

# ***SIT774 Web Technologies and Development***

## ***Workshop Week 10 - Node.js 3***

*Instructor : Ran Zhang*

Unit chair : Michael Hobbs

Instructor: Ran Zhang & Ruyi Li



# Key Concept

**Put your web application into use (10.1-10.5):** Create a database on the Node server and publish your website to an external host.

**Wrap up (10.6-10.10):** Reflect on what you have learnt in the unit and start to think about what you could do next to further develop your programming skills.

- ◆ Create a database
- ◆ Publish website

10.1 Welcome back	▼
Web Page	
10.2 Create a database on the Node server	▼
Web Page	
10.3 Creating a table, inserting, retrieving and displaying data	▼
Web Page	
10.4 Form data and database	▼
Web Page	
10.5 Publish your website to an external host VIDEO	▼
Web Page	
10.6 What now?	▼
Web Page	
10.7 Reflecting on your learning	▼
Web Page	
10.8 Learning Summary Report	▼
Web Page	
10.9 Submitting your portfolio VIDEO	▼
Web Page	
10.10 That's a wrap	▼
Web Page	

- ◆ Ontrack task- Learning summary
- ◆ Ontrack task- Final Portfolio



# 10.2 Create a database on the Node server

## Create two types of database:

- File database;
- In-memory database

### File database

#### First statement

```
var sqlite3 = require('sqlite3').verbose();
```

uses the `sqlite3` module (ie `require('sqlite3')`) to create a `sqlite3` instance via method `verbose()`.

#### Second statement

```
var db = new sqlite3.Database('myDB');
```

creates a database "myDB" as a file that is saved on the server.

This database is also called a file database. The communication between the code and the database "myDB" is now going through the object `db`.

### In-memory database.

The corresponding code is

```
var db = new sqlite3.Database(':memory:');
```

The statements within the following function block will use SQL commands to perform database operations, such as creating a table, inserting data into the table, retrieving data from the database and so on. Details are covered in the following step.

```
db.serialize(function() {  
    ...  
});
```

# 10.3 Creating a table, inserting, retrieving and displaying data

Suppose we need to create a table "User" in a database to store the following user data (with the fields *Name*, *Password* and *Option*):

Name	Password	Option
Jason	deakin2017	1
Karl	deakin2016	2
Belgrave	barby	3
Alice	cooldeakin	2

The NodeJS code shown below is for creating the file database, the table, entering and displaying the data.

```
let sqlite3 = require('sqlite3').verbose();
let db = new sqlite3.Database('myDB'); // file database

db.serialize(function() {

  db.run("CREATE TABLE IF NOT EXISTS User (name TEXT, password TEXT, option TEXT)");

  db.run("DELETE FROM User");
  db.run(`INSERT INTO User (name, password, option) VALUES ("Jason", "deakin2017")`);
  db.run(`INSERT INTO User (name, password, option) VALUES ("Karl", "deakin2016")`);
  db.run(`INSERT INTO User (name, password, option) VALUES ("Belgrave", "barby")`);
  // NOTE: The order of the fields relates to the order of the Values provided
  db.run(`INSERT INTO User (password, name, option) VALUES ("cooldeakin", "Alice", "2")`);

  // The SELECT operation is performed on the DB one row at a time and the function
  // is called for each row 'selected'
  console.log('Display all content from all rows of the DB');
  db.each("SELECT * FROM User", function(err, row) {
    console.log("[all] Name: " + row.name + " Password: " + row.password + " Option: " + row.option);
  });
  // Or you can select 'specific' fields from a data row
  console.log('Display only the name and option fields from all rows of the DB');
  db.each("SELECT name, option FROM User", function(err, row) {
    console.log("[subset] Name: " + row.name + " Option: " + row.option);
  });
});
db.close();
```

We create a **file database**, just as we learned in the previous step.

```
let sqlite3 = require('sqlite3').verbose();
let db = new sqlite3.Database('myDB'); // file database
```

