

WARSAW UNIVERSITY OF TECHNOLOGY

FACULTY OF MATHEMATICS AND INFORMATION SCIENCE



CLOUD COMPUTING 2021/2022

FINAL PROJECT - REPORT

Motorcycle Pricing System

Stanisław Matuszewski
323153
Amir Ali
317554
Mateusz Kierznowski
323152

June 24, 2022

1. Project Description

The idea of a project is to create a web app and connect it with a model instance. Users can provide information about their desired motorbike and the website will return a quotation based on features. The infrastructure workflow can be described by components.

2. Functional and Non-Functional Requirements

In this part, we explain the Functional and Non-Functional Requirements of the project:

2.1 Functional Requirements

1. Scrapping motorcycle data.
2. Dumping Data into a DynamoDB instance.
3. Working web app with possibility to quotation
4. Creating a well-performing model with a connection to Webapp
5. Preparing log from scrapping

2.2 Non-Functional Requirements

1. Each request should be processed within 15 seconds.

Core of services are lambda functions which are serverless tools which can ensure that every request is processed under the 15 second time threshold. The web application per request needs less than 0.5 seconds to correctly fetch predictions to the model and display it to the user. Scraper part pushes data to dynamodb in batches of 50 observations. Every batch can be maintained under 6 seconds (depending on data volume)

2. Implement DR scenarios for the system ensuring RPO at a level of 6 hours.

In the implemented system, we decided on the most budget disaster recovery option – Backup&Restore. Compared to Pilot Light, Warm Standby and Active-Active provides the simplest and cheapest way of recovering the system. This strategy is based on regular backups. In our case backups are made

instantly on the instance of the used DynamoDB table and are stored there for 35 days. Additionally, we provide logs of scrapped data on the s3 bucket.

3. Ensure high availability of the Motorbike dataset.

Motorbike dataset is stored in Dynamo databases which ensures that access to the data is simple, fast and highly available. Additionally, NonSql databases such as dynamodb help with the data structure. During scrapping, some users did not provide all information about the offer. In that case, MySql database will need to store a value (eg. "NA"). Dynamodb works well because it treats observation as a document and does not need to store any unnecessary values.

4. The System is easy to use. I.e User is just required to type and click then the result is displayed.

The interface is a very simple web application and very easy to use. The input is based on the number of 14 features and based on input gave the output of Motorbike price prediction. Display of results is fast due to the fact that lambda can handle requests under 0.5 seconds.

5. The system is scalable and Cost-effectiveness.

Scalability of application is provided by dynamodb scalability which dynamically adjusts database throughput capacity on your behalf in response to actual traffic patterns. Cost-effectiveness is provided due to usage of lambda. This serverless tool according to dynamodb documentation costs \$0.20 per 1M requests. Which is a little expensive, especially in comparison to the number of requests we need.

3. System Architecture

CC project architecture

piątek, 20 maja 2022
10:24



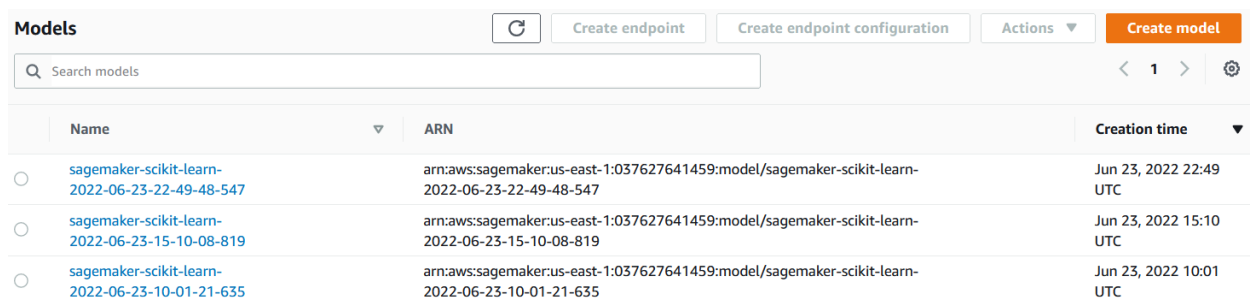
Figure 1: System Architecture of MotorBike Prediction Application

1. EC2 instance which holds a script for scraping data. After a run, it sends data to the s3 bucket in json format.
2. Then lambda catches that new data flows. It provides a test whether the run has been completed successfully, if not it asserts. Then lambda preprocess data and insert data into the MongoDB database.
3. Then we build the model using SageMaker endpoint. SageMaker is a fully managed service that provides the platform to build, train, and deploy ML models and it removes the heavy lifting from each step of the ML process to make it easier to develop high-quality models
4. The scrapper run can be repeated every day or every x day (preferably). SageMaker Model runs need to be created every month/2month to maintain price changes.
5. The frontend application is created using the Angular framework (v10.2.0). The user will have access to the motorcycle pricing form. The form consists of a sequence of inputs characterizing a given motorcycle for later evaluation. The client-side application communicates with the AWS

Lambda service via REST API. After sending a request to the AWS Lambda service, the valuation of the motorcycle from the model is returned in the response.

