

Jiayuan Yan 1904780

by Jiayuan Yan

FILE	17866_JIAYUAN_YAN_JIAYUAN_YAN_1904780_254686_183521861.DOCX (60.71K)		
TIME SUBMITTED	15-MAY-2020 04:35AM (UTC+0100)	WORD COUNT	3747
SUBMISSION ID	127698745	CHARACTER COUNT	21075

Web Services
Web Development
Assignment

Guide teacher: Dr Tim Bashford

Dr Kemi Ademoye

Dr Michael Dacey

<Jiayuan Yan>

<1904780>

Contents

1. “No server ” technology and network hosting.....	3
1. “No server ” technology	3
2. The connection between the “No server” technology and the network hosting.....	5
2. The circuit breaker pattern.....	7
1. The circuit breaker pattern	7
2. Circuit breaker mode in web development	10
3. RESTful web services.....	11
4. References	21

1. “No server” technologies and Web hosting.

1. “No server” technologies

"No server" technology does not actually require no servers. It doesn't need a server. The underlying server management is done by cloud service providers such as AWS, Azure, or Google." Serverless technology is a new way to host applications on an infrastructure that does not require end-user management, and it is the next step in the evolution of infrastructure services. It separates the underlying infrastructure from the developers, essentially virtualizes the run and operations management, commonly known as FaaS(function as a service), and the serverless architecture allows you to perform a given task without worrying about the server, virtual machine, or underlying computing resources.

Serverless technology can develop rapidly. The most iconic event is AWS Lambda service launched by amazon in 2014, which makes serverless technology become the mainstream trend. Serverless technology has been described as a powerful way of development, in which users replace virtual machines with computing power only when they need it, and can be removed immediately after use^[2].

For example, if a company needs to expand the server capacity within the enterprise, it can enter the event configuration through API request or file upload and trigger the serverless function. Once the event is complete, the server will be idle, waiting for the next action^[5].

Serverless technology has the following advantages:

First, high agility. Because developers do not need to deploy, manage, or extend servers when using them, they can forgo infrastructure management. This greatly reduces operational overhead. In addition, serverless is highly compatible with the microservices architecture, which provides significant agility benefits.

Second, scalability. No server upgrades and added computing resources are dependent on the DevOps team. Applications without a server can scale quickly and seamlessly to accommodate peak traffic. Conversely, these applications automatically shrink as the number of concurrent users decreases.

The third, different billing model. You only need to pay for computing resources when using a serverless platform. The serverless architecture introduces a true pay-per-view model, where customers pay only when they perform a function. The serverless billing model makes it an ideal choice for microservers with small load requirements and applications with "spikey" traffic patterns. Unlike traditional environments, there is no need to pay for virtual machines or containers that are often idle.

The fourth, Safety. The serverless architectural pattern provides security for users. Because it no longer manages servers, DDoS attacks are much less threatening, and automatic scaling without server functionality helps reduce the risk of such attacks.

Serverless computing is not just a niche solution used by tech companies. Has completely changed the way developers deploy and manage complex software, with a huge impact on how businesses deliver applications. The user group that will benefit the most is the application field of the Internet of things, which can realize the simultaneous use of computing resources by billions of terminal devices.

2. The connection between the “No server” technology and the Web hosting

1. Web hosting

Web hosting is a service that allows organizations and individuals to publish Web sites or Web pages on the Internet. Web hosting or Web hosting service providers are businesses that provide the technology and services needed to view Web sites or Web pages on the Internet. Web sites are hosted or stored on special computers called servers. When Internet users want to view a client's website, all they have to do is enter the client's website address or domain name into the browser. Their computers will then connect to the client's servers, and the client's web pages will be sent to them via a browser.

Web hosting is essentially renting a customer's web site to someone else's computer so you don't have to worry about all these technical and maintenance details.

Basically, customers can provide hosting plans for their own websites, tell hosting companies how much space they need and pay for it. Once the client's hosting account is in place, all files from the site can be uploaded to the client's web hosting computer.

This way, the customer does not have to maintain the computer all the time or worry about the technical aspects of managing the Web server, because the Web hosting company will do this on behalf of the customer. All the customer needs to do is pay an annual or monthly fee, and the Web hosting company will maintain all the work of the Web server^[6].

2.The connection between the “Serverless” technology and the Web hosting :

First, security factors. Compared with serverless technology, choosing network hosting may mean a greater risk. Since serverless technology no longer manages servers, DDoS attacks are much less threatening, and automatic expansion of serverless functions helps reduce the risk of such attacks. Web hosting, however, can lead to the theft or disclosure of customers' website information.

Second, efficiency. Compared with network hosting, serverless technology can bring more efficiency. Although network hosting does not require Web maintenance by customers themselves, it often leads to higher costs.

