BALTIC JAGT

4th semester

Melissa Ma
Adi Tanase

ERHVERVS AKADEMI SYDVEST

Table of Contents

Abstract

This report is for the fourth semester exam in Computer Science. Here we explain the process that stands behind developing a web site. The purpose of the project is to proof methodologies and technologies used within this entire semester in the JavaScript course.

Introduction

Problem statement

Project background

Baltic Jagt is a project in development, with aspirations to promote cheap hunting opportunities within the Baltic lands. Their main target is the Scandinavian market.

Project definition

The future company has done the agenda well while being in progress. Meaning that connections has been establish within the Baltic lands. By pulling the hunting cost information from the both markets, and have them in balance, it encourage the owner to bring the small firm alive.

But having knowledge and product in stock will not lead to a healthy company so a way to promote these will need to get on stage. That's where the developers, us, we can help by creating a modern website which will shorten the distance between customers' need.

Business benefits

Managing the data in a digital manner will increase the management performance.

Better visualization off the products will lead to customer loyalty.

A static location (office) won't be need from beginning, minimum costs to start-up

Vision Statement

Making the "Baltic Jagt" available on the web will open new sales possibilities and will prepare the Shop for incoming changes on the market. Scandinavians likes to travel as well as doing their hobbies. So we see these as an opportunity to promote the phrase: "Travel and enjoy your hunting for less".

List of features

1. CRUD functionalities for the database
2. Administration features
3. Visualizing the available items
4. Ability to keep the products in a Cart
5. Order products to hunt or package hunt (products plus accommodation)
8. Shop offer available for customer 24/24
9. Maintain sales order history to keep analysis of sales in order to display the most sold items
10. Maintain adequate database

Technologies

MEAN Stack

There are so many frameworks for building a web application. In our opinion, a good decision is to choose a rapid and scalable framework. So here we useing MEAN, it's full stack JavaScript, that ca run on any platform that has browser. MEAN is defined as being a group of MongoDB,Express,AngularJS and NodeJS. We're using JSON format to store data into MongoDB, then invoke Json query , after this , return the result as JSON to Angular and show to the client.

MongoDB

MongoDB is a database which is use JSON to store data.It's a NoSQL(non-relational) database. So we can save a lot of time from analyzing relationships among tables.

Express

Express is a simple nodejs web application framework, providing a set of component and module to help us building single and multi-pages.

Angular

AngularJS is a framework for frontend development. It make's HTML from static web page to become a dynamic web page. In the AngularJs program, it's easy to bind data with the following: {{}}.

Node.js

This is the platform we are using to build our website. It has proven to be fast and efficient during this entire project.

Version control

GitHub

GitHub is a free (public repositories) open source repository platform. In this environment are many programmers sharing code. It's a convenient and rapid way for us to track changes to source code.

We didn’t go wild regarding branching, we kept it simple and pushing only to the origin master branch.

Deployment

Openshift

Is hosting free our application in the public cloud, it has support for JavaScript app. After we create our application locally we uploaded on openshift, we add as well the MongoDb module. At the beginning we faced some issue regarding deployment on openshift. The way we worked around this was to get our git repositpoy first, copy the app into it, and from the same location do the following command:

```
yo angular-fullstack:openshift
```

And from the same location we commit and push the chances as followings:

- grunt build

  ```
  grunt buildcontrol:openshift
  ```

-

Architecture

At this chapter we will explain the construction of our project. The basic structure it has been created by using the Yeoman feature to generate a full-stack application. In our case we decided to use this generator provided by 'DaftMonk' from the internet. The pictures below illustrating the two main steps we approach in order to get our startup application.