We then use SQL commands to create a table "User" in the database, insert sample data into the table, retrieve all data from the table, and display the retrieved data on the console.

```
db.run("CREATE TABLE IF NOT EXISTS User (name TEXT, password TEXT, option TEXT)");
db.run("DELETE FROM User");
db.run(`INSERT INTO User (name, password, option) VALUES ("Jason", "deakin2017")`);
db.run(`INSERT INTO User (name, password, option) VALUES ("Karl", "deakin2016")`);
db.run(`INSERT INTO User (name, password, option) VALUES ("Belgrave", "barby")`);
// NOTE: The order of the fields relates to the order of the Values provided
db.run(`INSERT INTO User (password, name, option) VALUES ("cooldeakin", "Alice", "2")`);

// The SELECT operation is performed on the DB one row at a time and the function
// is called for each row 'selected'
console.log('Display all content from all rows of the DB');
db.each("SELECT * FROM User", function(err, row) {
  console.log("[all] Name: " + row.name + " Password: " + row.password + " Option: " + row.option);
});
// Or you can select 'specific' fields from a data row
console.log('Display only the name and option fields from all rows of the DB');
db.each("SELECT name, option FROM User", function(err, row) {
  console.log("[subset] Name: " + row.name + " Option: " + row.option);
});
```

**db.serialize()** makes sure the statements in the function(), which is called **callback function**, are executed in a serial manner.

```
db.serialize(function() {
```

After the operations on database are completed, the last statement **db.close()** closes the communication between the code and the database.

```
});
db.close();
```



Now, save the above source code as a Node.js file (eg *DBUser.js*) in the folder where your Node.js server is located in, then run the code as follow:

```
node DBUser.js
```

You will see the following results on the console, which mean the database “**myDB**” and table “*User*” are successfully created, and the sample data is now stored in the table.

```
D:\SIT774_Node > DBUser.js
Display all content from all rows of the DB
Display only the name and option fields from all rows of the DB
[all] Name: Jason Password: deakin2017 Option: 1
[all] Name: Karl Password: deakin2016 Option: 2
[all] Name: Belgrave Password: barby Option: 3
[all] Name: Alice Password: cooldeakin Option: 2
[subset] Name: Jason Option: 1
[subset] Name: Karl Option: 2
[subset] Name: Belgrave Option: 3
[subset] Name: Alice Option: 2

D:\SIT774_Node >_
```

When checking the Node.js server folder, you will find a file with the name “**myDB**”, which is a file SQLite database.



# 10.4 Form data and database -create a form web page

- Create a form web page
- Create a Node.js server
- Run the system
- Returning JSON Data

```
<!doctype html>
<html lang="en">

<head>
  <title>Express SQLite3 Demo</title>

  <meta charset="utf-8">
  <meta name="author" content="SIT774">
  <meta name="description" content="Express Sqlite3 Form Demo">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- Latest compiled and minified CSS -->
  <!-- CSS only -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha2/dist/css/bootstrap.min.c
    integrity="sha384-aFg/bzH65dt+w6FI2ooMVUpc+21e0SRygnTpmBvdBgSdnuTN7QbdgL+OapgHtvPp"
  </head>

<body>
  <div class="container">

    <h1> Online Form - Sqlite3 Demo</h1>

    <form action="/users" method="post">
      <div class="row mb-2">
        <label class="col-sm-2 col-form-label text-sm-end"
          for="customer">Name:</label>
        <div class="col-sm-10">
          <input type="text" class="form-control"
            id="customer" name="name">
        </div>
      </div>
      <div class="row mb-2">
        <label class="col-sm-2 col-form-label text-sm-end"
          for="pwd">Password:</label>
        <div class="col-sm-10">
          <input type="password" class="form-control"
            id="pwd" name="password">
        </div>
      </div>
      <div class="row mb-2">
        <label class="col-sm-2 col-form-label text-sm-end"
          for="radioBtnInput">Contact:</label>
        <div class="col-sm-10 my-auto" id="radioBtnInput">
          <div class="form-check form-check-inline">
            <input class="form-check-input" type="radio" name="contact" value="1"
              <label class="form-check-label" for="rb1">
              Option 1
            </label>
          </div>
          <div class="form-check form-check-inline">
            <input class="form-check-input" type="radio" name="contact" value="2"
              <label class="form-check-label" for="rb2">
              Option 2
            </label>
```

```

          </div>
          <div class="form-check form-check-inline">
            <input class="form-check-input" type="radio" name="contact" value="3"
              <label class="form-check-label" for="rb3">
              Option 3
            </label>
          </div>
        </div>
      </div>
      <div class="row">
        <div class="col-sm mx-auto d-flex justify-content-center">
          <button type="submit" class="btn btn-primary"
            value="Create Profile">Submit</button>
        </div>
        <div class="col-sm mx-auto d-flex justify-content-center">
          <button type="reset" class="btn btn-primary">Reset</button>
        </div>
      </div>
    </form>

    <hr>

    <p>Use the following button to retrieve information from DB</p>
    <form action="/users" method="get">
      <button type="submit" value="Get Users" class="btn btn-primary">Get Users</button>
    </form>
  </div>

  <!-- Bootstrap JavaScript files -->
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha2/dist/js/bootstrap.bundle
    integrity="sha384-qKXV1j0HvMUeCBQ+QVp7JcfG1760yU08IQ+GpUo5h1bpg51QRIuqHAJz8+BrxE/N"
    crossorigin="anonymous"></script>

</body>

</html>
```

**Attributes in the first form** `<form action="/users" method="post">` pass the information to the server that the data in the form will be sent using the **POST** method, and the server-side handler /user will accept and process the received form data.

