

SERVERLESS TECHNOLOGIES AND THE CIRCUIT BREAKER PATTERN

April 3rd 2019

Abstract

Serverless technologies and their impact on web hosting are critically analysed and compared with traditional hosting technologies. The Circuit Breaker Pattern and the reasons for implementing the pattern in modern web development are discussed and critically analysed.

Samuel Withers

Contents

1) Introduction	2
1.1 Traditional web development and hosting	2
1.2 Goals	2
2) Serverless technologies	3
2.1 Netflix case study.....	3
3) Circuit Breaker Pattern	5
3.1 How does the pattern work?.....	5
3.2 When to use the Circuit Breaker pattern?	7
4) References	8

1) Introduction

1.1 Traditional web development and hosting

When a company builds a website, in order for it to be accessed remotely, it must be hosted. A web host is the technology that provides a website with the functionality it needs to operate on the web. A website can be hosted on your own laptop if needed, however, it is a very unreliable solution and not suitable for any web service with significant traffic [1]. Traditionally, a company will have a server (or collection of servers) locally on site. A server is a specialised computer that provides functionality for other computers [1][2]. In the case of web hosting, this functionality is access to the web service being requested by the client computer.

The company hosting the web service is responsible for having the architecture in place to host the service, so if the usage of the service increases, the company must provide the architecture to host the greater number of requests coming through.

This means that traditionally, large companies would have large server room, or even warehouses, with dedicated servers to receive the requests from client machines and perform the functionality needed[2]. This leads to large amounts of hardware essentially being unused 90% of the time, or companies not possessing the hardware to cope with large spikes in requests to their web service, leading it to go down.

1.2 Goals

In this report, the use of serverless technologies in web hosting is explored and critically analysed. Appropriate use cases of the technologies are highlighted and a case study of a major company explored. Serverless technologies are compared with traditional web hosting technology.

The circuit breaker pattern and the use of the pattern in modern web development is explored and critically evaluated. The operation of the pattern is outlined and appropriate uses of the pattern highlighted and explored.

2) Serverless technologies

Serverless technologies are not, in fact, serverless. What they do is eliminate the need for local server hardware by using hardware from a cloud provider (AWS, Microsoft Azure, Google Cloud, etc). So rather than hosting a service on local server hardware, the service is run on servers hosted by one of the serverless providers [3]. This is not the only difference however; serverless technologies allow programs to be scalable. If you have server hardware locally to host your service, you need enough hardware to cope with peak usage of your service (arguably more for the event of failure of parts of hardware). If the service has a wide range of usage, this means at non-peak times, some of the hardware is simply sitting there not being utilised. Buying server hardware to host services is very expensive, so you want as much of it to be utilised at any given time as possible. Using a serverless provider allows the client to pay for the server hardware needed at any given time. If a service is under low usage, only a small amount of the serverless providers' resources will be used and the client will pay accordingly [3][4]. As the usage increases, the serverless provider allows the use of more hardware to accommodate the increase in usage. The client is charged based on how many resources are needed from the serverless provider.

So in essence, serverless technology is the practise of renting servers from providers, but only renting as many as are needed at any given time. This typically cuts down on costs for companies, and cuts down on unused hardware. The main draw for companies to use serverless technologies is that there is no large upfront cost for hardware and the costs associated with running the web service scale with the usage of the service, which is perfect as the use typically scales with the profits.

2.1 Netflix case study

Netflix is a video streaming platform predominantly for tv-shows and movies [5][6]. They also rent and sell DVD's, but this makes up a miniscule portion of their online usage that for this case study we shall ignore it. Netflix transfers petabytes of data every single day to millions of users [5][6]. When Netflix began the online streaming portion of the business, they were using in-house server hardware to host the service. As the use of their online streaming service increased, Netflix found they could not keep up with the increase in demand from buying in new hardware, and the cost of buying in new hardware was high. So over the course of seven years Netflix was essentially re-written and all functionality transferred to the AWS (Amazon web services) cloud and Netflix became a completely serverless cloud-based application [5][6].

Netflix typically sees large spikes in activity on Sunday afternoons (when the most people typically sit down and watch shows), being completely serverless allows them to not worry about the usage from customers becoming more than their servers can handle, as Amazon's technology scales the amount of hardware that Netflix rent from them for the duration of the spike [5][6].

The move to serverless technology has allowed Netflix to grow to the scale it has today without having any of its services impeded by large demand so has therefore increased uptime percentage of the service. It has also cut down on costs for the company making it more profitable [5][6].

3) Circuit Breaker Pattern

A circuit breaker in electronics is a switch designed to trip when an anomaly is detected. This is typically when the current in the circuit is too high or a short circuit is detected. A circuit breaker is designed to interrupt the flow of current through the circuit when a fault is detected with the purpose of preventing lasting damage to the circuit.

The circuit breaker pattern in software development serves a similar purpose. Faults in systems that occur due to predictable events, such as slow connections causing a call to a server to fail, are often easily fixed and often correct themselves after a short time. However, there are often failures in systems due to unanticipated events that may take much longer to fix. Without the circuit breaker pattern, a system may continuously try to call an operation that has stopped working, either because of the system failure or because it caused the system failure. The circuit breaker pattern will allow the system accept the operation is failing and handle the failure [7][8].

If a service is seeing abnormally high traffic (DDOS attack, or simply more requests per second than it can handle) this could cause one part of the system to fail, which could in turn cause parts of the system further along to fail. The circuit breaker pattern will allow the one part of the system to fail, and not waste resources on trying to call a failing operation repeatedly, so the rest of the system can continue to operate without that single operation [7][8].

In short, the circuit breaker pattern will prevent a system from repeatedly trying to perform an operation that is likely to fail. This allows a system to continue working without having to fix the fault, this is integral in a system that needs the highest percentage uptime possible. This means the problem can (hopefully) be fixed whilst the system is still operational, the circuit breaker pattern can then allow the program to detect the problem has been fixed and start performing that operation again as part of the system.

The circuit breaker pattern is similar to another pattern called the retry pattern. The retry pattern allows a program to retry a failed operation with the expectation that it will succeed [8]. This is crucially different to the circuit breaker pattern as it serves almost the opposite function. The circuit breaker pattern prevents an operation from happening that may continue to fail. It is possible to implement the retry pattern and circuit breaker pattern together, however the retry pattern must have logic implemented within it so it knows to not retry operations that the circuit breaker has flagged as not working.

