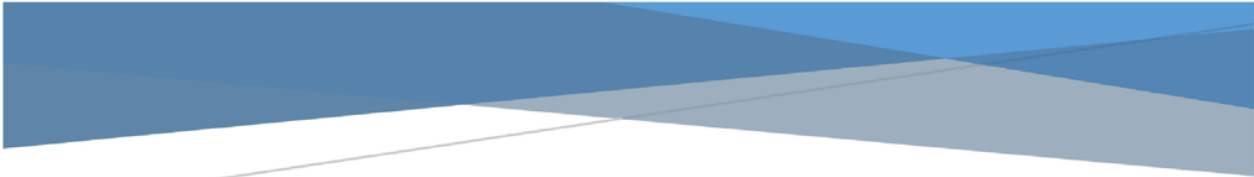


Final

by Anthony Njonge

FILE	12049_ANTHONY_NJONGE_FINAL_181292_1427619370.DOCX (1.52M)		
TIME SUBMITTED	08-MAY-2019 11:04 PM (UTC+0100)	WORD COUNT	6260
SUBMISSION ID	106702867	CHARACTER COUNT	43711



WEB SERVICES (ACCA7005) AND WEB DEVELOPMENT (ACCA7004)

REPORT STRUCTURE

PART I: RESTful AUCTION API WITH CLIENT SERVICE
PART II: RESTful BEST PRACTICE
**PART III: RELEVANCE OF SERVERLESS AND USE OF CIRCUIT
BREAKER PATTERN IN WEB DEVELOPMENT**

Anthony Njonge (1809703)

Contents

PART I	3
IMPLEMENTATION OF AUCTION API AND WEB UI	3
SUMMARY:	3
PART II	4
BEST PRACTICE FOR RESTful WEB SERVICES	4
INTRODUCTION:	4
RESTful BEST PRACTISE:	4
PART III	13
SERVERLESS	13
INTRODUCTION	13
ARCHITECTURE	14
ADVANTAGES	15
DISADVANTAGES	15
FUTURE WORK	16
CONCLUSION	16
CIRCUIT BREAKER PATTERN	16
WEB DESIGN PATTERNS	16
USE OF CIRCUIT BREAKER PATTERN	17
REFERENCES	19
APPENDIX	20
CODE	20

Figure 1 Richardson Maturity Model [5].....	5
Figure 2 Auction Site profile	5
Figure 3 Example of manipulation of resources	6
Figure 4 Error Catching.....	7
Figure 5 Easy to infer names	9
Figure 6 Connectedness example	11
Figure 7 Use of Parameters.....	12
Figure 8 Survey Trend of Cloud Computing [13]	13
Figure 9 Evolution of Technology [15]	14

PART I

IMPLEMENTATION OF AUCTION API AND WEB UI

SUMMARY:

The Auction API has been implemented using Spring Boot. The frontend is a Bootstrap-based client.

The application will be submitted in a zipped folder.

PART II

BEST PRACTICE FOR RESTful WEB SERVICES

INTRODUCTION:

Representational State Transfer(REST) is a kind of architectural style for distributed hypermedia system delivered through the web, the other being Simple Object Access Protocol(SOAP). In REST, the services (web services) typically software functions for tasks act on resources that are identified by the uniform resource locator(URL) address. RESTful web services are set apart from the rest of web use through the explicit use of HTTP standardised methods in invocation of the different operations [1].

REST-style architecture is made up of clients and servers. The client sends requests to the server which the server in turn interprets and return with an appropriate response back to the client. It is this nature of division of tasks that web services facilitate the creation of simpler smaller parts of the whole process without affecting the performance or delivery of the final system. Web services establish a layer of indirection that insulate clients on the how of fulfilling the requests [2].

RESTful BEST PRACTISE:

To consider the best practise for implementing RESTful web service it is important to start from work done by Fielding [3] who identified some constraints of what defines REST as

“REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.”

In addition to the named four, two other constraints are stateless and code on demand.

In QCon San Francisco 2008 conference [4], Richardson introduced the Richardson Maturity Model(RMM). Richardson shared his experiences of building enterprise RESTful web services and theorised that for a web service to be truly RESTful it needs a minimum of the following levels of Plain Old XML(POX), Resources, HTTP Verbs, and Hypermedia Controls.

The model stipulates that as the development of the RESTful web services increases in complexity and through the levels it should be possible to achieve maturity that is measurable. It is therefore arguable that in achieving RMM maturity, going through the levels should also result in best practise.

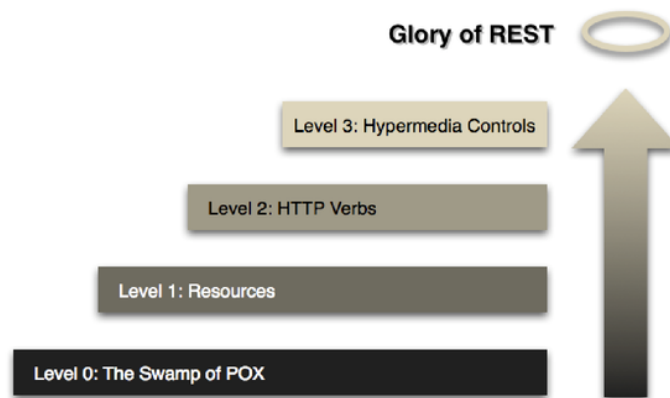


Figure 1 Richardson Maturity Model [5]

Identification

A RESTful web-service should be independent of what communication protocol is used in identifying it for its consumption. The uniform resource identifier (URI) used for identification should allow for any URI scheme to identify it. If a resource's URI changes then the server should be able to use hypermedia to redirect to the new URI [6]. Failure of identification is a red flag that identification and consumption are coupled.

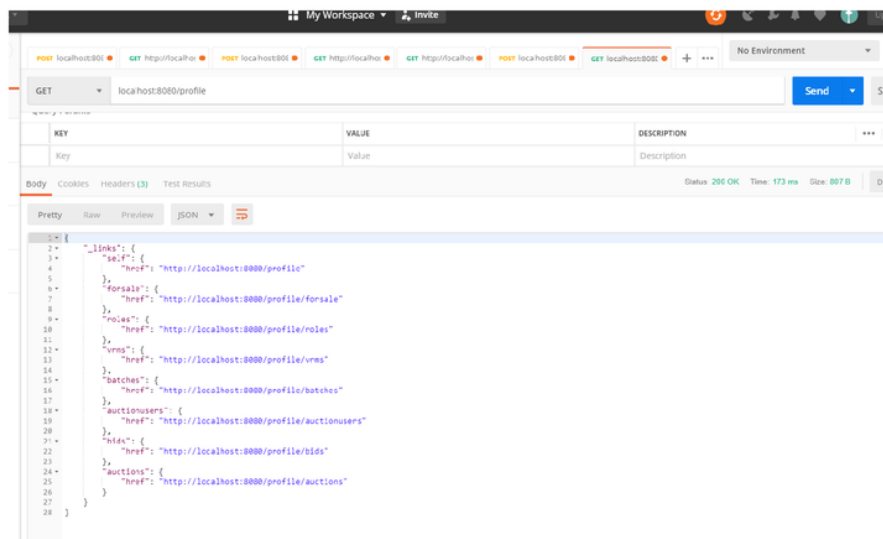


Figure 2 Auction Site profile

Manipulation of resources through representation

A resource can be objects or abstract representation of the objects. The representation is the instantaneous state of the resources. When the state is sent to the client, the state on the server can change without affecting future interactions. It

is the changing state of the interaction that enables the success of RESTful web-services. All these actions of manipulation should only be allowed if the client has express permission to do so.