**Attributes in the second form** `<form action="/users" method="get">` tell the server that once the *Get User* button is clicked, the server-side handler /user will process the request which is sent using the method **GET**.



# 10.4 Form data and database

## - Create a Node.js server

```
// Require the express web application framework (https://expressjs.com)
let express = require('express')

// Create a new web application by calling the express function
let app = express()

let sqlite3 = require('sqlite3').verbose();

// persistent file database "myDB".
let db = new sqlite3.Database('myDB');

// Get port from environment and store in Express.
let port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Normalize a port into a number, string, or false.
 */
function normalizePort(val) {
  let port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}

// Tell our application to serve all the files under the 'public_html' directory
app.use(express.static('public_html'))

// Here we are configuring express to use inbuilt body-parser as middle-ware.
app.use(express.urlencoded({ extended: false }));

// REST endpoint for STORING the user data into the DB as submitted form the form
app.post('/users', function (req, res, next) {
  let username = req.body.name;
  let password = req.body.password;
  let option = req.body.contact;

  console.log("Just received POST data for users endpoint!");
  console.log(`Data includes: ${username}, ${password} and ${option}`);

  // insert the form data into the table User
  let stmt = db.run('INSERT INTO User VALUES (${username}), (${password}), (${option})');

  // still display the default web page in public folder, i.e. index.html, for next data ent
  res.status(200).redirect('/');
});

// REST endpoint for getting all user data
app.get('/users', function (req, res) {
  let html = '';
```

```
// HTML code to display a table populated with the data from the DB

html += '<!doctype html><html lang="en">';
html += '<head>';
html += '<title>Bootstrap Express/SQLite3 Demo</title>';
html += '<meta charset="utf-8">';
html += '<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=1">';
html += '<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha2/dist/css/bootstrap.min.css" integrity="sha384-aFq/bzH65dt+6F1200MVUp21e0SRygnTpmBvdBgSdnuTN7QbdgL+OapgHtP8">';
html += '</head>';

html += '<body><div class="container">';
html += '<h3> The User Information Table </h3>';
html += '<table class="table">';
html += '<thead class="thead-dark"><tr>';
html += '<th>Name</th><th>Password</th><th>Option</th>';
html += '<tr></thead><tbody>';

// Retrieve data from table User on the server
// and display it in a web page table structure
db.all('SELECT * FROM User', function(err, rows) {
  if (err) {
    return console.error(err.message);
  }
  if (rows.length === 0) {
    console.log("Array is empty!")
    html += '<tr><td colspan="3"> No data found </td></tr>';
  } else {
    rows.forEach(function (row) {
      html += '<tr>';
      html += '<td>'+row.name+'</td>';
      html += '<td>'+row.password+'</td>';
      html += '<td>'+row.option+'</td></tr>';
    });
  }
  html += '</tbody></table>';
  html += '</div>';
  html += '</body></html>';
  res.send( html );
});

// Tell our application to listen to requests at port 3000 on the localhost
app.listen(port, function () {
  // When the application starts, print to the console that our app is
  // running at http://localhost:3000 (where the port number is 3000 by
  // default). Print another message indicating how to shut the server down.
  console.log(`Web server running at: http://localhost:${port}`)
  console.log("Type Ctrl+C to shut down the web server")
})
```

## 10.4 Form data and database

### - Create a Node.js server

In the code:

**the first method** `app.post('/users', function (req, res, next) {...});` is a server handler that deals with the data sent from the first form (ie method="post", action="/user").

**the second method** `app.get('/users', function (req, res) {...});` is another server handler that deals with the request sent from the second form (ie method="get", action="/user").

**NOTE:** If the form web page is saved in another name, for example `form.html`, in the `public` folder, the statement `res.status(200).redirect('/')`; can be changed to `res.status(200).redirect('/form.html')`;  
Save the above code as a Node.js file (eg `DBUserindex.js`) in the Node.js server folder.



