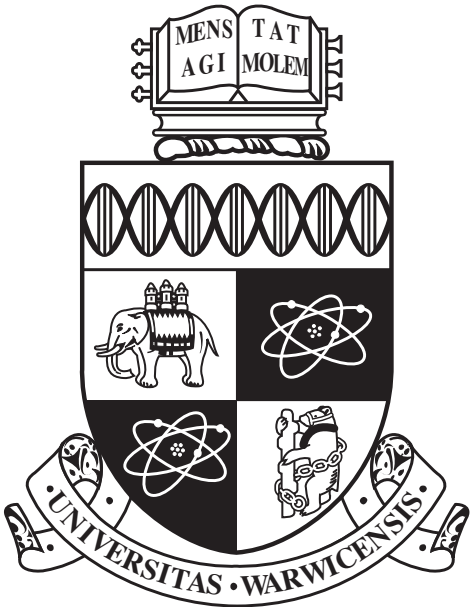
University of Warwick
Department of Computer Science

CS139

Web Development Technologies



Cem Yilmaz
May 23, 2022

Contents

1 HTML

1.1 Syntax

HTML stands for HyperText Markup Language and is semantic. This means that it describes the structure of the document and not the content. It is intended to modify the appearance of HTML elements and can be in fact frustrating to use for page layouts.

A lot of HTML is done with the <> brackets. For example,

```
<h1>Welcome to CS139</h1>
```

Listing 1: Heading

This would set the header tag to the text "Welcome to CS139". For this module, we will be using JSFiddle, that is available online.

1.1.1 Doctypes

Every HTML documents should have a doctype definition on top. In particular, HTML5 uses

```
<!DOCTYPE html>
```

Listing 2: DOCTYPE

It helps the browser to know what to expect.

1.1.2 Example HTML5 Document

```
<!DOCTYPE html>
<html>

  <head>
    <meta charset="UTF-8">
    <title>Title for Hello World </title> // Title that is seen at the top of the browser
  </head>
  <body>
    <h1>Hello world</h1> // Biggest header for the website
  </body>
</html>
```

Listing 3: Example Document

1.1.3 Head tag

This tag is used by the browser, web-crawlers and bots. IT includes meta-tags required by these applications and includes location of supporting documents e.g. JavaScript and CSS.

1.1.4 Text-encoding

Familiar with ASCII, but that is only 128 characters. In particular, UTF-8 has 107000 characters and is denoted with

```
<meta>
```

Listing 4: Text-encoding

1.1.5 Body tag

This is the tag where main information goes into that the user gets to read. Its syntax is

```
<body>
</body>
```

Listing 5: Body

1.1.6 Syntax

```
<a href="google.com">Google search</a>
```

Listing 6: Syntax

In the code above, a is the element name. The hyperlink in href is called the *Attribute*. The content is the *Google search* text. However, there are also empty tags e.g.

```
<meta charset="utf-8">
```

Listing 7: Empty Tag

1.1.7 Nesting Definitions

Definition 1.1. Child and parent

A syntax is a child if and only if there exists a tag that is at a lower level than the upper tag.

```
<body>
  <p>
  This is some text
  </p>
</body>
```

Listing 8: Parent Child

In particular, $< body >$ is the parent of $< p >$ and $< p >$ is the child of $< body >$.

Definition 1.2. Sibling

Sibling is when the tag is on the same level. For example,

```
<body>
  <p>
  This is some text
  </p>
  <p>
  Another text
  </p>
</body>
```

Listing 9: Sibling

In here, the p are siblings.

Similarly, the term descendants would be group of tags of in comparison to a tag that is a parent of all.

1.1.8 Lists

There are 3 types of lists:

- Ordered lists denoted with $< ol >$ and then listed items with $< li >$
- Unordered lists denoted with $< ul >$ and then listed items with $< li >$
- Description list would list terms and then list descriptions. In particular, the tags that are used are $< dl >$, $< dt >$ and $< dd >$ which are list, item and description respectively.You can also created nested list if you simply begin another list inside a list.

For example

```
<ul>
  <li> shopping </li>
  <ol>
    <li> eggs </li>
    <li> bread </li>
  </ol>
  <li> cooking </li>
</ul>
```

Listing 10: Lists

1.1.9 Hyperlinks

Hyperlinks are listing websites to a specific piece of text. For example

```
<a href="www.google.com"> This is google hyperlink </a>
```

Listing 11: Hyperlink

You can also hyperlink inside the website using ids. For example,

```
<body>
<h1> Links </h1>
<p id = "#top">
  This is some paragraph text
</p>
<a href="#top"> Go to top </a>
```

Listing 12: IDs

1.1.10 Images

You can also include images with a singular tag that use the src and alt attributes. For example,

```
<img src="http://warwick.ac.uk/logo.gif" alt="Warwick Logo" title="Warwick Logo" width=200
    height=60 />
```

Listing 13: Image embed

1.1.11 Character entities

```
  //Nonbreakable space
&lt; //<
&gt; //>
&copy; //Copyright symbol
&trade; //Trademark symbol
```

Listing 14: Character entities

1.1.12 Break Line

You can get a new line or break a line using the tag

```
<br>
```

Listing 15: Break

1.2 Semantic Mark-up

Some semantics include but are not limited to

```
<abbr>
<cite>
<time>
<span>
<div>
```

Listing 16: Semantics

That is, these do not change the looks but are important regardless. Usually, semantic mark-up refers to creating IDs on code to give meaning to piece of text and make it readable. In particular, the example in Hyperlinks with the ID is an example of a semantic mark-up.

1.3 Validation

You can make sure that your HTML is valid using a validation tool provided by W3C. The website is http://validator.w3.org

1.4 Types of Style Sheet

There are three different types of style sheet:

- Author created style sheets
- User style sheets
- Browser style sheets

1.5 Tables

A lot of data in HTML conveys data in rows and columns, i.e., a table.

```
<table>
  <tr>
    <td> The death of Marata </td>
    <td> Jacques-Louis David </td>
    <td> 1793 </td>
    <td> 162cm </td>
    <td> 128cm </td>
  </tr>
  <tr>
    <td> Burial at Ornans </td>
    <td> Gustave Courbet </td>
    <td> 1849 </td>
    <td> 314cm </td>
    <td> 663cm </td>
  </tr>
</table
```

Listing 17: Table

In here, $< /tr >$ stands for table row. $< td >$ stands for table data. Therefore cells in a row are declared by td, whereas rows are declared by tr. There is also $< th >$ which characterises table header. You can also add rowspan i.e.

```
<table>
  <tr>
    <th> Artist </th>
    <th> Title </th>
    <th> Year </th>
  </tr>
  <tr>
    <td rowspan="3"> Jacques-Louis David </td>
    <td> The death of Marat </td>
    <td> 1793 </td>
  </tr>
  <tr>
    <td> The Intervention of Sabine Woman </td>
    <td> 1789 </td>
  </tr>
  <tr>
    <td> Napoleon Crossing the Alps </td>
    <td> 1800 </td>
  </tr>
</table>
```

Listing 18: Rowspan

Column span works in a similar fashion. Furthemore

```
<thead>
// Top of the table. It is always the top rows of a table.
</thead>
<tbody>
// Middle of the table. It is always inbetween header and foot.
<tbody>
<tfoot>
// Sets the last row of the table. This will always be at the bottom of the table.
</tfoot>
```

Listing 19: Table Head and Footer

1.6 Forms

Sometimes we require to get data from the user, e.g. log-in page. It has the attributes:

- Action - the destination of the form data when submitted
- Method - the way in which the data is sent (GET or POST)
- Accept-charset - The charset accepted by the form

```
<form method="post" action="process">
  <fieldset>
    <legend> Details </legend>
    <p>
      <label> Title: </label>
      <input type="text" name="title" />
    </p>
    <p>
      <label> Country: </label>
      <select name="where">
        <option> Choose a country </option>
```

```
        <option> Canada </option>
        <option> Finland </option>
        <option> United States </option>
      </select>
    </p>
    <input type ="submit" />
  </fieldset>
</form>
```

Listing 20: A basic form

1.6.1 Attributes

- Type - the kind of input to accept
- Name - the name submitted to the server
- The value that you want the field to have
- Other attributes related to type...

```
<input type="text" name="city name" value="Warwick" >
<input type="text" name="uname" placeholder="what is your name" >
```

Listing 21: Attributes

1.6.2 Radio Buttons

Radio buttons must have the same name in a group and allows you to select only a singular option

```
<input type="radio" name="where" value="1">Coventry<br>
<input type="radio" name="where" value="2" checked>Warwick<br>
<input type="radio" name="where" value="3">Nottingham<br>
```

Listing 22: Radio Buttons

1.6.3 Dropdown menus

Dropdowns need $< option >$ elements with the $< select >$.

```
<select name="choices">
  <option>First</option>
  <option selected>Second </option>
  <option> Third </option>
</select>
```

Listing 23: Dropdown menus

1.6.4 Accessibility

Sometimes we prefer the use of tables for data, not layout. In these cases, we use the caption element. This connect cells with a textual description in the header. You can also use "id" and "label for" to easen the accessability

2 Introduction to Python/Flask

2.1 Flask and Python

Flask is a server-side web micro-framework and it mainly uses Python as its scripting language. Its extensions provide further functionality. For example,

- Dynamic HTML markup uses Jinja2
- Database access via SQLAchamy
- OThers such as Flask-WTF (forms) and Flask-Login (authentication)

It was released in 2010 and its version 2.0.2 was released in October. Reddit, Linkedin, Netflix and Pinterest use it.