In the designed system a user can be created in the system through the submission of a POST to /auctionusers.

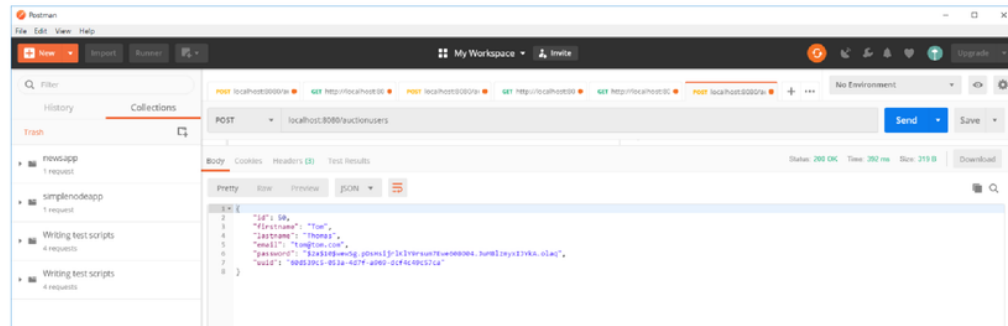


Figure 3 Example of manipulation of resources

Self-descriptive messages

A RESTful web-service should be clearly defined that the interaction to be self-descriptive, stateless and server-defined behaviour. All the information should be contained within the messages and responses during the interaction. If for example, the expected request body type is JSON, then the request header should identify the request body as containing the stated content type.

Hypermedia Constraint

This constraint is a result of meeting the requirements of the previous constraints. The application state is driven from the client side and any change is dictated by the responses received via a HTTP request. It is therefore only possible to change the application state from interactions with the RESTful web-service. If a change is evident from the server-side, the application state on the client can change accordingly.

Stateless

Stateless means that the state of the service to handle an incoming request is determined by the incoming request itself. This requires the stated URI, the body, any query parameters is contained in the request, and any headers available. The URI will give the location of the resource, while the body contains all the necessary state of the resource. After processing the request, the server should then be able to return an appropriate response back to the client dependent on the parameters received earlier on through the initial request. It is important to note that at this stage a status code is sent that corresponds to the HTTP status codes. For example, a successful response would be a 200 code or a 405 code would mean a "Method Not Allowed", or a 400 code would mean "Bad Request".

Cacheable

Cache allows for performance system improvements, allowing for the client to reuse a previous response if the state has not changed. It is therefore vital that to take advantage of this property that responses should include information about themselves and whether they can be cached. The cache information should allow for the client to identify whether the response being used needs a new request or not.

Error resolution

A restful web-service should be able to catch errors without failing. In the system implemented in Part I, this has been implemented on the duplicate plate, automation service when the service system is generating new plates.

```
@PostConstruct
@Transactional
public void addPlate() {

    Batch newBatch = new Batch();

    newBatch.setPrice(batchprice);
    batchRepository.save(newBatch);

    GenerateVrm generateVrm = new GenerateVrm();

    for( int i=0;i<batchnumber; i++){
        try{
            newBatch.setPrice(batchprice);
            batchRepository.save(newBatch);
            Vrm vrm = new Vrm();
            vrm.setBatch(newBatch);
            vrm.setVrmno(generateVrm.generateVrm());
            vrmRepository.save(vrm);
        } catch (Exception e) {
            Vrm vrm = new Vrm();
            vrm.setBatch(newBatch);
            vrm.setVrmno(generateVrm.generateVrm());
            vrmRepository.save(vrm);
        }
    }
}
```



Figure 4 Error Catching

No Versioning

If version is implemented in a RESTful webservices it adds to the difficulty of the consumption of the service. Version will create a requirement for the consumers of the services to always update the clients whenever a new version is released. Fielding states that "versioning an interface is just a polite way to kill deployed

applications [7]". A dead link to a service is therefore best avoided by not implementing any versioning during the deployment of the services. It is for this reason that no version has been implemented on the service.

Use of nouns for resource names

It is always advisable to first determine the resources in a RESTful webservices. This allows for planning to determine which resource will be exposed for use. Nouns should be used for the naming of the resources [8]. Use of nouns allows for the use of properties in relation to the services. For example in the auctioning system, some resources identified were each table of the database is a resource. Therefore, "auctionusers" representing the users is a resource, "vrmino" which represent the number plates is a resource. These resources allow for the CRUD actions of POST (create), GET(read), PUT & PATCH(update) and DELETE(delete). An example of the "vrmino" class which is an implementation of the vrmino is shown below:

```
@Data
@Entity
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
public class Vrm {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String vrmino;

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

    public Vrm() {}

    public String getVrmino() {
        return vrmino;
    }

    public void setVrmino(String vrmino) {
        this.vrmino = vrmino;
    }

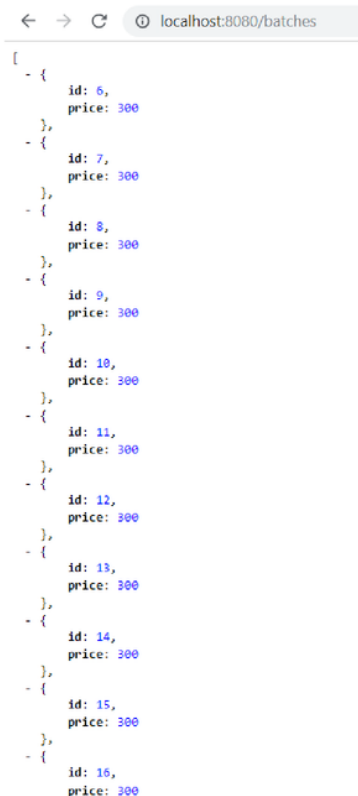
    public Batch getBatch() {
        return batch;
    }

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

    public Vrm(String vrmino, Batch batch) {
        this.vrmino = vrmino;
        this.batch = batch;
    }

    @Override
    public String toString() {
        return "Vrm{" +
            "id=" + id +
            ", vrmino='" + vrmino + '\'' +
            ", batch=" + batch +
            '}';
    }
}
```

Resource name should be concrete and domain-specific. This allows for the use of the system to be easily inferred. For example in the auctioning system to view "batches" it is possible to use the URI <http://localhost:8080/batches> to return the list of available batches.



```
localhost:8080/batches
[
  - {
    id: 6,
    price: 300
  },
  - {
    id: 7,
    price: 300
  },
  - {
    id: 8,
    price: 300
  },
  - {
    id: 9,
    price: 300
  },
  - {
    id: 10,
    price: 300
  },
  - {
    id: 11,
    price: 300
  },
  - {
    id: 12,
    price: 300
  },
  - {
    id: 13,
    price: 300
  },
  - {
    id: 14,
    price: 300
  },
  - {
    id: 15,
    price: 300
  },
  - {
    id: 16,
    price: 300
  }
]
```

Figure 5 Easy to infer names

Resources number restricted

The level of abstraction of the system will determine the complexity of the system. In the auction system, the number of resources is kept at a minimum to keep the overall system simple.