4. Building SageMaker endpoint

For the chosen model we build the SVM model which provides correct and pertinent predictions for requests sent by the user. Building procedure has been created via SageMaker library which ensures correct deployment of any sklearn model. In application SageMaker, after deploy request, sets the endpoint status to Creating. After it builds the endpoint, it sets the status to InService. Then SageMaker can process incoming requests from lambda. Working endpoints can be monitored via SageMaker resources interface. In our case:



The screenshot shows the AWS SageMaker console 'Models' page. At the top, there are buttons for 'Create endpoint', 'Create endpoint configuration', 'Actions', and 'Create model'. Below these is a search bar labeled 'Search models'. The main content is a table with columns: 'Name', 'ARN', and 'Creation time'. There are three models listed, each with a radio button to its left. The first model is 'sagemaker-scikit-learn-2022-06-23-22-49-48-547' with ARN 'arn:aws:sagemaker:us-east-1:037627641459:model/sagemaker-scikit-learn-2022-06-23-22-49-48-547' and creation time 'Jun 23, 2022 22:49 UTC'. The second model is 'sagemaker-scikit-learn-2022-06-23-15-10-08-819' with ARN 'arn:aws:sagemaker:us-east-1:037627641459:model/sagemaker-scikit-learn-2022-06-23-15-10-08-819' and creation time 'Jun 23, 2022 15:10 UTC'. The third model is 'sagemaker-scikit-learn-2022-06-23-10-01-21-635' with ARN 'arn:aws:sagemaker:us-east-1:037627641459:model/sagemaker-scikit-learn-2022-06-23-10-01-21-635' and creation time 'Jun 23, 2022 10:01 UTC'.

	Name	ARN	Creation time
<input type="radio"/>	sagemaker-scikit-learn-2022-06-23-22-49-48-547	arn:aws:sagemaker:us-east-1:037627641459:model/sagemaker-scikit-learn-2022-06-23-22-49-48-547	Jun 23, 2022 22:49 UTC
<input type="radio"/>	sagemaker-scikit-learn-2022-06-23-15-10-08-819	arn:aws:sagemaker:us-east-1:037627641459:model/sagemaker-scikit-learn-2022-06-23-15-10-08-819	Jun 23, 2022 15:10 UTC
<input type="radio"/>	sagemaker-scikit-learn-2022-06-23-10-01-21-635	arn:aws:sagemaker:us-east-1:037627641459:model/sagemaker-scikit-learn-2022-06-23-10-01-21-635	Jun 23, 2022 10:01 UTC

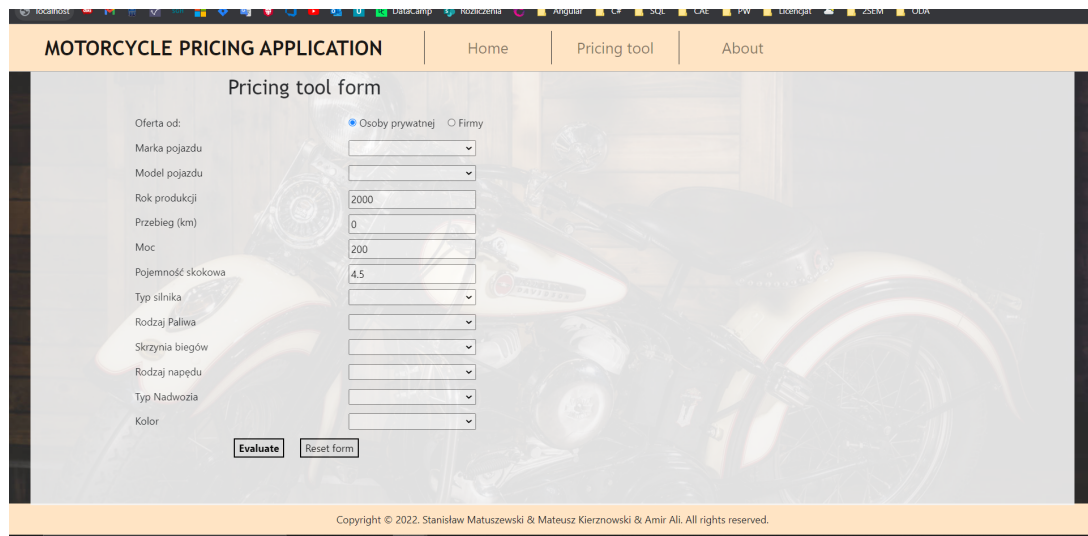
Final model: *sagemaker-scikit-learn-2022-06-23-22-49-48-547*. Whenever lambda wants to send requests to the endpoint it just needs to point the name and call SageMaker invoke scripts.