2.2 General Architecture

Python code is executed on a server and the output html is then sent to the client's browser.

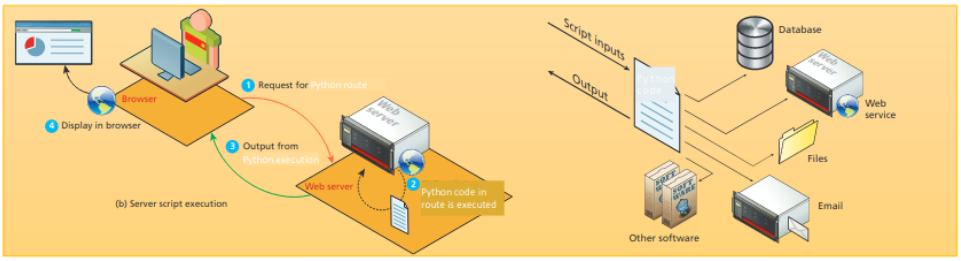


Figure 1: Architecture of web server

2.3 Flask and Python

=Python is used to execute a web server and is written within a .py file. HTMl code is written in .html file. Python uses indentation for scoping without braces or semicolons. Comments use ectomorph simple or triple apostrophes e.g.

```
133 # comment
134 ''' comment '''
```

Listing 24: Commenting

2.4 Variable Naming

Variables are duck=typed, i.e. dynamically and an integer. Variable names must begin with a letter or an underscore and cannot have spaces. Case sensitive value is not equal Value.

2.5 Python Data Types

Scalar types include

- Integer
- Float
- Complex
- Boolean
- String

And compound types

- Dictionary
- List and tuples
- Set
- Object

2.6 Python Containers

Python has 4 containers types: Lists, Sets, Dictionaries and Tuples.

- Lists are ordered and mutable, uses $[]$
- Sets are unordered and cannot have duplicates, uses $set\,()$
- Tuples are ordered and immutable, uses $(,)$
- Dictionaries are associative arraways - unordered values are references by their key, use $\{\}$

In particular, for examples, dictionaries can have "keys", e.g.

```
135 ages = {"Adam":27, "Dean":20, "Louise":30}
```

Listing 25: Keys

Then,

```
136 ages["Adam"] = 27
137 ages.update("Dean",20)
```

Listing 26: Keys 2

The key, which is their name, is linked to data which is their age.

2.7 Python Functions

You can define your own functions so that code is reusable. It is essentially creating methods. The code is

```
138 def <function name> (arg1, arg2...):
```

Listing 27: Functions

And these can return any object.

2.8 Dealing with Forms

Form values are submitted via a GET or POST method.

- GET - encodes the parameters in the url, some of you have already discovered this
- POST - encodes form values in the message body.

Flask stores these values in the args(GET) or the form(POST) of the request object and we can pull out the values and do whatever we wish with them.

2.9 Templates in Flask

Web pages don't need to be created in Python and we can create HTML pages and render them from Python instead. An HTML page in Flask is called a Template and is stored in the Templates folder. IT is rendered using

```
render_template(<pagename>)
```

Listing 28: Template Render

And these templates have Jinja2 scripting. That is, these are not just HTML and they can be scripted and accept data from Python. For example,

- Statements/scripts are written in $\{\%\%\}$
- Expressions evaluated to the output are written in $\{\{\}\}$
- Comments are written in $\{\#\#\}$

And this is useful because we can put shared elements into each page easily, i.e. headers, footers, menus etc. Furthermore, by keeping them in one place we can edit and have changes made in all pages.

2.10 Object Oriented Python

You can make python object oriented

```
from flash impot Flask
app = Flash(__name__)

class Person :
  name = "unset"

  def get_name(self):
  return self.name;

  def set_name(self, new_name):
  self.name = new_name;

@app.route('/oop')
def oop():
  p = Person()
  p.set_name("Jane")
  return p.get_name()
```

Listing 29: Object Oriented Python

Is an example of such, where we have a member variable called name and two functions get_name and set_name. To call methods on objects, we use the syntax

```
object.method(<args>)
```

Listing 30: Methods in objects

The . is used regularly in Python to represent hierarchies (classes, modules etc.) If you want to import constructors, then you wrap the name of the constructor around double underscore. i.e. __name__.
Lastly, there is no such thing as private or protected in python. Instead, in variables, we name them with _name which is protected and __name which is a private variable. Lastly, inheritance can be done using

```
class Admin(user):
```

Listing 31: Inheritance

This creates a subclass by using a class as a parameter to a class and you can also call parent class by

```
super().method()
```

Listing 32: Parent class

3 Databases

A web page can have the same structure: text,images,forms,tables,etc. even when the content is different. Web servers get the different data from a database. For databases, we use SQL. SQL stands for structured query language and is the language which you interact with Relational DataBase Management Systems (RDBMS). SQL is also a standard, however, database vendors extend their variants of SQL. It is possible to make a website with noSQL, and anther way to store is key-value pairs. No need for indexes and fast retrieval through other means e.g. hash-function. It is the same idea as dictionaries in Python.

Definition 3.1. Database

A data base is

- A structure collection of data
- Arranged into tables
- Data may be queried

In a to-do list, we would require to store

- Users
- Lists
- ListItems

And each of these will be stored in tables.

3.1 Management

Databases often require permissions to access the data, such as users, passwords, access rights and etc. Managing the database is a complex process and is beyond the scope of this web-dev module. For simplicity we will use SQL called sqlite3.

3.2 Database table

A table is placed to store data of the same type. You decide on the columns in the table, and these usually represent the data which you will store. The rows of the table are the data that is entered. A table has a fixed number of columns, but may have an unlimited number of rows. You can think of it like a spreadsheet.
For example

Example 3.1. Creating users table

Table 1: Example Database

Name	Datatype
email	varchar(50)
nickname	varchar(20)
biography	text
password*	varchar(25)

Note that it isn't the best way to store password but it is sufficient for current purposes

3.3 Relating data

For example, in an online forum, Users require to write leading to posts. Posts are a part of threads which belong to forums. Sites have many forums. Forum features must be view all posts by a user and get all threads in a forum. It should be able to find all users who posted in forum A, but not forum B on Tuesday.

3.4 SQL Datatypes

The SQL datatypes are

- integer - whole numbers
- real - decimal numbers
- text
- blob - binary data
- varchar(length) - fixed length test
- datetime - date and time
- date

3.5 Unique Rows

We designate a column in a table as the primary key of the database. It requires that each column is unique in that row. Most databases have an auto increment from 1. Designating a column as an integer primary key creates an auto incrementing value in SQLite.

3.6 SQL Operations

You can test SQL operations using the website https://sqliteonline.com The basic operations we are going to consider are:

3.6.1 Creating a table

```
160 CREATE TABLE <table name>(<column 1 datatype>, ...<columnN datatype>);
```

Listing 33: Creating table

Example 3.2. Creating users table

```
161 CREATE TABLE users(email varchar(50), nickname varchar(20), biography text, password
        varchar(25));
```

Listing 34: Creating Users Tabl

And if we want to have an auto incrementing primary key for our example

```
162 CREATE TABLE users(id integer primary key, email varchar(50), nickname varchar(20),
        biography text, password varchar(25));
```

Listing 35: Primary key

3.6.2 Inserting data

To insert data into the table we use the insert statement as follows:

```
163 INSERT INTO <table name> VALUES(value1, ..., valueN);
```

Listing 36: Inserting data

Example 3.3. Inserting data to table with ID

```
INSERT INTO users VALUES(NULL, "apc@dcs.warwick.ac.uk", "Adam", "Lecturer..", "
    123asdh2qd");
```

Listing 37: Inserting data to table with ID

3.6.3 Updating data

We can update existing database records using the UPDATE function as seen below:

```
UPDATE <table name> SET <column> = <new value> ... ;
```

Listing 38: Updating data

But if we require to run this we update the column for all rows in the database. Sometimes we need to select the right row

```
UPDATE <table name> SET <column> = <new value>, ... WHERE <column> = <value>;
```

Listing 39: Selecting the right row to update

Example 3.4. Editing existing values

```
UPDATE users SET password = "12345", nickname = "andrew" WHERE id = 2
```

Listing 40: Updating users

3.6.4 Deleting data

The delete command is defined as follows:

```
DELETE FROM <table name> WHERE <column> = <value>
```

Listing 41: Deleting data

Note that if you forget the WHERE close, all rows are deleted.

Example 3.5. Users deletion

```
DELETE FROM users WHERE id = 3
```

Listing 42: Deleting data from USERS

This will delete the rows with id $= 3$

3.6.5 Selecting data

We can limit our queries by adding WHERE clause.

```
SELECT email from users WHERE ("newsletter"="true");
```

Listing 43: Selecting data

i.e., this will create a table with only email column with people whose newsletter is true.

3.6.6 Joining tables

We can join tables by extracting column data from any table that we desire. In particular,

```
172 SELECT * FROM posts WHERE ("user_id" = 1)
```

Listing 44: Joining tables

That is, we could query them from multiple sources.