- Doing the install

```
npm install -g generator-angular-fullstack
```

- Running

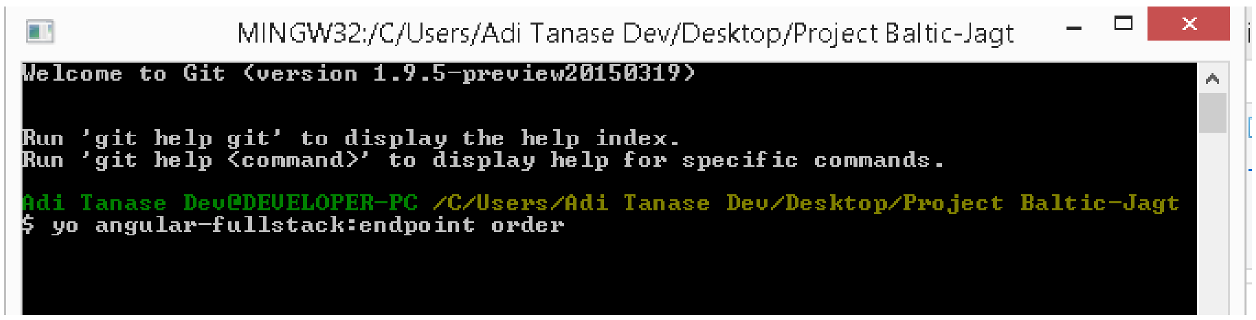
```
yo angular-fullstack [app-name]
```

Of course that, in order to get this far we had to do some prerequisite installation like the technologies we described at the previous chapter (see 'Technologies' page 3)and the yeoman.

We got some errors when we want to run it locally, basically the server will not start. So what we did it was to delete the node modules folder and the bower components. Reinstall them again and voila everything was just up and running.

Server side

At this level we are fetching data and served at the client side. The are many cool thing inside this side, but we will synthetize, and only talk about few of them. First we began by generating end points. Starting simple by issuing the following command:



MINGW32:/C/Users/Adi Tanase Dev/Desktop/Project Baltic-Jagt

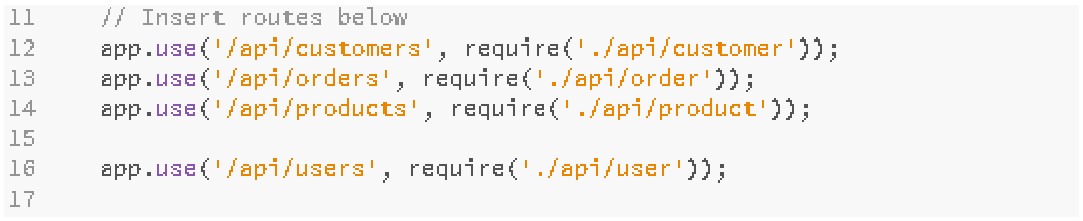
```
Welcome to Git (version 1.9.5-preview20150319)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

Adi Tanase Dev@DEVELOPER-PC /C/Users/Adi Tanase Dev/Desktop/Project Baltic-Jagt
$ yo angular-fullstack:endpoint order
```

That will create the order endpoint in the api folder with the necessary classes. We will pick one of them, the best candidate we think it has a certain amount of influence is the ‘order.model.js ‘. At this level we find the order Schema, this type of construction is introduced by Mongoose. What it does, is to map to a MongoDB collection and it will define a structure for the documents within our collection.

Another class which definitely is worth mentioning is ‘route.js’. The figure below is a piece of this class, here we have the routes for our endpoints. Data will be retrieved at client side with the identic urls.



```
    // Insert routes below
    app.use('/api/customers', require('./api/customer'));
    app.use('/api/orders', require('./api/order'));
    app.use('/api/products', require('./api/product'));

    app.use('/api/users', require('./api/user'));

```

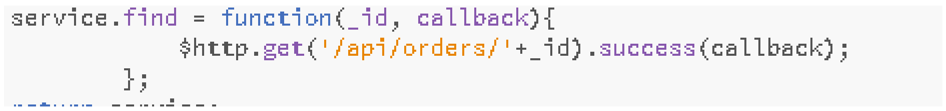
Client side

One of the great things regarding this side is lose coupling from the server side, a great advantage if another server is added. This kind of decoupling allows us to develop back-end and front-end independently from each other. Coming back to our client, angular prove to be a really big gain, fast and intuitive.

We used again the power of generator to create routes as following:

- Order
- Customer
- Store
- Shopping cart

If we take the Order route for example this will contain controller, view, css file, and a routing class. From this initial point we will insert our code until we will get our desired product. The same route it has as well a service with purpose to get the json data from the server. The figure is illustrating a part of the OrderService class.



```
service.find = function(_id, callback){
            $http.get('/api/orders/'+_id).success(callback);
        };
```

Using the generator it's reduced the amount of time for developing, instead of creating manual structure for our routes.

