Web Technologies: Report

Jannick Hemelhof, Roberto Ristuccia, Youssef Boudiba

December 21, 2015

Contents

1 Introduction

Intro with some info about the project: basic idea, how we handled being in a group, etc.

2 Design

Design of our web app, maybe show a diagram or two

3 Handling of Requirements

3.1 User can register/log in

3.2 User has a profile

3.3 User can manipulate data

Look at/search/add and edit

3.4 Social aspect is needed

3.5 Usage of AJAX

3.6 Form validity

While thinking about handling this requirement our initial thoughts saw form validation as something that would happen on the server side of our application. Rereading the assignment gave some other insights and discovering the very neat validator.js[1] library showed us that client side validation could look good and was straightforward to implement.

- Server side: We have built in some checks that ensure a consistent status of our database: When registering a username needs to be unique and the required fields need to be filled out. The check for required fields is also present on the client side but since someone can manipulate requests to maybe remove some data after the client side check an insurance policy was needed on the server side. These checks for required fields are present all over the server side where data is being received.
- Client side: We added some basic checks: Required fields need to be filled out and, by using the HTML5 input fields, designated input fields need to have a value that corresponds with the type of field. When there is an error (e.g. a required field wasn't filled in) the field is marked and an error is shown explaining what the problem is. This is possible thank to the validator.js library.

3.7 CSS

3.8 HTML5 features

For this requirement we went ahead and implemented following HTML5 specific features:

- Local Storage: In order to remember the logged in user across different sessions we're using the local storage feature to place a token. That way we can check if there's a user logged in at the moment.
- Geolocation: A user who has just seen a UFO out in the field might be too shocked to remember where he is at the moment. By using the HTML Geolocation API we can just use the location of the device on use that data when submitting the sighting.

[1] Validator, for Bootstrap 3: `http://1000hz.github.io/bootstrap-validator/`

- Notification: Showing the user that an operation was successful would be possible by letting a custom form/modal pop up but we were aiming for a more native solution. By using the Notifications API (supported by Chrome, Firefox, and Safari) we can offer the user a confirmation that his request was handled in a native environment.
- Input fields/Validators: In our input forms we use some new HTML5 input types e.g. email but also the required attribute.

3.9 Web Services

Our initial idea was to provide weather data for each sighting. Sadly, we realised that weather data is mostly aimed at forecasting and not at historical information for a specific date. We then decided to Imgur API[2] since our goal from the start was the ability for the user to upload an image when posting a sighting. Hosting images ourselves isn't really scalable so using this external web service helps us a lot. Usage of the Imgur API happens on two occasions:

- Home page: When a user lands on our homepage, we sent a GET request to the API in order to receive 50 images taken from the UFO gallery. That way we can show the user something interesting but also relevant. That way we might have even room for sponsored images.
- New Sighting: When posting a new sighting a user should be able to provide an image of what he/she has just seen. As said before, we're not interested in hosting these images ourselves so that's where the angular-imgur-upload[3] library comes in. After the user has selected an image to upload, we use this library to sent it to Imgur. We then save the url to the image in our sighting, a url we get from the response of the Imgur API.

3.10 Provide data

Providing our data so others can use it is a straightforward and easily accomplished task. We went further with that idea and provide a simple API so developers can develop their own applications for managing UFO sightings using our infrastructure. We support following commands:

Idea	Type	Path	Auth header	Parameters	Return value
Register a user	POST	/signup	no	• username • password • email	• auth token
Login a user	POST	/login	no	• username • password	• auth token

[2]The Imgur API: `http://api.imgur.com`
[3]Angular.js Imgur Upload service: `https://github.com/purple-circle/angular-imgur-upload`

Get user info	GET	/user	no	• username	• username • email • firstname • lastname
Update user info	PUT	/user	yes	• username • email • firstname • lastname	
Get sightings	GET	/sightings	no		• sightings
Add sighting	POST	/sightings	yes	• title • description • day • month • year • url • coordinate • author	
Post comment	POST	/comment	yes	• content • author • sightingID	
Get sighting	GET	/sighting	no	• sightingID	• sighting

Update sighting	PUT	/user/sightings	yes	• sightingID • title • description • date	
Delete sighting	DELETE	/user/sightings	yes	• sightingID	

3.11 Responsive design

3.12 Google Maps

We used the Google Maps API on two occasions:

- New sighting: When a user is reporting a new sighting he should be able to tell where he saw it. He has three options for that:
  - Coordinates: He can input longitude + latitude coordinates for the location.
  - Address: He can input a normal address e.g. Pleinlaan 9, Etterbeek. When previewing the location this address is matched using the Google Maps API and coordinates + address are automatically filled in with the matched location.
  - Device location: Thanks to HTML5 Geolocation API the user can use the location of his device to automatically fill in the coordinates.

  Previewing the location will show a Google Map with the requested location marked.
- Sighting page: When a user is on the detailed sighting page the location where the sighting was seen is shown on a Google Map (location is marked).

4 Conclusion