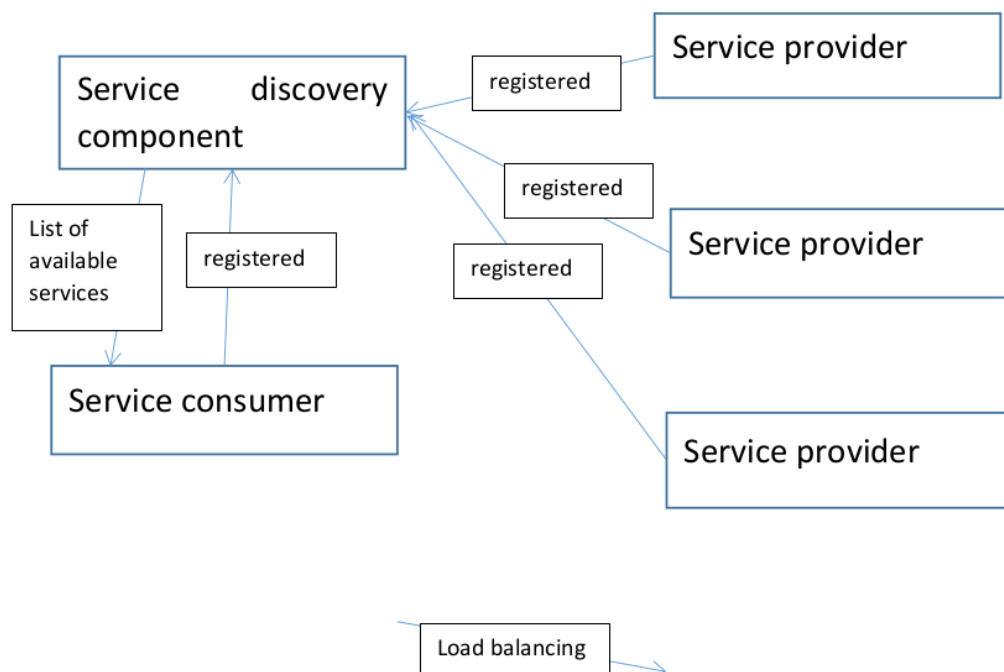
2. The circuit breaker pattern

1. The circuit breaker pattern

Circuit breaker is a circuit of the components, the role is when the circuit appears a short circuit or instantaneous current is too large and other problems, the circuit breaker directly trip, when the circuit repair the problem, must manually switch the circuit breaker to the state of connection, the whole circuit to restore normal power^[1].

Simulating the application of circuit breaker in the circuit, the circuit breaker mode in cloud computing comes into being.

As shown in the figure below:



Assume that the service provider response is very slow, so the consumer to the provider's request thread will be wait for, know the service returns, under the scenarios of high concurrent load, if don't do any processing, this kind of problem is likely to cause all threads processing user requests the depletion of resources, and cannot respond to user request further.

If the service consumer is also a provider of another service, there may be a cascading reaction that makes other next level service consumers unavailable, resulting in an avalanche effect.

Circuit breaker mode came into being:

Similar to the electrical short circuit, trip. If the current is too large, the circuit will be cut off automatically to avoid heating up the circuit, burning the circuit or electrical appliances.

When a dependent service has a large number of timeouts, it does not make much sense to allow new requests to be accessed, but to consume existing resources unnecessarily. For example, if we set the timeout time to 1 second, if a large number of requests are not answered within 1 second in a short period of time, it usually means an exception. At this point there is no need for more requests to access the dependency, and we should use circuit breakers to avoid wasting resources.

Circuit breaker can realize rapid failure, if over a period of time to detect many similar mistake, will force the multiple calls later fail fast, no longer request depends on service, so as to prevent the application execution may fail in the attempt to keep operating, so that the application can continue without having to wait for correct mistakes. The breaker mode

also enables the application to diagnose whether the error has been fixed, and if so, the application will attempt to invoke the operation again.

The breaker mode is similar to a proxy for operations that are prone to errors. This proxy can log several errors in the most recent call and then decide to continue with the allow operation or simply return an error.

Advanced circuit breakers have at least three states:

Closed: circuit breaker in the default case is closed, and the circuit breaker itself "with" counting function, when the error occurs once, counter will also carry out the action of "accumulation", to a certain number of errors occur circuit breaker will be "open", this time will also enable a timer internally, once time will switch into a state of half open.

Open: in the open state any request is "directly" rejected and an exception is thrown.

Half-open: the circuit breaker allows partial requests in this state, and if these requests are successful, it means that the error is no longer present and is "switched back" to the off state and the count is "reset". If any error occurs in the request, it is returned to the "on" state and the timer is retimed, giving the system a break.

2. Circuit breaker mode in web development

Circuit breaker mode for use:

Let's say an application is communicating with a database hundreds of times per second, and the database suddenly has an error, and the programmer doesn't want to keep accessing the database at the time of the error. The error is therefore handled directly before waiting for the TCP connection to expire and into the normal terminator (rather than the direct terminator). In simple terms, the circuit breaker detects errors and "prevents" the application from repeatedly calling a service that is nearly unresponsive (unless the service is secure enough to retry the connection)^[3].

In web development, the design mode of using circuit breakers usually needs to pay attention to the following points:

1. A circuit breaker should record all failed (and possibly successful) requests. So that the administrator can monitor it encapsulates the health of the operation.
2. In the open state, rather than simply failing very quickly, the circuit breaker is also able to record specific information about each request to the journal and schedule these requests when. Remote resources or services become available for playback.
3. The "HTTP503 service is not available" defined by the HTTP protocol. It can be a response that is returned on a specific Webserver if the requested service is currently unavailable. This response can contain additional information, such as the expected duration of the delay.

4. The circuit breaker should be configured to match the operation of the protective possible recovery mode^[4].

3. RESTful web services

REST itself is an architectural style designed for distributed hypermedia systems, not a standard. In a RESTful system, the server exposes the resource with a URI, and the client USES four Http predicates to access the resource. Because the client receives the resource, they are placed in a certain state. When they access a new resource, usually by clicking on the next connection, they change, or transition, their state. For work, REST assumes that resources can be represented using a common standard syntax^[6].

Traditional Web applications are mostly B/S architectures

The B/S architecture has the advantage of being easy to deploy, but it is not ideal in terms of user experience. To improve this situation, you can introduce REST

REST adds three new specifications to the existing architecture: unified interfaces, layered systems, and on-demand code.

1. Unified interface: the core feature of the REST architectural style is to emphasize that there is a unified interface between components, which is represented in the REST world, where everything on the network is abstracted into resources, and REST is to operate on resources through a common linker interface. The advantage of this design is to ensure that the services provided by the system are decoupled, greatly simplifying the system, thus improving the interaction and reusability of the system.

In addition, REST optimizes for common Web situations so that REST interfaces are designed to efficiently transfer large amounts of hypermedia data, which leads to the fact that REST interfaces are not optimal for other architectures^[7].