5. Web application

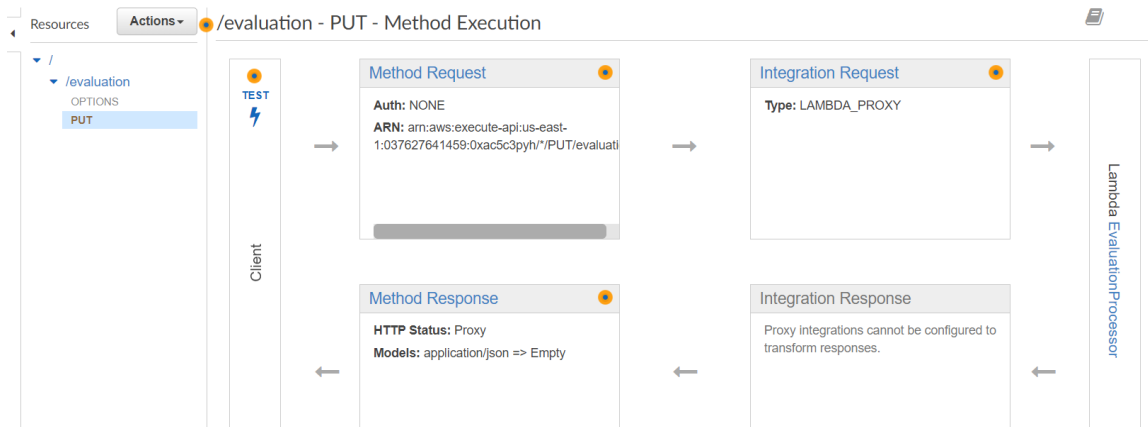
The frontend application was created entirely with use of the Angular framework (v10.2.0). Since it is a relatively big SPA framework, our application has the possibility to expand its functionalities, use cases and easily deal with growing project complexity. User interface consists of 3 main tabs - Home, Pricing tool and About. The main part of an app is a Pricing Tool, which provides the user the possibility to represent motorcycles as a set of 15 features.

There are two buttons below the input fields:

- Evaluate - to send entered motorcycle to prediction
- Reset form - to reset the form to its initial state