3.7 Flask - SQLAlchemy

3.7.1 Advantages and Disadvantages of SQLite3

Table 2: Advantages and Disadvantages

Advantages	Disadvantages
Public domain (free and open source)	Lack of support
Easy to install	Not fully ACID compliant
Easy to work with	Does not support concurrent writes

3.7.2 Connecting to a SQLite database

Connectiong to a SQLite database is simple and requires no usernames, passwords, hostnames or ports to deal with.
A connection to the database from the Flask 'app' can be made using

```
173 from flask_sqlalchemy import SQLAlchemy
174 db = SQLAlchemy()
175 db.init_app(app)
```

Listing 45: Importing SQLite

You may not run code by calling methods on newly created connection stored in db.
However, the 'app' needs to be configured first with some SQLAlchemy values, i.e.

```
176 app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
177 app.config['SQLALCHEMY_DATABASE_URI'] = f'sqlite:///{db_abs_location}'
```

Listing 46: Configuration

And the location of the database must be an absolute filename:

```
178 import os
179 basedir = os.path.abspath(os.path.dirname(__file__))
180 db_abs_location = os.path.join(basedir, 'sqlalchemy.sqlite')
```

Listing 47: Location of DB

3.7.3 Populating the database

The database requires a schema and some default data. For this, we use a Python file containing the schema and it can be found in Moodle. In particular, for the schema on forums in moodle we have:

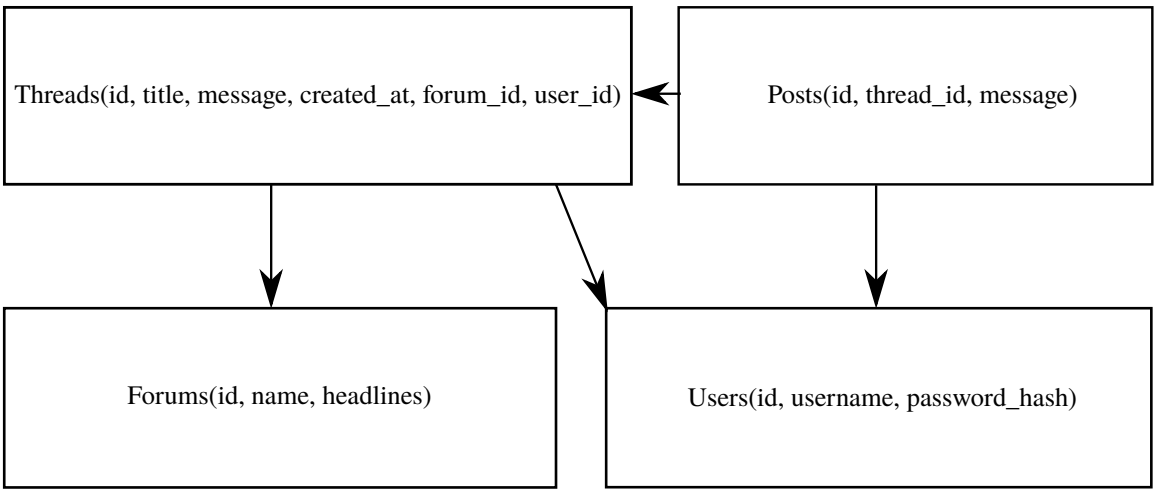


Figure 2: schema

3.7.4 Differences between SQL and SQLAlchemy

We use $String(n)$ instead of $varchar(n)$. Furthermore, when we create a table, we define it using the $Model$ class.

Example 3.6. Table using SQLAlchemy

```
        class Users(db.Model):
    __tablename__='users'
    id = db.Column(db.Integer, primary_key = True)
    email = db.Column(db.String(50))
    nickname = db.Column(db.String(20))
    biography = db.Column(db.Text())
    password_hash = db.Column(db.String(25))
```

Listing 48: Table using SQLAlchemy

And to actually create these tables, we then need to run the code

```
@app.route('/create')
def create_db():
  db.create_all()
  return "tables created"
```

Listing 49: Table creation

Alternatively,

```
with app.app_context():
  db.create_all()
```

Listing 50: Table creation alternative

3.7.5 Executing Queries

Once a connection is created you can query the database and there are two ways to submit a query with SQLAlchemy:

- $db.session$ which is suitable for result-less queries (add, delete, update)
- $Model.query$ which begins a query on a Table(Model). This is the same as SELECT.

3.7.6 Database Example (Adding data, linking)

A database example can be found in moodle can be executed using

```
194 from forumschema import db, Users, Forums, Threads, Posts, dbinit
195
196 db.init_app(app)
197
198 with app.app_context():
199   db.create_all()
200   dbinit()
```

Listing 51: Executing example

In particular, the file in case missing is as following:

```
201 # import SQLAlchemy
202 from flask_sqlalchemy import SQLAlchemy
203
204 # create the database interface
205 db = SQLAlchemy()
206
207 # a model of a forum for the database
208 class Forum(db.Model):
209     __tablename__='forums'
210     id = db.Column(db.Integer, primary_key=True)
211     name = db.Column(db.Text())
212     headline = db.Column(db.Text())
213
214     def __init__(self, name, headline):
215         self.name=name
216         self.headline=headline
217
218
219 # a model of a forum for the database
220 class Thread(db.Model):
221     __tablename__='threads'
222     id = db.Column(db.Integer, primary_key=True)
223     title = db.Column(db.Text())
224     message = db.Column(db.Text())
225     created_at = db.Column(db.DateTime())
226     forum_id = db.Column(db.Integer(), db.ForeignKey('forums.id') )
227
228     def __init__(self, title, message, created_at, forum_id):
229         self.title=title
230         self.message=message
231         self.created_at=created_at
232         self.forum_id=forum_id
233
234
235
236 from datetime import datetime
237 import time
238
239 def dbinit():
```

```
    db.session.add(Forum("Lab Sessions", "Discuss Lab Sessions here"))
    db.session.add(Forum("Lectures", "Discuss lectures here"))
    db.session.add(Forum("General", "Discuss anything else here"))

    posttime = datetime(2022,2,4,18,0)

    db.session.add(Thread("Lab 2 help", "Anybody good at CSS?",posttime,1))
    db.session.add(Thread("Can you smell doughnuts?", "I think so",posttime,1))
    db.session.add(Thread("Python win", "This is true",posttime,1))

    time.sleep(1)
    db.session.add(Thread("Great lecture", "Best. Lecture. Ever!", datetime.now(),2))
    time.sleep(1)
    db.session.add(Thread("I LOVE CS139", "It is so good", datetime.now(),2))

    db.session.commit()
```

Listing 52: File

3.7.7 Add data to the forum

The index pages requires a number of forums from the forum table, and to write a query to fetch everything from the forums we write

```
SELECT * FROM FORUMS;
```

Listing 53: Fetch All

However in SQLAlchemy instead we write this as

```
rows = Forums.query.all()
```

Listing 54: SQLAlchemy Fetch All

And when a set of results is returned, each row is a data object. The columns of the row can be accessed by name. In particular,

```
forums = Forums.query.all()
s = ""
for entry in forums:
  s += f"{entry.name}<br>"
```

Listing 55: Access of data

3.7.8 Serialisation in Python

Serialisation enables data to be moved in and out of variables, including objects. This is useful if you want to store complex data in your database or a file. Serialisation in Python comes in two forms:

- import pickle
- import json

It is useful for transferring the data.

3.8 GET request and response

Request a resource e.g. a file
Response has 2 sections

1. A header containing the status of the request
2. An optional body containing the response to the request

3.8.1 Response Codes

The response codes for the requests are:

- $2\#\#$ coes are for successful responses,
- $3\#\#$ are for redirection-related responses,
- $4\#\#$ are for client errors,
- $5\#\#$ are for server errors.

where hashtags are numbers. However, it is particular important to remember the following response codes:

- 200 - OK Success
- 301 - Moved Permanently
- 404 - Not found
- - Internal server error

4 Internet Protocols

Computers need to communicate in the same language and TCP/IP exists in every operating system and this allows for such communication. However, be aware that networking is an entirely separate discipline and web dev just needs an awareness of these protocols - it's the curtain between your web-app and the rest of the world.

4.1 Layers of protocol

1. HTTP: Hypertext Transfer Protocol - used for web communication
2. FTP: File Transfer Protocol - used for transferring files between computers
3. POP/IMAP/SMTP - Email-related protocols for transferring and storing mail
4. DNS: Domain Name System - protocol used for resolving domain names to IP addresses

4.2 IP Address

4.2.1 IPv4

IP addresses needed by every computer on the internet. For example, IPv4 uses $4 \times 8-$bit numbers.

- 127.0.0.1 - localhost
- 192.168.*.* and 10.0.*.* are for private networks
- There are other reserved addresses and ranges

All IPv4 numbers, i.e., 256^4 addresses are gone.

4.2.2 IPv6

We now use IPV6, for which uses $8 \times 16-$ bit numbers which results in 340 undecillion addressses. It looks like, for example: $2001 : 0db8 : 0000 : 0000 : 0000 : ff00 : 0042 : 8329$

4.3 Network Address Translation (NAT)

A device connects to a router's internal address and then router opens a port and connects to the external resource. Data received is then passed through the router to the originating device

4.4 Domain Name System (DNS)

Domain name system servers are a 'contacts list' for internet servers. It translates human readable name to an IP address and it is a hierarchic distribution naming system for resources on the internet.

Example 4.1. Domain Namespace

$$\underbrace{\text{server1}}_{\text{Fourth-level domain}} . \overbrace{\text{www}}^{\text{Third-level domain}} . \underbrace{\text{funwebdev}}_{\text{Second-level domain (SLD)}} . \overbrace{\text{com}}^{\text{Top-level Domain (TLD)}} \tag{1}$$

That is, it is a tree of domain spaces. Zero or more resources records exist at each node of tree and the root zone is the TLD, and is controlled by the Assigned Numbers Authority. Registrars assigned domain names beneath the TLD and check with interNIC, the canonical source for the domain names.

4.5 Resolving a domain name

A network host is primed with the addresses of the root name server. The client sends a query to a root name server and query the top level domain server for the authoritative nameserver for the domain. Last step repeats for each additional subdomain request.