2. Layered system: the addition of layered system rules improves the independence between layers, sets boundaries for the complexity of the entire system, and insulates new servers from legacy clients by encapsulating legacy services, thus improving the scalability of the system.

3. Code on demand: REST allows you to extend client functionality. For example, extend client functionality by downloading and executing code in the form of an applet or a script. But while this improves system scalability, it also reduces visibility. So it's just an optional constraint on REST.

REST describes an architectural style for designing Web applications. It is a set of architectural constraints and principles that an application or design that meets these constraints and principles is RESTful. RESTful Web Services are what we call RESTful Web Services.

REST presents the following design guidelines :

Resources are specified by uris: in a Web application, everything should have a unique ID, and the unifying concept of an ID is a URI. A URI constitutes a global namespace, and using a URI to identify your key resources means they get a unique, global ID.

Use HTTP methods explicitly: REST requires developers to use HTTP methods explicitly and in a manner consistent with the protocol definition. This basic REST design principle establishes a one-to-one mapping between create, read, update, and delete (create, read, update,

and delete, CRUD) operations and HTTP methods. According to this mapping:

To create a resource on the server, use the POST method.

To retrieve a resource, use the GET method.

To change the state of a resource or to update it, use the PUT method.

To DELETE a resource, use the DELETE method.

Multiple representations of resources: multiple representations of resources for different needs. The multiple representations described here include XML, JSON, HTML, and so on. That is, the server side needs to provide external resource representation in a variety of formats for different clients to use. For example, a mobile application can use XML or JSON to communicate with the server, while a browser can understand HTML.

Stateless: requests on the server side should be stateless, and complete, standalone requests do not require the server to retrieve any type of application context or state while processing the request. Stateless constraints make server changes invisible to the client because the client is not dependent on the same server in two consecutive requests. A client from a server receives a document contains links, when it wants to do some processing, this service goes down, may be a hard disk broken and was taken to repair, may be software need to upgrade to restart - if the client visit to receive a link from this server, it will not detect the background server has changed.

REST is not just a new architecture, it's a new way of thinking about Web development: using urls to structure systems. In REST, all urls correspond to resources, and as long as the URL is well-designed, the architecture of the system it presents is good. This is similar to TDD(Test Driven Development), which designs the interface of the system through Test cases. Each Test case represents a series of user requirements. Developers don't need to write the functionality to begin with, they just need to express the functionality they need to implement in the form of test cases. This is similar to the REST way of designing the system structure through urls. We only need to design reasonable urls according to the requirements. These urls do not have to link to the specified page or complete some behavior, as long as they can intuitively show the user interface of the system. Based on these urls, we can easily structure the system. From the conceptual point of view of the REST architecture, anything that can be abstracted into a resource can be specified as a URL, and the developer needs to work on how to abstract the user requirements into a resource and how to abstract accurately. Because the more precise the resource abstraction, the better it is for REST applications, this is very different from the traditional action-based thinking in MVC development patterns. A well-designed URL can not only help developers understand the structure of the system more clearly, but also facilitate users to remember and identify resources, because the URL is simple and meaningful enough. According to the previous design pattern, many urls are followed by a heap of parameters, which is also very inconvenient for users^[8].

I'll show you some of my code below to show how I followed the RESTful web service implementation's best practice recommendations.

@RestController

```
@RequestMapping("/commodity")
@CrossOrigin(origins = "*", maxAge = 3600)
public class CommodityController {

    @Autowired
    private ICommodityService commodityService;

    @GetMapping("/commodities")
    public DataBody query() {
        return commodityService.selectCommodity();
    }

    @GetMapping("/{id}")
    public Commodity queryById(@PathVariable("id") Long id) {
        return commodityService.selectById(id);
    }

    @PutMapping
    public Integer updateById(@RequestBody Commodity commodity) {
        return commodityService.updateById(commodity);
    }

    @PostMapping
    public Integer saveCommodity(@RequestBody Commodity commodity) {
        return commodityService.saveCommodity(commodity);
    }

    @DeleteMapping("/{id}")
    public Integer deleteCommodity(@PathVariable("id") Long id) {
        return commodityService.deleteById(id);
    }

    @PostMapping("/buy")
    public String buyCommodity(@RequestBody BuyData buyData) {
        return commodityService.buy(buyData);
    }
}
```

I use the POST method to create on my server all the relevant information about the goods, use the GET method to retrieve a goods, and the corresponding to the ID of the goods, use the PUT method, so that I can for goods and as a administrator ID to edit and update the information, use the DELETE method can make me as the administrator to DELETE goods and ID information.

```
@RestController
@RequestMapping("/user")
@CrossOrigin(origins = "*", maxAge = 3600)
public class UserController {
    @Autowired
    private IUserService userService;

    @PostMapping
    public User queryUserByLoginNameAndPwd(@RequestBody User user) {
        return userService.selectUserByLoginNameAndPwd(user);
    }
}
```

In this part of the code, I used the AUTO and POST methods to retrieve and create the user's account and password information.

```
@Select("select id, name, price, number, image, description from sys_commodity")
List<Commodity> selectAll();

@Select("select id, name, price, number, image, description from sys_commodity where id = #{id}")
Commodity selectById(@Param("id") Long id);

@Select("select id, name, price, number, image, description from sys_commodity where name like '%${name}%' and id != #{id}")
List<Commodity> selectByName(@Param("name") String name, @Param("id") Long id);

@Update("update sys_commodity set name = #{commodity.name}, price = #{commodity.price}, number = #{commodity.number}, image = #{commodity.image},
```

```
description = #{commodity.description} where id = #{commodity.id}")
Integer updateById(@Param("commodity") Commodity commodity);
```

```
@Insert("insert into sys_commodity(name, price, number, image, description)
values(#{commodity.name}, #{commodity.price}, #{commodity.number},
#{commodity.image}, #{commodity.description})")
Integer insert(@Param("commodity") Commodity commodity);
```

```
@Select("select count(*) from sys_commodity")
Long queryCount();
```

```
@Delete("delete from sys_commodity where id = #{id}")
Integer delete(@Param("id") Long id);
```

```
// postgres. Query the current sequence id
@Select("SELECT curval('sys_commodity_id_seq'::regclass) AS id")
Long queryId();
```

