password')          -> users.changePassword

app.post('/auth/signup')             -> users.signup
app.post('/auth/signin')             -> users.signin
app.get('/auth/signout')             -> users.signout

app.put('/users/assign-campaign/:userId') -> users.assignCampaign
```

POST

```
{
    title: {
            type: String,
            default: '',
    },
    description: {
            type: String,
            default: ''
    },
    photoURL: {
            type: String,
            trim: true,
            default: ''
    },
    created: {
            type: Date,
            default: Date.now
    },
    userRating: [{
            rating: {
                    type: Number,
                    max: 10
            },
            userId: {
                    type: Schema.ObjectId,
                    ref: 'User'
            }
    }],
    judgeRating: [{
            rating: {
                    type: Number,
                    max: 10
            },
            userId: {
                    type:Schema.ObjectId,
                    ref: 'User'
            }
    }],
    owner: {
            type: Schema.ObjectId,
            ref: 'User'
    },
    campaignObject: {
            type: Schema.ObjectId,
            ref: 'Campaign'
    }
}
```

CAMPAIGN

```
{
    title: {
            type: String,
            default: ''
    },
    rules: {
            type: String,
            default: ''
    },
    description: {
            type: String,
            default: ''
    },
    identifier: {
            type: String,
            trim: true,
            default: ''
    },
    campaignEnd: {
            type: Date,
            default: Date.now
    },
    campaignStart: {
            type: Date,
            default: Date.now
    },
    created: {
            type: Date,
            default: Date.now
    }
}
```

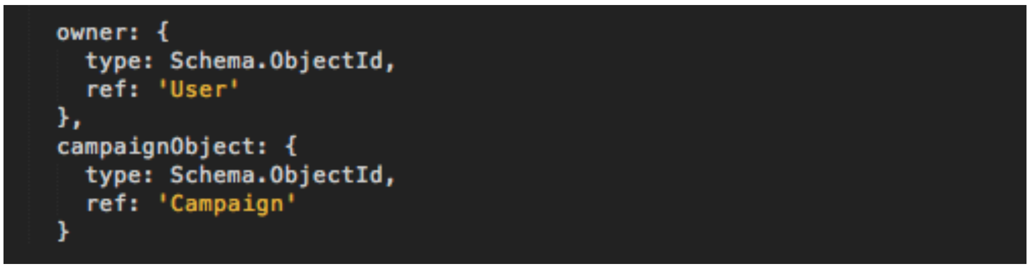
USER

```
{
    firstName: {
            type: String,
            trim: true,
            default: ''
    },
    lastName: {
            type: String,
            trim: true,
            default: ''
    },
    email: {
            type: String,
            trim: true,
            match: [/.+\@.+\..+/, 'Please fill a valid email address'],
            unique: true,
            default: ''
    },
    bio: {
            type: String,
            default: ''
    },
    photoURL: {
            type: String,
            trim: true,
            default: 'img/users/default.png'
    },
    level: {
            type: String,
            trim: true,
            default: 'admin'
    },
    password: {
            type: String,
            default: ''
    },
    salt: {
            type: String
    },
    provider: {
            type: String,
            required: 'Provider is required'
    },
    providerData: {},
    additionalProvidersData: {},
    updated: {
            type: Date,
            default: Date.now
    },
    created: {
            type: Date,
            default: Date.now
    },
    iOSToken: {
            type:String,
            default:''
    },
    campaignObject: {
            type: Schema.ObjectId,
            ref: 'Campaign'
    }
}
```

DATABASE SCHEMAS

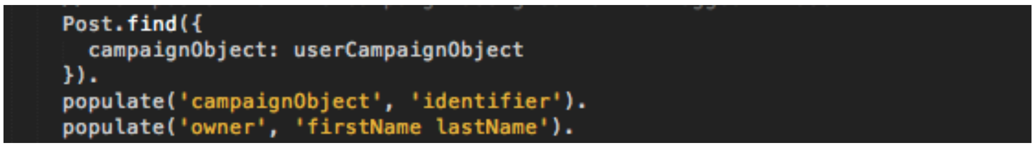
In the current stage of development schemas used are for **users, posts** and **campaigns.** Data modelling is done with the help of Mongoose. Documents contain properties of type **ObjectId**, which allow referencing from one schema to another.

For example, a **Post** document references to the two other schemas: **campaigns** and **users.**



```
owner: {
  type: Schema.ObjectId,
  ref: 'User'
},
campaignObject: {
  type: Schema.ObjectId,
  ref: 'Campaign'
}
```

Schema referencing adds a 'join-like' functionality to the documents (SQL JOIN). When retrieving data from the database the **populate** method is used to add properties that belong to **users** or **campaigns** directly on the post document:



```
Post.find({
  campaignObject: userCampaignObject
}).
populate('campaignObject', 'identifier').
populate('owner', 'firstName lastName').
```

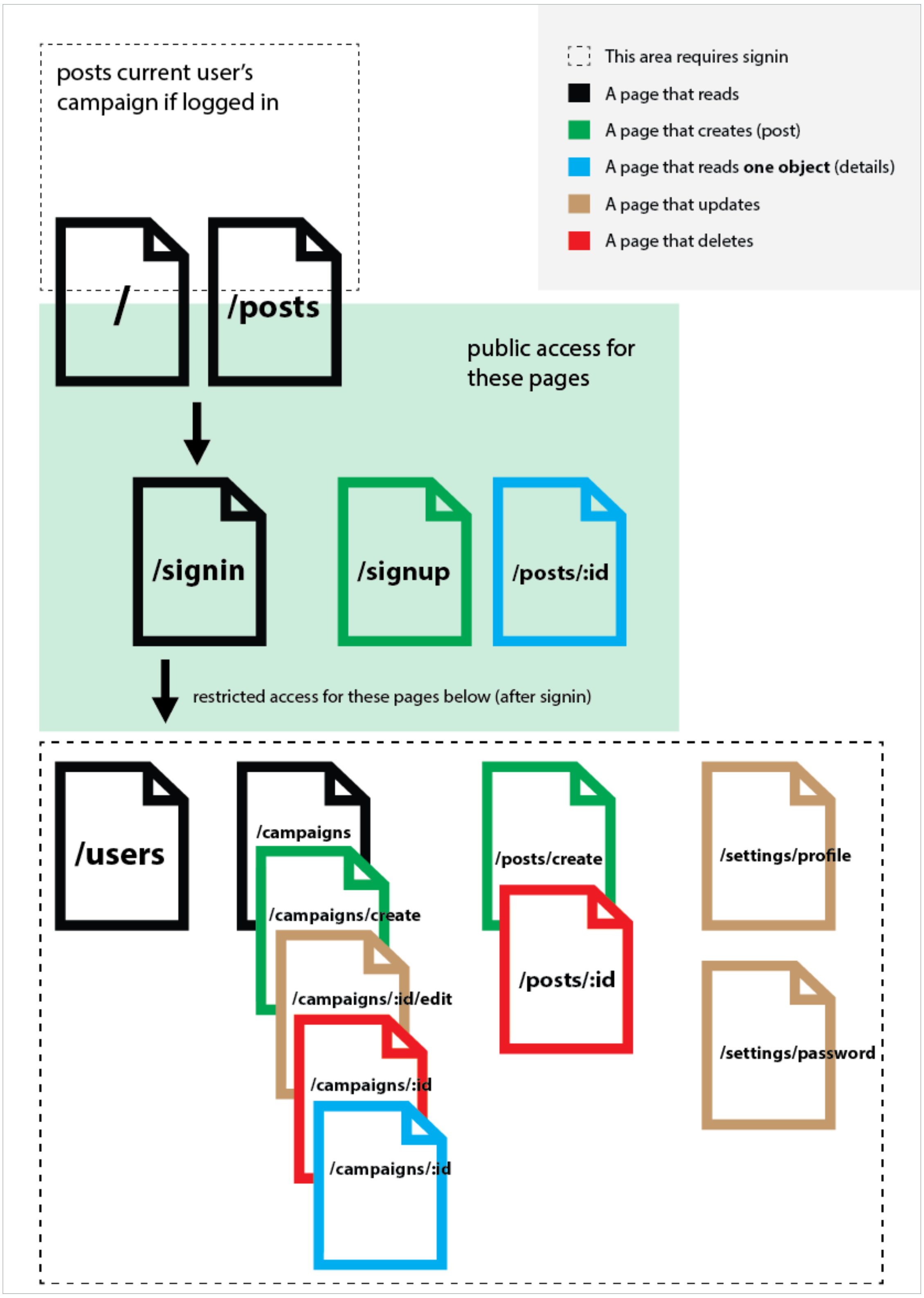
In this example, all posts belonging to the campaign represented by the variable **userCampaignObject** are populated with the **identifier** defined in the **campaign** schema. Moreover, each post also gets populated with the **firstName** and **lastName** of the owning user.

