Department of
COMPUTER SCIENCE
HONG KONG BAPTIST UNIVERSITY

COMP7980 – Dynamic Web and Mobile Programming
COMP7270 – Web and Mobile Programming

Chapter 3 HTTP & Express Framework

Course Instructors: Dr. Ma Shichao, Dr. Zhang Ce, and Mr. Jiang Jintian

Announcement

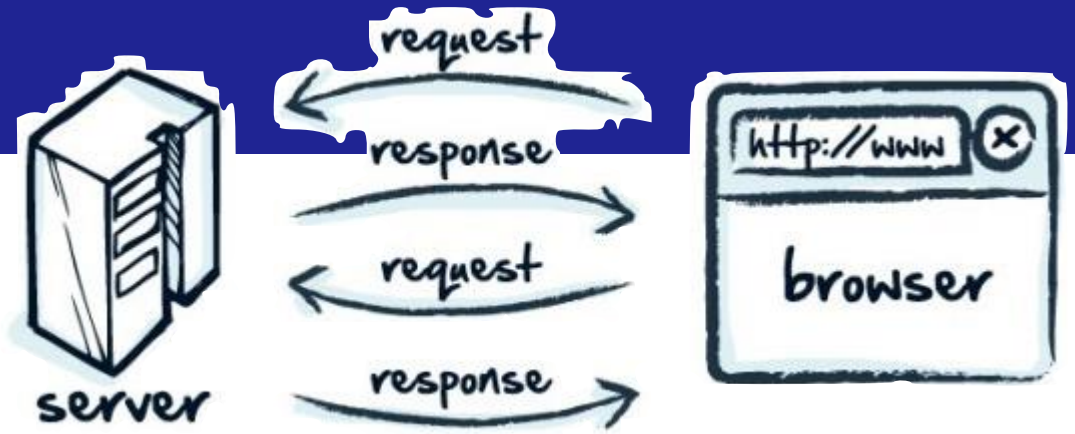
- Group Project
  - See the Project Guideline on Moodle
  - Each group consists of 2-3 students
  - Please join a group using the Google's online Excel Form

  Group Form

Agenda

- **HTTP** Protocol
- HTML Form Submission
- **Node.js** - JavaScript runtime environment
- Web Application Development with **Express**
- **NoSQL** Database

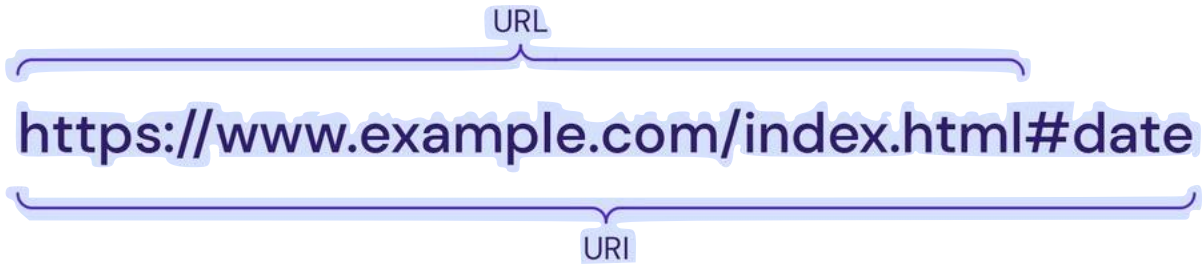
HTTP Protocol



- The **Hypertext Transfer Protocol** (HTTP) is an application-level protocol used for communication between **web browsers** and **web servers**. It serves as the foundation for data communication on the World Wide Web.
- HTTP is a **request-response** protocol, where a client (typically a web browser) sends a request to a server, and the server responds with the requested data.

Key features of the HTTP protocol

- **Stateless**: HTTP is a stateless protocol, meaning that **each request from the client is independent** and **does not retain any information about previous requests**. This allows for scalable and distributed systems.
- **Uniform Resource Identifier** (URI): HTTP uses URIs to **identify and locate resources** on the web. **URIs include URLs** (Uniform Resource Locators) that specify the address of a specific resource, such as a web page or an image.