In this part of the code, I used the Select, Delete, Insert and Update methods to retrieve data from the database, Delete the corresponding commodity information, and query and Insert the ID information of the current sequence.

```
@Mapper
public interface RelatedCommodityMapper {
    @Select("select related_commodity_id from sys_related_commodity where commodity_id
= #{id}")
    List<Long> selectByCommodityId(@Param("id") Long id);

    @Insert("insert into sys_related_commodity(commodity_id, related_commodity_id)
values(#{commodityId}, #{relatedCommodityId})")
    @Options(useGeneratedKeys = true, keyProperty = "id", keyColumn = "id")
    Integer insert(@Param("commodityId") Long commodityId,
@Param("relatedCommodityId") Long relatedCommodityId);

    @Update("update sys_related_commodity set commodity_id = #{commodityId},
related_commodity_id = #{relatedCommodityId} where id = #{id}")
    Integer update(@Param("id") Long id, @Param("commodityId") Long commodityId,
@Param("relatedCommodityId") Long relatedCommodityId);
```

```
@Delete("delete from sys_related_commodity where commodity_id = #{commodityId}")
Integer deleteByComId(@Param("commodityId") Long commodityId);
}
```

This part of the code is used to retrieve information from the database to display other related items after retrieving one item.

```
public class Commodity {
    private Long id;

    private String name;

    private BigDecimal price;

    private Long number;

    private String image;
```

In this section of the code, I define the name, quantity, price, and so on of the item.

```
public class User {
    private Long id;
    private String loginName;
    private String password;
    private String isAdmin;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getLoginName() {
```

```

        return loginName;
    }

    public void setLoginName(String loginName) {
        this.loginName = loginName;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getIsAdmin() {
        return isAdmin;
    }

    public void setIsAdmin(String isAdmin) {
        this.isAdmin = isAdmin;
    }
}

```

This part of the code I used to achieve the user's login account and password creation.

```

@Autowired
private CommodityMapper commodityMapper;

@Autowired
private RelatedCommodityMapper relatedCommodityMapper;

@Autowired
private UserMapper userMapper;

```

This part of the code is used to query the relevant information of the goods.

```

public Long getId() {
    return id;
}

```

```
}

public void setId(Long id) {
    this.id = id;
}

public Long getCommodityId() {
    return commodityId;
}

public void setCommodityId(Long commodityId) {
    this.commodityId = commodityId;
}

public Long getRelatedCommodityId() {
    return relatedCommodityId;
}

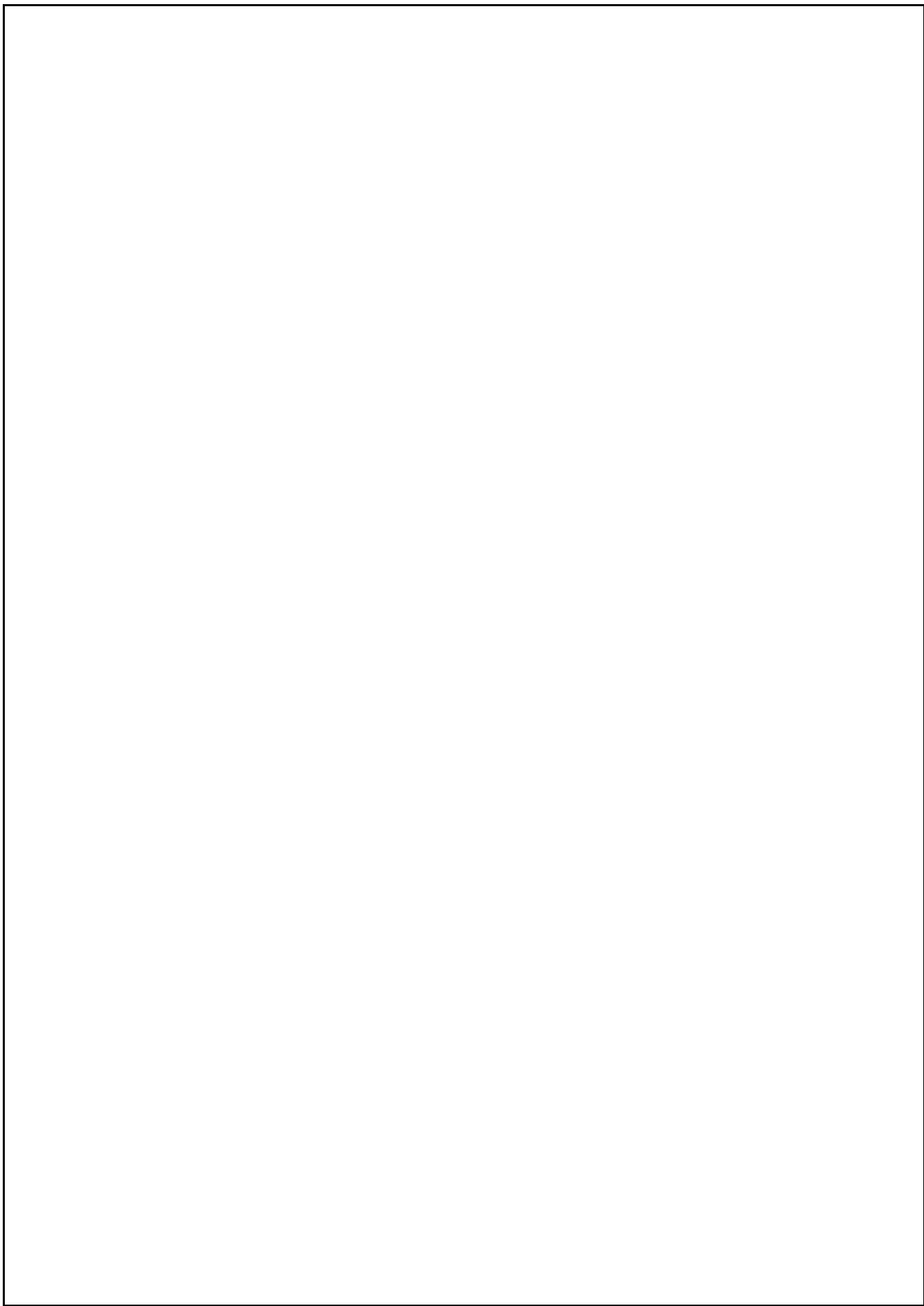
public void setRelatedCommodityId(Long relatedCommodityId) {
    this.relatedCommodityId = relatedCommodityId;
}
```