The schemas also contain validation logic and in some cases, manipulate strings (or pre-process before writing to the database). The parameter **trim** is set to true to remove white spaces from a string. In other situations, the **max** parameter is used to limit number values.

Custom validation is used together with **Passportjs,** which checks for the used strategy or in the case of users, if a password is of a certain length.

Lastly, regular expressions are used to validate the format of strings, such as the email address. The full database schema used at this stage is defined as follows:

SITE PAGES OVERVIEW



The public area of the website is made out of 4 pages:

- the homepage **/** which shows posts from all campaigns (if a user is logged in, only the posts from his campaign will be shown)

- the posts page **/posts** which is similar to the homepage only displays the posts in a list view without images and offers search functionality (also if a user is logged in, posts from his campaign will be shown only)

- the detailed view post page **/posts:id** which allows viewing a post's detailed information (if logged in extra actions will be displayed on this page, such as delete)

- the signup page **/signup** which allows users to register a new account (once a user signs up he will be automatically signed in to his account)

- the signing page **/signin** which allows users to sign in to their account if they are already registered

The private area of the website contains administration sections for users (own account), posts and campaigns:

- **campaigns** can be created, read as a list, read individually (detailed), updated and deleted

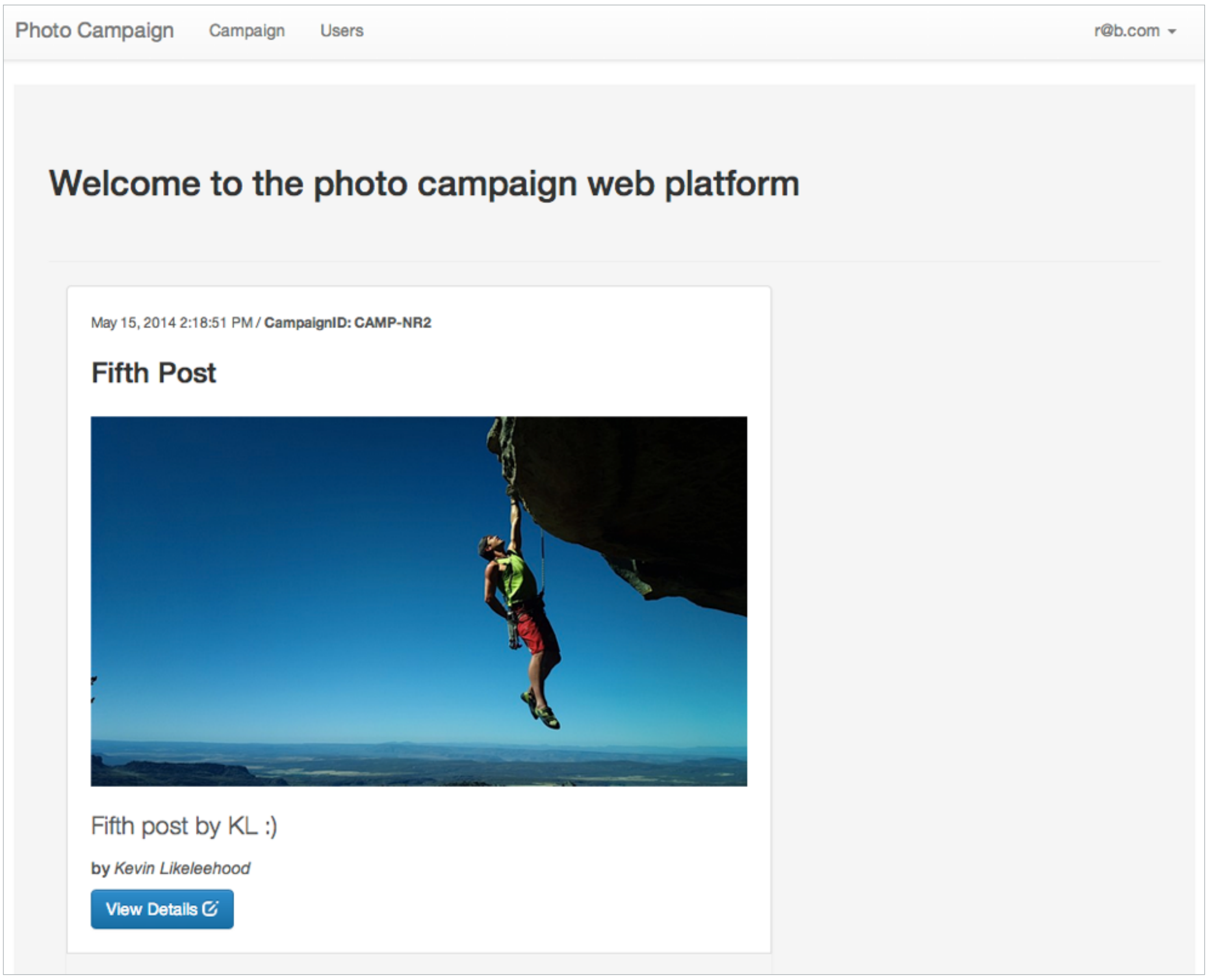
- **posts** can be created (although this functionality is reserver for the iOS app, the REST API makes it easy to implement the same functionality for the web application) or deleted.

- **users** can be viewed as a list and can access a settings module: allows editing the profile and the password.