Example 4.2. Wikipedia

The DNS recurses asks for the IP address of the DNS $www.wikipedia.org$. To do so, it connects to the root nameserver. The rootname server redirects to the .org nameserver. The .org nameservers then redireects to wikipedia as required.

4.6 Domain Name Caching

Not every request from every web user is sent to root nameservers as this would take a massive load. In partice, NS record caching is used to overcome this throughout internet. For example, your ISP will have the IPs of common domain names like google.com and bbc.co.uk to avoid overload. Time to live (TTL) dictates how long a DNS record should remain valid.

4.7 Uniform Resource Locators

$$\overbrace{\text{http}}^{\text{protocol}} : // \underbrace{\text{www.funwebdev.com}}_{\text{Domain}} / \overbrace{\text{index.php?}}^{\text{path}} \underbrace{\text{page=17}}_{\text{Query String}} \# \overbrace{\text{article}}^{\text{fragment}} \tag{2}$$

4.8 IP Address, Domain Name and URL

DNS is case insensitive and so the URL is also case insensitive. IP Address can be used for the domain. Ports, on the other hand, can also be given. We add : (colon) in an integer to specificy the port i.e.,

http://www.funwebdev.com:5000/index.php

4.9 Path

When making an HTTP or FTP request, it is a file that you are requesting and / is the root folder on that server, just like your local file system. If a file is not specified, then webserver will choose a file for you, by default this is usually the $index.html$. The path IS case sensitive.

4.10 Query string

This provides information to the web servers and is a sequence of key-value pairs after a question mark, separated by ampersands. For example

Table 3: Data

Key	Value
page	12
colour	red

Would be represented by

...index.php?page=12&colour=red

We can access data that is passed via get or post methods through the request object in Flask and conventionally we use

- GET when requesting data : /forum?page=1
- POST when sending data: forms

4.11 Fragment

Fragment is a method of requesting a portion of a page and used by the HTML anchor tag $< a >$, for example

...index.php#article

4.12 Hypertext Transfer Protocol

HTTP is an application protocol for hypermedia information systems and it standardises the request-response message. Its development coordinated by Internet Engineering Task Force (IETF) and World Wide Web Consortium (W3C). Version HTTP/1.1 is in common use. HTTP/2 published in 2015. HTTP/3 drafted in 2020.

4.13 Web browser

The resource provided by the web server will be raw data, usually text or encoded binary. A web browser interprets the data and displays it in a recognised form:

- It is converting HTML page data and rendering it as a readable web page
- Each browser has different method of doing this, but they 'loosely' conform to most of the standard

Local browser caching also exists, for which reduces the amount of data being transferred to the root nameservers. These also contain additional features which include, but are not limited to, phishing detection, URL autocomplete etc.

4.14 Stateless-ness

HTTP is a stateless protocol and in a local process, the computer knows about the current state of memory and instructions. In a web process, the server treats every request as a completely new request with no historical state. Transfer the next 'state' back to the end user for their use. For this, we have 3 solutions:

- Query String
  - URL Path Rewriting
- Cookies
- Sessions

4.14.1 Query String

Because query string was mentioned earlier, we will focus on the URL path in here instead. Dynamic URLs i.e., query strings are pretty essential part of web application development. Static URLs, on the other hand:

- The appearance of search terms within the URL does seem to improve its relative position in SEO
- Another benefit to static URLs is that users tend to prefer them
- Can use URL rewriting to make static URLs be dynamic

4.14.2 Cookies

Cookies is data unique to the client that is created by the server. The server sends a cookie back to the client when the first request is made and the client incorporates that cookie in subsequence requests. The client incorporates that cookie in subsequence requests. These usually expire when the browser session ends and are restricted to the domain e.g., mysite.com. In flask:

- The server-side (Flask) sets the cookie in the response

- make$_{r}esponsetocreatetheresponseset_{c}ookietoaddthecookietotheresponseheader$

```
@app.route('/setcookie')
def methoud():
  username=request.args.get('name')
  message=request.args.get('mesg')

  resp = make_response(render_template_string("{{message}}", \
    message=f'Responding to "{message}" sent by "{username}"'))

  resp.set_cookie('usercookie', username, max_age=60)

  return resp
```

Listing 56: Cookies

This cookie grabs the name and message from the query string and generates a response and inputs the defined message.
For checking cookies we do a server-side access on request.cookies for data it set last time. In particular,

```
def checkcookie():
  name = request.cookies.get('usercookie')
  if name is None:
    name = 'not found'
  return f'<h1>usercookie: {name}</h1>'
```

Listing 57: Checking Cookies

A demo of this exists in the CS139 moodle page.

4.14.3 Sessions

Sessions store data just for the time that a user is active in the website. This data is synchronised securely with the server and server can store the session information with other distributed methods e.g. Memcache. It is especially useful for user authentication: login and logout.
In flask, we use session object to save information for the session. In particular,

```
from flask import Flask, request, redirect, render_template, session
app = Flask(__name__)
app.secret_key = 'any string you want'

@app.route('/')
def method():
  if 'usename' in session:
```

```
    username = session('username')
    return f'Logged in as {username}<br>'+\
      '<b><a href="/logout">Logout</a>'

  return 'You need to log in <br>'+\
    '<b><a href="/login">Login</a>'
```

Listing 58: Session Flask

We can store data in a session using a dictionary and put in values by writing

```
session['key'] = value
```

Listing 59: Session object

And retrieve them with

```
value = session['key']
```

Listing 60: Session object retrieval

Python can test dictionary membership with in $< value >$ in session. So for example, we could do

```
if 'user_id' not in session:
  return redirect('index');
```

Listing 61: Example

Which would test for a userid key set in the session. Finally, to close the session we can simply pop() the individual session keys. Or we can clear() all the keys. In fact, flask provides a way for a session to end when the browser closes also by using

```
session.permanent = False
```

Listing 62: Flask browser close session

And to summarise:

- To create a session in Flask add entries in the session object
- Create a session variable in the object e.g.

```
  session['username']='Adam'
```

Listing 63: Creating session variable

- Release the session variable using pop() e.g.

```
  session.pop('username', None)
```

Listing 64: Release session

- and to finally set the secret key:

```
  app.secret_key = 'any string you want'
```

Listing 65: Secret key

5 CSS

5.1 Cascade

Styles are applied in the following order:

- Browser default
- External style sheet
- Internal style sheet
- Inline styling

However, in terms of inheritance, only some properties are inherited because it is complicated.

5.2 Syntax

The commands modify the styling of your HTML, and for instance,

```
h1 {
  color: blue;
  font-size: 12px;
}
```

Listing 66: CSS example

5.3 Selectors

Notice how all $< h1 >$ tags will be modified the same way. A class selector can be used to modify just some HTML elements: we use fullstops to denote a class. An ID selector can e used to modify unique HTML element as well. These are done by denoting classes and then modifying the style of these classes

```

```

Listing 67: Class

```

```

Listing 68: ID

```

```

More information can be found in https://www.w3schools.com/cssref/css$_s electors.asp$

5.4 Pseudo-elements

A pseudo-selector targets a particular state or relationship e.g. the $< a >$ tag. An example of pseudo-element is the following:

```
<style>
p::selection {background-color:green;}
</style>
```

Listing 69: Pseudo selector

5.5 Box Model

All elements on a page are boxes. They all have width and height. Block level elements start and end with a new line. For example, $< h1 >$, $< p >$, $< table >$ are all block level elements. That is, they would become separate lines.
There are also inline elements, that is, boxes which are not in separate lines. Examples include $< a >$, $< img >$ etc.

6 Security

6.1 Login

To prevent unwanted access to a web page, we could add something along the lines of

```
308 from flask_login import login_required
309
310 @login_required #denotes that login is required for a webpage
311 login_manager.login_view="login" #where to redirect if not logged in
```

Listing 70: Login

6.2 Sanitising Query Strings

Sometimes query strings are not as expected, that is:

- Parameter may not exist or,
- Parameter does not contain a value or,
- Query string parameter value isn't correct type or is out of range or,
- Value is required for a database lookup but the provided value does not exist in the db

These are expected errors and require error checking.

6.3 HTTPS

HTTPS was used until quite recently when it changed in 2011. All requests and all responses used plain text by default and then HTTPS was introduced. It is a protocol for secure communication over a network with an extra protocol on HTTP called SSL/TLS protocol. In other words, information is not encrypted.
A client and server build a TCP (point-to-point) connection. The client sends a number of specifications to the server:

- What version of SSL/TLS is being executed
- Which cipher suites it wants to use
- Which compression method to use

The server selects the highest SSL/TLS version supported by both, picks a cipher suite and optionally a compression method. The server sends its certification and it must either be trusted explicitly OR a party that the client trusts. After verifying the server's certification, an asymmetric key is exchanged. Both client and server setup a key for symmetric encryption and the client tells the server that all communication must be encrypted. The server verifies the MAC address (used for authentication) is correct and the message can be correctly decrypted. Handshake is then complete.

6.3.1 Disadvantages

Speed - setting up the encrypted connections can be expensive. Furthermore, some servers have dedicated encryption hardware to speed up the encryption operations.

6.4 The role of certificates

Creating a secure connection with a server would require knowing its public key. However it would be impractical to store the public keys for every server in the internet. The solution is certificate authorities (CA). In other words, a third party would be required to store all public keys, in order to ensure that the source is legitimate. The issue is, however, that anyone can become a CA and you may be the only one that trusts it.
HTTPS should be used whenever sensitive data is being passed between client and server. Was used solely for e-commerce but now is being used widely.