3.1 How does the pattern work?

A circuit breaker implemented in a program acts in a very similar way to an actual circuit breaker in an electronic circuit. However, the circuit breaker in an electronic circuit only has two states, on and off. It either allows current to flow through or allows absolutely no

current to flow through. The circuit breaker pattern instead has three states; closed, open and half-open.

Closed means that the application operates as normal and the operation the pattern is tied to functions as normal.

The circuit breaker moves to the open state when too many failures have occurred on the operation within a certain time limit. When open, operation will immediately fail without the resources being spared to send another request.

After a certain amount of time in the open state, the circuit breaker will transfer to the half-open state. In this state a limited number of requests are allowed to pass through as a check to see whether the operation is working correctly again. If the operation fails any of these requests, the circuit breaker is put back in the closed state, however, if the requests are successful, it is assumed the problem has been fixed and the breaker is put back into the closed state [7][8].

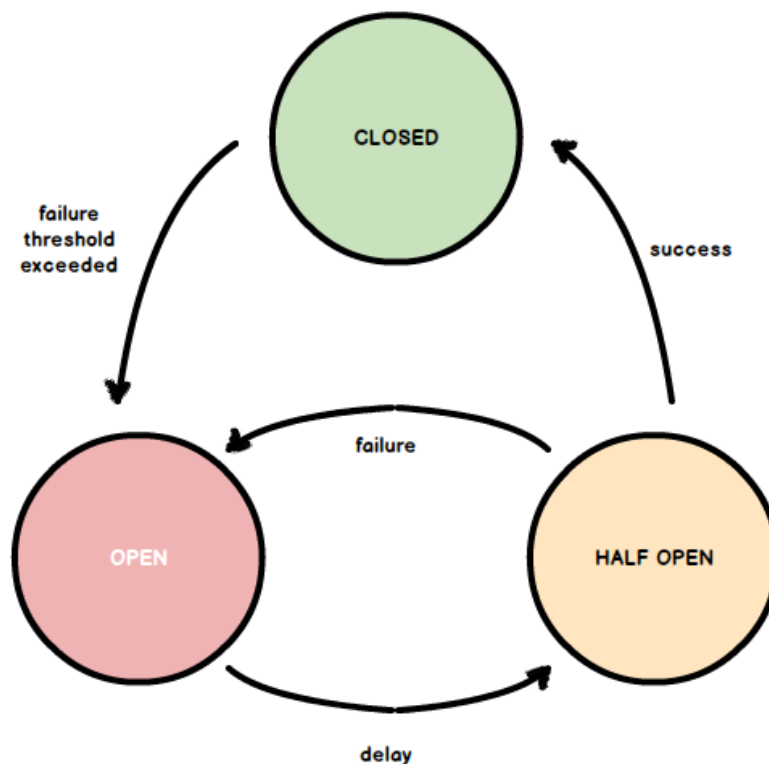


Figure 1: Diagram of the basic operations of the circuit breaker pattern [9]

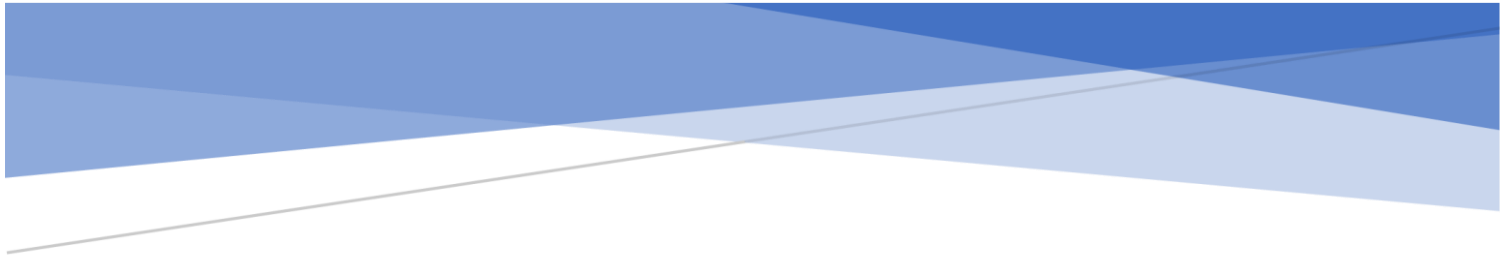
3.2 When to use the Circuit Breaker pattern?

The circuit breaker pattern is often used in web development. This is because often a web page is populated by sources accessed from other parts of the web. If one of these external sources is not working correctly or is down, then this should not cause the web service to fail. The circuit breaker pattern will allow the website to continue operating as normal without wasting computing resources making calls to the external source that is not responding.

For example, a web service using external sources could be configured with a time-out period, where if the external source didn't respond for 30 seconds the service will assume the source is unavailable and throw an exception. However, if the external source has completely failed, this means every user must have the service active for 30 seconds before getting the exception. This can take up a lot of traffic and resources for the service and could potentially cause an overload, this is where the circuit breaker comes in. The circuit breaker will detect that the external source has failed, and stop the service from making calls to it until it is fixed. This saves the web service from potential crashes and keeps it operational.

4) References

- [1] "How to Host a Website from Your Computer - Make A Website Hub", *Make A Website Hub*, 2019. [Online]. Available: <https://makeawebsitehub.com/host-website-computer/>. [Accessed: 01- Apr- 2019].
- [2] I. Cohen, *The Ultimate Guide to Web Hosting*. Host Trader, 2014.
- [3] S. Burns, "Serverless computing: What it is and why it matters", *Computer Weekly*, 2018. [Accessed 5 April 2019].
- [4] *Introduction to Serverless Computing with AWS Lambda*. Contino, 2019.
- [5] "Serverless Case Study — Netflix", 2019. [Online]. Available: <https://dzone.com/articles/serverless-case-study-netflix>. [Accessed: 04- Apr- 2019].
- [6] N. Lambda, "Netflix & AWS Lambda Case Study - Amazon Web Services (AWS)", *Amazon Web Services, Inc.*, 2019. [Online]. Available: <https://aws.amazon.com/solutions/case-studies/netflix-and-aws-lambda/>. [Accessed: 05- Apr- 2019].
- [7] M. Nygard, *Release it!*. Pragmatic Bookshelf, 2007.
- [8] "Circuit Breaker pattern - Cloud Design Patterns", *Docs.microsoft.com*, 2019. [Online]. Available: <https://docs.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker>. [Accessed: 05- Apr- 2019].
- [9] *circuit breaker pattern for microservices*. 2019.