Consistency in resource naming

To avoid complexity, it is good practice to avoid a mixture of singular and plural names. The auction services developed makes use of only plural names.

Use of JSON

JSON – Javascript Object Notation is a widely used data format that is increasingly pervasive in the consumption of RESTful webservices. JSON is preferred to other data exchange formats due to it being easy, reliable and fast to use and implementation [9] [10]. JSON is used to encode nested data structures allowing for the ease of transfer and consumption.

Headers

Developing RESTful services with the ability of utilising accept and content-type headers ensures that future formats can be easily integrated. This can easily be done by the use of libraries rather than an extensive redevelopment and redeployment for the new format. In the developed service the use of Jackson with Hibernate support was used to marshal JSON request and responses.

```
// only the admin should be able to initiate a bid session
@Entity
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
public class Bid implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "userid")
    //private int userid;
    private Auctionuser userid;
```

Other headers are also present including Origin headers which are used to control the interactions from cross-origin interactions. Accept-Encoding headers allow a RESTful webservice to determine whether to use compressing data methods. Compressing data allows for the transfer of large data packets which therefore helps the performance of the interaction with reduced network traffic.

Allow Cross-origin resource sharing(CORS)

This follows closely with the best practise of use of headers. The setting of the Access-Control-Allow-Origin to "*" allows for the RESTful webservice to be available across different origins. It allows for the client request to access the HTTP methods. It is for this reason why having to set this setting to allow all requests is recommended for public RESTful webservices.

In the developed service, through the use of default Spring Boot configurations of @RequestMapping or the child configurations of @PostMapping or @GetMapping, the service by default allows CORS.

Connectedness

When a client accesses a webservice that has a response that contains a collection, it is important that the returned response is in very least give a point of reference as to the location of the current location and also provide pointers to other parts of the service available to the client. For a collection, this might involve the use of pagination and a link showing the current location of the link. An example from the developed service is accessing the link list of "vrms".

```
276 *      "vrm": {
277 *        "href": "http://localhost:8080/vrms/20"
278 *      },
279 *      "batch": {
280 *        "href": "http://localhost:8080/vrms/20/batch"
281 *      }
282 *    },
283 *    "_links": {
284 *      "first": {
285 *        "href": "http://localhost:8080/vrms?page=0&size=20"
286 *      },
287 *      "self": {
288 *        "href": "http://localhost:8080/vrms?page=1&size=20",
289 *        "templated": true
290 *      },
291 *      "next": {
292 *        "href": "http://localhost:8080/vrms?page=1&size=20"
293 *      },
294 *      "last": {
295 *        "href": "http://localhost:8080/vrms?page=44&size=20"
296 *      },
297 *      "profile": {
298 *        "href": "http://localhost:8080/profile/vrms"
299 *      },
300 *      "search": {
301 *        "href": "http://localhost:8080/vrms/search"
302 *      }
303 *    },
304 *    "page": {
305 *      "size": 20,
306 *      "totalElements": 894,
307 *      "totalPages": 45,
308 *      "number": 0
309 *    }
310 *  }
311 * }
312 * }
313 * }
```

Figure 6 Connectedness example

Date/Time

Date/Time should be taken into consideration since different formats, timezone etc have a different meaning dependent on the client and webservice. It is therefore advisable to use standardised time like using the ISO8601 for dates and times.

Use of Parameters

The available webservice should allow the use of parameters for extending the resource [11]. An example in the developed service is extending the vrms resource by searching for specific "batches" by batchid.

```

GET http://localhost:8080/vrms/search/findVrmByBatch?batchid=6

Pretty Raw Preview JSON

114      },
115      {
116      },
117      {
118          "vrmmo": "C070 AFG",
119          "_links": {
120              "self": {
121                  "href": "http://localhost:8080/vrms/9"
122              },
123              "vrm": {
124                  "href": "http://localhost:8080/vrms/9"
125              },
126              "batch": {
127                  "href": "http://localhost:8080/vrms/9/batch"
128              }
129          },
130      },
131      {
132          "vrmmo": "CL20 ABB",
133          "_links": {
134              "self": {
135                  "href": "http://localhost:8080/vrms/10"
136              },
137              "vrm": {
138                  "href": "http://localhost:8080/vrms/10"
139              },
140              "batch": {
141                  "href": "http://localhost:8080/vrms/10/batch"
142              }
143          },
144      },
145      {
146          "_links": {
147              "self": {
148                  "href": "http://localhost:8080/vrms/search/findVrmByBatch?batchid=6"
149              }
150          }
151      }

```

Figure 7 Use of Parameters

Application Security

Webservices resources can be secured in the same manner normal enterprise applications can be secured. The principles of security involved authorisation and authentication. Authorisation answers the question of "Does this user(Principal) have the requested permissions on the resources?" while authentication answers the question of "Does the credentials offered match credentials saved for use of the services?". Other RESTful security considerations should include, validation of any request payload, restrictions of verbs (GET, POST, PUT, DELETE) actions, and order of instructions received.

PART III SERVERLESS INTRODUCTION

Why serverless?

Recent surveys [12] [13] have shown that a majority of organisations have enterprise application or a section of their enterprise infrastructure already being delivered through the cloud and more are intending to follow in the near future. But why?

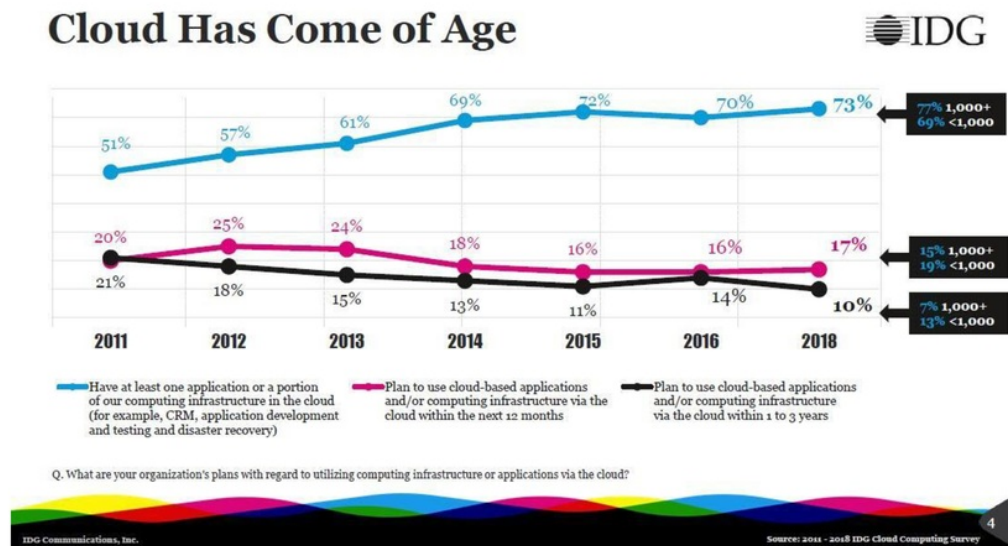


Figure 8 Survey Trend of Cloud Computing [14]

Serverless is a misnomer. Serverless does not mean having no servers are in use. Serverless architecture is also known as serverless computing is a new software design pattern that started to resolve the issues of infrastructure delivery.