6.5 Man-in-the-middle attacks

Data was sent via HTTP may be subject to man in the middle attack. A man in the middle attack consists of the following stages:

- Alice and Bob would like to have a conversation with Michael helping them
- Bob sends his key to Michael and Michael responds to Alice with his own key
- Alice encrypts her message using Michael's key, believing it to be Bob's. Michael can read the message and forward a modified message using Bob's key.

6.6 SQL Injection

Even with encryption connections, nefarious instructions can still be transmitted. More specifically, snippets of SQL are sent via user submitted fields to attempt get the server to execute the rogue SQL. For example,

```
312 qry = f"SELECT * FROM employees WHERE name = '{username}';"
```

Listing 71: SQL Injection

You would not want to have similar code in your TODO list applications!
Now we consider a username of ' or '1'='1 This username would cause your program to create this string query:

```
313 SELECT * FROM employees WHERE '' or '1'='1'
```

Listing 72: SQL Injection Name

And the output from this would be all employees in the database.

6.6.1 Solution: Escape SQL Characters

You could evaluate and escape all user input. For example, username = username.replace("","""") but replacing characters is really bad as there are too many special characters and scenarios to consider. In other words, one forgotten parameter is then a vulnerability.

6.6.2 Solution: Parameterised statements

The query string is not built at runtime, it is written with placeholder values to which variables are bound. Placeholders are restricted by type, and thus invalid if an incorrect strings is passed to them. Values are bound to placeholders.

6.7 Database permissions

Restrict access to system tables, we need to limit exposure of tables on a per-user basis. For RDBMS with permissions (not SQLite).

6.8 Session Hijacking

There are a number of ways that an attacker can hijack a user session which include

- Session sniffing - All unencrypted network traffic is vulnerable to network traffic. Cookies which store a user's data could be reconstructed and then used by an attacker to impersonate the user
- Session fixation - Attacker sets the user's session ID to one known to him. Then waits for the user to log on.

6.9 Cross Site Scripting (XSS)

XSS allows users to inject client side script into webpages. It is accounted for about 84% of all security vulnerabilities documented by Symantec in 2007. Non-persistent data provided by client is used immediately by server-side scripts without proper sanitisation, thus content is uploaded is included back into the pages. It is essentially like crafting a GET request. For example,

```
username = request.args.get('name')
hello_html = f"<p>Hi {username}</p>"
```

Listing 73: XSS

This is an awful script because the input is no filtered and the output is not escaped.
Because the remote script is run on your page in the browser, it has access to everything. It can steal cookies for session hijacking and change the HTML in the page, e.g. add forms for payments.
To deliver XSS attack, simply the user a link, with the <a href set to the crafted URL. Content is inserted into pages, appearing as if from the correct site.

6.9.1 Solution: Filtering the input using jinja2 markup

```
from jinja2 import Markup
badstring = '<script>alert("Hello from XSS") </script>'
badstring = Markup(badstring).striptags()
```

Listing 74: Markup

This will remove html and similar tags We can also escape the output to be extra safe by

```
from jinja2 import escape
badstring = '<script>alert("Hello from XSS") </script>'
displaystring = escape(badstring)
```

Listing 75: Escape

You can also use Jinja2 filters such as

```
{{ username | striptags | title }}
{{ username | escape }}
```

Listing 76: Jinja2 Filters

and if something is definitely safe then

```
{{ no_xss_here | safe }}
```

Listing 77: Jinja2 Safe Filter

6.10 Convert special characters to HTML

Flask with Jinja2 automatically escapes variables so they will show up correctly on a template. However, in HTML {{ and }} characters will be interpreted. Avoid this with either

{{'{{'}} text {{'}}'}}

Or otherwise use the {% raw %{{ text }}{% endraw %} You should sanitise your data only if you output it. Store the data as raw as you can and you may want to output it in a different format later on.

6.11 Persistent XSS

Data provided by attacker is stored blindly, without proper escaping. You could insert script tags into a text area which would get rendered later into any page where that text appears. The script may not have any visible side effects, but could parse information from the page and inject HTML content freely.

7 JavaScript

7.1 Javascript

Definition 7.1. Javascript

Javascript is an interpreted programming language and it allows you to write programs that run on the client side rather than the server side. It is implemented to some degree in all browsers. However, it can also be run in the server side.

The purpose of javascript is to dynamically create HTML. That is, changing existing HTML on a page. It can also create response to user events such as clicks, drags, key presses, touches and multitouches. It can also be used to validate input and uploading data in the background. To insert javascript into a page it must be inserted between <script> and </script> tags. These javascript segments are executed when the page loads. Some basic syntax:

```
<script type="text/javascript" charset="utf-8">
document.write("Hello") // prints "hello" in the webpage
alert("Hello") // creates a "Hello" pop-up on your browser
console.log("Hello") // prints a "Hello" in your browser's console
</script>
```

Listing 78: basic javascript syntax

7.1.1 Variables

Variables in Javascript are loosely typed, meaning that we do not need to declare the datatype when we declare the names. e.g.,

```
<script type="text/javascript" charset="utf-8">
var x = 2;
var y = " Hello";
document.write(x+y);
</script>
```

Listing 79: variables

There are a few rules:

- Variables are to be prefixed with var
- Names must start with letters, $ or _
- Names are case sensitive

7.1.2 Datatypes

Though we don't express them directly, Javascript has these types:

- Strings - text inside single or double quotes
- Numbers - both decimals and integers are of type number
- Boolean - true/false
- Arrays

- Objects
- Null - an empty variable
- Undefined - a variable which is not set
- Function

7.1.3 Javascript Arrays

Arrays are zero indexed like in Java. We declare an array by calling new Array();

```
var animals = new Array();
animals[0] = "Dog";
animals[1] = "Cat";
animals[2] = 4;
```

Listing 80: Array

Arrays can store different types of objects. You can also ask for length of an array like in java by invoking

```
var size = animal.length;
```

Listing 81: length

and you can ask for the index of a given value. Note that if duplicates exist it will give the smaller number.

```
var index = animals.indexOf("Cat");
```

Listing 82: index of array

7.1.4 Javascript objects

Javascript objects are essentially collections of properties. These can be though of as key-value pairs. We can create them inline such as

```
var person = {first_name:"Adam", last_name:"Chester"}
```

Listing 83: Javascript Objects

And values can be functions. Or have a constructor function:

```
function Fruit(name,colour,countryOfOrigin) {
this.name = name;
this.colour = colour;
this.countryOfOrigin = countryOfOrigin;

this.description = function() {
  return "This" + name + "is " + colour + "from" + countryOfOrigin;
};
}
var f = new Fruit("apple","green","United Kingdom");
```

Listing 84: Javacsript constructor

Notice how it is similar to a Java class. However, you can also arbitrarily create new properties on objects, for example. for our previous definition we could just write

```
f.foo = "pink unicorns";
f.fly = function() {
  console.log("Flying");
  //Other magic stuff
}
```

Listing 85: Arbitrary properties

7.1.5 Javascript functions

Functions are defined using the function keyword.

```
function sayHello() {
  alert("Hello");
}
```

Listing 86: functions

which may also take parameters to create something such as

```
function sayHelloTo(name) {
  alert("Hello " + name);
}
```

Listing 87: parameters

7.1.6 Control Flow

If works as in java

```
if (x == 6) {
}
```

Listing 88: if statement

If else is also the same as in java

```
if (x == 6) {
  alert("Found a 6");
} else {
  alert("No 6");
}
```

Listing 89: if else statement

The for loop similarly follows a java style

```
for (i=0;i<10;i++) {
  alert(i);
}
```

Listing 90: for loop

The while loop also

```
while (x<5) {
  alert("Hello");
  x++;
}
```

Listing 91: while loop

7.1.7 Finding an element in Javascript

There are a number of ways to refer to a HTML element in JavaScript:

- by their ID

```
377     document.getElementById("mainContent");
```

Listing 92: ID Reference

- by their tag name

```
378     document.getElementsByTagName("p");
```

Listing 93: Tag name reference

- by their class

```
379     document.getElementByClassName("subtitle");
```

Listing 94: Class reference

7.1.8 Changing HTML content

Changing HTML content is most easily done by setting the innerHTML property of an element, that is

```
380 document.getElementById("mainHeading").innerHTML = "Changed it";
```

Listing 95: innerHTML

You may also store elements in variables to avoid having to call the document.getElementBy...

```
381 var heading = document.getElementById('heading');
382 heading.innerHTML = "Help";
```

Listing 96: innerHTML 2

7.1.9 Changing attributes of an element

In addition to changing the contents of an element, we can also alter its attributes. In the following example, the smiley.gif is replaced with landscape.gif:

```
383 <!DOCTYPE html>
384 <html>
385 <body>
386 <img id="image" src="smiley.gif">
387 <script>
388 document.getElementById("image").src="landscape.gif";
389 </script>
390 </body>
391 </html>
```

Listing 97: Attribute change

7.1.10 Changing Styling of an Element

The style applied to an element can be accessed through the style property syntaxed as

```
document.getElementById("name").style
```

Listing 98: Style property

We can set/change styling setting the attribute of the style that we wish to modify, for example

```
document.getElementById("name").style.color="blue";
```

Listing 99: Styling change

And this would change the colour of the element in the page.