Serverless technologies and the circuit breaker pattern

April 3rd 2019

Abstract

Serverless technologies and their impact on web hosting are critically analysed and compared with traditional hosting technologies. The Circuit Breaker Pattern and the reasons for implementing the pattern in modern web development are discussed and critically analysed.

Samuel Withers

Contents

1) Introduction	11
1.1 What is a RESTful web service?	11
1.2 Goals	11
2) RESTful Architectural Constraints	12
2.1 Uniform Interface	12
2.2 Client-server architecture	12
2.3 Statelessness	13
2.4 Cacheability	14
2.5 Layered system	15
.....	16
.....	16
2.6 Code on demand	17
3) Evaluation of REST	18
4) References	19

1) Introduction

1.1 What is a RESTful web service?

REST stands for Representational State Transfer. It is a software architectural style for networked applications. However, the primary use is build web services of any scale. REST has become a widely used technology in web development due to the flexibility, scalability and efficiency [1].

REST was introduced in the year 2000 by Roy Fielding in his PhD dissertation. It was created with the purpose of having a model for web architecture that was consistent and scalable. Roy himself outlines the goals of REST as such “REST emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems.” [2 – page 17]

Most new web services are built using REST, this is due to REST being designed with scalability and reusability of components in mind. In an era when a web service may be maintained and added to for years at a time and service hundreds of millions of clients, these principles are of utmost importance.

REST is essentially a well-defined set of constraints to keep services consistent with one another and to allow the service to work in the most efficient and sensible way possible [1].

1.2 Goals

In this report, the best practices of RESTful web services are outlined and examples of each are shown. The architectural constraints of working in a RESTful way are discussed and the benefits and drawbacks are discussed.

2) RESTful Architectural Constraints

A truly RESTful web service has six architectural constraints: Uniform Interface, Client-server architecture, Statelessness, Cacheability, Layered System and Code on demand [1][2].

In this section, an overview of each constraint is given and appropriate examples from a RESTful web service are provided and how the constraint is adhered.

2.1 Uniform Interface

A Uniform Interface is a crucial part of any RESTful web service and defines how the client and server interact with one another [3].

The constraint has four guiding principles, the first being 'resource-based' where, within REST, a resource is an object with a type and has some data associated with it. The resource would then be accessed by a URI and represented in a format such as JSON. RESTful service is based on a collection of these resources [3] [4].

The second principle is 'actions on resources through these representations' where if, for instance, the service holds the representations of the resources in JSON format, it holds enough information to modify or delete the resource [3].

Thirdly, 'self-descriptive messages' occur when resources may be represented in multiple formats, therefore, each resource needs representation in a decoupled message from the resource itself [3].

And finally, 'Hypermedia as the Engine of Application State' which is commonly referred to as HATEOAS. The client can only transition state within the service through actions that are defined by hypermedia on the server, for example, hyperlinks in text [3].

2.2 Client-server architecture

The principle behind the client-server constraint is the 'separation of concerns', the design principle for separating parts of a program into distinct sections so that each section is responsible for a different 'concern' and doesn't have any information about the others. This allows a program to be easily maintained and modified [5].

Client-server means that the client is not concerned with data storage and has no information about data storage. All the data storage is handled on the server side. On the other side, the server has no information about the user interface, this means that logic from the server side can be easily transferred to another system with a completely different user interface if needed [1] [2].

Separation of concerns is a very important principle in programming, and is put into great effect in RESTful web services, allowing individual parts of web services to be reusable and easily maintained and or updated.

```
@RepositoryRestResource(collectionResourceRel = "plates", path = "plates")
public interface PlatesRepository extends PagingAndSortingRepository<Plate, Integer> {
}
```

Figure 2: Repository in the Java backend responsible for communicating with the database [6]

Figure 1 shows a complete implementation of a REST service to write to a database. This is in Java on the server-side of the service. There are repositories for each type of object in the service that needs to be saved to the database, meaning all information about data storage is contained within the server side of the service. This means the client side of the service simply accesses data through a set of pre-defined endpoints, keeping the service secure and the system loosely coupled.

2.3 Statelessness

The Statelessness constraint states that each request from the client to the server must contain all the information needed to process the request and the state of the session is held by the client. Essentially, no session information is retained by the receiver and the in the case of client-server architecture, the server is the receiver [1].

```
function saveBid() {
    $.ajax({
        type: "POST",
        url: "http://localhost:8080/bidsmade",
        data: {"bidAmount": $('#bidMade').val(), "auctionId": $('#auctionID').val(), "batchId": $('#batchNumber').val()},
        success: function () {
            alert("Post Success");
        },
        error: function () {
            alert("The post failed");
        }
    });
}
```

Figure 3: Example of a JavaScript request to the database [6]

Figure 2 shows a JavaScript function making a POST to the backend. The JavaScript does this only with the knowledge of the column names in the database. All the JavaScript needs is the correct endpoint to be specified in the url field, and the correct column names specified in the data field. This is all the information the client needs to pass to the server to process the request. The state of the session is still held by the client.

2.4 Cacheability

Cacheability is simply responses defining themselves as cacheable or uncacheable. This is to prevent clients from having access to data that is either old or inappropriate. If caching is done incorrectly it can impact the performance of the service due to each client-server interaction including unnecessary cached data meaning each data transfer takes longer than needed [7][1].

```
function showAuction() {  
  $.ajax({  
    cache:false,  
    type: "GET",  
    url: "http://localhost:8080/batch/" + $('#batchNumber').val() + "/auction",  
    contentType: "application/json",  
    dataType: "json"  
  }).then(function (data) {  
    $('#auctionID').text("<h2 class=\"text-center\">" + data.auctionid + "</h2>");  
  });  
}
```

Figure 4: An example of a GET request with caching set to false [6]

The JavaScript function shown in Figure 3 has caching set to false as the data accessed from the GET request does not need to be stored at the point the function is called, therefore, the data does not need to be cached. Caching the data accessed with this function would cause the data to be loaded at times where it does not need to be loaded, leading to a marginally slower system. Whilst the slow down for what amounts to a single piece of text is imperceptible, it is good practice to follow this constraint. RESTful web services are built with scalability in mind, so as the service scales up, so will the amount of data transferred meaning well-defined caching will keep the service from losing performance unnecessarily.

