BENR2423

Database and Cloud System

Chapter 4:

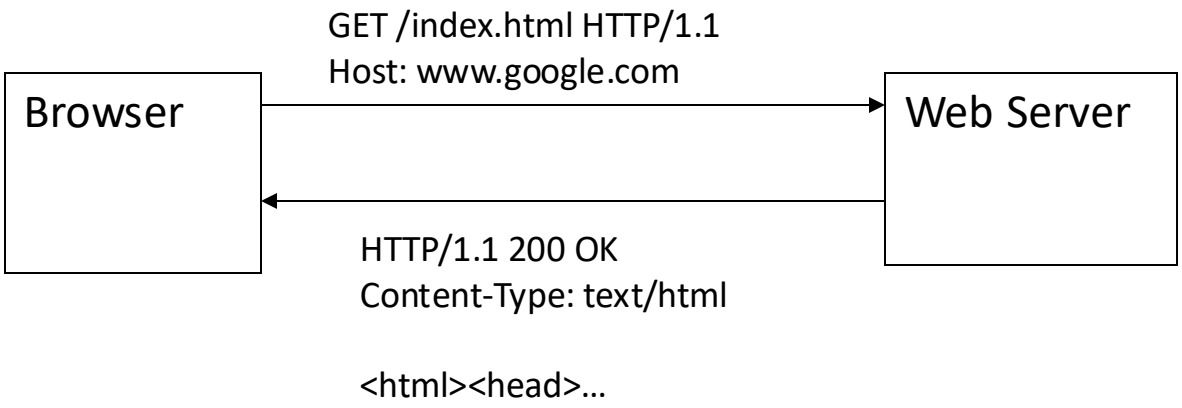
RESTful API

Learning Outcomes

- To understand the fundamental of HTTP request protocol
- To differentiate the HTTP verbs and use them appropriately in Restful API
- To develop Restful API using Nodejs and Express Framework

Hypertext Transfer Protocol (HTTP)

- A communications protocol for transmitting hypermedia documents, such as HTML.
- Designed for communication between web browsers and web servers.

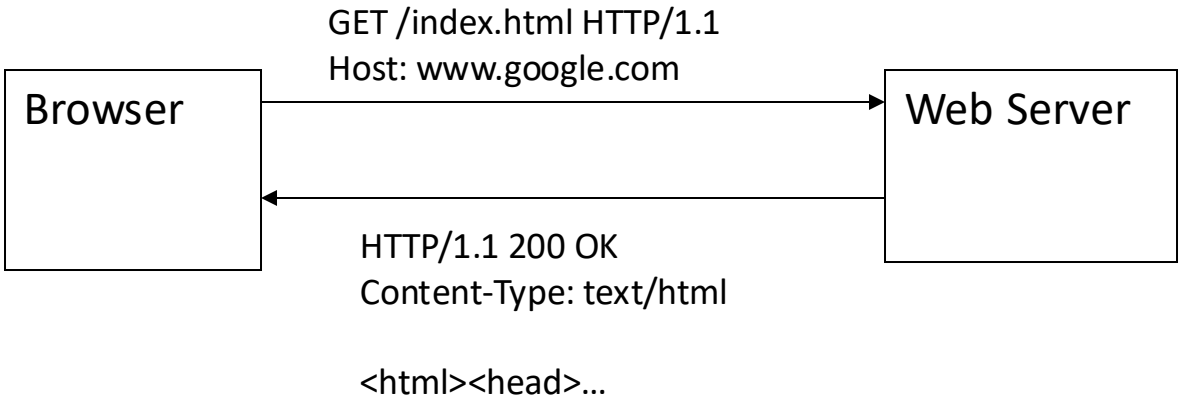


- Basic HTTP request methods (HTTP verbs)
  - GET
  - POST
  - DELETE
  - PATCH
  - PUT
  - CONNECT
  - OPTION
  - TRACE