7.1.11 Responding to events

- A user clicking the mouse
- The web page loading
- Mouse moves over an element
- Form submitted
- User keypress

https://www.w3schools.com/jsref/dom$_{o}bj_{e}vent.asp$

7.1.12 Event Handling Code

To handle an event, the event type and the code that we wish to run is added to the element

```
<h1 onclick="this.innerHTML='changed'">Click Me!</h1>
```

Listing 100: onclick

If there is a lot of code to execute on an event then this approach may become cumbersome. As an alternative, we can place the code that we'd like to call into a function call the function on the onclick handler. For example

```
<!DOCTYPE html>
<html>
<head>
<script>
function changetext(id){
id.innerHTML="Oooops!";
}
</script>
</head>
<body>
<h1 onclick="changetext(this)">Click on this text!</h1>
</body>
</html>
```

Listing 101: onclick

We can attach event handlers to elements programmatically. That is,

```
<h1 id='header'>Click me!</h1>
<script>
document.getElementById('header').onclick=function() {alert("hello");}
</script>
```

Listing 102: Event handlers

7.1.13 Form Validation

Forms have a special event handler called onsubmit. We can use the onsubmit handler to call a validation function, which will check if the form is valid before it is submitted to the server.

7.2 The HTML DOM

The HTML document object model (DOM) is a tree of the objects within a document. With a programmable object model Javascript can alter any element it wishes to.

8 jQuery

As we have learned in Javascript, we can add it using the <script> tag, however, this has several problems:

- It cannot be reused across pages
- It cannot be cached by the browser

We can refer to external javascript files in a similar way we do to CSS files by

```
412 <script src="myjs.js" type="text/javascript" charset="utf-8"></script>
```

Listing 103: External script

The src attribute is used to express which javascript file should be loaded.

Definition 8.1. jQuery

jQuery is a JavaScript library that was launched in 2006. It is used in over 80% of the 10000 most visited sites on the web.

Its main use for the fact that it is used to manipulate the DOM, to do event handling, to do animation and lastly AJAX. It is the best of all works across browsers. We can include jQuery by the reference

```
413 <script src="js/jquery-3.6.0.min.js" type="text/javascript" charset="utf-8"></script>
```

Listing 104: jQuery inclusion

In most scripts, you will want to your scripts to execute after the DOM has loaded. The browser fires an event when this occurs. We'll use jQuery's ready function to kick things off for us. To use jQuery functions, we must invoke a jQuery object. This is done by using the $ function. In the case of our DOM ready event, it looks like:

```
414 $(document).ready( function() {...});
```

Listing 105: jQuery ready

We call the factory method $ which creates a jQuery object that we can call other functions on - in this case ready. jQuery 3.0 simplifies this to just

```
415 $( function() {} );
```

Listing 106: jQuery 3.0

In other words, $ is a variable name of a function that is defined in jQuery, as mentioned before, we are able to use symbols for functions.

8.1 Objects

When we use the jQuery factory function, we are creating a jQuery object. The object wraps the HTML entity and provides it with additional functionality. Nearly all jQuery methods return a jQuery object, so methods can be chained. We need jQuery objects when we work with the DOM across multiple browsers is a pain in the ass.

```
var target = document.getElementById("target");
target.innerHTML = "<td>Hello <b>World</b>!</td>";
```

Listing 107: Table

Setting the HTML of a <tr> as we saw last lecture. This used to fail under some version of internet explorer. Manipulating the DOM often requires complex navigation techniques. Imagine inserting a new element after an existing one:

```
var target = document.getElementById("target");
var newElement = document.createElement("div");
target.parentNode.insertBefore(target.nextSibling, newElement)
```

Listing 108: new element

We can get these magical objects using

- Use factory method - and a little CSS i.e., jQuery can use CSS selectors to identify elements
- $\$('H1')$ - gets all of the $H1$ elements on a page
- $\$(\text{"}\#header\text{"})$ - gets the element with an id of header
- $\$(\text{".}title\text{"})$ - gets the elements with a class of title

Because we can get multiple queries for our selectors, we need a method of selecting them:

```
$('H1')[0]
$('H1:first')
$('H1').first()
```

Listing 109: jQuery selector

All select the first output. However, note that in raw Javascript you can use the innerHTML attribute. jQuery objects all have a html() method which returns the html of an element. For example,

```
var html = $('#sample').html();
// or something like
$('#sample').html("new html here");
```

Listing 110: jQuery HTML

8.2 Adding/removing from the DOM

We can add content at the end of an element using the append() method:

```
$('#wrapper').append("More content");
```

Listing 111: Append

We can also add content at the beginning by using prepend(). We also remove an element through remove():

```
$('#wrapper').remove();
```

Listing 112: Remove jquery

8.3 DOM traversal

jQuery includes the following methods for traversing the DOM:

- parent() - gets the element's parent
- children() - gets the element's children
- siblings() - gets the element's siblings

8.4 jQuery Styling

We can also manipulate the CSS of an element through the following jQuery methods:

- addClass("newclass") - adds a class to the element
- removeClass("newclass") - removes a class
- toggleClass("newclass") - adds/removes a class depending on current state
- hasClass("newclass) - tests if class is applied to an element

We can also access the size of elements through

- height()
- width()

We can also access css values using the css method such as

```
429 var bg = $('p').css('background-color');
```

Listing 113: CSS values

And similarly, we can change the css values using second parameter

```
430 $('p').css('background-color',"green");
```

Listing 114: CSS values changing

8.5 jQuery objects are not live

Note that if we selected <p> elements on a page:

```
431 var paragraphs = $('p');
```

Listing 115: selecting p

And add some more <p>, paragraphs would still be the same - it does not know that new elements were created. This must be refreshed by simply

```
432 paragraphs = $('p')
```

Listing 116: refresh

8.6 Events in jQuery

We saw how to couple events to user actions in the last lecture by

```
<button onclick="alert('Hello')"> Say hello </button>
```

Listing 117: Events

But there are a couple of problems with this:

- The viewcode (HTML) is tied to the event code (JS). Updating the script means updating the HTML
- It is not scalable

jQuery makes it easy to bind events to HTML elements. Therefore to bind an event to an element we

1. Give the element an ID
2. Tell jQuery that something should happen on this event type
3. Give jQuery the code to run when the event occurs

In HTML

```
<button id='hello'> Say hello </button>
```

Listing 118: event

And in JS we would have

```
$(function() {
  $("#hello").click(function( event ) {
    alert("Hello");
  })
});
```

Listing 119: event

Some events you can edit are:

- focusin - user clicks on the element
- focusout - user leaves element
- keydown
- keyup
- keypress - an up/down combination
- click
- dbclick
- hover
- and much more

We can add multiple handlers for a single event types and there is no specific orders at which they would occur. The browser makes no guarantees about the order which handlers are called.

8.7 Event Bubbling in JS

Recall that DOM elements can be nested within each other. In the previous example the button was nested under the body element. The main idea is that when an event is triggered on an element, it is then triggered on the parents of that element. The event is said to "bubble up" through the parents. That is, for example, consider background with a box with a button in it. If there are events on all three of these elements and the client presses on the button, because the box is a child of background and button is a child of box, all 3 events will trigger. This can be stopped by using the command

```
event.stopPropagation();
```

Listing 120: stopPropogation

after your function event.

8.8 Event Capturing

In all browsers there are two stages of event processing:

1. The event goes down (from root to child)
2. and then bubbles up (from child to parent)

This course will not worry about capturing.

8.9 Animation

jQuery makes it easy to add in animation to your HTML pages. This should be done with care.

9 AJAX (Json and XML)

Definition 9.1. AJAX

Asynchronous JavaScript and XML (AJAX) is a collection of inter related web technologies. IT is a client side technique that allows users to send actions to a server and receive updates to the page without a full refresh.

Client requests are sent asynchronously:

- They are made in the background
- The return of data may happen at an unknown point in the future

The user can do other things while the request is happening and they perceive a faster response. Note that some requests may be sent synchronously if desired.

9.1 Extensible Markup Language (XML)

Designed to be human and machine readable and is designed to emphasise simplicity, generality and usability over the internet. It is textual data format and some examples include:

- RSS
- Atom
- SOAP

XML is much like HTML, both have start and end tags for elements and both may have attributes. An element may contain:

- other elements
- text

- attributes
- All of the above in any combination

Valid names may contain letters, numbers and other characters and cannot start with a number or a punctuation and cannot start with letters XML and cannot contain spaces.

9.1.1 Attributes

Attributes provide additional information about the data. For example, they may contain metadata i.e., which version of the software created the elements etc. This is not part of the data but may be helpful to software developed to parse the files. Attributes must be quoted.

9.1.2 XML Sample

All XML files must include a root node and records are stored under this node

```
<library>
  <book id='1'>
    <title>Much Ado About Nothing</title>
    <author>
      <forename>William</forename>
      <surname>Shakespeare</surname>
    </author>
    <publicationdate>
      <day></day>
      <month></month>
      <year>1543</year>
    </publicationdate>
  </book>
  <book id='2'>
    <title>Programming in C</title>
    <author>
      <forename>Stephen</forename>
      <surname>Kochan</kochan>
    </author>
    <publicationdate>
      <day>12</day>
      <month>6</month>
      <year>2003</year>
    </publicationdate>
  </book>
</library>
```

Listing 121: Library database

XML has no pre-defined tags, instead, we can define these data structures.

9.1.3 Well-formed XML

XML is said to be well-formed when:

- The document contains only unicode characters
- none of the special characters appear e.g. <, +
- Begin, end and empty element tags are correctly nested
- Element tags are case-sensitive
- A single root element contains all other elements

9.1.4 Defining data in XML

When creating XML document, you may define any elements that you require. Consider the scenario where you are tasked with defining the XML markup for an online music services. Define appropriate tags to represent an album and write a sample XML document for an album

9.2 JavaScript Object Notation (JSON)

JSON was developed for the purpose of sending structured data over the network. Smaller than the equivalent XML, faster and easier to parse and it is an alternative to XML.