WEB FRONT-END AND BACK-END INTERFACE

photocampaign.net

The web front-end is a hub of all posts from all campaigns for users that are not authenticated. Once a user logs in, only posts from his current campaign (or campaigns) are shown.



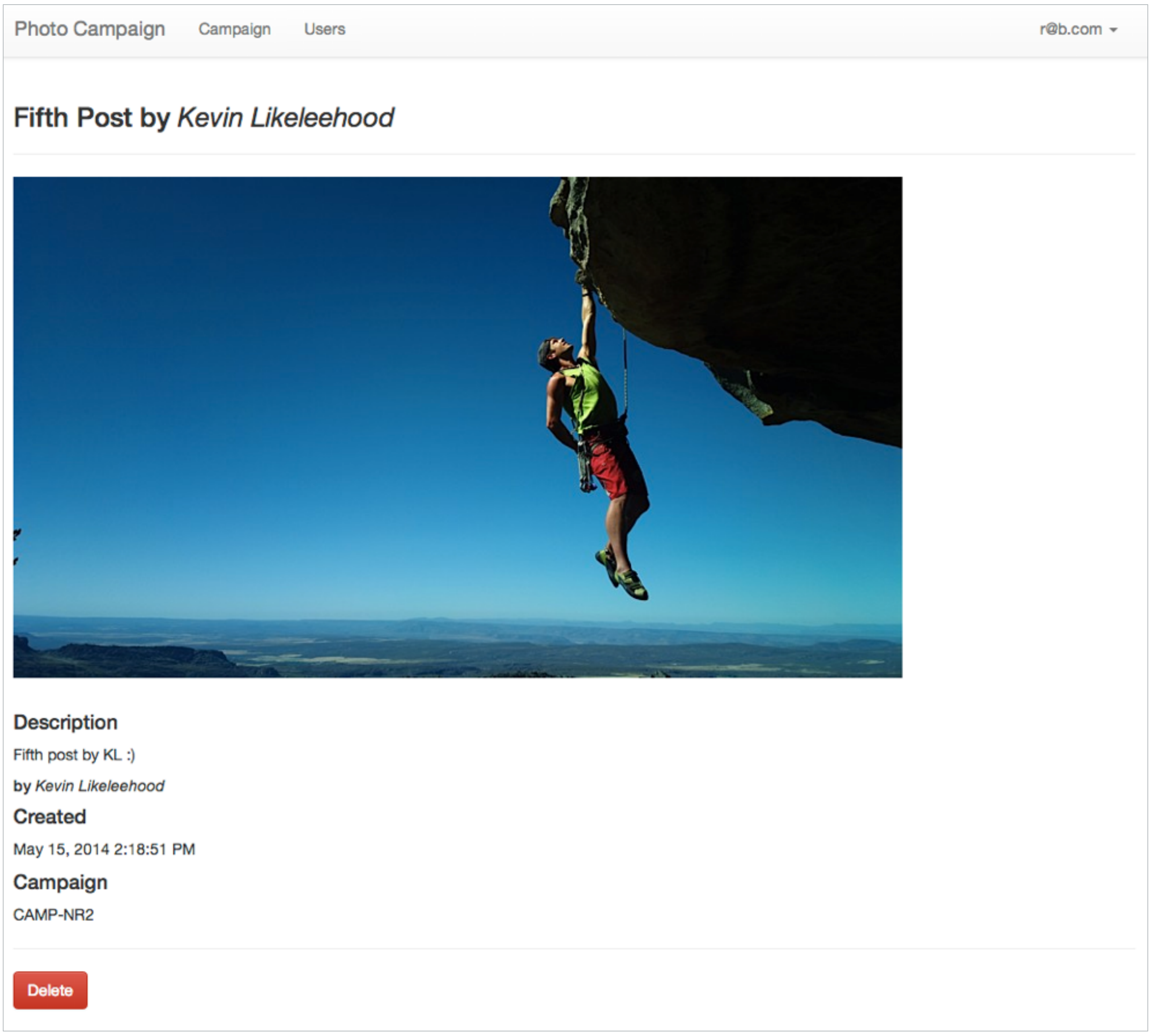
Welcome to the photo campaign web platform

May 15, 2014 2:18:51 PM / **CampaignID: CAMP-NR2**

Fifth Post

Fifth post by KL :)

by *Kevin Likeleehood*



Fifth Post by *Kevin Likeleehood*

Description

Fifth post by KL :)

by *Kevin Likeleehood*

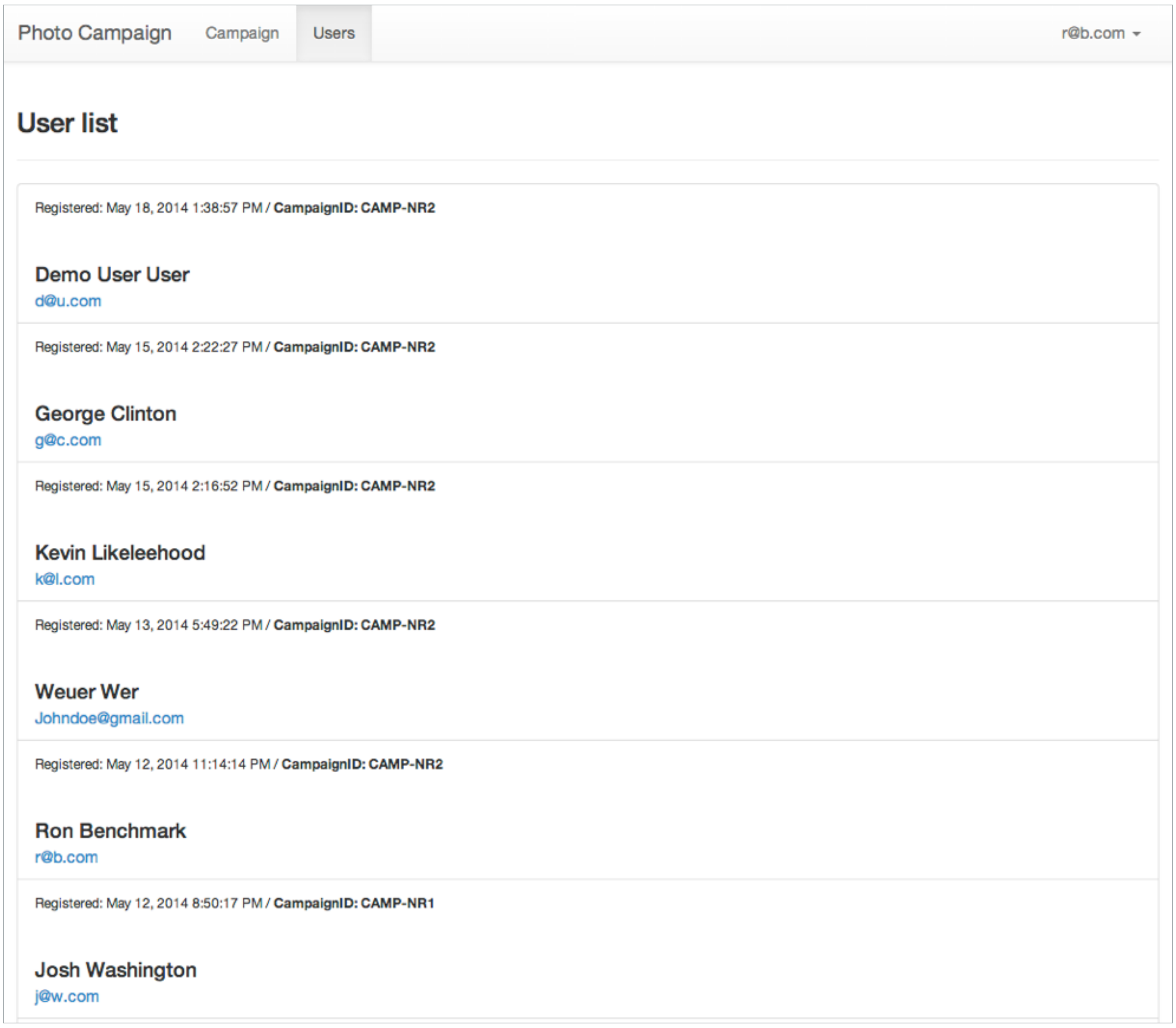
Created

May 15, 2014 2:18:51 PM

Campaign

CAMP-NR2

Each post can be viewed in detail and if authorized, in can be removed by the current logged-in user. The detailed view shows the title, description, author (name and surname), created date and campaign identifier. In later versions, detailed views will include comments as well as user and judge ratings.