Key features of the HTTP protocol

- **Methods**: HTTP defines several methods that **indicate the type of action the client wants to perform** on a resource. The most common methods are **GET** (**retrieve a resource**), **POST** (**send data to be processed**), PUT (update a resource), DELETE (remove a resource), and HEAD (retrieve metadata about a resource).
- **Status Codes**: HTTP uses status codes to **indicate the outcome of a request**. These codes provide information about whether the request was successful, encountered an error, or requires further action. For example, a status code of **200 indicates a successful request**, while **404 indicates that the requested resource was not found**.

Key features of the HTTP protocol

- **Headers**: HTTP headers are **additional information sent along with the request or response**. Headers can contain metadata, **authentication credentials**, caching directives, and other details that facilitate communication between the client and server.
- **Cookies**: HTTP supports the use of cookies, which are **small pieces of data** stored on the client-side by the server. Cookies allow servers to maintain session information and **remember user preferences** across multiple requests.

IP Addresses

- Every machine on the Internet has to have **a unique IP address** so that communications can be routed to the correct computer.
- IPv4: **32-bit binary number**
  - e.g. 123.45.67.89
  - Each number ranges from 0 to 255.

Domain Names & Local Machine

- **Domain name**
  - Domain names are **mapped to IP addresses**.
  - Easier to remember hkbu.edu.hk -> 158.182.0.81
- To reference the **local machine**, we can use
  - **127.0.0.1** (IP address) or **localhost** (domain name)

Port

- Different **Internet services** will use different **ports**
  - Web server usually uses **port 80**.
  - **Ports 8000 and 8080** are common ports for software providing **http services** that is not a core HTTP server.

Port Number	Protocol	Application
20	TCP	FTP data
21	TCP	FTP control
22	TCP	SSH
25	TCP	SMTP
53	UDP, TCP	DNS
80	TCP	HTTP (WWW)
110	TCP	POP3
443	TCP	SSL

HTML Form Submission

```
<form action="https://www.httpbin.org/post" method="POST">
  <input name="email" type="email">

  <input name="numTickets" type="number" min=1 max=4>

  <button type="submit">Submit</button>
</form>
```

- Form is used to pass data to a server.
- The **submit button** will trigger the submission.
- action specifies **where** the form data will be submitted to.
- method specifies the **HTTP request method** for sending form data.

Name Attribute

```
<input name="numTickets" type="number">
```

- The name attribute specifies the name of an <input> element.
- The name attribute could be used to reference elements in client-side JavaScript, or to **reference form data in form submission**.
  - Note: Only form elements with a name attribute will be included in form submission.

Checkbox

☐ Option 1 `<input name="box" type="checkbox" value="dummy">`

- When this checkbox is clicked, this form element name, **together with its value**, is submitted to the server. box = "dummy"

- If the box is NOT clicked, this form element **won't be submitted**.

☑ `<input name="box" type="checkbox" value="dummy" checked>`

- To check the box, put the checked attribute in the opening tag.

HTTP Method - GET

http://server/path/endpoint?input1=value1&input2=value2&...

- GET is the **default method** used by HTML forms if no method is specified.
- When using the GET method, the form data **is appended to the URL** as **query parameters**. This means that the form data is visible in the URL, which can **have implications for security and privacy**.
- GET requests are typically used for **retrieving data** from the server and are **idempotent**, meaning that multiple identical requests will have the same effect as a single request.
- GET requests **can be bookmarked**, cached, and shared, as the **form data is part of the URL**. However, there is a limit on the length of the URL that can be transmitted, so **large amounts of data are not suitable** for GET requests.