2.5 Layered system

Layered system is another constraint commonly used across software development. The concept of a layered system is that components within a system are layered in a hierarchical arrangement in such a way that lower layers provide functionality that supports the functions of higher layers [8] [1].

This can be used in a RESTful web service to make sure the client cannot tell if it is connected directly to the end server. The layers are a set of intermediary servers that can help to improve system performance by holding commonly cached data.

Figure 4 shows a class that is used to randomly generate a string in the format of a registration plate. This class has no information about what a Plate object is, or what a Batch object is, it simply exists to provide functionality to classes higher in the hierarchy.

Figure 5 then shows the plate generator class being used to generate a 'batch'. This is noteworthy as the batch generator is aware of what the plate generator is but does not yet know what Plate and Batch objects look like, neither does it need to. This contributes to the layered system by keeping information from different components separate until needed in the hierarchy.

```
/**
 * Class to generate a random license plate with the correct letter number format
 * @return the randomly generated license plate
 */

public String generatePlate (int first, int second){
    reg = "";
    for (int i = 0; i < 2; i++) {
        char ch1 = (char) (Math.random() * 26 + 'A');
        reg = reg + String.valueOf(ch1);
    }

    //Used to generate specific integers for specific batches
    reg = (reg + String.valueOf(first));
    reg = (reg + String.valueOf(second));

    reg = reg + " ";

    for (int i = 0; i < 3; i++) {
        char ch2 = (char) (Math.random() * 26 + 'A');
        reg = (reg + String.valueOf(ch2));
    }
    return reg;
}
```

Figure 5: Class used to generate a random string in format of a registration plate [6]

```
/**
 * Generates batches of number plates. Returns an arraylist of strings that contains the plate numbers
 */
public class BatchGenerator {
    private PlateGenerator pg = new PlateGenerator();
    ArrayList<String> plateBatch = new ArrayList<>();

    public ArrayList<String> generateBatch(int first, int second) {
        if(plateBatch.size() > 0)
            plateBatch.clear();

        for (int i = 0; i < 5; i++) {
            String plate = pg.generatePlate(first, second);
            plateBatch.add(plate);
        }
        return plateBatch;
    }
}
```

Figure 6: Class used to generate five random registration plate strings [6]

Figure 6 shows where the plate generator and batch generator are used and is an example of a hierarchical arrangement of functionality. 'plateBatchMaker' calls generateBatch, which in turn calls plateGenerator. The plateBatchMaker is then used to save plates to the plate repository. Components lower down in the hierarchy can easily be changed as they are only concerned with the information contained within themselves and not that of components higher up.

```
private ArrayList<String> plateBatchMaker(int first, int second) { return bg.generateBatch(first, second); }
//Outer for loop to creates 10 batches and assign them with an auction
for (int i = 0; i < 100; i++) {
    int first = i % 10;
    int second = i % 5;

    batch = new Batch(auctionList.get(new Random().nextInt(auctionList.size())));
    br.save(batch);
    //Inner for loop to populate each batch with 5 plates
    for (String plate: plateBatchMaker(first, second)) {
        pr.save(new Plate(plate, batch));
    }
}
```

Figure 7: The highest place in the hierarchy that the plate generator and batch generator are used [6]

2.6 Code on demand

This constraint is optional, so a web service can still be considered RESTful even if it does not conform to Code on Demand.

Code on Demand is the process of the server transferring logic that can be executed to the client to add to or change the functionality of the client. JavaScript, for example, is a client-side scripting language that can be used to add functionality to a web service on the client's end [1].

As this is an optional constraint, there are many web services that do not use Code on Demand. The DVLA for example, often try to create services with as little client-side functionality as possible, as some users of the service may disable all client-side functionality in their browsers.

```
function makeBid() {  
    var x = document.getElementById("bid").value;  
    var y = document.getElementById("bidMade").value;  
    if (x <= y + 1) {  
        alert("Value cannot be less or same as current max bid");  
        return false;  
    }  
    document.getElementById("bidMade").value = document.getElementById("bid").value;  
    document.getElementById("bid").value = "";  
}
```

Figure 8: Code on Demand constraint being adhered to with JavaScript [6]

Figure 7 shows an example of a JavaScript version that satisfies the Code on Demand constraint. This function is not run when the program starts as it is run client side when the client performs the action that calls the 'makeBid()' function. This function simply moves a value from one place to another, however this functionality is available to the client at any time as the code is run client side.

3) Evaluation of REST

In his PhD dissertation, Dr. Roy Fielding said “REST provides a set of architectural constraints that, when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems.” [2 – page 95] As this statement holds up almost perfectly nineteen years later is a testament to how well designed and defined the constraints of RESTful services are.

As Dr Fielding said, REST services were designed with scalability in mind, and with the explosion of the internet since 2000, this design philosophy has certainly been proven. It was almost impossible to predict back in 2000 the size and scale of some of the web services we have today, but with the majority of public API's following a RESTful architecture, REST is here to stay.

Until around 2010, a large portion of web services were still built using Simple Object Access Protocol (SOAP), an alternative web service interface to REST, however most modern web services are now built using REST [9].

That is not to say REST is flawless and superior for every situation as an argument can still be made for SOAP services even today. Firstly, SOAP is built on stateful operations, as opposed to the stateless operations of REST and also contains built-in methods to increase security in transaction in asynchronous operations. As such, SOAP can still benefit developers if a service with high transaction reliability is needed [9].

REST has mostly taken over from SOAP as the web service architecture of choice due to numerous reasons. REST is very simple to understand as it is built around the basic CRUD (create, read, update, delete) operations. This makes it very simple for other developers to look at a REST service and understand the basics of the service.

REST also supports many data formats, but with JSON being the standardised method for consuming API information, REST's ease of use with JSON makes the architecture far easier to use and more standard [1].