Initially, any developer who delivered software to be deployed through the internet needed to also consider the implications of how to host and how to deliver the application for use. Developers had to plan for the hardware, software, bandwidth, infrastructure, location of where to base the hardware. Over time the demand for more and more applications to be "hosted" on the cloud in data centers in different location meant that the financial cost and time was proving to be a distraction to delivering cloud applications.

The services from traditional hosting companies such as RackSpace, Heroku, Microsoft, IBM, Amazon, and Google changed to start offering a service, Infrastructure as a Service(IaaS). IaaS has removed the need for developers to think about the provisioning of the infrastructure to more pertinent issues of system performance. Serverless has provided the advantages of pay-per-use billing, provision of autoscaling and abstraction of infrastructure.

ARCHITECTURE

Serverless architecture is all about building systems on the cloud. It mainly involved three services; Backends as a Service(BaaS) and Functions as a Service(FaaS) on top of the main service of IaaS.

The basic tenet is having not to be involved in managing the servers and having the providers (owners) of the hardware, take charge of providing a server for use; provision. The providers ensure that the servers uptime and updates of dependencies are managed without visibility to the end users. The focus is on renting space on servers rather than ownership.

Time of deployment of the service is also an important fact. As a given, provisioning of services is required to be fast in the range of minutes rather than the days or at least hours of configuration it used to take enterprises.

Serverless is an event-driven model where the application runs in containers that are started through the triggering of events. The application logic and any business logic is handled by functions running as a service. As more demand for the service appears, the system is able to increase (scale) by deploying more functions and vice versa in terms of low demand [15].

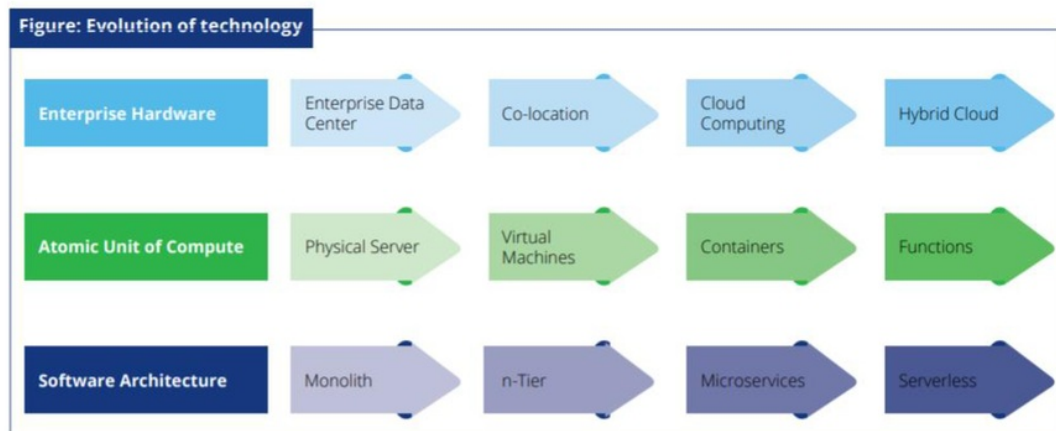


Figure 9 Evolution of Technology [15]

To be able to deploy application serverless the traditional stack has had to be transformed where the network, compute and database stack is transformed into an API gateway, FaaS, and BaaS.

API gateway which is programming that act as a gatekeeper to the application, acts as the single point of access to the application. It allows for the encapsulation of the application and allows for the working to be hidden to the clients. Examples of API gateway services include AWS API Gateway or Azure API Management. FaaS is the ability to provide custom server-side software that responds to event triggering. This layer runs the business logic. Examples of FaaS include AWS Lambda, Azure

Functions and Google Cloud Functions . BaaS is the provision of backends i.e. databases, messaging platforms, user management and other highly scalable data and logic components. Examples of Baas include AWS DynamoDB and Azure CosmosDB.

ADVANTAGES

Costs:

Before IaaS, the cost to provision hardware for use for internet facing applications involved the need for technical expertise in-house or the ability to hire the technical expertise. This expertise was required to build, configure, deploy and maintain the servers. In Serverless, all the time, effort and relative cost are handled by the providers. It has greatly reduced the actual cost of deploying application since enterprises are having to leave the task of provisioning physical hardware to third-parties.

Risk:

In the age of cloud computing, the expectation is that the systems are expected to be available 24/7. It is this expectation that any downtime could be disastrous for any application failure. With serverless computing, the use of applications such as Docker and Kubernetes it is possible for a system to be maintained in deliver by spinning up of new instances of the service automatically without intervention. All these systems allow for the risk of failure to be mitigated.

Scaling:

As the costs of delivering computing services decrease, it is possible to provision more servers in real-time without the worry of the cost. With the model of pay-per-use pricing, it allows for budgeting to be visible at the time of provisioning the services making it easier for scaling decisions to be made.

Lead time:

Serverless allows for agile delivery, continuous delivery and integration to work seamlessly with the deployed application as any new changes that require redeployment of the application require little lead time [16]. This allows for the delivery of more software and could be argued better software. The quick lead time allows for organisations and developers to be able to try ideas quickly and be allowed to fail fast.

DISADVANTAGES

State:

The management of state is tricky [16]. FaaS is stateless and therefore issues of persistency of data is critical. To overcome this disadvantage it requires the use of externalisation of state outside of the functions, which requires an extra service.

Execution duration:

FaaS functions are restricted on time of instruction time. If a function exceeds the set limits it always results in a failure of services. This can be detrimental to the delivery of services. To overcome this disadvantage and take advantage of the cheap cost of delivery, the developers are able to take advantage of the higher memory allocations available.

Cold starts:

Cold starts are the times when functions are invoked for the first time. During this initial process, the functions take considerable time to execute and will therefore manifest as delayed performance in the service. With services that are always being invoked this is less of a worry as the function are always readily available. For the less commonly used services, a heartbeat system could be used to keep invoking an unused function.

FUTURE WORK

Due to the nature of serverless computing, dependency on the vendors could lead to a problem of lock-in. It has, therefore, become obvious that there is need for streamlining the deployment of applications in an easier and quicker manner and if necessary from one vendor to the next. It is because of this need that offerings such as the Serverless Framework [17, 18] have appeared. More developments are expected in this field due to its early stage status.

CONCLUSION

Serverless computing is here to stay and adoption of it can only increase. With the new developments in the Internet of Things (IoT), Microservices, Big Data, integration of development operations(DevOps) to cloud operations, serverless computing offers all the benefit of agile computing where the delivery of working software is easier, quicker and cheaper.

CIRCUIT BREAKER PATTERN

WEB DESIGN PATTERNS

The definition of a design pattern as defined by the "gang of four" (GoF) is *"description of communicating objects and classes that are customized to solve a general design problem in a particular context [19]"*.

The properties of a design pattern are defined by a name, the description of the problem that it helps to tackle, the solution after the use of the pattern and the end result and trade-offs from its use.