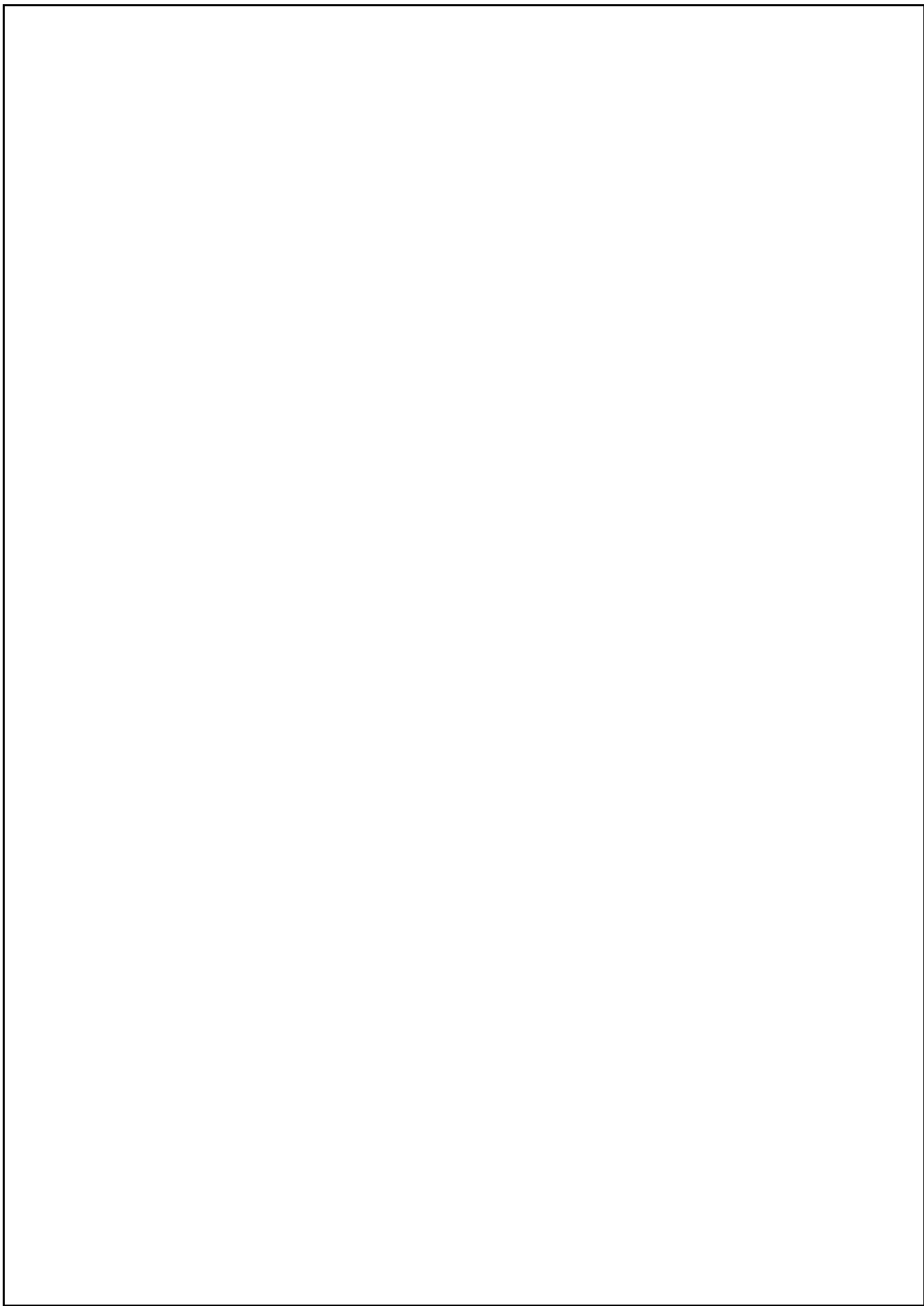
In this part of the code, the retrieval attribute of the item ID is used to retrieve other items related to the item being queried.