User list

Registered: May 18, 2014 1:38:57 PM / **CampaignID: CAMP-NR2**

Demo User User
d@u.com

Registered: May 15, 2014 2:22:27 PM / **CampaignID: CAMP-NR2**

George Clinton
g@c.com

Registered: May 15, 2014 2:16:52 PM / **CampaignID: CAMP-NR2**

Kevin Likeleehood
k@l.com

Registered: May 13, 2014 5:49:22 PM / **CampaignID: CAMP-NR2**

Weuer Wer
Johndoe@gmail.com

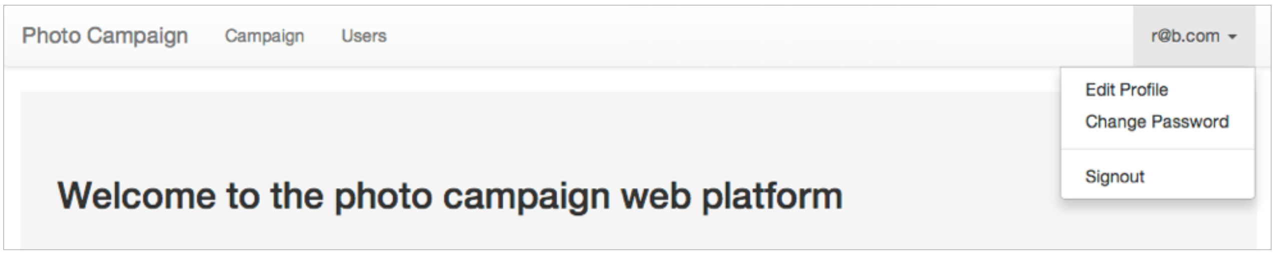
Registered: May 12, 2014 11:14:14 PM / **CampaignID: CAMP-NR2**

Ron Benchmark
r@b.com

Registered: May 12, 2014 8:50:17 PM / **CampaignID: CAMP-NR1**

Josh Washington
j@w.com

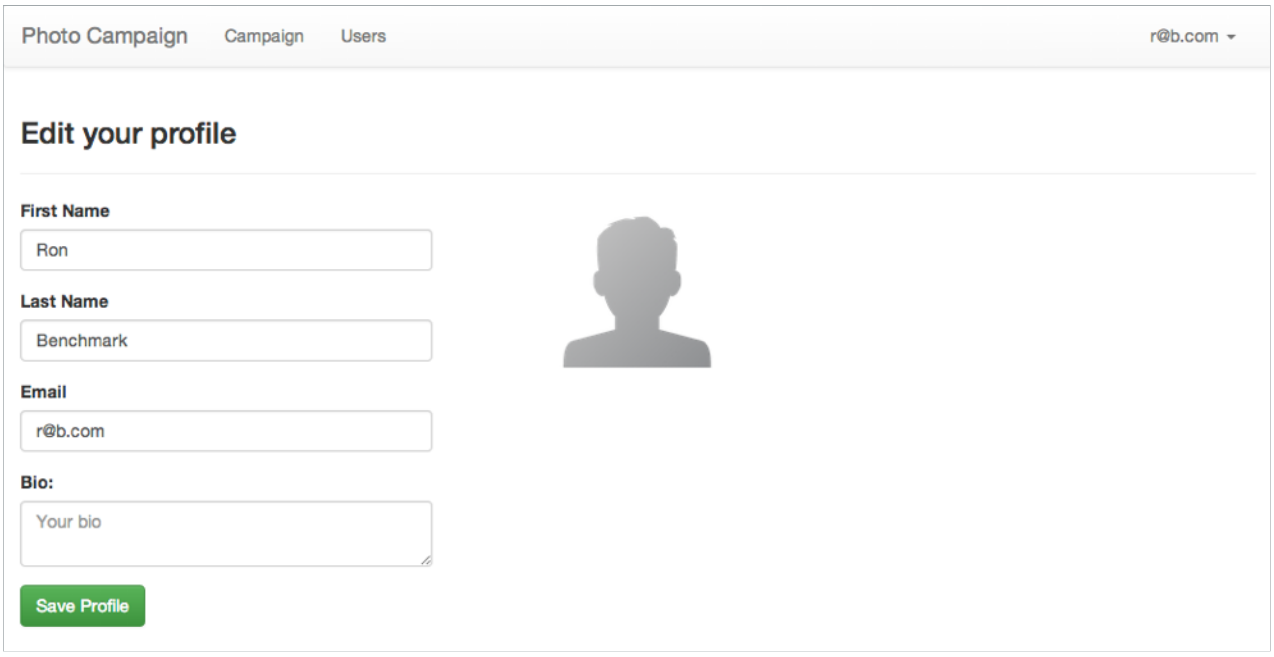
Administrators have access to a user page, which functions as an overview of the currently registered users. The REST API supports user CRUD operations, but users can only edit their own account information as of this version.



Welcome to the photo campaign web platform

Additionally to the profile page, the users have access to a password change page from the top-right menu.