9.2.1 Datatypes

JSON's basic datatypes are

- Number
- String
- Boolean
- Array
- Object (unordered collection of key value pairs)
- null (empty)

9.2.2 Syntax Rules

- Data is in key-value pairs
- Data is comma separated
- Curly braces hold objects
- Square brackets hold arrays
- Whitespace is not relevant - use it to make things easier to read

9.2.3 Example

```
{
  "employees" : [
    {
      "employee_number" : 12312,
      "first_name" : "Adam",
      "last_name" : "Chester"
    },
    {
      "employee_number" : 12313,
      "first_name" : "Matt",
      "last_name" : "Davies"
    }
  ]
}
```

Listing 122: Example

9.2.4 How AJAX Works

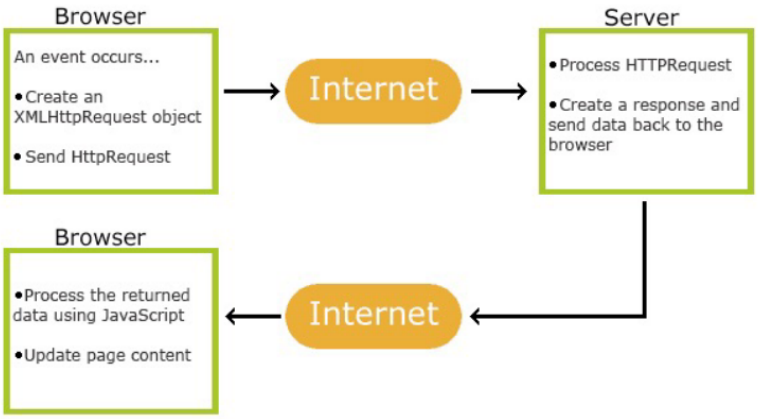


Figure 3: How AJAX Works

9.2.5 The XMLHttpRequest Object

All modern browsers support the MLHttpRequestObject. It is this object that exchanges data behind the scenes. In javascript, this object can be created using

```
481 var xmlhttp = new XMLHttpRequest();
```

Listing 123: XMLHttpRequest

9.2.6 Sending an XMLHttpRequest to the server

To send a request to the server, we must first open the request

```
482 xmlhttp.open("method","url","asynchronous");
```

Listing 124: open request

- Method - GET or POST to the URL
- URL - the resource we're interested in
- asynchronous - true/false

Secondly we use the send function

```
483 xmlhttp.send(); // for GET requests - url contains the query string
484 xmlhttp.send(string); // for POST requests
```

Listing 125: Send function

This sends request to the server

9.2.7 Keeping track of the request

When the request is sent, an onreadystatechange event is triggered every time the state changes. The readyState attribute holds the status of the event as an integer

1. request not initialised
2. server connection established

3. request received
4. processing request
5. request finished and response complete

This event fires 5 times during the request for update.

9.2.8 Processing the server response

The response from the server can be accessed through the following attributes of XMLHttpRequest:

- responseText - the response data as a string
- responseXML - the response data as XMl

The status of the request can be obtained through the $status$ property.

9.2.9 Example

Here we concern ourselves with readyState 4 and a status of 200 (OK).

```
485 xmlhttp.onreadystatechange=function() {
486   if (xmlhttp.readyState==4 && xmlhttp.status==200) {
487     document.getElementById("myDiv").innerHTMl=xmlhttp.responseText;
488   }
489 }
```

Listing 126: Asynchronous response

9.2.10 Handling JSON responses

To create a javascript object from a JSON resposne, you can call the JSON.parse function. This function is included as part of all modern browsers. The complementary function is JSON.stringify which creates a JSON string from a javascript object.

9.2.11 What about jQuery

Writing raw javascript to do these AJAX requests is quite longwinded. jQuery has some useful shorthand notation that we can use to make AJAX requests. jQuery has an ajax function that we can use to make requests a little more palletable.

```
490 $.ajax({
491   url = "/update"
492   data: "{first_name: 'Andrew', last_name: 'Hague'}",
493   success: updatePage();
494 });
```

Listing 127: AJAX in jQuery

The hash of options here includes

- url
- data - the data to be passed to the server
- success - a callback function to execute when request completes

Ajax GET requests have a shorthand notation:

```
$.get(url,sucess_callback);
//example
$.get("/status", function() {
  $('#status').html("OK");
});
```

Listing 128: GET Ajax jQuery

And for post we use

```
$.post(url,data,success_callback);
//example
$.post("/createTodo",
  {"title":"Buy Milk", "done":false},
  function(data) {
    var item = JSON.parse(data)
    var clone = $('#todos tbody tr').last().html();
    clone.children().first().html(item.title);
    $('#todos tbody').append(clone);
    }
);
```

Listing 129: POST Ajax jQuery

10 Web Development Hardware and Design

10.1 Usability and Accessibility

Web developers must:

- Work for an employer/client
- Be profitable
- Collaborate with non-programmers
- Work to deadlines
- Maintain a reputation

10.1.1 Golden rules of Interaction design

We must understand the computer - its limitations, capacities, tools and platforms. Understand people - their psychology, social aspects and human error. And we need to understand human error - the err is to be human. When an aspect of an interface is unclear, it is often blamed on user error and an additional paragraph is added to the help documentation or help screens. This means

Definition 10.1. User-centered design

1. Put the user first
2. Keep the user in the centre
3. Remember the user at the end

10.1.2 Software development lifecycle

1. Requirements capture
2. Design
3. Implementation
4. Verification
5. Maintenance

10.1.3 Requirements capture

Elicitation - we need to work with stakeholder (product owner) and conduct interviews and questionnaires to ask users. Ethnography - we need to observe users and lastly consider scenarios and stories - start, normal activity, what goes wrong, what else can be happening and how it ends.

10.1.4 Types of requirements

Functional
A service provided by the system and what it should not do
Non functional
Constraints on the functions and what can affect the whole system

10.1.5 Specification

What the system does, not how it does it. Describe using:

- Natural language - easy for anyone
- Structure language - pseudocode
- Graphical notation - paper prototypes
- Design language - not for everyone
- Mathematical specification - very precise

Paper prototypes are quick and cheap to create and easy to change and iterate often. You are also able to get usability feedback from real users.

10.1.6 Usability

Usability is made out of

- Learnability - How easy is it for users to accomplish basic tasks the first time they encounter the design?
- Satisfaction - How pleasant is it to use the design?
- Efficiency - Once users have learned the design, how quickly can they perform tasks?
- Errors - How many errors do users make, how severe are these errors, and how easily can they recover from these errors
- Memorability - When users return to the design after a period of not using, how easily can they re-establish efficiency?

10.1.7 Usability Concepts

Visibility - can the user see how to use the website. The more visible the function, the easier it is for the user to use the function. The relationship between the position of control and what it does, is also vital to ease of use
Affordance - Related to visibility is the concept of affordance. It was used by Norman in 1992 as a technical term that refers to the properties of an object - what sort of operations can be done with a particular object
Perceived affordance - which of the buttons, for example, would denote a specific function?
Constraints - The concept of restraining the kind of user interaction that can take place at any one time. A common way of achieving this is to deactivate certain menu options by shading, thereby restricting their use. This prevents users from using it and thereby making mistakes.Constraints can also take of the form:

- Physical - physical objects constrain the way they are used: DVD disk fits into DVD drive, not into USB port; Keyboard keys can usually only be pressed
- Logical - this refers to the users' understanding of the way world works. Making actions and their effect obvious allows users to deduce what happens next
- Cultural - Cultural conventions are learned conventions - colour red for danger or warnings is a typical western cultural convention. Most cultural constraints are arbitrary, but once learnt they are accepted as universal within that culture.

10.1.8 Feedback

Feedback is giving to the user information on what action has been done and what has been accomplished allowing the user to feel at ease and able to continue to operate successfully. Providing feedback of the right sort can aid visibility. Types of feedback include:

- Visual - wait, an hourglass symbol on the interface
- Auditory - beep when a mistake is made by the user
- Tactile - touch of a keyboard, mouse
- Verbal - warnings given in sound, or welcome message

10.1.9 Mapping

Relations to the relationship between controls and their effects in the world. Good natural mapping examples include steering wheels, slider bars, cursor keys. Some poor mapping examples are taps and light switches - which switch switches which light?

10.1.10 Consistency

Similar operations should use similar elements for similar tasks. For example, cut and paste are always in the edit menu. File menu top left, help in top right in windows.

10.1.11 Usability Heuristics

1. Visibility of system status - keep users informed of status by providing appropriate feedback (visual, auditory or tactile as appropriate)
2. Match between system and real world - Use user's language rather than system terms
3. User control and freedom - Provide "escape routes for users"
4. Consistency and standards - avoid making users wonder if different words mean the same thing
5. Error prevention - prevent users from making errors in the first place where possible. Think about user's existing model and how it can be utilised
6. Recognition rather than recall - make objects, actions and options visible
7. Flexibility and efficiency of use - provide accelerators that allow experts to do things faster
8. Aesthetic and minimalist design - avoid using information that is irrelevant or rarely needed

9. Help users recognise, diagnose and recover from error - use plain language to describe the nature of error and its solution. Be informative and non-technical, indicate something has gone wrong and provide potential solution
10. Help and documentation - provide information that is easily searched and provides help with concrete solutions.

10.1.12 Accessibility

The inclusive practice of making websites useable by people of all abilities. It is a legal requirement (Equality Act 2010).