# 10.4 Form data and database - Create a Node.js server

## Run the system

Now we can run the system by following the steps below.

### Step 1: Launch the server

In Command Prompt (Windows) or Terminal (macOS), go to the Node.js server folder (eg *SIT774\_Node*), and execute the command:

```
node DBUserindex.js
```

You should see the message shown below, which means the server is running.

```
D:\SIT774_Node> node DBUserindex.js  
Example app listening on port 3000
```

### Step 2: Open the form page (ie *index.html*) in a browser

Open a web browser and visit the URL address <http://localhost:3000>. The form web page *index.html* will display on the screen.

### Step 3: Enter user data into the form and submit the form to the server

Enter the name and password, select an option in the form (eg *Student* / *HappyDeakin* / *1*), and click the button "Submit" as shown below:

After submission, you can continue to enter other user data into the form and submit it to the server.

### Step 4: Display all user data in the browser (client)

Click the button "Get Users" in the form page, all user data (including those sample data entered in before) in the server database will be displayed in the browser as shown below:

Name	Password	Option
Jason	deakin2017	1
Karl	deakin2016	2
Belgrave	barby	3
Alice	cooldeakin	2
Student	HappyDeakin	1

This example shows all necessary techniques for creating links between a form web page (client) and a database on the server side.

# 10.4 Form data and database

## -Returning JSON Data

### Returning JSON Data

You may have noticed from the above example that it can be a little cumbersome to return a HTML response to the calling client (browser). In some cases, it may be simpler/clearer to return only the 'data' from a GET or POST request, i.e., just the data obtained from the database SELECT command. A common format for such data in web communication is JSON (JavaScript Object Notation). An example of JSON data being returned from a GET request showing information on the current *Bitcoin* prices, is shown below:

- <https://api.coindesk.com/v1/bpi/currentprice.json>

```
{
  "time": {
    "updated": "Sep 16, 2021 00:59:00 UTC",
    "updatedISO": "2021-09-16T00:59:00+00:00",
    "updateduk": "Sep 16, 2021 at 01:59 BST"
  },
  "disclaimer": "This data was produced from the CoinDesk Bitcoin Price Index (USD). Non-UK",
  "chartName": "Bitcoin",
  "bpi": {
    "USD": {
      "code": "USD",
      "symbol": "$",
      "rate": "47,919.5250",
      "description": "United States Dollar",
      "rate_float": 47919.525
    },
    "GBP": {
      "code": "GBP",
      "symbol": "£",
      "rate": "34,603.1682",
      "description": "British Pound Sterling",
      "rate_float": 34603.1682
    },
    "EUR": {
      "code": "EUR",
      "symbol": "€",
      "rate": "40,544.8539",
      "description": "Euro",
      "rate_float": 40544.8539
    }
  }
}
```

Other address can accept parameters, here a random joke is returned but is restricted to 'Two Parts' and 'Safe' (clean). Note if you call this link, you may get a different result.

- <https://v2.jokeapi.dev/joke/Any?type=twopart&safe-mode>

```
{
  "error": false,
  "category": "Spooky",
  "type": "twopart",
  "setup": "Why didn't the skeleton go for prom?",
  "delivery": "Because it had nobody.",
  "flags": {
    "nsfw": false,
    "religious": false,
    "political": false,
    "racist": false,
    "sexist": false,
    "explicit": false
  },
  "id": 183,
  "safe": true,
  "lang": "en"
}
```

These address, combined with the format of the REQUEST and RESPONSES, defines an **API** (application programmer interface) and enables the development of web based applications.

The use of the data returned from these APIs is then left open to the developer making it more flexible in how (and what content is) finally presented in the browser window. These could be linked to an event handler for a **Button** on the page, where the request is issued on a 'click' with the response data used to populate an element in the page.

Some more example APIs include:

- Current XKCD Comic <http://xkcd.com/info.0.json>
- Name/Country of Origin Probability (given a first name as a parameter) <https://api.nationalize.io/?name=michael>
- NASA Random Result [https://api.nasa.gov/planetary/apod?api\\_key=DEMO\\_KEY](https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY)

In all the examples listed above, they are publicly accessible (with no authentication keys needed) APIs. Except for the NASA example, where an **api\_key** is needed and in this case a **DEMO\_KEY** is used, but a registered user could obtain their own individual key.