In conclusion, REST has become the standard for building web services in the modern era and was designed in such a robust way that the architecture will most likely be the standard for years to come, due to the unparalleled scalability and reusability of components.

4) References

- [1] L. Richardson and S. Ruby, *RESTful web services*. Beijing: O'Reilly, 2007.
- [2] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures", PhD, UNIVERSITY OF CALIFORNIA, IRVINE, 2000.
- [3] S. Allamaraju, *RESTful web services cookbook*. 2010.
- [4] "Resources", *Restful-api-design.readthedocs.io*, 2019. [Online]. Available: <https://restful-api-design.readthedocs.io/en/latest/resources.html>. [Accessed: 07- May- 2019].
- [5] E. Elliott, *Programming JavaScript applications*. Beijing: O'Reilly, 2014.
- [6] Own work
- [7] N. Darkeye and g. padavala, "REST API Caching Guide – REST API Tutorial", *Restfulapi.net*, 2019. [Online]. Available: <https://restfulapi.net/caching/>. [Accessed: 07- May- 2019].
- [8] S. Wilson et al., "REST Principles and Architectural Constraints – REST API Tutorial", *Restfulapi.net*, 2019. [Online]. Available: <https://restfulapi.net/rest-architectural-constraints/>. [Accessed: 07- May- 2019].
- [9] "SOAP vs. REST Comparison: Differences in Performance, APIs & More", *Stackify*, 2019. [Online]. Available: <https://stackify.com/soap-vs-rest/>. [Accessed: 07- May- 2019].

Appendix

jquery min.js file not included due to size and it not being my own work

```
package com.example.demo;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

import javax.persistence.*;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.List;

/**
 * Class to build auction object
 */
@Entity
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
@Table(name="auction")
public class Auction implements java.io.Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int auctionid;
    private LocalTime starttime;
    private LocalTime endtime;

    //One to many mapping, auction has many batches
    @OneToMany(fetch = FetchType.LAZY, targetEntity=Batch.class, mappedBy =
"auction")
    private List<Batch> batch;

    public Auction () {
    }

    public int getAuctionid() {
        return auctionid;
    }

    public void setAuctionid(int auctionid) {
        this.auctionid = auctionid;
    }

    public LocalTime getStarttime() {
        return starttime;
    }

    public void setStarttime(LocalTime starttime) {
        this.starttime = starttime;
    }

    public LocalTime getEndtime() {
        return endtime;
    }

    public void setEndtime(LocalTime endtime) {
        this.endtime = endtime;
    }

    public List<Batch> getBatch() {
        return batch;
    }

    public void setBatch(List<Batch> batch) {
        this.batch = batch;
    }
}
```

```

        public Auction(LocalTime starttime, LocalTime endtime) {
            this.starttime = starttime;
            this.endtime = endtime;
        }
    }

package com.example.demo;

import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

/**
 * Repository interface for auctions
 * Responsible for mapping java code to the database
 */
@RepositoryRestResource(collectionResourceRel = "auction", path = "auction")
public interface AuctionRepository extends PagingAndSortingRepository<Auction,
Integer> {
}

package com.example.demo;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

/**
 * Class to build a Batch object
 */
@Entity
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
@Table(name="batch")
public class Batch implements java.io.Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int batchid;

    //One batch has many plates
    @OneToMany(fetch = FetchType.LAZY, targetEntity=Plate.class, mappedBy =
"batch")
    private List<Plate> plate;

    //Many plates have one auction
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "auctionid_fk")
    private Auction auction;

    public Batch() {
    }

    public Batch(Auction auction) {
        this.auction = auction;
    }

    public int getBatchid() {
        return batchid;
    }

    public void setBatchid(int batchid) {
        this.batchid = batchid;
    }

    public List<Plate> getPlate() {
        return plate;
    }
}

```

```

    public void setPlate(List<Plate> plate) {
        this.plate = plate;
    }

    public Auction getAuction() {
        return auction;
    }

    public void setAuction(Auction auction) {
        this.auction = auction;
    }
}
package com.example.demo;

import java.util.ArrayList;
import java.util.List;

/**
 * Generates batches of number plates. Returns an arraylist of strings that
 * contains the plate numbers
 */
public class BatchGenerator {
    //Calls plate generator to generate the plates within the array list
    private PlateGenerator pg = new PlateGenerator();
    ArrayList<String> plateBatch = new ArrayList<>();

    public ArrayList<String> generateBatch(int first, int second) {
        //Clears the arraylist after each set of five plates has been created
        if(plateBatch.size() > 0)
            plateBatch.clear();

        //Creates five (mostly) random number plates and adds them to the arraylist
        for (int i = 0; i < 5; i++) {
            String plate = pg.generatePlate(first, second);
            plateBatch.add(plate);
        }
        return plateBatch;
    }

    public BatchGenerator() {
        this.plateBatch = plateBatch;
    }
}
package com.example.demo;

import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

/**
 * Repository interface for batch
 * Responsible for mapping java code to the database
 */
@RepositoryRestResource(collectionResourceRel = "batch", path = "batch")
public interface BatchRepository extends PagingAndSortingRepository<Batch, Integer>
{
}
package com.example.demo;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

import javax.persistence.*;

/**
 * Class to build a Bid object
 */
@Entity

```

```

@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
@Table(name="bid")
public class Bid implements java.io.Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int bidid;
    private int bidamount;

    //One bid has one auction
    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "auc_bid_fk")
    private Auction auction;

    //One bid can only apply to one batch
    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "batch_bid_fk")
    private Batch batch;

    public Bid(int bidamount, Auction auction, Batch batch) {
        this.bidamount = bidamount;
        this.auction = auction;
        this.batch = batch;
    }
}

package com.example.demo;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

/**
 * Controller class for bids as they must be posted from front end
 */
@RestController
public class BidController {

    //Create each repository needed
    private BidRepository bidr;
    private BatchRepository batr;
    private AuctionRepository aucr;

    @Autowired
    public BidController (BidRepository bidr, BatchRepository batr,
AuctionRepository aucr) {
        this.bidr = bidr;
        this.batr = batr;
        this.aucr = aucr;
    }

    //Specify the end point to make the bids on
    @PostMapping("/bidsmade")
    public void makeBid (int bidAmount, int batchId, int auctionId) {
        Batch batch = batr.findById(batchId).get();
        Auction auction = aucr.findById(auctionId).get();

        Bid bid = new Bid(bidAmount, auction, batch);
        bidr.save(bid);
    }
}

package com.example.demo;

import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.web.bind.annotation.PostMapping;

/**