User profile page example:



Edit your profile

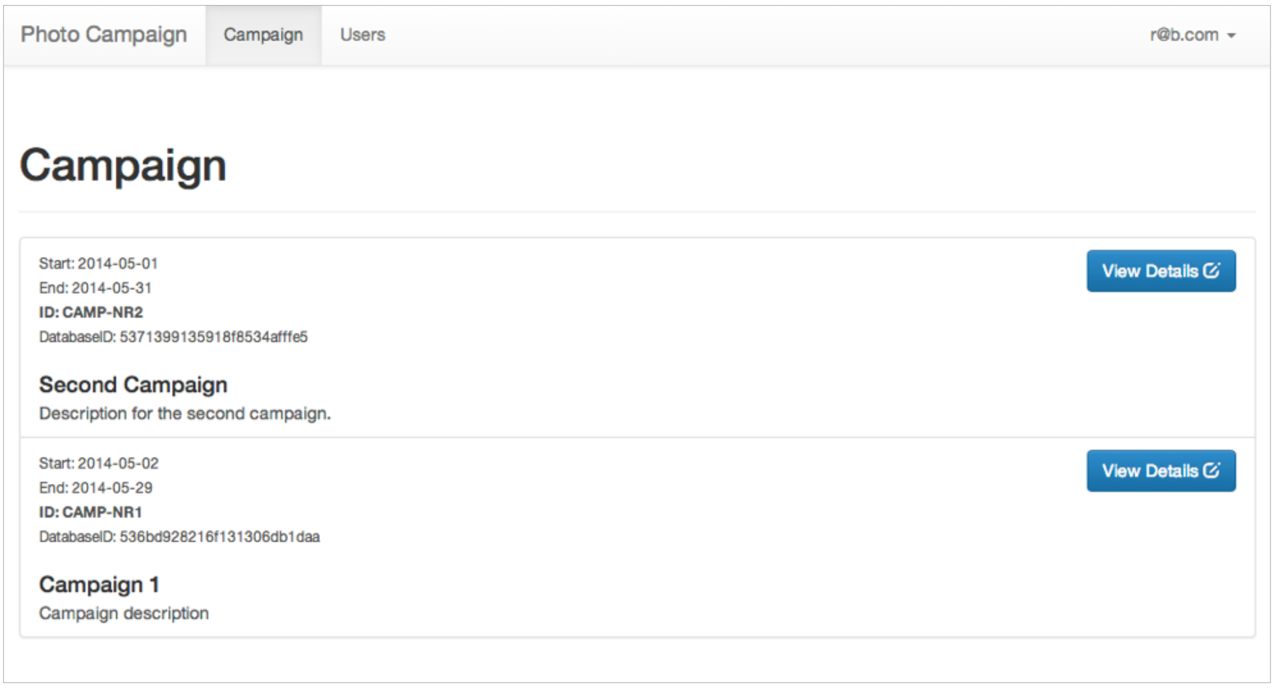
First Name: Ron

Last Name: Benchmark

Email: r@b.com

Bio: Your bio

Administrators have access to a campaign overview. Once a new campaign is created, it will appear in a list and can be edited at any time.



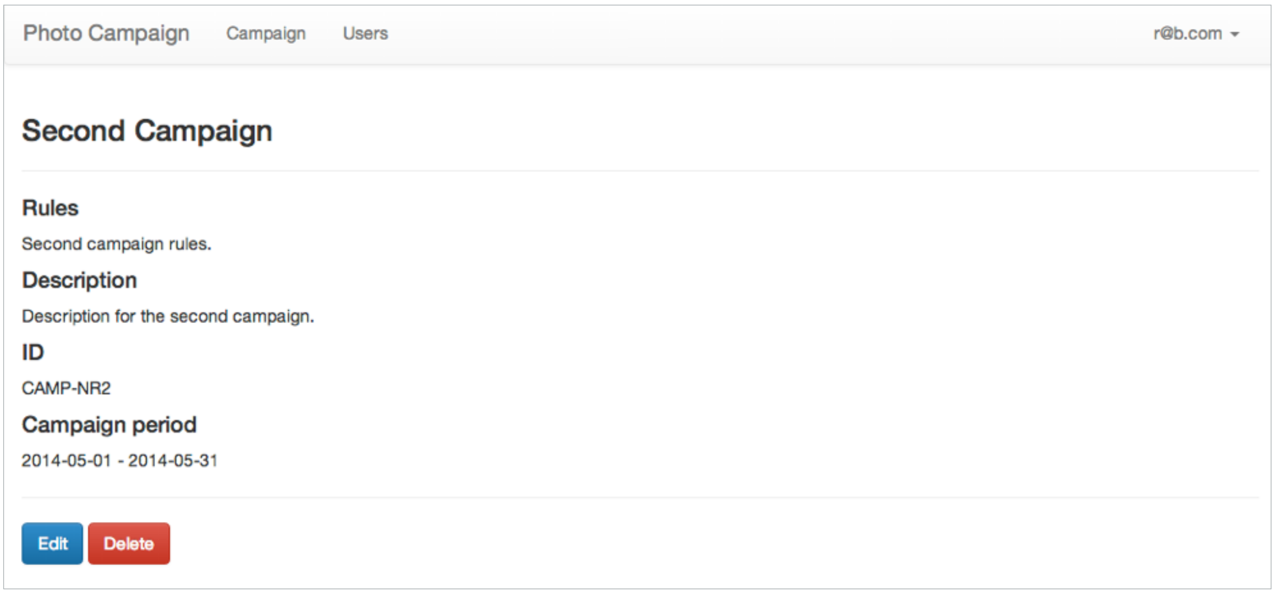
Campaign

Start: 2014-05-01
End: 2014-05-31
ID: CAMP-NR2
DatabaseID: 5371399135918f8534afffe5

Second Campaign
Description for the second campaign.

Start: 2014-05-02
End: 2014-05-29
ID: CAMP-NR1
DatabaseID: 536bd928216f131306db1daa

Campaign 1
Campaign description



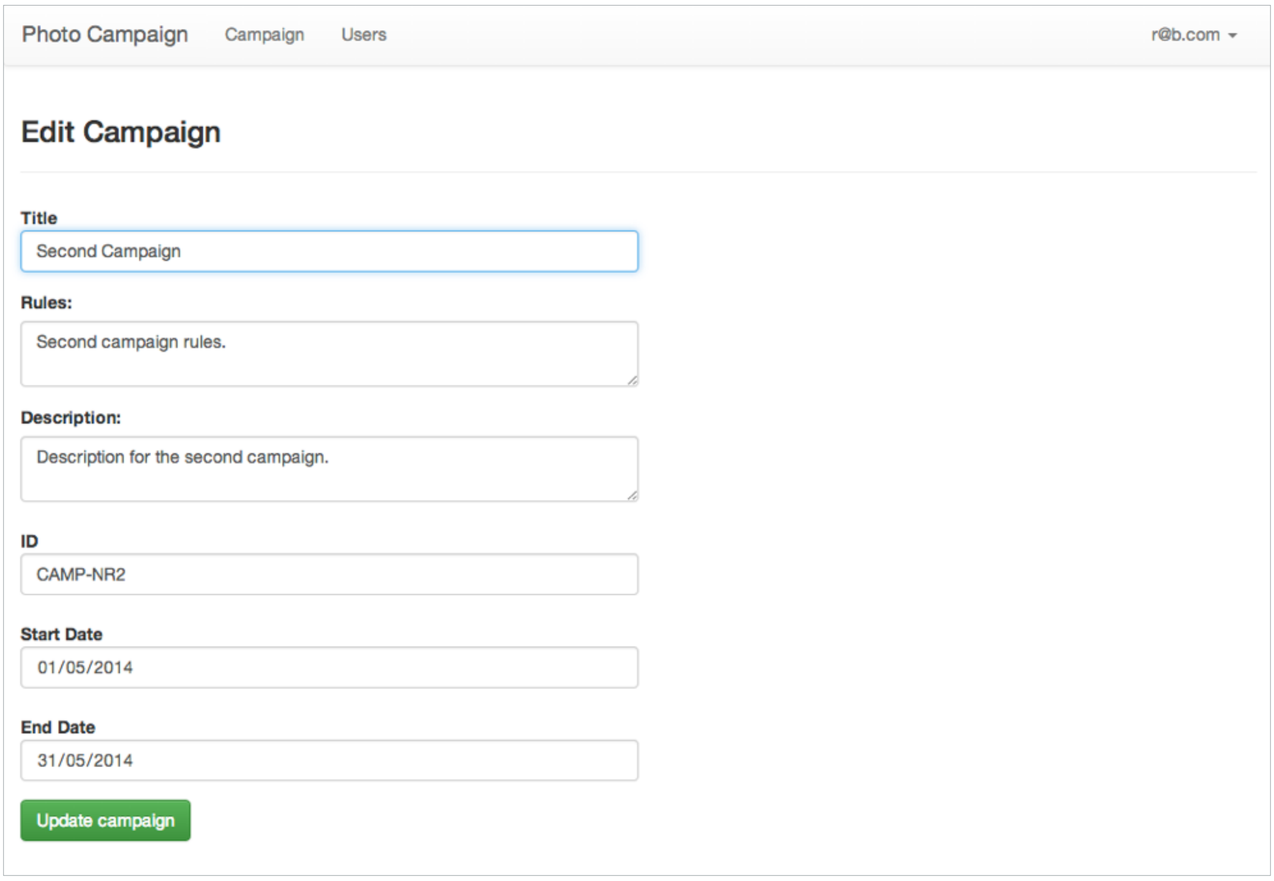
Second Campaign

Rules
Second campaign rules.