- Meet level AA of the web content accessibility guidelines (WCAG 2.1)
- Otherwise; you could be sued for discrimination

At a basic level, a site is coded semantically using HTML, has text equivalents for images and meaningful link names.

10.1.13 Accessibility Needs

- Visual - blindness, poor vision, colour blindness
- Motor - difficulty in using hands (tremors, muscle slowness, loss of fine muscle control)
- Auditory - deafness / hearing impairments
- Seizures - epilepsy brought on by flashing lights
- Cognitive - developmental disabilities, learning difficulties, memory problems and attention

Some ways of making accessible is by increasing image size, making critical information bold and links underlined, making clickable areas large

10.1.14 Assistive technologies

- Screen reading software - reads the content of the screen to the user using synthesised speech.
- Braille terminals output text as braille
- Screen magnification
- Speech recognition
- Large trackballs

10.1.15 WCAG 2.1

Web content accessibility guidelines (WCAG) are published by the Web Accessibility Initiative. WCAG 2.0 was publishedi n 2008 and became an ISO standard in 2012. WCAG 2.1 was published in 2018. It contains four principles:

1. Perceivable - information and user interface components must be presentable to users in ways they can perceive
2. Operable - user interface components and navigation must be operable
3. Understandable - information and the operation of user interface must be understandable
4. Robust - content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies

10.2 Evaluation and Testing

Evaluation is the process of gathering data about the usability of a design, the characteristics of the user and the activities the user will do next. The environment and the context where the evaluation will take place and the nature of the artefact being examined.

10.2.1 Formative vs Summative Evaluation

- Formative evaluation - evaluation of an existing product, predictive evaluation, feedback from evaluation
- Summative evaluation - evaluation at the completion of a product

10.2.2 Measuring usability

Examples include ratio of successes to failures, time to complete task, number of errors user makes and number of times the user expresses frustration or satisfaction.

10.3 Web architecture and performance

10.3.1 Software Stack

Even our simple application uses several software components. Software components that work together are referred to as a stack. For example, web server (aoache), Applications (python/flask) and database (SQLite). There is also three tier architecture involved:

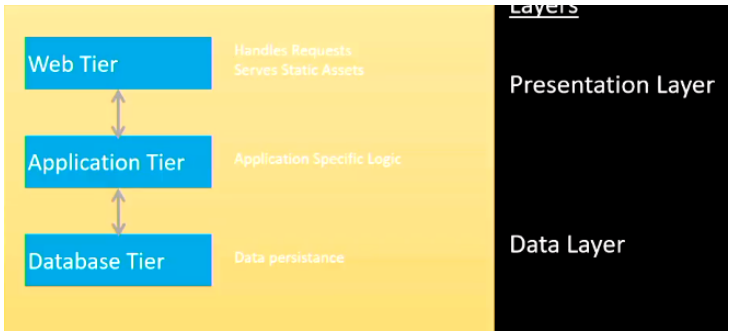


Figure 4: Three tier architecture

10.3.2 Servers

When a system ran out of capacity, you bought a bigger system. This is known as vertical scaling/scaling up. This is still practical to a certain degree. Even 1U servers can support huge quantities of RAM and have large disks. There are 42 units in a standard rack. We do not really buy servers anymore though. The advantages are:

- Can buy minimally specified system now and upgrade RAM or add disks as you grow
- Minimizes your capital expenditure on hardware

The disadvantages are:

- The whole machine must be replaced when its total capacity is exceeded.
- Upgrading/replacing the machine leads to downtime

A more modern approach to this is the database system is placed on a different server.

- RDBMS work best when they can cache large quantities of data in RAM
- This does however increase the latency of every request as they must now be sent over the network

Multiple copies of our web server and application are allocated to different servers

- These can be reasonably low capacity servers

And we have then a load balancer which redirects the user to a lowest traffic server. The algorithms can be used to balance the requests:

- Round robin - requests are sent to each server in turn
- Weighted round robin - requests are sent to each server in turn, as determined b an associated weight for each server
- Least active server - requests are sent to most lightly loaded server

However, the are disadvantages:

- More server maintenance is required
- Requires more server expenditure

the advantage is that

- Relatively cheap to add capacity - just add a server to load balancer
- Failure of web servers can easily be handled

10.3.3 Cloud based infrastructure

Cloud based infrastructure is rather:

- Cheap, reduces capital expenditure i.e., no need to pay for anything upfront as it scales as you use it
- Easily scaling - we can add resources as required, so no need to worry about capacity management

10.3.4 Additional software components

Caches

- Results of heavy computation maybe cached (stored)
- This means that computation does not need to have to be carried out again
- For example, memcached or varnish.

Message queues

- Some applications have long running tasks
- Waiting for this to be done would take a significant amount of time
- Applications can drop messages into a queue, which are then processed by software which listens to the queue

10.3.5 Performance

Performance can be measured using:

- End to end response time - the time taken from issuing a request to it being returned
- Site response time - this measures the time taken for the server to process a request
- Request throughput - the number of requests processed by a server per second

Note that response times do not necessarily increase linearly, therefore bulletpoint 1 and bulletpoint 2 are separate. Failure to have good performance an cause in page abandonment, good first impressions and loser users results in lost revenue. Now, which one to use?
We can test for concurrent users, end to end response time or throughput. Furthermore, we can measure end-to-end response time is sending a lot of request to the server and measure the time taken. The response time is an average of these. This needs to be done for each request type with appropriate tools.
Throughput can be measured by primarily concerning with how many requests your system can service per unit time. Need to consider the range of request types in your application. Each request will represent a different

proportion of application actions. We'd usually send a large number of requests, requests should be executed at varying levels of concurrency to investigate the throughput of the system and plot the results and view the performance curve.
Concurrent users can be tested by considering what a typical user does in a session, how many times they visit each page and what is a reasonable think time, where think time is time taken by a user to process information. An exponentially distributed think-with a mean of 7 seconds is common.

10.3.6 Typical application growth path

1. Start small - single server
2. Add in a dedicated server
3. Add in a second application server and load balance
4. Scale as required

11 Supporting Mobile

There has been a vast increase in the amount of mobile web traffic over the past few years. This is largely down to our handheld, IoT, smart home devices. The widespread adoption of these causes significant issues for web developers such as

- Small screen sizes
- Touch based interactions
- Headless web services

There are two main techniques which are

- Responsive web design
- Creation of separate pages for mobile platforms and web services

11.1 Responsive web design

For responsive web design, we focus on interactive web pages. The central notion for this is that one design should work across all designs e.g., desktops, tablets and mobile phones. This works by utilising an advanced feature of CSS3 called $media$ queries. You can use these to asses the capabilities of the device:

- Height/width of browser window
- Height/width of device
- Orientation - landscape/portrait
- Resolution

You can use the media queries in the CSS file directly or we can add links to different CSS files in the head element. We simply add a section which is delimited by an $@media$ tag. Max-device-width can also be used for a browser screen which doesn't normally change size.

```
511 @media only scree and (max-width:480px) {
512   div#wrapper {
513     width: 400px;
514   }
515   div#header {
516     background-image: url(media-quieries-phone.jpg);
517     height: 93px;
518     position: relative;
519   }
520   div#header h1 {
521     font-size: 140%;
```

```
  }
  #content {
    float: none;
    width: 100%;
  }
  #navigation {
    float: none;
    width: auto;
  }
}
```

Listing 130: Media queries

We do this as it saves a lot of hassle rewriting whole pages. You can simply use your existing content and it will be laid out in the best format for the device.

11.2 CSS Layout and Flexbox

We can combine with media query for responsive layout. Flexbox can be found in flexbox.help link website.

11.3 Creating a separate mobile layout

You might choose to have an entirely separate site for mobile content. This was previously a very common approach. You can do a redirect based on the browser's UserAgent which is sent in the header. User agent is accessed by

```
request.user_agent
```

Listing 131: User Agent

IT is sent as a part of the HTTP request and contains more information than one would think. We can access the user agent in python with the code above. Use this information to build a different user experience. You could then check it against known value for mobile devices.
However, there is already a library in flask which can help use recognise the device being used instead of accessing it from request.user_agent.string which is flask-mobility.

```
from flask_mobility import Mobility
from flask_mobilidy.decorators import mobile_template

@app.route('/mobile')
@mobile_template('{mobile/}index.html')
def mob(template):
  return render_template(template)
```

Listing 132: Flask-mobility

11.3.1 Pros

- Easy to create separate fixed width views
- Hard to cater for all sizes of screen

11.3.2 Cons

- May not work if new user agents are created
- May be lots of work to double the pages

12 Being a web developer

12.1 Testing

There are various types of testing:

- Functional testing
- Usability testing
- Compatibility testing
- Performance testing
- Security testing

12.1.1 Functional Testing

- Test all pages for errors
- Test all forms, check the results of the operation and check your validations are working appropriately.

12.1.2 Usability Testing

- Is navigating your application easy?
- Is the content correct?
- Link to help docs?
- Have you considered accessibility

12.1.3 Compatibility Testing

Browsers - we've only been really testing with Chrome and Firefox, what about edge, safari and opera? Or mobile devices, tablets (MS surface). What happens if a user is using an unsupported browser?

12.1.4 Performance Testing

How fast is your application? Can you optimise your apps such as caching data/assets, more efficient queries

12.1.5 Security Testing

is your application vulnerable to SQL injection attacks? Especially important if your chosen framework uses raw SQL. What is your update policy and what about your software stack?

12.1.6 When to run tests

It is recommended that the tests are ran frequently. Tests are ran manually after implementing every feature. Continuous Integration (CI), our tests run automatically when we check code into central repository.