Implementation

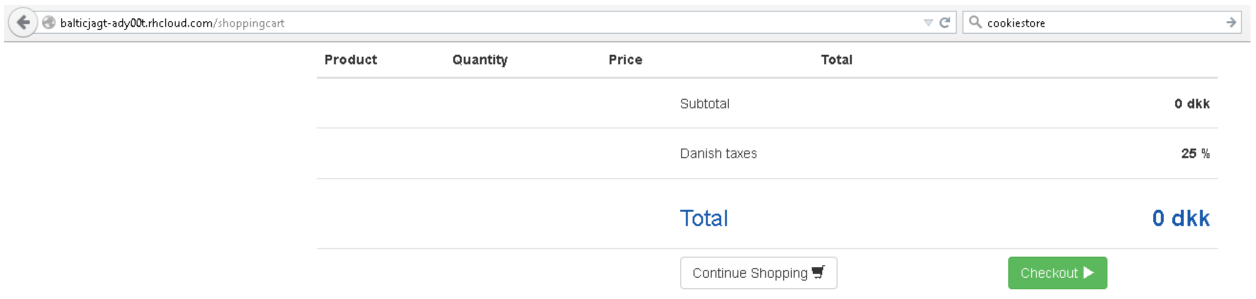
In this part we will select some code from our project to highlight what we thought it has been most challenging. Some of the issues we encountered and the current stage of Baltic Jagt website will be summarize as well.

Shopping Cart

The implemented cart in this project is basically a list that contains items selected by the user while he is shopping. When the user is done with shopping, he will normally review the list of products, quantities, and prices are correct. Once the customers is ready, he will be able to check out. At this level he will have to input his shipping information (name, address etc.).

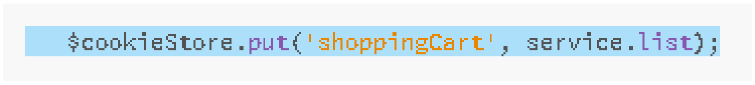
Starting from this definition we develop this flow of process. We started simple, by creating a service for our cart and adding a list.

We did the skeleton using an ordinary list with CRUD functionalities and for appearance adding templates from 'Bootsnip'. The picture below is a snapshot of the result we got after we implement add the necessary CSS and html code from the bootsnip website.



Product	Quantity	Price	Total	
		Subtotal		0 dkk
		Danish taxes		25 %
		Total		0 dkk

Starting from a working skeleton now we had to do it functional. Angular it's providing great options to develop great features. The ordinary list it became a value into a cookie, a necessary approach in order to maintain data during the shopping session. The code below is illustrating how w we stored our cart with cookieStore.



```
  $cookieStore.put('shoppingCart', service.list);
```

We add a list of products with the key ‘shoppingCart’, this way we will be able to do CRUD functionalities within this key. A fast solution for having our cart survive until the order is created.

Problems

We faced some issue regarding wiping the products from the cart, we are calling ‘wipping’ because we managed to delete our cookie but the cart is still not empty. Solution in this case was to reload the page. We approach the following piece of code:

```
$route.reload();
```

Which it didn’t quite behave the way we wanted, what we suggest is to determine the path we want to reload and do a windows location reload. That we want to implement but within the time range we decided to do it when the site is almost done and the polishing will occur.

Current phase

At this moment the Baltic Jagt website is a working structure, main features are solid defined and ease of future code implementation.

The current integrated code it has accomplish so far , a functional full process shopping cart and creating an order while iterating the number of items stored in the cart.

Conclusion

Referring to the knowledge achieved for this entire process we are quite happy with the result. MEAN stack is powerful and its architecture is helping us to get the product fast and dynamic. The current stage of the website is not full with features, but we are aware how to develop the rest of them. In cause of that we say that we focused on developing solid few features and understand the process itself due the generated amount of code. We apprehended skills from installing the right tools (MEAN stack) till scaffolding (YO), build script (Grunt), and manage client-side dependencies (Bower).

Overall we are confident on using MEAN for getting Baltic Jagt website for commercial use, and developing other websites with this type of architecture.

Appendix

References

https://github.com/DaftMonk/generator-angular-fullstack

http://nodejs.org/api

http://www.tutorialspoint.com/html/index.htm

http://www.tutorialspoint.com/html5/index.htm

http://www.tutorialspoint.com/css/index.htm

http://www.tutorialspoint.com/javascript/index.htm

http://yeoman.io