Description
Description for the second campaign.

ID
CAMP-NR2

Campaign period
2014-05-01 - 2014-05-31

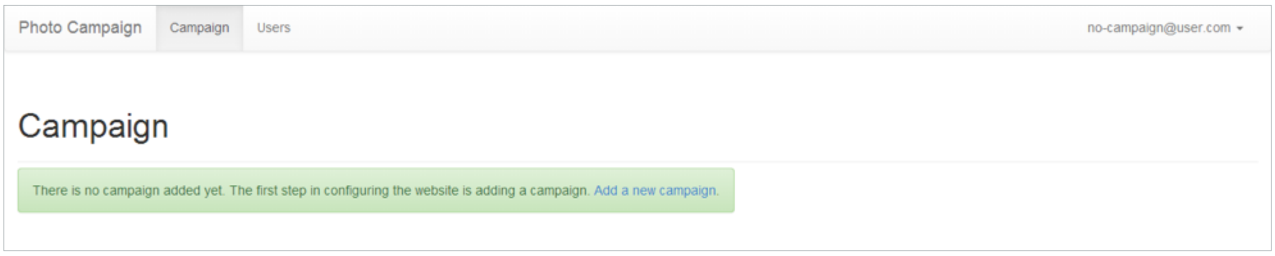


Edit Campaign

Title	Second Campaign
Rules:	Second campaign rules.
Description:	Description for the second campaign.
ID	CAMP-NR2
Start Date	01/05/2014
End Date	31/05/2014

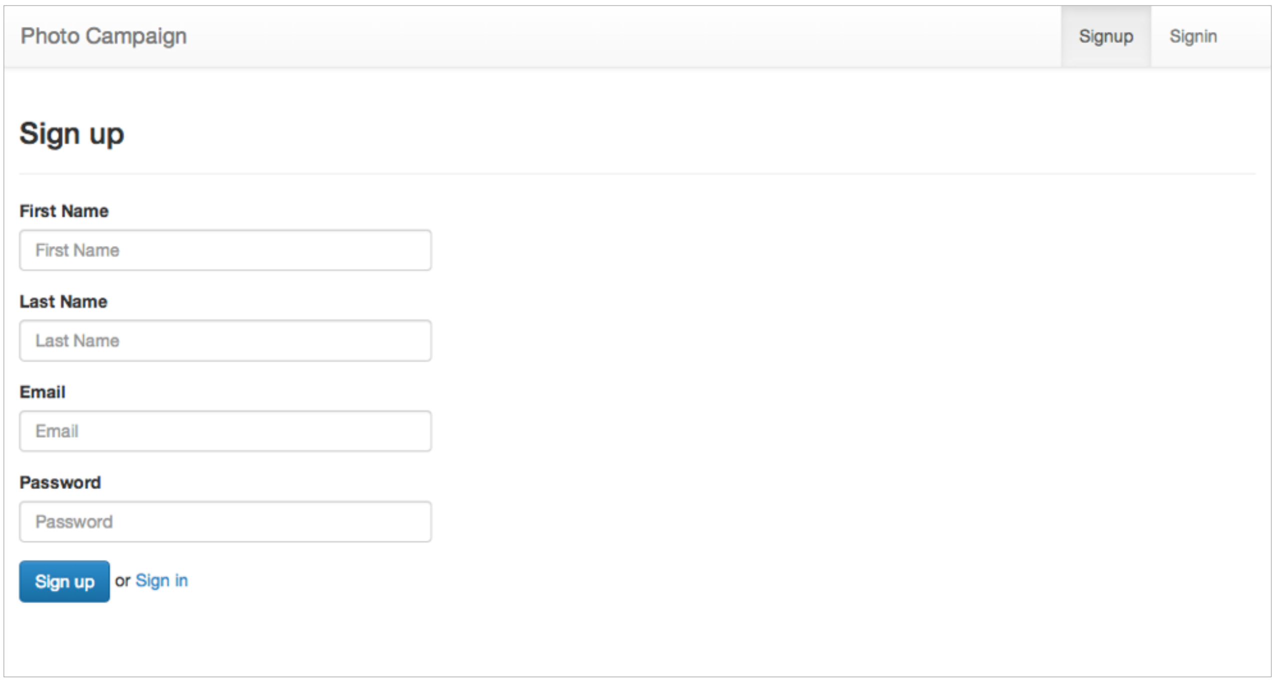
Deleting a campaign can also be done when viewing it's details.

If no campaign is active, users will be registered without belonging to one and can be assigned to a campaign later on. This implemented functionality comes in useful when a campaign expires and it’s users can be re-allocated to another campaign. By doing this, users don’t have to register again.



Campaign

There is no campaign added yet. The first step in configuring the website is adding a campaign. Add a new campaign.