HTTP Method - POST

- POST sends the form data **in the body of the HTTP request**, rather than appending it to the URL. This makes POST requests **more secure for sensitive data**, as the data is not directly visible in the URL.
- POST requests are **not idempotent**, meaning that multiple identical requests may have different effects on the server. This is because POST requests are often used for actions that **modify data on the server**, such as **creating a new resource** or updating existing data.
- POST requests are **not bookmarkable or shareable** directly from the URL, as the form data is not part of the URL itself.
- There are **no inherent restrictions on the length of data** that can be sent in a POST request, making it suitable for transmitting larger amounts of data.

More on JavaScript

JavaScript

- **Loosely typed**
  - The same variable could be **re-assigned to values of different types**.
- Variable naming rules
  - Names are **case sensitive**.
  - Names begin with **a letter** or the **underscore** character.

JavaScript Types

- To parse a string into a number, we can use the parseInt() function.

```
numVariable = parseInt(stringValue);
```

- If the input string doesn't start with a number, NaN **(Not-A-Number)** will be returned.
- We can check for this value with the isNaN() function.

```
isNaN(numVariable)
```

JavaScript Object

- JavaScript objects can be thought of **as a collection of properties**.
  - Each property is **identified by a key value** and **can hold values of any type**, including other objects.
  - Additionally, objects can also **contain functions as properties**.
  - However, when an object **does not include any functions** and primarily serves to store and manipulate data, it is often referred to as a "**plain object**" or a "**data object**".
  - To access this **person's city**, we can use **person.address.city**

```
var person = {
  name: "John Doe",
  age: 30,
  occupation: "Software Engineer",
  isStudent: false,
  hobbies: ["reading", "playing guitar", "hiking"],
  address: {
    street: "123 Main St",
    city: "Anytown",
    country: "USA"
  }
};
```

Falsy Values

- A value is either “**truthy**” or “**falsy**”.
- A variable **without a value assigned**, is of type **undefined**.

```
> var undefinedVar;
undefined
> typeof undefinedVar
'undefined'
```

```
> if (undefinedVar)
        foo = "It's truthy";
        else foo = "It's falsy"
"It's falsy"
```

Falsy Value	Type
false	Boolean
0	Number
NaN	Number
'' or "" (empty string)	String
null	Object
undefined	Undefined

https://www.sitepoint.com/javascript-truthy-falsy/

Equality Comparisons

==	true	false	0	''
true	true	false	false	false
false	false	true	true	true
0	false	true	true	true
''	false	true	true	true

===	true	false	0	''
true	true	false	false	false
false	false	true	false	false
0	false	false	true	false
''	false	false	false	true

Node.js, Express & MongoDB

Node.js

- Node.js is an open-source, **server-side JavaScript runtime** environment that allows developers to run JavaScript code on the server.
- Node.js has a vast ecosystem of **open-source modules and packages** available through the **Node Package Manager (NPM)**. These modules provide various functionalities that can be easily integrated into Node.js applications, allowing developers to leverage the work of others and accelerate the development process.
- Node.js is commonly used for **building web servers, APIs, real-time applications** (such as chat applications and gaming servers), streaming applications, and microservices. It has gained popularity among developers for its performance, scalability, and the ability to **use JavaScript as a full-stack language**.

Express

```
router.post('/form', function (req, res) {

    var response = {
        header: req.headers,
        body: req.body
    };

    res.json(response);
});
```

- Express is a fast, minimalistic, and flexible **web application framework for Node.js**. It provides a robust set of features and utilities for building web applications and APIs. Express.js is built on top of the Node.js core HTTP module, **simplifying the process of handling HTTP requests and responses**.
- **Routing**: Express.js allows developers to **define routes for handling specific HTTP requests** (such as GET, POST, PUT, DELETE) and their corresponding actions.