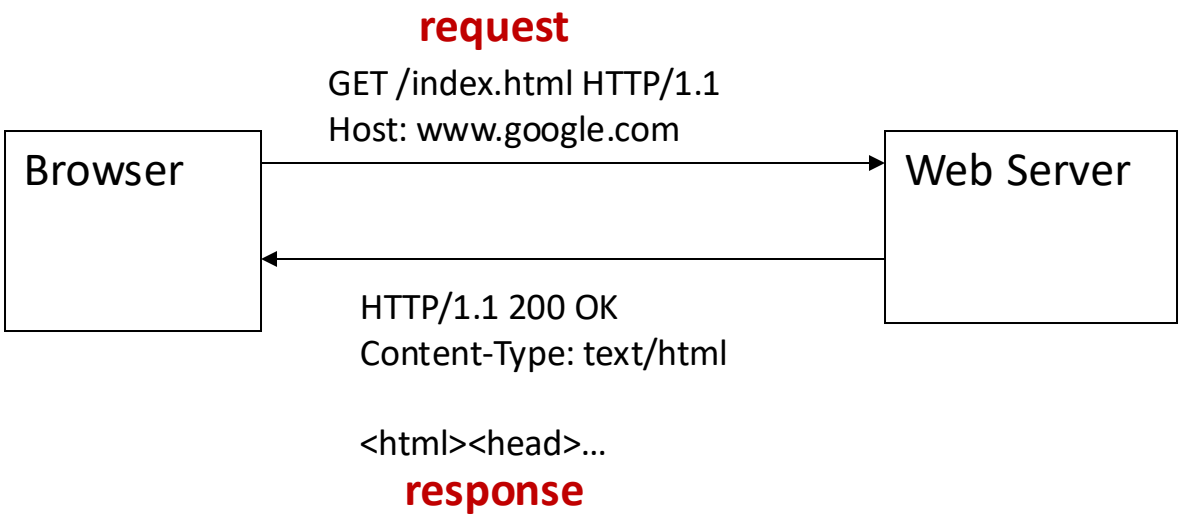
Hypertext Transfer Protocol (HTTP)

Representational State Transfer (REST)

- A style of software architecture and approach to communications often used in web services development
- A Web service that follows these guidelines is called RESTful

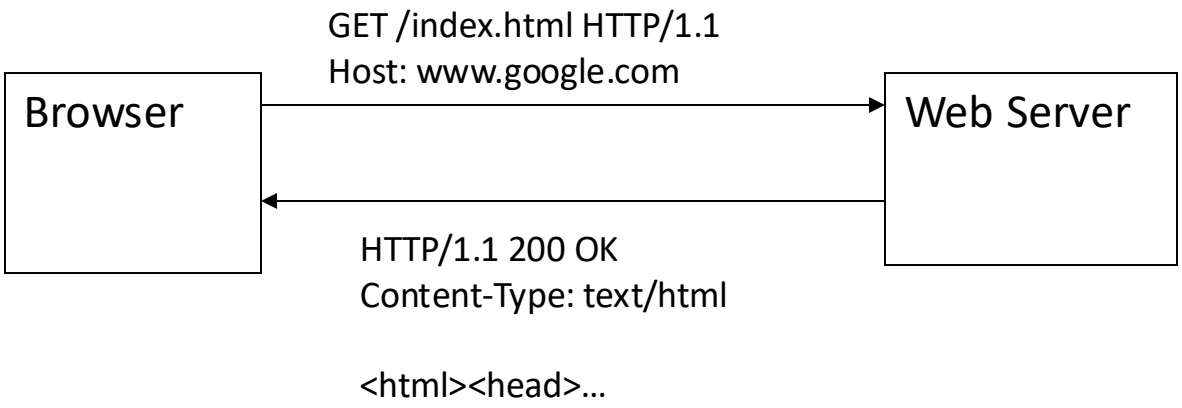


- A Web service must provide its **Web resources** in a textual representation
- The **Web resources** should be linked to a specific URL
- Each URL is called a **request** while the data sent back to you is called a **response**