Sign up

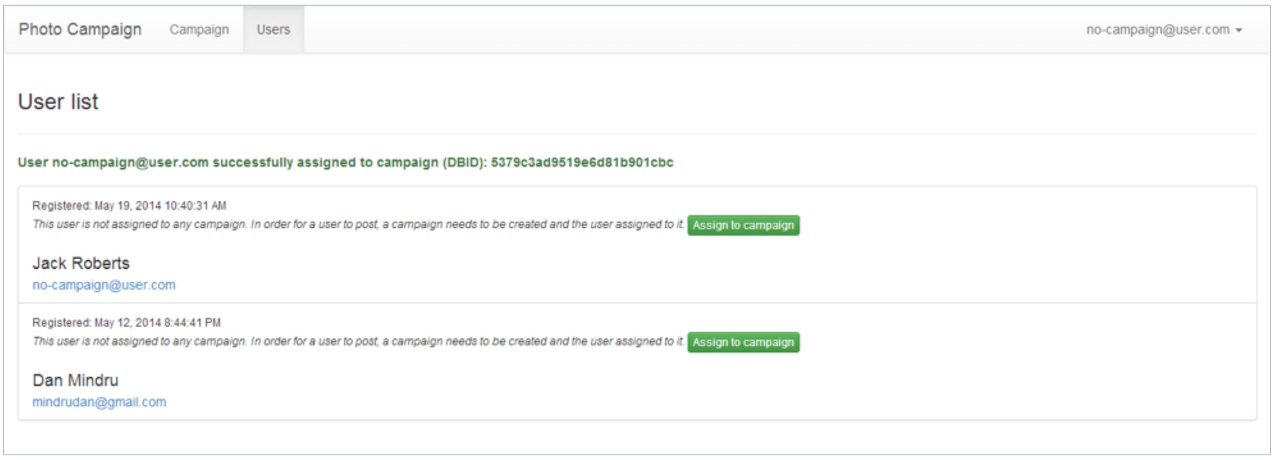
First Name

Last Name

Email

Password

Sign up or Sign in



User list

User no-campaign@user.com successfully assigned to campaign (DBID): 5379c3ad9519e6d81b901cbc

Registered: May 19, 2014 10:40:31 AM
This user is not assigned to any campaign. In order for a user to post, a campaign needs to be created and the user assigned to it. Assign to campaign

Jack Roberts
no-campaign@user.com

Registered: May 12, 2014 8:44:41 PM
This user is not assigned to any campaign. In order for a user to post, a campaign needs to be created and the user assigned to it. Assign to campaign

Dan Mindru
mindrudan@gmail.com

The entire application is built on Twitter Bootstrap's grid system. This results in a responsive user interface that works great on mobile devices. The responsive interface becomes very useful when it comes to signing-up on the platform, an action that is currently supported only on the web (not in the iOS app).

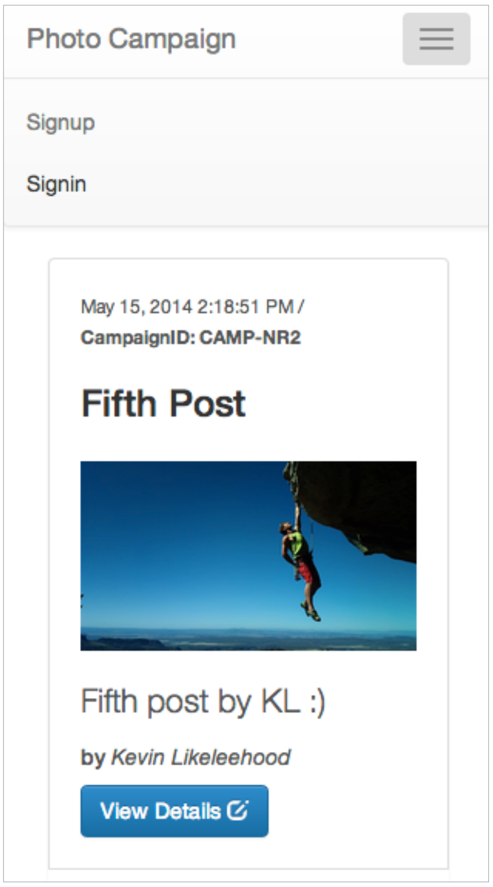


Photo Campaign

Signup

Signin

May 15, 2014 2:18:51 PM / CampaignID: CAMP-NR2

Fifth Post

Fifth post by KL :)

by *Kevin Likeleehood*

View Details

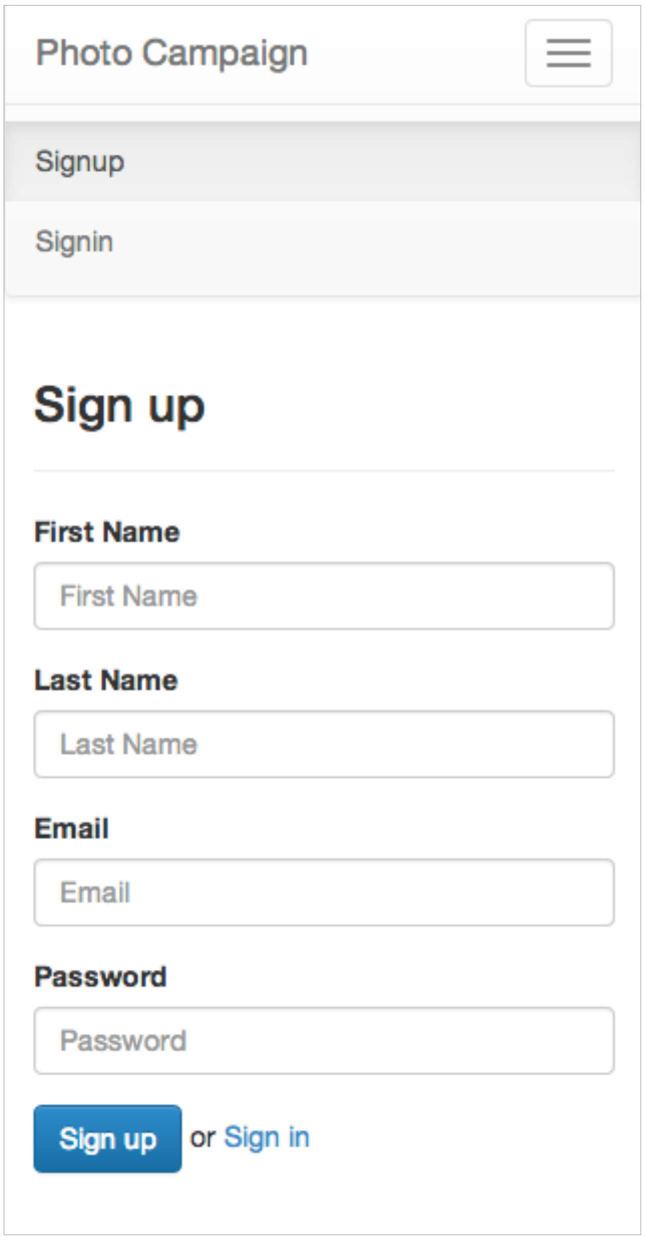


Photo Campaign

Signup

Signin

Sign up

First Name: First Name

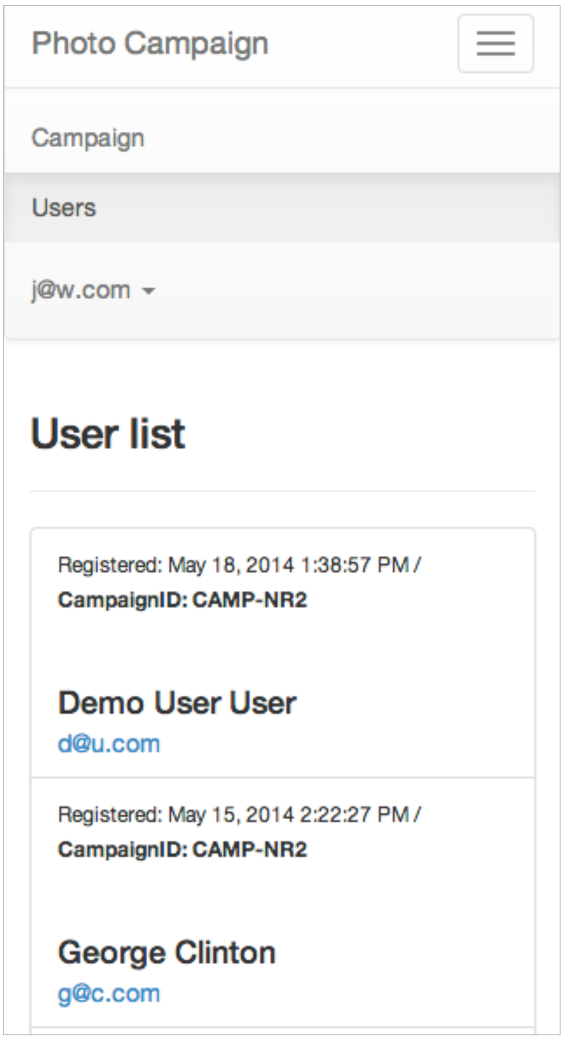
Last Name: Last Name

Email: Email

Password: Password

Sign up or Sign in

Other mobile views presented below: (user list left, editing a campaign on the right)



