HOWEST

THESIS

Automated deployment and performance analysis of a white-label web application

Author:
Niek CANDAELE

Supervisor:
Thomas CLAUWAERT

February 17, 2021

"Funny or thought provoking quote goes here"

Someone, somewhere

HOWEST

Abstract

Bachelor of applied computer science

Automated deployment and performance analysis of a white-label web application

by Niek CANDAELE

A case study and practical implementation of a white-labeled web application. Starting with an existing application, proceeding with analysis of the current implementation and problems, investigating potential solutions and finally implementing them

Acknowledgements

Thanks to Thomas Clauwaert, Serge Morel, en de rest . . . :)

Contents

List of Figures

List of Tables

List of Abbreviations

AWS **A**mazon **W**eb **S**ervices

For/Dedicated to/To my…

Chapter 1

Intro

1.1 How it used to be

Stampix is a startup that prints photos. Stampix' customers are companies, these companies buy printcodes which they can then distribute to their users in context of marketing or loyalty campaigns. Every client gets their own branded web application.

This involves:

- Storing all brand-related content in a CMS (Contentful)
- Pulling in all that content during app runtime
- Deployments for new clients require a lot of manual configuration / dev work
- There's no automated tests, which can cause broken deployments if not careful

This had a few problems which I will explain in detail later ...

1.2 Solutions

Following are the methods used to improve this workflow. Each method will probably get it's own detailed chapter later?

- Using a static site generator to build web app and assets during build time
- Automated testing (Selenium-like / snapshots / unit)
- Deploying each built application to AWS

Chapter 2

Research

2.1 Static site generators

Static site generators take your app and build in before serving to users. This means users receive plain HTML files. This moves a large computation burden from run time to build time which results in significantly faster load times for users. Furthermore, this approach allows for more aggressive and efficient caching.

2.1.1 Nextjs

Nextjs[1] is a React framework. Not explicitly a static site generator but has support for it

2.1.2 Gatsby

Gatsby[2] is a static site generator at heart. It might be harder to do runtime stuff with Gatsby

2.1.3 Umi.js

2.2 Automated testing

We create lots of different webapps, a different one for each client. Currently, we have no automated tests. This should change

Automated testing gives developers confidence their changes did not break anything. It can spot bugs before the code is even released

2.2.1 Selenium

Selenium[3] is an established project

2.2.2 Cypress

Cypress[4] is pretty new and 'cool'

[1] https://nextjs.org/
[2] https://www.gatsbyjs.com/
[3] https://www.selenium.dev/
[4] https://www.cypress.io/

2.2.3 Protractor

2.2.4 Playwright

Playwright[5] is

2.3 Deployment

2.3.1 Ansible

2.3.2 Terraform

2.3.3 Serverless

2.3.4 AWS Cloud Development Kit

[5] https://playwright.dev/

Appendix A

Performance reports

A.1 Initial performance

Lighthouse scores? Some trace info? Other performance indicators?