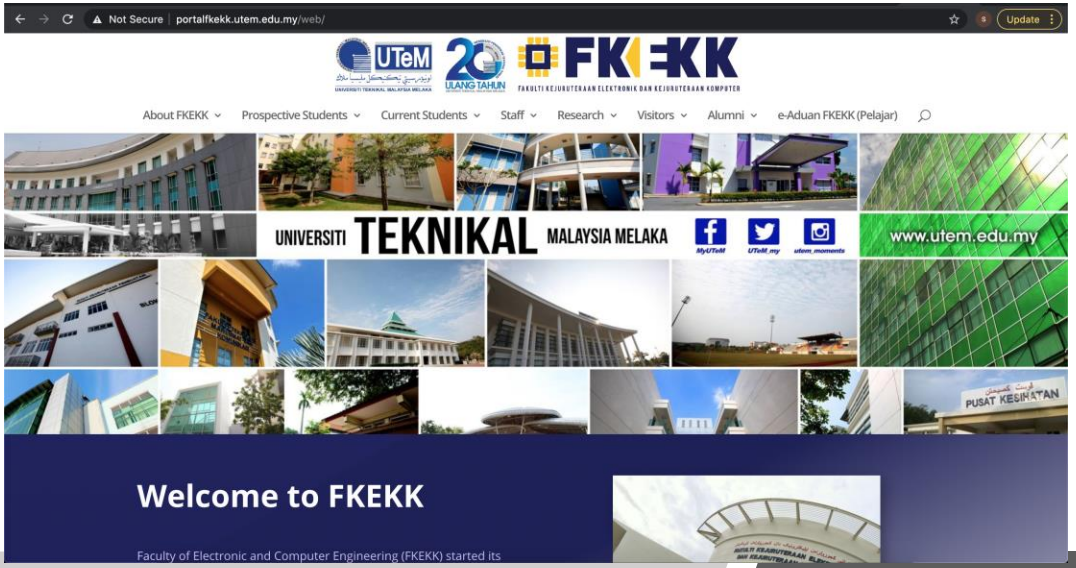
Representational State Transfer (REST)

- The Anatomy of A Request
  - The endpoint : www.google.com
  - The method
  - The headers
  - The data (or body)



Representational State Transfer (REST)

- REST uses URI to identify resources
  - http://portalfkekk.utem.edu.my/web/index.php?search=utem
  - http://portalfkekk.utem.edu.my/web/index.php/deans-welcome-note/
  - http://portalfkekk.utem.edu.my/web/index.php/prospective-students/programme-offered/benr/



The Endpoint

- The methods provide meaning for the request to perform four basic actions: Create, Read, Update and Delete (CRUD):
  - GET : Read Request
  - POST : Create Request
  - DELETE : Delete Request
  - PATCH : Update Request

The Method (HTTP Verbs)

HTTP GET

- This is the default request method to get a resource from a server.
- If you perform a **GET** request, the server looks for the data you requested and sends it back to you.
- In general, a **GET** request performs a **READ** operation.

GET http://benr2423.com/books

Retrieve all books

HTTP POST

- This request is used to create a new resource on a server.
- If you perform a **POST** request, the server creates a new entry in the database and tells you whether the creation is successful.
- In general, a **POST** request performs an **CREATE** operation.

POST http://benr2423.com/student

body: { name: 'Soo', matric: 'B0201110011’}

Create a new student

- This request is used to update a resource on a server.
- If you perform a **PATCH** request, the server updates an entry in the database and tells you whether the update is successful.
- In general, a **PATCH** request performs an **UPDATE** operation.

PATCH http://benr2423.com/student/B0201110011

body: { name: 'Soo YG', matric: 'B0201110011'}

Update student info with matric B0201110011

HTTP PATCH

HTTP DELETE

- This request is used to delete a resource from a server.
- If you perform a **DELETE** request, the server deletes an entry in the database and tells you whether the deletion is successful.
- In other words, a **DELETE** request performs a **DELETE** operation.

DELETE http://benr2423.com/student/B0201110011

Delete student with matric B0201110011

- Headers are used to provide information to both the client and server.
- It can be used for many purposes, such as authentication and providing information about the body content.
- References:

  https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

The Headers

- The data (sometimes called **body** or **message**) contains information you want to be sent to the server.
- This option is only used with **POST**, **PATCH** or **DELETE** requests

The Data (Or Body)

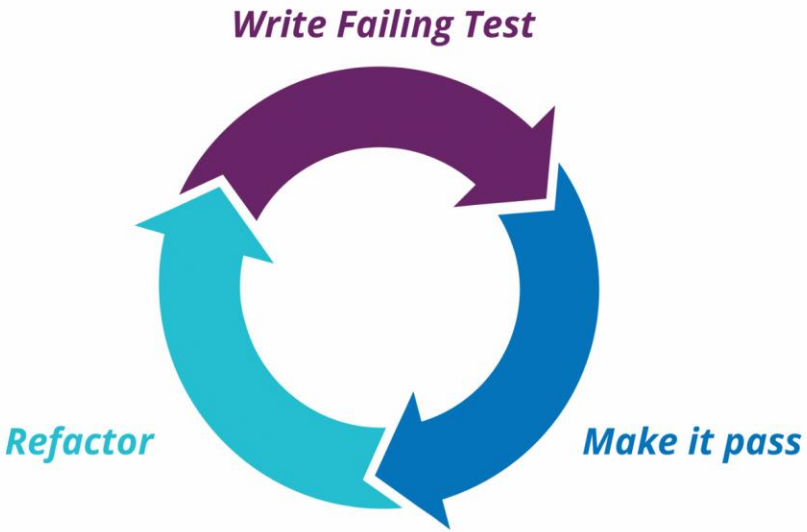
- An API is an **application programming interface**. It is a set of rules that allow programs to talk to each other
- Developer creates the API on the server and allows the client to talk to it