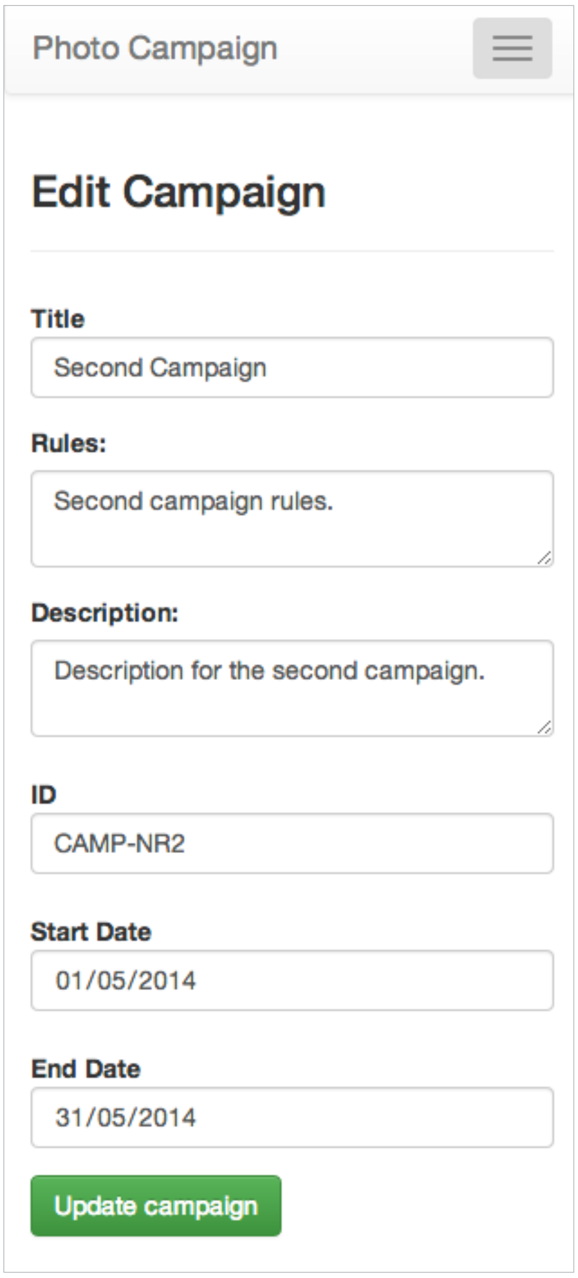
User list

Registered: May 18, 2014 1:38:57 PM / **CampaignID: CAMP-NR2**

Demo User User
d@u.com

Registered: May 15, 2014 2:22:27 PM / **CampaignID: CAMP-NR2**

George Clinton
g@c.com



Edit Campaign

Title
Second Campaign

Rules:
Second campaign rules.

Description:
Description for the second campaign.

ID
CAMP-NR2

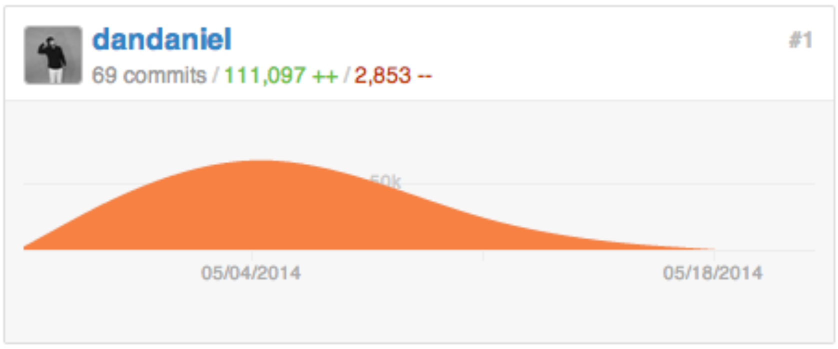
Start Date
01/05/2014

End Date
31/05/2014

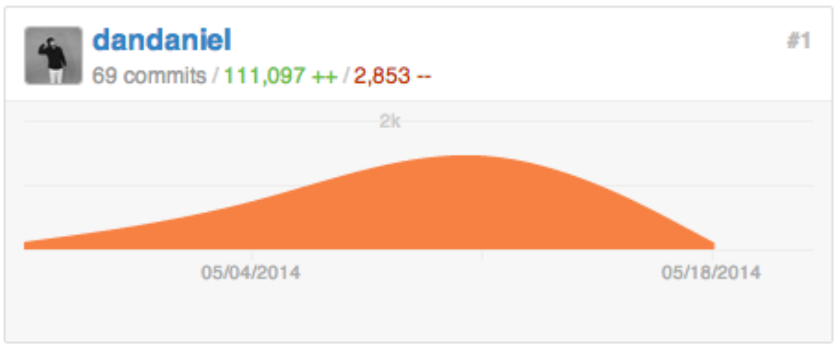
VERSION CONTROL

Github has been used for the development of the entire platform (both iOS and web). Commits were always made with a working application (never commits with broken functionality). In the initial stages of development commits where made for larger pieces of functionality. Later in development, commits where made for smaller changes and finally once the application was deployed on the web server, small commits where made for debugging and bug fixing. Commits always contained descriptive messages of what changed in order for bugs to be tracked easily.

The repository contained one branch and had a total of 76 commits including merges. It can be seen that more additions where made on the beginning of the project:



More deletions where made to the end of the project:



CONCLUSION

Concluding this report, the demo video is available with a more detailed view of the platform:

youtube.com/watch?v=xlu2rq1LZaY