Homer et.al [20] with the help of the developer community was able to summarise the most common problems areas during cloud application development. The areas

identified were availability, data management, design and implementation, messaging, management and monitoring, performance and scalability, resilience and security.

To mitigate these issues, it is helpful for developers to make use of design patterns.

USE OF CIRCUIT BREAKER PATTERN

Circuit breaker pattern is a design pattern used to improve stability and resilience of an application [20] by providing a means of handling faults. It is used to ensure that a failing service does not cascade to other services being offered elsewhere in the overall system. An example of a circuit breaker pattern is the use Netflix Hystrix.

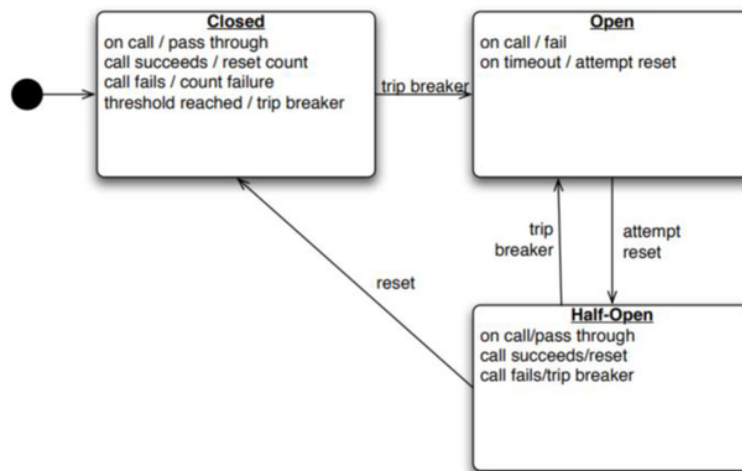


Figure 10 Circuit Breaker State Transition [21]

The states of the application are either Closed, Open or Half-Open [21]. The changes in the state of the circuit breaker will need to be logged which can then be referenced to check on the state of the application. This allows for the identification of issues that need to be resolved. On state closed the requests are allowed to pass through to an operation. A counter is maintained of how many failures occur during passing of the instructions to the operation. On open the instruction **is failed immediately**. On half-open a limited number of requests are allowed to pass through. If the feedback is that the operation was a success the state changes to closed and the counter reset.

The circuit breaker pattern is however different to the retry pattern. The retry pattern enables the application to keep retrying an operation with the aim that it will succeed while circuit breaker pattern prevents an application from performing an operation that has been logged as failing [20].

To implement the circuit breaker pattern, consideration should be given to exception handling, what types of exceptions and the logging of the exceptions. There needs to be a method of how recoverability of the service is activated. When a service fails testing of failed operations on the open state to change to half-open can be implemented through a pinging system rather than a timer methodology. The HTTP status code of 503 Service Unavailable.

To get the best outcome from the use of this pattern it is advisable to complement its use with the Retry Pattern and the Health Endpoint Monitoring Pattern with the latter returning information about the status of a service.

REFERENCES

- [1] R. S. Richardson L., RESTful Web Services, Sebastopol: O'Reilly Media, Inc., 2007.
- [2] R. Daigneau, Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services, Upper Saddle River: Pearson Education, Inc., 2012.
- [3] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures - Doctor of Philosophy Dissertation - UNIVERSITY OF CALIFORNIA, IRVINE," 2000. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf. [Accessed 16 April 2019].
- [4] Q. Conf, "Q Conference 2008," Q Confs, 2008. [Online]. Available: <https://qconsf.com/sf2008/sf2008/presentation/Introducing%2bReal-World%2bREST.html>.
- [5] M. Fowler, "Richardson Maturity Model," 18 March 2010. [Online]. Available: <https://martinfowler.com/articles/richardsonMaturityModel.html>. [Accessed 20 04 2019].
- [6] A. M. Richardson L., Restful Web APIs, Sebastopol: O'Reilly Media, Inc., 2013.
- [7] R. Fielding, "Scrambled Eggs," in *Evolve The Adobe Digital Marketing Community*, San Diego, 2018.
- [8] S. Vinoski, "RESTful Web Services Development Checklist," *IEEE INTERNET COMPUTING*, pp. 94-96, 11 2008.
- [9] V. Joshi, "Choosing JSON Over XML," Cloud Elements, Denver, 2017.
- [10] D. I. Zone, API Management: Comparative Views of Real-World Design, Durham: DZONE, 2018.
- [11] P. e. a. Giessler, "Best Practices for the Design of RESTful Web Services," in *The Tenth International Conference on Software Engineering Advances*, Barcelona, 2015.
- [12] IDG, "Understand Organizations' Cloud," 2018. [Online]. Available: <https://resources.idg.com/download/executive-summary/cloud-computing-2018>. [Accessed 20 04 2019].
- [13] L. Columbus, "State Of Enterprise Cloud Computing, 2018," Forbes, Jersey City, 2018.
- [14] IDG, "Understand Organizations' Cloud," IDG Communications, Inc, Framingham, 2018.
- [15] C. A. P. R. Barik R., "Serverless Computing – Architectural Considerations and Principles," Deloitte Touche Tohmatsu India LLP, Mumbai, 2018.
- [16] M. C. J. Roberts, What Is Serverless?, Sebastopol: O'Reilly Media, Inc, 2017.
- [17] P. Sbarski, Serverless Architectures on AWS, Shelter Island: Manning Publishers, 2017.
- [18] S. Framework, "Serverless Framework Documentation," 2019. [Online]. Available: <https://serverless.com/framework/docs/getting-started/>. [Accessed 01 05 2019].
- [19] R. H. R. J. J. V. E. Gamma, Design Patterns: Elements of Reusable Object Oriented Software, Reading: ADDISON-WESLEY PUBLISHING COMPANY, 1995.
- [20] A. Homer, Cloud Design Patterns - PRESCRIPTIVE ARCHITECTURE GUIDANCE FOR CLOUD APPLICATIONS, Seattle: Microsoft, 2014.
- [21] M. Nygard, Release It! Design and Deploy Production-Ready Software, Raleigh: The Pragmatic Bookshelf, 2007.