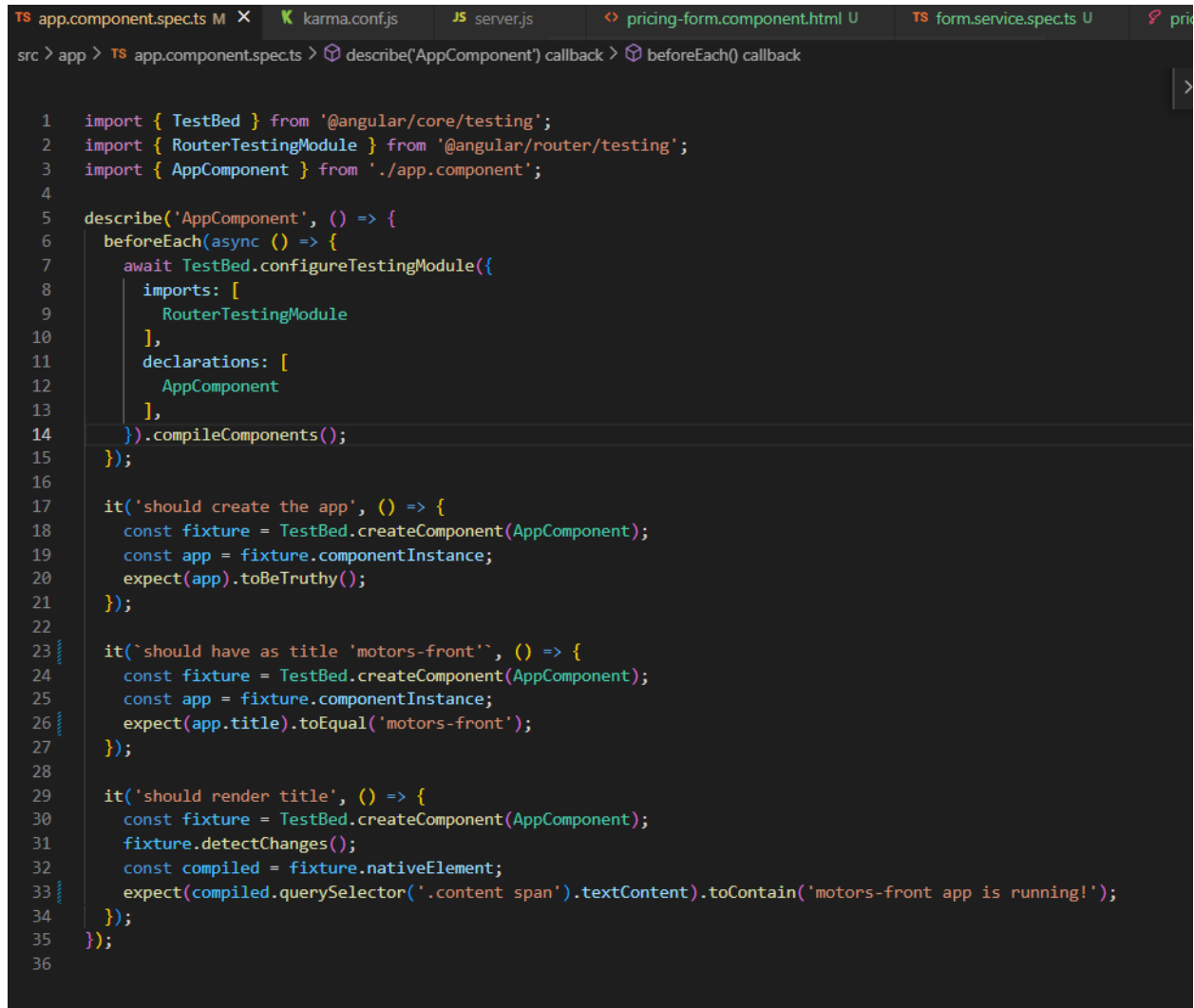
After clicking the *Evaluate* button, the PUT request for the specific endpoint on API Gateway is sent. The body of this request includes the JSON object representing the entered motorcycle. It triggers the execution of a dedicated lambda *EvaluationProcessor* function. As a response, it returns the predicted value of a model.



Application code includes basic unit tests carried out within the Angular framework. The whole application is stored on AWS as S3 bucket. The application takes 17,2 MB of memory, 15 of which are for internal multimedia files(videos, images).

Unit tests

As part of the Angular application, we implemented basic unit testing mechanism in files *.spec.ts. Each Angular component has its own dedicated test file, which verifies if loading and rendering of elements related to it was finished successfully.



```
src > app > TS app.component.spec.ts > describe('AppComponent') callback > beforeEach() callback

1  import { TestBed } from '@angular/core/testing';
2  import { RouterTestingModule } from '@angular/router/testing';
3  import { AppComponent } from './app.component';
4
5  describe('AppComponent', () => {
6    beforeEach(async () => {
7      await TestBed.configureTestingModule({
8        imports: [
9          RouterTestingModule
10         ],
11        declarations: [
12          AppComponent
13         ],
14      }).compileComponents();
15    });
16
17    it('should create the app', () => {
18      const fixture = TestBed.createComponent(AppComponent);
19      const app = fixture.componentInstance;
20      expect(app).toBeTruthy();
21    });
22
23    it('should have as title \'motors-front\'', () => {
24      const fixture = TestBed.createComponent(AppComponent);
25      const app = fixture.componentInstance;
26      expect(app.title).toEqual('motors-front');
27    });
28
29    it('should render title', () => {
30      const fixture = TestBed.createComponent(AppComponent);
31      fixture.detectChanges();
32      const compiled = fixture.nativeElement;
33      expect(compiled.querySelector('.content span').textContent).toContain('motors-front app is running!');
34    });
35  });
36
```

6. Technologies and Libraries

As presented on the system diagram we have used many AWS tools which provide all necessary functions to easily deploy our application. For different parts we used the following technologies:

Scraper Part:

- boto3

- Beautiful Soup 4.9.0
- Pandas

Model Part:

- Sagemaker
- Numpy
- Scikit-Learn
- TensorFlow
- Pickle

Web Application:

- Angular 10
- Bootstrap 4
- PrimeNG (free Angular's components library)