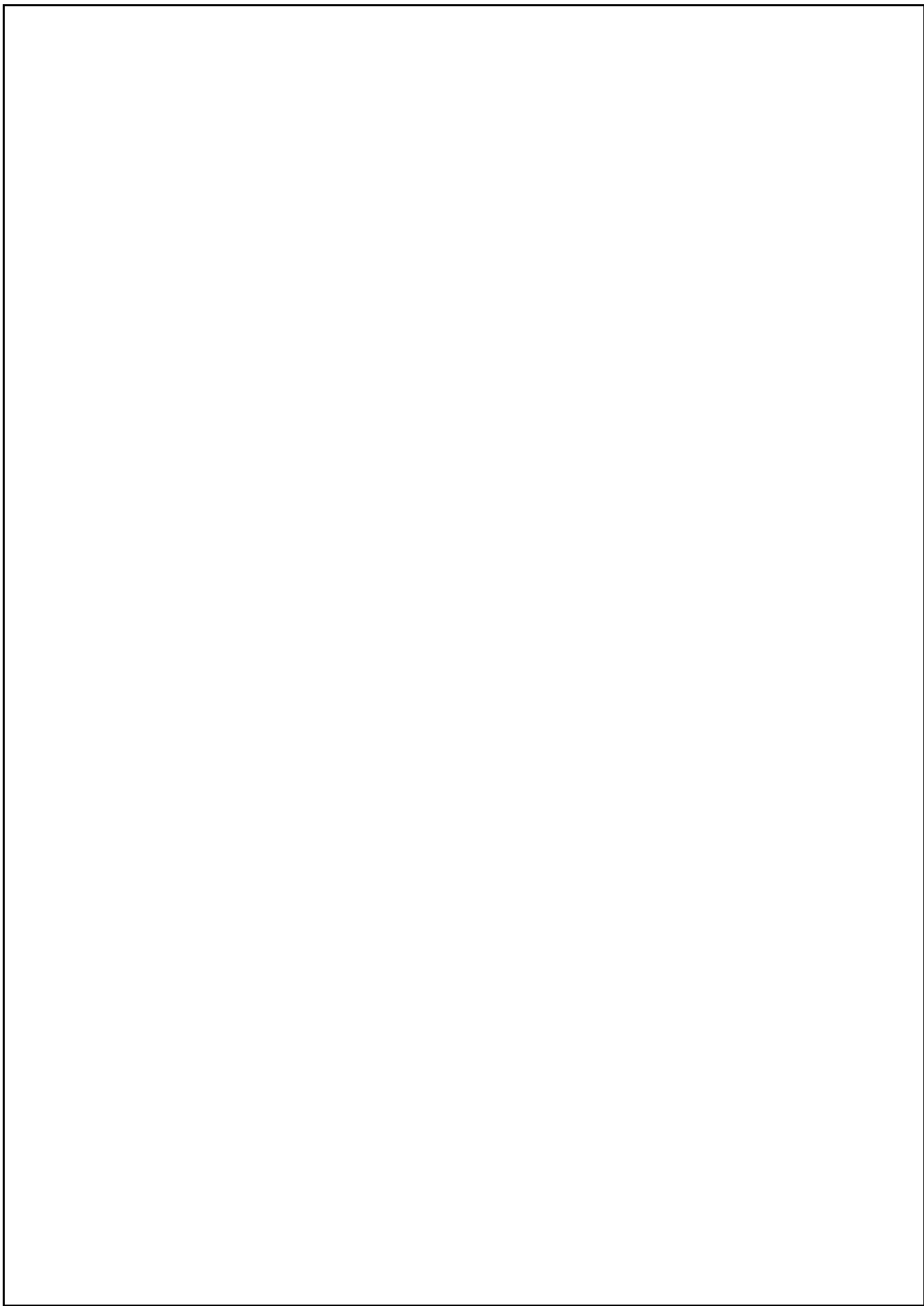
Summary: the code I wrote basically follows the principles of RESTful web services, implements everything with a unique ID (user ID, product ID, etc.), explicitly USES the HTTP method, and USES it in the same way as the protocol definition. The specific code and its implementation can be found in the code I uploaded.