APPENDIX

CODE

Auction class

```
package com.webservicesandwebdev.auction.domain;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Data;

import javax.persistence.*;
import java.io.Serializable;
import java.time.LocalDate;

/**
 * The auction class for the list of auctions
 */
@Data
@Entity
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
public class Auction implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "price")
    private Bid price;

    private LocalDate date; //private LocalDate date;
    //private LocalTime timestart;
    //private LocalTime timeend;
    private boolean active;

    @OneToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "id")
    private Batch batchid;

    public void setBidprice(Bid bidprice) {
        this.price = bidprice;
    }

    public LocalDate getDate() {
        return date;
    }

    public void setDate(LocalDate date) {
        this.date = date;
    }

    // public LocalTime getTimestart() {
    //     return timestart;
    // }

    // public void setTimestart(LocalTime timestart) {
    //     this.timestart = timestart;
    // }

    // public LocalTime getTimeend() {
    //     return timeend;
    // }

    // public void setTimeend(LocalTime timeend) {
    //     this.timeend = timeend;
    // }

    public boolean isActive() {
        return active;
    }

    public void setActive(boolean active) {
        this.active = active;
    }

    public Batch getBatchid() {
        return batchid;
    }
}
```

```

    }

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

    // //to reformat the date into String
    // public void setTimestart(String auctiontimestart)
    // {
    //     setTimestart(LocalTime.parse(auctiontimestart));
    // }
    // //to reformat the date into String
    // public void setTimeend(String auctiontimeend)
    // {
    //     setTimeend(LocalTime.parse(auctiontimeend));
    // }

    public void setDate(String auctiondate)
    {
        setDate(LocalDate.parse(auctiondate));
    }

    public Auction(Bid bidprice, LocalDate date, boolean active, Batch batchid) { //LocalTime
        timestart, LocalTime timeend, LocalDate date,
        this.price = bidprice;
        this.date = date;
        // this.timestart = timestart;
        // this.timeend = timeend;
        this.active = active;
        this.batchid = batchid;
    }

    @Override
    public String toString() {
        return "Auction{" +
            "price=" + price +
            ", date=" + date +
            // ", timestart=" + timestart +
            // ", timeend=" + timeend +
            ", active=" + active +
            ", batchid=" + batchid +
            '}';
    }

    public Auction(){
    }
}

```

AuctionUser Class

```

package com.webservicesandwebdev.auction.domain;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Data;
import org.hibernate.validator.constraints.Length;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import javax.persistence.*;
import javax.validation.constraints.Email;
import javax.validation.constraints.NotEmpty;
import java.util.Optional;
import java.util.Set;
import java.util.UUID;

/*
This class defines the users of the service
*/
@Data
@Entity
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
public class Auctionuser implements java.io.Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    // @OneToOne(fetch = FetchType.LAZY)
    // @JoinColumn(name = "userid")

```

```

//private Bid bid;

private String firstname;

private String lastname;

private String email;

private String password;

private java.util.UUID uuid = java.util.UUID.randomUUID();
//private String role;

//private int active;
//@ManyToMany(cascade = CascadeType.ALL)
// @JoinTable(name = "user_role", joinColumns = @JoinColumn(name = "user_id"), inverseJoinColumns
= @JoinColumn(name = "role_id"))
// private Set<Role> roles;

public Auctionuser(){
}

public Auctionuser(String firstname, String lastname, String email, String password) {
    this.firstname = firstname;
    this.lastname = lastname;
    this.email = email;
    this.password = password;
    this.uuid = UUID.randomUUID();
}

public String getFirstname() {
    return firstname;
}

public void setFirstname(String firstname) {
    this.firstname = firstname;
}

public String getLastname() {
    return lastname;
}

public void setLastname(String lastname) {
    this.lastname = lastname;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
    String hashedPassword = passwordEncoder.encode(password);
    this.password = hashedPassword;
}

@Override
public String toString() {
    return "Auctionuser{" +
        "id=" + id +
        ", firstname='" + firstname + '\'' +
        ", lastname='" + lastname + '\'' +
        ", email='" + email + '\'' +
        ", password='" + password + '\'' +
        ", uuid=" + uuid +
        '}';
}
}

```

Vrm Class

```
package com.webservicesandwebdev.auction.domain;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Data;

import javax.persistence.*;

/*
VRM class
*/
@Data
@Entity
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
public class Vrm {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String vrmno;

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

    public Vrm() {}

    public String getVrmno() {
        return vrmno;
    }

    public void setVrmno(String vrmno) {
        this.vrmno = vrmno;
    }

    public Batch getBatch() {
        return batch;
    }

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

    public Vrm(String vrmno, Batch batch) {
        this.vrmno = vrmno;
        this.batch = batch;
    }

    @Override
    public String toString() {
        return "Vrm{" +
            "id=" + id +
            ", vrmno='" + vrmno + '\'' +
            ", batch=" + batch +
            '}';
    }
}
```

Bid Class

```
package com.webservicesandwebdev.auction.domain;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

import javax.persistence.*;
import java.io.Serializable;

/*
Bid class that defines the bids.
*/

// only the admin should be able to initiate a bid session
@Entity
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
public class Bid implements Serializable {
    @Id
```

```

@GeneratedValue(strategy = GenerationType.IDENTITY)
private int id;

@ManyToOne(fetch = FetchType.LAZY) //onetoone
@JoinColumn(name = "userid")
private Auctionuser userid;

@ManyToOne(fetch = FetchType.LAZY) //onetoone
@JoinColumn(name = "batchid")
private Batch batchid;

private Float price;

public Auctionuser getUserid() {
    return userid;
}

public void setUserid(Auctionuser userid) {
    this.userid = userid;
}

public Batch getBatchid() {
    return batchid;
}

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

public Float getPrice() {
    return price;
}

public void setPrice(Float price) {
    this.price = price;
}

public Bid(Auctionuser userid, Batch batchid, Float price) {
    this.userid = userid;
    this.batchid = batchid;
    this.price = price;
}

public Bid(){
}

@Override
public String toString() {
    return "Bid{" +
        "userid=" + userid +
        ", batchid=" + batchid +
        ", price=" + price +
        '}';
}
}

```

BatchView Class

```

package com.webservicesandwebdev.auction.domain;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonProperty;

import javax.persistence.*;
import javax.validation.constraints.NotNull;
import java.io.Serializable;

/*
This is the Batchview class that corresponds to the postgres view.
*/
@Entity
@Table(name = "batchview")
@JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
public class BatchView implements Serializable {

    //private static final long serialVersionUID = 1L;
    @Id
    @Column(name = "id")
    private Long vrmviewid;

```

```

        @NotNull
        @Column(name = "vrmmo", unique = true)
        private String vrmmo;

        @JsonProperty
        public Long getVrmviewid() {
            return vrmviewid;
        }

        @JsonProperty
        public String getVrmmo() {
            return vrmmo;
        }

        public BatchView(Long vrmviewid, @NotNull String vrmmo) {
            this.vrmviewid = vrmviewid;
            this.vrmmo = vrmmo;
        }

        public BatchView() {}
    }

```

CYMRU Generate class

```

package com.webservicesandwebdev.auction.model;

import java.io.Serializable;
import java.util.*;

/*
 * This class is called by the VRMServices class in generation of CYMRU plates
 */
public class GenerateVrm implements Serializable {

    int batchnumber = 10;

    List<String> city = Arrays.asList("CA", "CB", "CC", "CD", "CE", "CF", "CG", "CH", "CI", "CJ", "CK", "CL",
    "CM", "CN", "CO", "CP", "CR", "CS", "CT", "CU", "CV", "CW", "CX", "CY");

    List<String> memorytag = Arrays.asList("19", "69", "20", "70", "21", "71");

    List<String> alphabet =
    Arrays.asList("A", "B", "C", "D", "E", "F", "G", "H", "J", "K", "L", "M", "N", "O", "P", "R", "S", "T", "U", "V", "W", "X",
    "Y");

    String[] BadWords =
    {"CNT", "FCK", "BLW", "BXT", "FOF", "POP", "CCK", "SHT", "GER", "END", "OMY", "FER", "RTY", "WTF", "ASM", "NTS", "HAG",
    "NOB", "PLE", "NKY", "AGS", "LAH", "STK"};

    String[] ValueWords = {"HUW", "LEE", "TOM", "CMY", "OSP", "LUV"};

    String lastLetters(int batchnumber) {
        String letters = "";

        for (int i = 1; i < batchnumber+1; i++) {
            String c = alphabet.get((int) (i*Math.random()*6));

            letters += c;
        }

        return letters;
    }

    private boolean unwantedWord(String letters) {
        for (int i = 0; i < BadWords.length; i++) {
            if (letters.equals(BadWords[i])) {
                return true;
            }
        }

        return false;
    }

    int valplate = 0;
    private int valuedWord(String letters) {
        for (int i = 0; i < ValueWords.length; i++) {
            if (letters.equals(ValueWords[i])) {
                return valplate++;
            }
        }
    }