```

```

* Repository interface for bid
* Responsible for mapping java code to the database
*/
@RepositoryRestResource(collectionResourceRel = "bids", path = "bids")
public interface BidRepository extends PagingAndSortingRepository<Bid, Integer> {
}

package com.example.demo;

import org.springframework.web.bind.annotation.RequestMapping;

/**
 * Controller class to handle the html endpoints
 * Each end point is given it's own html to load when it's hit
 */
@Controller
public class Controller {

    @RequestMapping("/login")
    public String Generate() {
        return "Login.html";
    }

    @RequestMapping("/bid")
    public String Bid() {
        return "Bid.html";
    }
}

package com.example.demo;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import javax.persistence.*;

@Entity
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
@Table(name="customer")
public class Customer implements java.io.Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int customerid;
    private String email;
    private String password;

    public Customer(String email, String password) {
        this.email = email;
        this.password = password;
    }
}

package com.example.demo;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

/**
 * Controller class for customers
 * Needed as customers are posted from front end to database
 */
@RestController
public class CustomerController {

    private CustomerRepository custr;

```

```

    @Autowired
    public CustomerController (CustomerRepository custr) {
        this.custr = custr;
    }

    @PostMapping("/customer")
    public void makeCustomer (String email, String password) {

        Customer customer = new Customer(email, password);

        custr.save(customer);
    }
}
package com.example.demo;

import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.web.bind.annotation.PostMapping;

/**
 * Repository interface for customers
 * Responsible for mapping java code to the database
 */
@RepositoryRestResource(collectionResourceRel = "customers", path = "customers")
public interface CustomerRepository extends PagingAndSortingRepository<Customer,
Integer> {
}
package com.example.demo;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.PostMapping;

import javax.annotation.PostConstruct;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Random;

/**
 * Service used to fill the database with data generated by java classes
 */
@Service
public class DataFillerService {

    private BatchGenerator bg = new BatchGenerator();

    private ArrayList<String> plateBatchMaker(int first, int second) {
        return bg.generateBatch(first, second);
    }

    private DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ISO_LOCAL_TIME;

    //Start and end times hard coded for all three auctions
    //Unfortunately long due to the nature of trying to get just a time, and not a
data+time
    private String auction1Start = "09:00:00";
    private LocalDateTime auc1Start = LocalDateTime.parse(auction1Start,
dateTimeFormatter);
    private String auction1End = "09:30:00";
    private LocalDateTime auc1End = LocalDateTime.parse(auction1End, dateTimeFormatter);
    private String auction2Start = "12:00:00";
    private LocalDateTime auc2Start = LocalDateTime.parse(auction2Start,
dateTimeFormatter);
    private String auction2End = "12:30:00";

```

```

        private LocalDateTime auc2End = LocalDateTime.parse(auction2End, dateTimeFormatter);
        private String auction3Start = "15:00:00";
        private LocalDateTime auc3Start = LocalDateTime.parse(auction3Start,
dateTimeFormatter);
        private String auction3End = "15:30:00";
        private LocalDateTime auc3End = LocalDateTime.parse(auction3End, dateTimeFormatter);

        private Auction auction1 = new Auction(auc1Start, auc1End);
        private Auction auction2 = new Auction(auc2Start, auc2End);
        private Auction auction3 = new Auction(auc3Start, auc3End);

        private List<Auction> auctionList = Arrays.asList(auction1, auction2,
auction3);

        //Initializing all repositories needed
        private final PlatesRepository pr;
        private final BatchRepository br;
        private final AuctionRepository ar;

        @Autowired
        public DataFillerService (PlatesRepository pr, BatchRepository br,
AuctionRepository ar) {
            this.pr = pr;
            this.br = br;
            this.ar = ar;
        }

        //Method used to create and save data to the database
        @PostConstruct
        @Transactional
        public void fillData() {

            //Save the auctions
            ar.save(auction1);
            ar.save(auction2);
            ar.save(auction3);

            Batch batch;

            //Outer for loop to creates 10 batches and assign them with an auction
            for (int i = 0; i < 100; i++) {
                int first = i % 10;
                int second = i % 5;

                batch = new Batch(auctionList.get(new
Random().nextInt(auctionList.size())));
                br.save(batch);
                //Inner for loop to populate each batch with 5 plates
                for (String plate: plateBatchMaker(first, second)) {
                    pr.save(new Plate(plate, batch));
                }
            }
            //Plate and batch hardcoded for testing purposes, see cucumber tests
            batch = new Batch(auction3);
            br.save(batch);
            pr.save(new Plate("CV57 ORC",batch));
        }
    }
}
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@SpringBootApplication
@EnableTransactionManagement
public class LicensePlateAuctionApplication {

```

```

    public static void main(String[] args) {
        SpringApplication.run(LicensePlateAuctionApplication.class, args);
    }
}
package com.example.demo;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import javax.persistence.*;

/**
 * Class to create Plate object
 */
@Entity
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
@Table(name="plate")
public class Plate implements java.io.Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int plateid;
    private String platenumber;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "batchid_fk")
    private Batch batch;

    public Plate() {
    }

    public Plate(String platenumber) {
        this.platenumber = platenumber;
    }

    public Plate(String platenumber, Batch batch) {
        this.platenumber = platenumber;
        this.batch = batch;
    }

    public Batch getBatch() {
        return batch;
    }

    public void setBatch(Batch batch) {
        this.batch = batch;
    }

    public int getPlateid() {
        return plateid;
    }

    public void setPlateid(int plateid) {
        this.plateid = plateid;
    }

    public String getPlatenumber() {
        return platenumber;
    }

    public void setPlatenumber(String platenumber) {
        this.platenumber = platenumber;
    }
}
package com.example.demo;

public class PlateGenerator {
    private String reg = "";