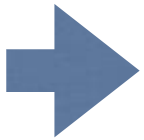
Express

- **Template Engines**: Express supports various template engines, such as Pug (formerly known as Jade), **EJS (Embedded JavaScript)**, Handlebars, and more. Template engines enable the **dynamic generation of HTML** or other markup languages, simplifying the process of **rendering views and generating dynamic content**.

```
<table>
  <% for (var booking of bookings) { %>

    <tr>
      <td><%= booking.email %></td>
      <td><%= booking.numTickets %></td>
    </tr>

  <% } %>
</table>
```



```
<table>
        <tr>
          <td>tony@stark.com</td>
          <td>2</td>
        </tr>
        <tr>
          <td>bruce@wayne.com</td>
                <td>1</td>
        </tr>
</table>
```

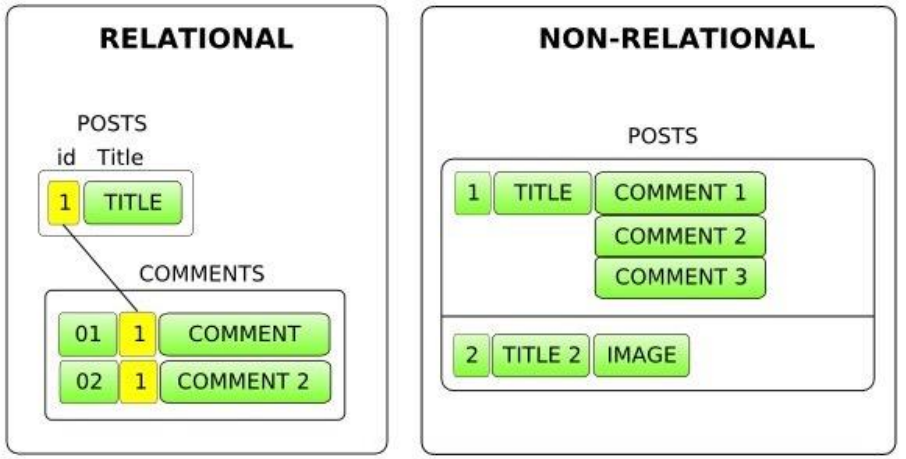
Response to client

The Request Object

- Under Node.js/Express, the req object represents the HTTP request and has properties:

req.params	**Route parameters** (e.g., /:id)
req.query	**Query parameters**, form data submitted via GET
req.body	**Request body**, form data submitted via POST

NoSQL Database



- A NoSQL (Not Only SQL) database is a type of database management system that provides a **non-relational data model** for storing and retrieving data.
- NoSQL databases allow for **dynamic schema design**, meaning that each record or **document can have its own structure** without requiring a predefined schema. This flexibility is beneficial when dealing with data that may have varying attributes or when the schema needs to evolve over time.

Collection and Document

Relational	NoSQL
Database	Database
Table	**Collection**
Row	**Document**
Column	Field

MongoDB

- **_id** field is reserved for **primary key** in MongoDB.
- MongoDB uses ObjectId as the default value of **_id** field of each document, which is generated while the creation of any document.
- An ObjectId is 12-byte long.

```
{
   _id: ObjectId("5099803df3f4948bd2f98391"),
   name: { first: "Alan", last: "Turing" },
   birth: new Date('Jun 23, 1912'),
   death: new Date('Jun 07, 1954'),
   contribs: [ "Turing machine", "Turing test", "Turingery" ],
   views : NumberLong(1250000)
}
```

MongoDB

- MongoDB provides a powerful and **flexible query language** that supports a wide range of operations for retrieving, modifying, and aggregating data. The query language includes capabilities for filtering, sorting, joining, and performing complex aggregations, making it suitable for a variety of use cases.

```
db.users.insertOne(          ← collection
   {
      name: "sue",           ← field: value
      age: 26,               ← field: value
      status: "pending"      ← field: value
   }                         } document
)
```

```
db.users.find(                          ← collection
   { age: { $gt: 18 } },                ← query criteria
   { name: 1, address: 1 }              ← projection
).limit(5)                              ← cursor modifier
```