```

```

    }
    }
    return valplate;
}

int addedVal = valplate *100;

/*
 * Choose the city and memorytag for the plates
 */
String choosencity = city.get((int)(Math.random() * 24));
String choosenmemorytag = memorytag.get((int)(Math.random() * 6));

String generateVrm() {

    String licensePlate;
    String letters;
    do {
        letters = lastLetters(3);
    } while (unwantedWord(letters));

    licensePlate = choosencity + choosenmemorytag + " " + letters;
    return licensePlate;
}

public GenerateVrm() {
    this.valplate = addedVal;
}
}

```

VRM Service

```

package com.webservicesandwebdev.auction.model;

import com.webservicesandwebdev.auction.domain.Batch;
import com.webservicesandwebdev.auction.domain.Vrm;
import com.webservicesandwebdev.auction.repositories.AuctionRepository;
import com.webservicesandwebdev.auction.repositories.BatchRepository;
import com.webservicesandwebdev.auction.repositories.VrmRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.annotation.PostConstruct;
import javax.transaction.Transactional;

/*
 * This service generates the CYMRU plates.
 * This service can be extended to UK plates with the input of city tags from the cities entity in the
 * db.
 */
@Service
public class VrmService {

    private VrmRepository vrmRepository;

    private BatchRepository batchRepository;

    @Autowired
    public VrmService(VrmRepository vrmRepository, BatchRepository batchRepository) {
        this.vrmRepository = vrmRepository;
        this.batchRepository = batchRepository;
    }

    int batchnumber = 10;
    float batchprice = batchnumber * 30;

    @PostConstruct
    @Transactional
    public void addPlate(){

        Batch newBatch = new Batch();

        newBatch.setPrice(batchprice);
        batchRepository.save(newBatch);

        GenerateVrm generateVrm = new GenerateVrm();
    }
}

```

```

        for( int i=0;i<batchnumber; i++){

            try{
                newBatch.setPrice(batchprice);
                batchRepository.save(newBatch);
                Vrm vrm = new Vrm();
                vrm.setBatch(newBatch);
                vrm.setVrmno(generateVrm.generateVrm());
                vrmRepository.save(vrm);
            } catch(Exception e){
                Vrm vrm = new Vrm();
                vrm.setBatch(newBatch);
                vrm.setVrmno(generateVrm.generateVrm());
                vrmRepository.save(vrm);
            }

        }
    }
}

```

CONTROLLERS

Auction

```

package com.webservicesandwebdev.auction.controller;

import com.webservicesandwebdev.auction.domain.Auction;
import com.webservicesandwebdev.auction.model.*;
import com.webservicesandwebdev.auction.repositories.AuctionRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

/*
 * Auction controller
 */
@RestController
public class Auctioncontroller {

    @Autowired
    private AuctionRepository repository;

    Auctioncontroller (AuctionRepository repository){
        this.repository = repository;
    }

    AuctionList auctionList = AuctionList.getInstance();

    public Auctioncontroller(){
        auctionList = AuctionList.getInstance();
    }

    @GetMapping("/auctions/{id}")
    public Auction auction(@PathVariable int id){
        return auctionList.GetAuction(id);
    }

    @GetMapping("/auctions")
    public List<Auction> allAuctions(){
        return auctionList.GetAllAuction();
    }

    @PostMapping(path = "/auctions")
    AuctionList auctionList(@RequestBody Auction auction){
        auctionList.AddAuction(auction);
        return auctionList;
    }

}

```

Batch

```

package com.webservicesandwebdev.auction.controller;

import com.webservicesandwebdev.auction.repositories.BatchRepository;
import com.webservicesandwebdev.auction.domain.Batch;
import com.webservicesandwebdev.auction.model.BatchList;

```

```

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

import java.util.List;
import java.util.Optional;

/*
Batch controller
*/
@RestController
public class Batchcontroller {
    private BatchRepository batchRepository;

    BatchList batchList = BatchList.getInstance();

    public Batchcontroller(BatchRepository batchRepository){
        this.batchRepository = batchRepository;
    }

    @GetMapping("/batches/{id}")
    public Optional<Batch> batch(@PathVariable int id){
        return batchRepository.findById(id);
    }

    @GetMapping("/batches")
    public Iterable<Batch> allBatchList(){
        return batchRepository.findAll();
    }
}

}

Bid Controller
package com.webservicesandwebdev.auction.controller;

import com.webservicesandwebdev.auction.domain.Bid;
import com.webservicesandwebdev.auction.model.BidList;
import com.webservicesandwebdev.auction.repositories.BidRepository;
import org.springframework.web.bind.annotation.*;

import java.util.Collection;

/*
Bid Controller
*/
@RestController
public class Bidcontroller {
    private BidRepository repository;

    BidList bidList = BidList.getInstance();

    public Bidcontroller() {
        bidList = BidList.getInstance();
    }

    Bidcontroller (BidRepository repository){
        this.repository = repository;
    }

    @RequestMapping("/bids/{id}")
    public Bid bid(@PathVariable int id) throws Exception{
        return bidList.GetBid(id);
    }

    @RequestMapping("/bids")
    public Collection<Bid> allBidsList() throws Exception{
        //return bidList.GetBidList();
        return repository.findAll();
    }

    @PostMapping(path = "/bids")
    BidList bidList(@RequestBody Bid bid){
        bidList.AddBid(bid);
        return bidList;
    }
}

}

```

User Controller

```
package com.webservicesandwebdev.auction.controller;

import com.webservicesandwebdev.auction.model.UserList;
import com.webservicesandwebdev.auction.domain.Auctionuser;
import com.webservicesandwebdev.auction.repositories.AuctionUserRepository;
import org.springframework.ui.ModelMap;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.List;
import java.util.Optional;

/*
This is the user controller class
*/
@RestController
public class Usercontroller
{
    private AuctionUserRepository repository;

    UserList users = UserList.getInstance();

    UserController(AuctionUserRepository repository) {
        this.repository = repository;
    }

    @GetMapping("/auctionusers/{id}")

    public Optional<Auctionuser> auctionUserRepository(@PathVariable int id){
        return repository.findById(id);
    }

    @GetMapping("/auctionusers")

    public List<Auctionuser> auctionUserRepository(){
        return repository.findAll();
    }

    @PostMapping(path= "/auctionusers", consumes = "application/json", produces =
"application/json")
    Auctionuser newUser(@RequestBody Auctionuser userUser) throws Exception {
        try {
            return repository.save(userUser);
        } catch (Exception e){
            return userUser;
        }
    }

    @RequestMapping(value = "/addUser", method = RequestMethod.POST)
    public String submit(@Valid @ModelAttribute("auctionuser") Auctionuser auctionuser,
        BindingResult result, ModelMap model) {
        if (result.hasErrors()) {
            return "error";
        }
        model.addAttribute("firstname", auctionuser.getFirstname());
        model.addAttribute("lastname", auctionuser.getLastname());
        model.addAttribute("password", auctionuser.getPassword());

        return "userView";
    }
}
```