```

```

    /**
     * Class to generate a random license plate with the correct letter number
     format
     * @return the randomly generated license plate
     */

    public String generatePlate (int first, int second){
        reg = "";
        //Randomly generates first two letters
        for (int i = 0; i < 2; i++) {
            char ch1 = (char) (Math.random() * 26 + 'A');
            reg = reg + String.valueOf(ch1);
        }

        //Used to specify certain numbers for certain batches
        reg = (reg + String.valueOf(first));
        reg = (reg + String.valueOf(second));

        reg = reg + " ";

        //Randomly generates last three letters
        for (int i = 0; i < 3; i++) {
            char ch2 = (char) (Math.random() * 26 + 'A');
            reg = (reg + String.valueOf(ch2));
        }
        return reg;
    }

    public PlateGenerator() {
        this.reg = "";
    }

    public String getReg() {
        return reg;
    }

    public void setReg(String reg) {
        this.reg = reg;
    }
}

package com.example.demo;

import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

/**
 * Repository interface for plates
 * Responsible for mapping java code to the database
 */
@RepositoryRestResource(collectionResourceRel = "plates", path = "plates")
public interface PlatesRepository extends PagingAndSortingRepository<Plate,
Integer> {
}

package com.example.demo;

import org.springframework.context.annotation.Configuration;
import org.springframework.data.rest.core.config.RepositoryRestConfiguration;
import org.springframework.data.rest.webmvc.config.RepositoryRestConfigurerAdapter;

/**
 * Class to expose id's of each object class
 */
@Configuration
public class RepositoryConfig extends RepositoryRestConfigurerAdapter {

    @Override

```

```

        public void configureRepositoryRestConfiguration(RepositoryRestConfiguration
config) {
            config.exposeIdsFor(Plate.class);
            config.exposeIdsFor(Batch.class);
            config.exposeIdsFor(Auction.class);
            config.exposeIdsFor(Bid.class);
            config.exposeIdsFor(Customer.class);
        }
    }
}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Bid</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
    <meta name="viewport" content="width=device-width, height=device-height,
initial-scale=1.0, shrink-to-fit=no"/>
</head>
<body>
<div class="jumbotron text-center" style="margin-bottom:0">
    <h1>Regucation</h1>
    <p>Bid on customised license plates!</p>
</div>

<nav class="navbar navbar-expand-sm bg-dark navbar-dark">
    <a class="navbar-brand" href="/">Home</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#collapsibleNavbar">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="collapsibleNavbar">
        <ul class="navbar-nav">
            <li class="nav-item">
                <a class="nav-link" href="/login">Login</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="/bid">Bid</a>
            </li>
        </ul>
    </div>
</nav>

<div class="container" style="margin-top:50px">
    <div>
        <h1 class="text-center">This is the bidding page, select the batch of your
dreams and try to win!</h1>
        <div class="row">
            <div class="card border-dark mb-3 text-center mx-auto" style="width:
18rem;">
                <div class="card-body">
                    <h5 class="card-title">Choose the batch to bid on from 1 to
100</h5>
                    <div class="input-group mb-3">
                        <input id="batchNumber" type="number" class="form-control"
aria-label="bidAmount" value="">
                        <div class="input-group-append">
                            <input id="batchSelectButton" class="btn btn-primary"
type="submit"
                            onclick="showPlates(), showAuction()">
                        </div>
                    </div>
                    <h2 id="batchPlates">
                    </h2>
                    <h2> AuctionID:

```

```

        </h2>
        <input id="auctionID" type="number" readonly>
    </div>
</div>
<div class="card border-dark mb-3 text-center mx-auto" style="width:
18rem;">
    <div class="card-body">
        <h5 class="card-title">Make your bid here!</h5>
        <p class="card-text">Click submit to preview your bid</p>
        <div class="input-group mb-3">
            <input id="bid" type="number" class="form-control" aria-
label="bidAmount" value="" min="0">
            <div class="input-group-append">
                <input id="bidButton" class="btn btn-primary"
type="submit"
                onclick="makeBid()">
            </div>
        </div>
    </div>
    <div class="card-body mx-auto">
        <h5 class="card-title">Your latest bid</h5>
        <p class="card-text">Click submit to confirm your bid</p>
        <div class="input-group mb-3">
            <input id="bidMade" type="number" class="form-control"
aria-label="bidAmount" value="" readonly>
            <div class="input-group-append">
                <input id="makeBidButton" class="btn btn-primary"
type="submit"
                onclick="saveBid()">
            </div>
        </div>
    </div>
</div>
<div>
    <p class="text-center">Batches with a '1' in them (eg, 1, 11, 21
etc) will have a number starting with 0. '2' will have a 1, '3' a 2 and so on.
    Use this to decide which batch you would like to bid on</p>
</div>
</div>
<div class="row">
</div>
</div>
<div class="jumbotron text-center" style="margin-bottom:0">
    <div class="footer-copyright text-center py-3">© 2019 Copyright:
        <a href="/"> Regucation.com</a>
        <a>Page last updated on 07/05/19 by Samuel Withers</a>
    </div>
</div>
<script src="jquery-3.4.0.min.js"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"
integrity="sha384-
U02eT0CpHqdSjQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDs4x0xIM+B07jRM"
crossorigin="anonymous"></script>
<script src="FrontEnd.js"></script>

</body>
</html>

```

```

//Displays example plates when page is loaded
$(document).ready(function () {
    $.ajax({
        type: "GET",
        url: "http://localhost:8080/plates",
        contentType: "application/json",
        dataType: "json"
    }).then(function (data) {
        for (var i = 0; i < 5; i++) {
            $('#GetPlates').append("<h2 class=\"text-center\"> " +
data._embedded.plates[i].platenumber + "</h2>");
        }
    });
});

//Used to show plates from a specific batch
function showPlates() {
    $.ajax({
        type: "GET",
        url: "http://localhost:8080/batch/" + $('#batchNumber').val() + "/plate",
        contentType: "application/json",
        dataType: "json"
    }).then(function (data) {
        for (var i = 0; i < data._embedded.plates.length; i++) {
            if (i === 0) {
                $('#batchPlates').text(data._embedded.plates[i].platenumber);
            } else {
                $('#batchPlates').append("<h2 class=\"text-center\"> " +
data._embedded.plates[i].platenumber + "</h2>");
            }
        }
    });
}

//Used to show the auction for a specific batch
function showAuction() {
    $.ajax({
        type: "GET",
        url: "http://localhost:8080/batch/" + $('#batchNumber').val() + "/auction",
        contentType: "application/json",
        dataType: "json"
    }).then(function (data) {
        $('#auctionID').val(data.auctionid);
    });
}

//Used to save latest bid to database
function saveBid() {
    $.ajax({
        type: "POST",
        url: "http://localhost:8080/bidsmade",
        data: {"bidAmount": $('#bidMade').val(), "auctionId":
$('#auctionID').val(), "batchId": $('#batchNumber').val()},
        success: function () {
            alert("Post Success");
        },
        error: function () {
            alert("The post failed");
        }
    });
}

//Used to save customer details to the database
function saveCustomer() {
    $.ajax({
        type: "POST",
        url: "http://localhost:8080/customer",
        data: {"email": $('#Email').val(), "password": $('#Password').val()},