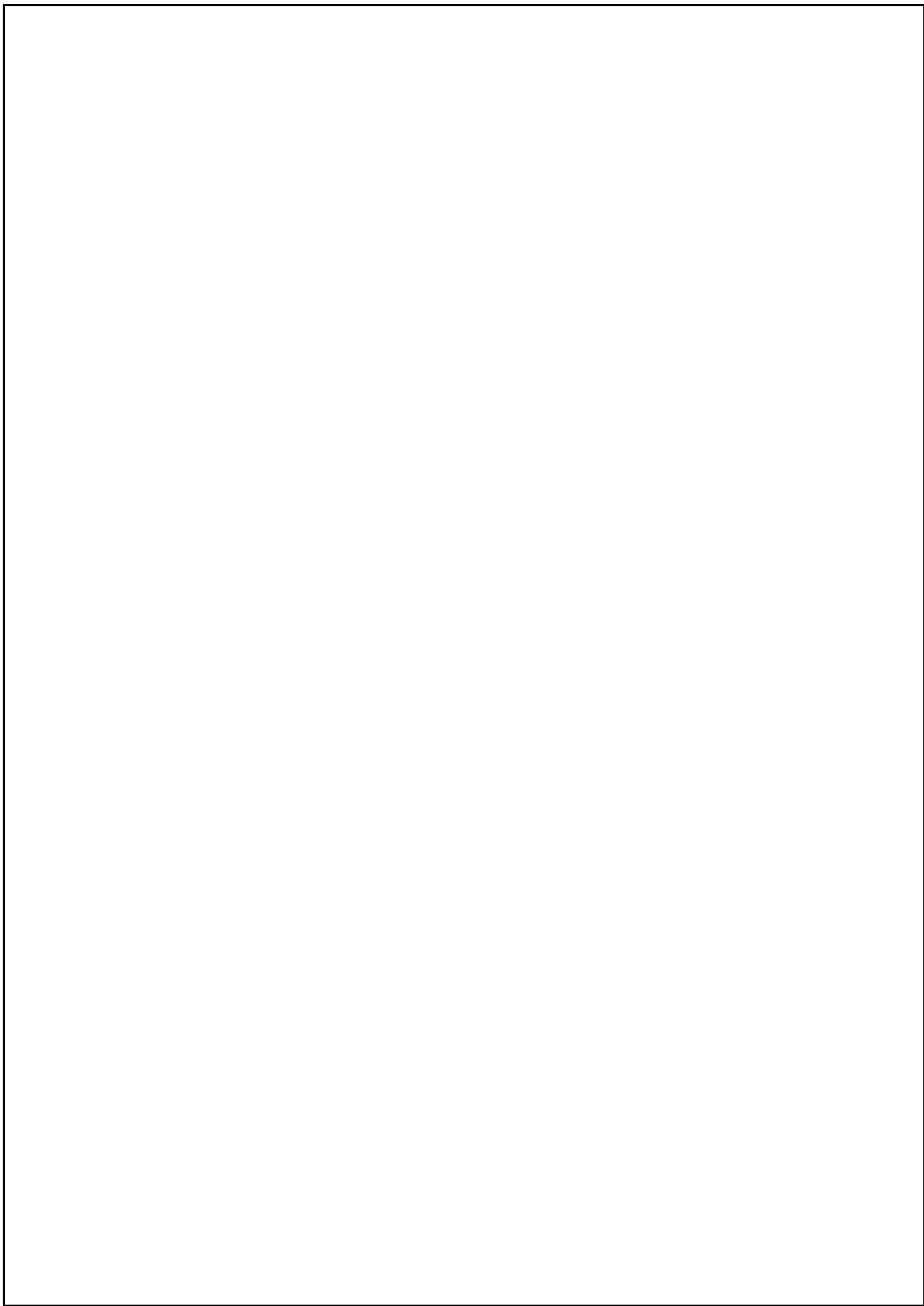
4. References

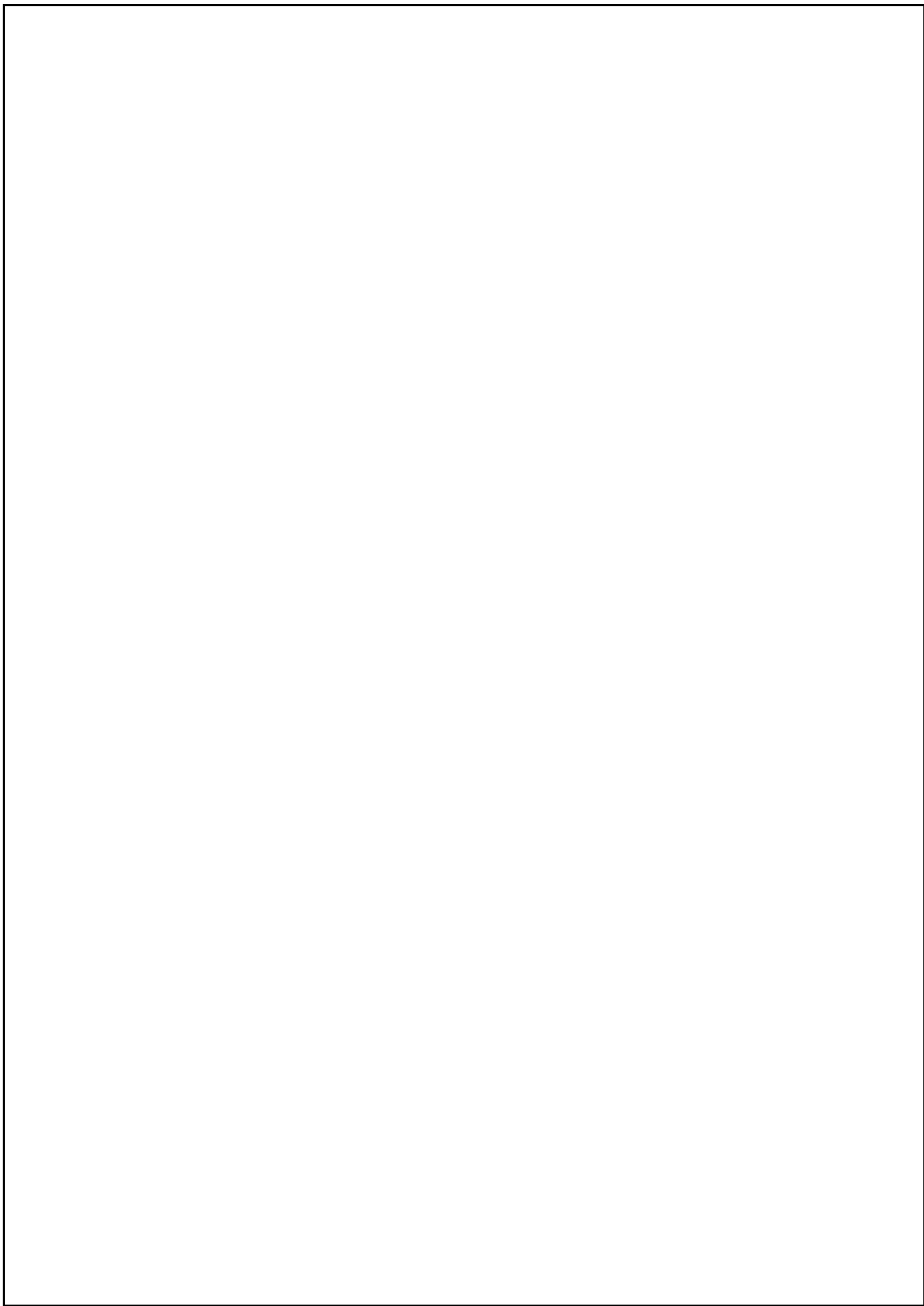
- [1]G. Leone, L. Cristaldi 和 S. Turrin, 《A data-driven prognostic approach based on statistical similarity: An application to industrial circuit breakers》, *Measurement*, Vol 108, pp. 163, 2017.
- [2]N. Savage, 《Going Serverless》, *Association for Computing Machinery. Communications of the ACM*, Vol 61, issue 2, pp 15 - 16, 2018.
- [3]D. Gentsch, A. Klaska, Zhanwei Tu 和 Yurong Lv, 《Innovative thermoplastic pole design sets new reliability and environmental standards for vacuum breaker technology》, 2011, pp 534 - 539.
- [4]J. Zhang, M. Liu, K. Wang 和 L. Sun, 《Mechanical Fault Diagnosis for HV Circuit Breakers Based on Ensemble Empirical Mode Decomposition Energy Entropy and Support Vector Machine》, *Mathematical Problems in Engineering*, Vol 2015, pp 1 - 6, 2015.
- [5]P. Castro, V. Ishakian, V. Muthusamy 和 A. Slominski, 《The rise of serverless computing》, *Communications of the ACM*, Vol 62, issue 12, pp 44 - 54, 2019.
- [6]TIER Industry Report - Portal Management, Data Processing, Web Hosting & Related Services. Taiwan Institute of Economic Research.
- [7]Xinyang Feng 和 Ying Fan, 《Research on Application of RDFa in RESTful Web Services》, 2012, pp 266 - 269.
- [8]H. Li, 《Research on Restful Web Services in Java》, *Applied Mechanics and Materials*, Vol 135 - 136, pp 806 - 808, 2011.
- [9]R. Daigneau, *Service design patterns: fundamental design solutions for SOAP/WSDL and RESTful Web services*. Upper Saddle River, NJ: Addison-Wesley, 2012.