VRM Controller

```
package com.webservicesandwebdev.auction.controller;

import com.webservicesandwebdev.auction.domain.Auctionuser;
import com.webservicesandwebdev.auction.repositories.BatchViewRepository;
import com.webservicesandwebdev.auction.repositories.VrmRepository;
import com.webservicesandwebdev.auction.domain.Vrm;
import com.webservicesandwebdev.auction.services.VrmServicesView;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.Collection;
import java.util.Collections;
```

```

import java.util.Optional;

/*
The VRM controller
*/
@RestController
public class Vrmcontroller {

    @Autowired
    private VrmRepository vrmRepository;

    @Autowired
    private BatchViewRepository batchViewRepository;

}

```

REPOSITORIES

Auction

```

package com.webservicesandwebdev.auction.repositories;

import com.webservicesandwebdev.auction.domain.Auction;
import com.webservicesandwebdev.auction.domain.Auctionuser;
import com.webservicesandwebdev.auction.domain.Vrm;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.data.rest.core.annotation.RestResource;

import java.util.List;

/*
This is the Auction Repository
*/
@RepositoryRestResource(collectionResourceRel = "auctions", path = "auctions")
public interface AuctionRepository extends JpaRepository<Auction, Integer> {

    @Override
    @RestResource(exported = false)
    void deleteById(Integer id);

    @Override
    @RestResource(exported = false)
    void delete(Auction entity);

    @Override
    @RestResource(exported = false)
    void deleteAll(Iterable<? extends Auction> entities);

    @Override
    @RestResource(exported = false)
    void deleteAll();

    /*
    this gives a list of all active auctions
    */
    @Query(value = "select * from auction where active = 'true'", nativeQuery = true)
    List<Auction> findByActive(@Param("active") boolean active);

    /*
    this gives the auction by batch id
    */
    @Query(value = "select * from auction where batchid = ?1", nativeQuery = true)
    List<Auction> findAuctionsByBatchid(@Param("batchid") int batchid);

}

```

AuctionUser

```

package com.webservicesandwebdev.auction.repositories;

import com.webservicesandwebdev.auction.domain.Auctionuser;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

```

```

import org.springframework.data.rest.core.annotation.RestResource;

import java.util.Collection;

/*
This repository return interacts with the auction user entity
*/
@RepositoryRestResource(collectionResourceRel = "auctionusers", path = "auctionusers")
public interface AuctionUserRepository extends JpaRepository<Auctionuser, Integer> {
    @Override
    @RestResource(exported = false)
    void deleteById(Integer id);

    @Override
    @RestResource(exported = false)
    void delete(Auctionuser entity);

    @Override
    @RestResource(exported = false)
    void deleteAll(Iterable<? extends Auctionuser> entities);

    @Override
    @RestResource(exported = false)
    void deleteAll();

    Auctionuser findByEmail(String email);
}

```

BATCH

```

package com.webservicesandwebdev.auction.repositories;

import com.webservicesandwebdev.auction.domain.Batch;
import com.webservicesandwebdev.auction.domain.Bid;
import com.webservicesandwebdev.auction.domain.Vrm;
import org.hibernate.Session;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.data.rest.core.annotation.RestResource;

import java.util.Collection;
import java.util.List;

/*
BatchRepository interacts with the Batch entity
*/

@RepositoryRestResource(collectionResourceRel = "batches", path = "batches")
public interface BatchRepository extends JpaRepository<Batch,Integer> {
    @Override
    @RestResource(exported = false)
    void deleteById(Integer id);

    @Override
    @RestResource(exported = false)
    void delete(Batch entity);

    @Override
    @RestResource(exported = false)
    void deleteAll(Iterable<? extends Batch> entities);

    @Override
    @RestResource(exported = false)
    void deleteAll();

    // @Query(value = "select * from batch", nativeQuery = true)
    // List<Bid> findAllById();
}

```

BATCHVIEW

```

package com.webservicesandwebdev.auction.repositories;

import com.webservicesandwebdev.auction.domain.BatchView;
import org.springframework.data.jpa.repository.JpaRepository;

```

```

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

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.Optional;
/*
Batchview repository interacts with the nonJPA batch entity
*/

@RepositoryRestResource(collectionResourceRel = "forsale", path = "forsale")
public interface BatchViewRepository extends JpaRepository<BatchView, Long> {

}

```

BID

```

package com.webservicesandwebdev.auction.repositories;

import com.webservicesandwebdev.auction.domain.Auction;
import com.webservicesandwebdev.auction.domain.Bid;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

import java.util.List;

/*
BidRepository for the bid repository
*/
@RepositoryRestResource(collectionResourceRel = "bids", path = "bids")
public interface BidRepository extends JpaRepository<Bid, Integer> {

    /*
    this gives the bids for a batch id
    */
    @Query(value = "select * from bid where batchid = ?1", nativeQuery = true)
    List<Bid> findAllByPrice(@Param("batchid") int batchid);

    /*
    this too gives the bids for a batch id
    */
    @Query(value = "select * from bid where batchid = ?1", nativeQuery = true)
    List<Bid> findBidByBatchid(@Param("batchid") int batchid);

}

```

VRM

```

package com.webservicesandwebdev.auction.repositories;

import com.webservicesandwebdev.auction.domain.Vrm;
import org.springframework.data.gemfire.repository.query.annotation.Trace;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.data.rest.core.annotation.RestResource;

import java.util.Collection;
import java.util.List;

/*
The VRM repository for the VRM entity
*/
@RepositoryRestResource(collectionResourceRel = "vrms", path = "vrms")
public interface VrmRepository extends JpaRepository<Vrm, Integer> {

    @Override
    @RestResource(exported = false)
    void deleteById(Integer id);

    @Override
    @RestResource(exported = false)
    void delete(Vrm entity);

}

```

```
    @Override
    @RestResource(exported = false)
    void deleteAll(Iterable<? extends Vrm> entities);

    @Override
    @RestResource(exported = false)
    void deleteAll();

    @Query(value = "select * from vrm where batchid = ?1", nativeQuery = true)
    List<Vrm> findVrmByBatch(@Param("batchid") int batchid);
}
```

Final

GRADEMARK REPORT

FINAL GRADE

74/100

GENERAL COMMENTS

Instructor

Part 1(50%): 30

Part 2 (25%): 23

Part 3 (25%): 21

Part 1

Automatic generation of number plates. Checks bad and high value words, adds £100 for a high value plate on last 3 characters. Register, but no login system, though I suspect this would have come next give more time. A good effort overall.

Part 2

Very good description but I would like to have seen you include some reference to design patterns.

Part 3

I wanted a little more on how FaaS and the CB pattern works.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18



Comment 1

...and what happens when it fails?

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34