REST API

- HTTP status codes let you tell the status of the response quickly.
- The range from 100+ to 500+. In general, the numbers follow the following rules:
  - 200+ means the request has succeeded.
  - 300+ means the request is redirected to another URL
  - 400+ means an error that originates from the client has occurred
  - 500+ means an error that originates from the server has occurred

HTTP Status Codes And Error Messages

An iterate process that begins with writing a test code for an application or function before starting out to write the application.



Test Driven Development (TDD)

The implementation of TDD to individual function or unit of an application.

```
describe("A suite is just a function", function() {
  var a;

  it("and so is a spec", function() {
    a = true;

    expect(a).toBe(true);
  });
});
```

Specs

Suites

Unit Test

- They are all designed to write, run tests and report results of those tests.
- Example: Mocha[1], Jasmine[2] and Jest[3].
- In general, they all have similar syntax.

[1] https://mochajs.org/

[2] https://jasmine.github.io/setup/nodejs.html

[3] https://jestjs.io/

Testing Framework

Common Matchers

```
test('two plus two is four', () => {
  expect(2 + 2).toBe(4);
});
```

```
test('adding positive numbers is not zero', () => {
  for (let a = 1; a < 10; a++) {
    for (let b = 1; b < 10; b++) {
      expect(a + b).not.toBe(0);
    }
  }
});
```

Matchers

Truthiness

```
test('null', () => {
  const n = null;
  expect(n).toBeNull();
  expect(n).toBeDefined();
  expect(n).not.toBeUndefined();
  expect(n).not.toBeTruthy();
  expect(n).toBeFalsy();
});
```

```
test('zero', () => {
  const z = 0;
  expect(z).not.toBeNull();
  expect(z).toBeDefined();
  expect(z).not.toBeUndefined();
  expect(z).not.toBeTruthy();
  expect(z).toBeFalsy();
});
```

Matchers

Truthiness

```
test('null', () => {
  const n = null;
  expect(n).toBeNull();
  expect(n).toBeDefined();
  expect(n).not.toBeUndefined();
  expect(n).not.toBeTruthy();
  expect(n).toBeFalsy();
});
```

```
test('zero', () => {
  const z = 0;
  expect(z).not.toBeNull();
  expect(z).toBeDefined();
  expect(z).not.toBeUndefined();
  expect(z).not.toBeTruthy();
  expect(z).toBeFalsy();
});
```

Matchers

Numbers

```
test('two plus two', () => {
  const value = 2 + 2;
  expect(value).toBeGreaterThan(3);
  expect(value).toBeGreaterThanOrEqual(3.5);
  expect(value).toBeLessThan(5);
  expect(value).toBeLessThanOrEqual(4.5);

  // toBe and toEqual are equivalent for numbers
  expect(value).toBe(4);
  expect(value).toEqual(4);
});
```

Matchers

Numbers

```
test('adding floating point numbers', () => {
  const value = 0.1 + 0.2;
  //expect(value).toBe(0.3);           This won't work be
  expect(value).toBeCloseTo(0.3); // This works.
});
```

Matchers

Strings

```
test('there is no I in team', () => {
  expect('team').not.toMatch(/I/);
});

test('but there is a "stop" in Christoph', () => {
  expect('Christoph').toMatch(/stop/);
});
```

Matchers

Array

```
const shoppingList = [
  'diapers',
  'kleenex',
  'trash bags',
  'paper towels',
  'milk',
];

test('the shopping list has milk on it', () => {
  expect(shoppingList).toContain('milk');
  expect(new Set(shoppingList)).toContain('milk');
});
```

Matchers

Exception

```
function compileAndroidCode() {
  throw new Error('you are using the wrong JDK');
}

test('compiling android goes as expected', () => {
  expect(() => compileAndroidCode()).toThrow();
  expect(() => compileAndroidCode()).toThrow(Error);

  // You can also use the exact error message or a regexp
  expect(() => compileAndroidCode()).toThrow('you are using the wrong JDK
  expect(() => compileAndroidCode()).toThrow(/JDK/);
});
```

Matchers