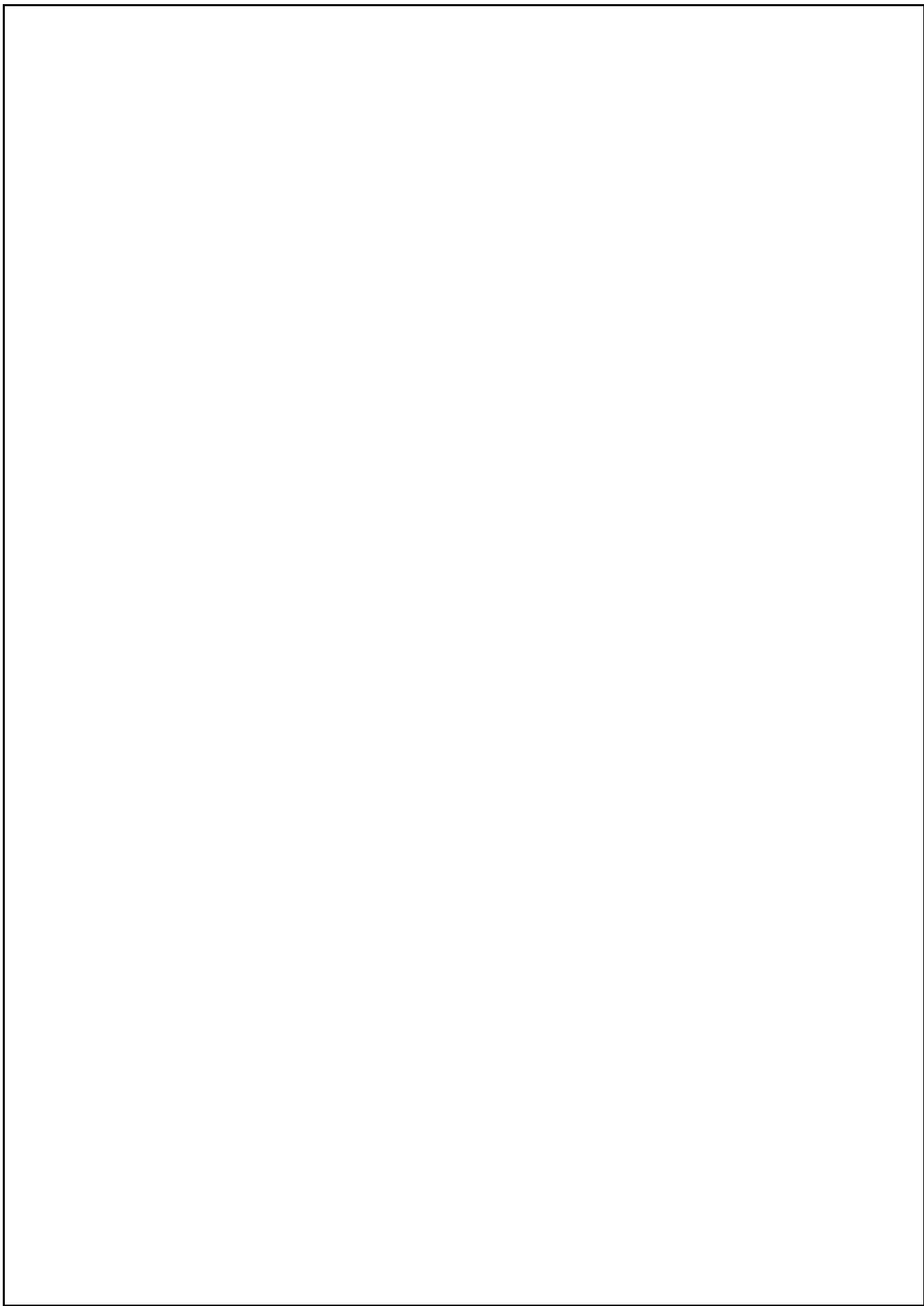












FINAL GRADE

74/100

GENERAL COMMENTS

Instructor

Implementation (50): 33
REST (25): 21

Serverless (25): 20

Feedback

REST discussion could have provided more evaluation of the benefits and drawbacks of web services.

Generally the discussion of FaaS is good, although you could have identified any drawbacks.

Web service provides required functionality with interesting ORM tool. Client provides some ajax functionality e.g. GET and POTS

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11



Comment 1

Define this

PAGE 12



Comment 2

Is this a specific feature of REST or just an example application??

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27
