HikingWorld



Nikolaj Ørberg Hansen, Matti Andreas Nielsen & Victor-Emil Rossil Andersen

Software Development – System Integration – 1st semester spring 2016

31/03-2016

Indhold

[Brief Introduction 3](#_Toc447227596)

[Technologies 4](#_Toc447227597)

[Application Programming Interface (API) 5](#_Toc447227598)

[Server specifications 7](#_Toc447227599)

# Brief Introduction

The goal of the application is to offer a variety of hiking to customers. These hiking trips will be composed like a charter trip, where the trip is arranged in advance. Every trip will be accompanied by a hiking guide and every trip will be rated in terms of difficulty. It is not possible to mix up different hiking trips, since they will be predefined. All trips will have a fixed price. The trips on the website will be visualized by presenting the user with a route map of the specific trips.

We also want to involve the users on the website. It will be possible for users to rate trips, recommend future trips and so on. The specifics are still yet to be decided. We might want to add a real time chat that will connect a logged in user instantly to an administrator if the administrator is available. In the real world this would not be a 24/7 service, but rather a service provided during the day time on certain days.

Users will usually just order one trip at a time, but our shopping cart will function as a regular web shop shopping cart where it is possible to add multiple trips. During the summer vacation some people might want to go for different trips during the summer. However, we want to add some restrictions. If a user tries to add two trips to the shopping cart, which lies within the same interval the order will be rejected.

# Technologies

Our main programming language throughout this project will be Node.js. Node.js is a server side language for developing web applications and is built on top of the Javascript core with added functionality. Node.js comes with a premade http server inside its standard library. This is the one we will be using for handling http requests, which will be the main protocol for our API. We will go in depth with our API in the next section.

We have chosen to implement the RESTful architecture in our application. To us the RESTful architecture provides us with a lot of freedom in how we will develop this application. At the moment we have a plan that we will be using JSON as the primary format of data. That is also a reason why we have chosen REST. If we had chosen SOAP we would be required to use XML as the format for all the data. The only way to bypass this would have been to wrap JSON inside of XML. This would require the network to transport heavy chunks of data around. By using REST, we will be granted the freedom to design and develop the application in a way that the client can decide its own content type for the response data. If we implement this functionality a developer using the API will be able to choose which format the data should be in.   
  
In addition to our own API we will also be using the Google Maps API for showing the predefined routes that the website will offer to the costumers. The Google Maps API is also accessible as a public REST service. The Google maps API is well documented and pretty easy to use. If we were to develop this functionality ourselves, we would have to drop out of school. We will also be using a weather API, but we have not decided on which one to use yet. The data from this API will show the weather prognoses of the different routes.

Another web integration we will be using is the continuous integration build cycle of Jenkins CI. This is not a tool that will provide functionality to our application, but rather a tool that will strengthen our development cycle. Jenkins CI is a tool that can automate a lot of tasks for us. In our situation we do not want to run all unit tests and integration tests locally on our computers. This will take time and pause the current development. Instead we will deploy Jenkins CI on our server which will run tests and build our project remotely every time we push to out git repository. It will then give a status report of the remote build and inform us of any errors. Jenkins CI will be called via a webhook which is basically a http request that will trigger whenever Github detect that a push has been made.

# Application Programming Interface (API)

Our node.js http server listens for requests, when it receives a request it creates a request and response pair of the corresponding node.js classes http.ClientRequest and http.ServerResponse.   
The request contains all of the information that the client provided us, like the URL, http method, form data and headers.  
While the response object contains all of the methods we need to give back an answer to the client. The way our http server works right now is that   
we have 3 main endpoints consisting of route, API and a default. The route gives back a redirect 301 http response to the request route, the API calls our restful API which we will be discussing further in a second and the third endpoint assumes that if none of the other endpoints was called its either a static file or an invalid endpoint. The following section will discuss the API endpoint exclusively.  
  
When a request's URL contains the string "/api/" the request and response objects are forwarded to the API controller, this controller then further splits up the URL to see which resource its looking for, if it finds a valid resource it will forward the request and response object to that resources controller which will process the request and write a response back to the client. The processing of a request will often involve a database call and wrapping the response data correctly.  
  
Now we will discuss the current status of our RESTful API. Take in consideration that this API is not finished at all and still needs A LOT of work.  
  
**The restful convention we follow is:**  
GET 200 /api/resource/:id  
POST 201 /api/resource  
put 200 /api/resource/:id  
DELETE 200 /api/resource/:id  
  
**Our current server endpoints:**  
GET 200 /api/route/:id  
GET 200 /api/route  
  
GET 200 /api/review  
  
POST 200 api/user/login  
POST 201 api/user/register  
POST 201 api/review  
POST 201 api/user/:id/buytrip  
  
PUT 200 api/user/:id  
PUT 200 api/review/:id  
  
DELETE 200 api/user/:id  
DELETE 200 api/review/:id  
DELETE 200 api/routes/:id  
  
**Client requests:**   
GET Google translate: [Https://www.googleapis.com/language/translate/v2?q=some\_text?target=](Https://www.googleapis.com/language/translate/v2?q=some_text%3Ftarget%3D)"dk"?key=some\_private\_key  
GET Weather API request  
GET Google map API  
GET/POST/PUT/DELETE [https://www.mattinielsen.com/api/\*\*/](https://www.mattinielsen.com/api/**/)\*\*  
  
**github webhook:**  
PUSH [https://jenkins.it-kartellet.dk](https://jenkins.it-kartellet.dk/) forwarded internally to --> 10.10.10.15:8081 where our production server is hosting jenkins.

# Server specifications

The application will be hosted on a Linux server owned by our group member Matti. The server only allows connections through SSH with a public/private key combination. Regular username/password connectivity has been deactivated on the server for better security. Every person in the group will store a private key locally. The server will store all the public keys, which allows for authentication when connecting remotely. The Linux server is configured to only allow access on port 8081 for Jenkins CI, port 23 for SSH, port 22 for SFTP and port 3000 for node.js. This will disallow remote clients to access other applications than the ones that we want to make publically available. Jenkins will be public for remote access, but will require credentials. This way, unauthorized users will not be able to access Jenkins.