```

```

        success: function () {
            alert("Post Success");
        },
        error: function () {
            alert("The post failed");
        }
    });
}

//Jquery wasn't working, did normal javascript instead
//Used to check bid is higher than last bid
function makeBid() {
    var x = parseInt(document.getElementById("bid").value);
    var y = parseInt(document.getElementById("bidMade").value);
    if (x < y + 1) {
        alert("Value cannot be less or same as current max bid");
        return false;
    }
    document.getElementById("bidMade").value =
document.getElementById("bid").value;
    document.getElementById("bid").value = "";
}

// function makeBid() {
//     var x = $('bid').val();
//     var y = $('bidmade').val();
//     if (x <= y + 1) {
//         alert("Value cannot be less or same as current max bid");
//         return false;
//     }
//     $("bidMade") = $("bid").value;
//     $("bid").value = "";
// }

//Logic for current time found at https://tecadmin.net/get-current-date-time-
javascript/
//Unused

// function countdownTimer(duration) {
//     var todayDate = new Date();
//     var currentTime = todayDate.getHours() + ":" + todayDate.getMinutes() + ":"
+ todayDate.getSeconds() + 30;
// }

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Home</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
    <meta name="viewport" content="width=device-width, height=device-height,
initial-scale=1.0, shrink-to-fit=no"/>
</head>
<body>
<div class="jumbotron text-center" style="margin-bottom:0">
    <h1>Regucation</h1>
    <p>Bid on customised license plates!</p>
</div>

<nav class="navbar navbar-expand-sm bg-dark navbar-dark">
    <a class="navbar-brand" href="/">Home</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-

```

```

target="#collapsibleNavbar">
  <span class="navbar-toggler-icon"></span>
</button>
<div class="collapse navbar-collapse" id="collapsibleNavbar">
  <ul class="navbar-nav">
    <li class="nav-item">
      <a class="nav-link" href="/login">Login</a>
    </li>
    <li class="nav-item">
      <a class="nav-link" href="/bid">Bid</a>
    </li>
  </ul>
</div>
</nav>

<div class="container" style="margin-top:50px">
  <div>

    <h2 class="text-center">Do you want to stand out from the crowd? Well this
is your chance! Regucation is your
    one stop shop for a specialised number plate, so sign up now!</h2>

    <a href="/login" class="btn btn-primary btn-lg btn-block" role="button"
aria-disabled="true">Login/Sign Up</a>

    <h3 class="text-center"> Here are some examples of our wonderful plates
</h3>

    <div id="GetPlates">
    </div>
  </div>
</div>

<div class="jumbotron text-center" style="margin-bottom:0">
  <div class="footer-copyright text-center py-3">© 2019 Copyright:
    <a href="/"> Regucation.com</a>
    <a>Page last updated on 07/05/19 by Samuel Withers</a>
  </div>
</div>

<script src="jquery-3.4.0.min.js"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"
integrity="sha384-
UO2eT0CpHqdsJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>
<script src="FrontEnd.js"></script>

</body>
</html>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Login</title>
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
  <meta name="viewport" content="width=device-width, height=device-height,
initial-scale=1.0, shrink-to-fit=no"/>

```

```

</head>
<body>
<div class="jumbotron text-center" style="margin-bottom:0">
  <h1>Regucation</h1>
  <p>Bid on customised license plates!</p>
</div>

<nav class="navbar navbar-expand-sm bg-dark navbar-dark">
  <a class="navbar-brand" href="/">Home</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#collapsibleNavbar">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="collapsibleNavbar">
    <ul class="navbar-nav">
      <li class="nav-item">
        <a class="nav-link" href="/login">Login</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="/bid">Bid</a>
      </li>
    </ul>
  </div>
</nav>

<div class="container" style="margin-top:50px">
  <h1>Sign up here</h1>
  <form>
    <div class="form-group">
      <label for="Email">Email address</label>
      <input type="email" class="form-control" id="Email" aria-
describedby="emailHelp"
        placeholder="Enter email">
      <small id="emailHelp" class="form-text text-muted">We'll never share
your email with anyone else, probably.</small>
    </div>
    <div class="form-group">
      <label for="password">Password</label>
      <input type="password" class="form-control" id="Password"
placeholder="Password">
      <small id="passwordHelp" class="form-text text-muted">Passwords are
stored in plain text just like the big companies, all for your peace of
mind.</small>
    </div>
    <button type="submit" class="btn btn-primary"
onclick="saveCustomer()">Submit</button>
  </form>
</div>

<div class="jumbotron text-center" style="margin-bottom:0">
  <div class="footer-copyright text-center py-3">© 2019 Copyright:
    <a href="/"> Regucation.com</a>
    <a>Page last updated on 07/05/19 by Samuel Withers</a>
  </div>
</div>

<script src="jquery-3.4.0.min.js"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"
integrity="sha384-
U02eT0CpHqdsJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEeFF/nJGzIxFdsf4x0xIM+B07jRM"

```

```
        crossorigin="anonymous"></script>
<script src="FrontEnd.js"></script>

</body>
</html>
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://localhost:5432/license
spring.datasource.username=postgres
spring.datasource.password=
spring.jpa.database-platform=postgres
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
spring.jpa.show-sql=false
spring.jpa.open-in-view=false
spring.resources.static-locations[0]=file:src/main/resources/static/
spring.resources.static-locations[1]=classpath:/static/
```