# Returning the User data as in JSON form

## Returning the User data as in JSON form

In our form & database example, to return the data in a JSON format we would add a new route handler in our express server, e.g., `app.get('/usersapi'...`, simply return the response from the `db.all()` command. As shown below:

```
// REST endpoint for getting all user data JSON response
app.get('/usersapi', function (req, res) {
  // Retrieve data from table User on the server
  // and return it in a JSON response
  db.all('SELECT * FROM User', function (err, rows) {
    if (err) {
      return console.error(err.message);
    } else {
      res.send(rows);
    }
  });
});
```

The resulting output in the client browser for the URL (<http://localhost:3000/usersapi>) would be:

```
[
  {
    "name": "Jason",
    "password": "deakin2017",
    "option": "1"
  },
  {
    "name": "Karl",
    "password": "deakin2016",
    "option": "2"
  },
  {
    "name": "Belgrave",
    "password": "barby",
    "option": "3"
  },
  {
    "name": "Alice",
    "password": "cooldeakin",
    "option": "2"
  }
]
```

# 10.5 Publish your website to an external host

## VIDEO

Watch this video to see how to publish your node app.

So, you've created an awesome web app using NodeJS. You've got a database, forms, and you're ready to share your web app with the world! But there's a problem... It's only running on localhost! How can we have our app running on the internet so that anyone can visit it!?

In this video, we're going to take a look at Heroku for publishing our node app.

Heroku is free for students and small projects, and it's easy to setup with very well documented guides!



# 10.6 What now?

## What now?

Congratulations on making it to the end of week 10.

Before we do a final wrap up, here is a quick list of the things you should aim to complete before the conclusion of the unit:

- Make sure you have completed all tasks you wish to include in your portfolio.
- Upload any tasks to OnTrack which aren't already uploaded, and work with your tutor to get these tasks marked as Complete.
- Prepare your Learning Summary Report, and make sure to provide evidence that you have achieved all of the unit learning outcomes.
- Create your portfolio on OnTrack, and review what has been created to make sure everything is in place.
- Relax!

## 10.7 Reflecting on your learning

Compare them to something more recent and think about **what it was like** when you were working on those tasks. Are there **any differences** in page and code quality? structure? If you created these again would it **be easier or more challenging**?

For the assessment in this unit you need to **demonstrate that you have met the unit's learning outcomes**.

So, now is probably a good time to review these outcomes and to **make sure you have evidence** that you have achieved these. As you review each outcome think about the tasks you have completed and how these helped you develop and demonstrate these outcomes. Make sure that you have evidence for each outcome, ideally multiple things so that you can have some depth to what you are demonstrating you have achieved.

## 10.8 Learning Summary Report

You need to **document your reflections** and the outcomes you have achieved in the Learning Summary Report.

This report **outlines the grade you are applying for**, together with justifications for **why** you should receive this grade.

This document will be at the front of your portfolio and puts forward your perspective on what you have learnt, how it demonstrates the unit learning outcomes, along with your reflections on what you got out of the unit more personally.

In unit site **10.9 Submitting your portfolio video**, we will show you how to prepare your portfolio using OnTrack and submit your work for final assessment.



# Ontrack task- 10.1P: Server Database for Website Project

In this task, you are required to extend your web server from **Task 9-1P** to allow for the feedback to be stored in a DB for retrieval even if the web server has to be restarted.

In addition to the storing of the feedback, your web page should also provide a method to retrieve the contents of the database through the addition of another button. A sample screenshot of what your main page should look like is shown below:

dKin Caps

A young and inspiring organisation whose primary purpose is to bring you the highest quality headwear available.

### dKin Caps: Membership

Signing up to become a member of the **dKin Caps** community is very rewarding. Please use the web-based form below to give us your details and you will receive some great offers from us!

#### Membership

Please complete the form below to provide some valuable feedback on your favourite ice cream.

Firstname:

Surname:

Email:

Mobile#:

Number of caps owned: ☐ No caps yet ☐ Between 1 and 10 caps ☐ Between 11 and 29 caps ☐ More than 30 caps

Favourite Cap(s):

Comments:

#### List Feedback

The following button will issue a **GET** request to the `/membershipdetails` route to Retrieve all the members details stored in the DB.

© 2024 A webpage footer with important info!

Task10.1.1 Modified form page with extra buton

Initially, the database should be empty, so clicking the *Retrieve Membership Details From Database* button should display an 'No data found' message. An example of this is shown below:

dKin Caps

A young and inspiring organisation whose primary purpose is to bring you the highest quality headwear available.

### Membership List

The table below lists all the registered members of the **dKin Caps** community. Accessing this data, stored in a persistant database, is performed on the server.

ID	Firstname	Surname	Email	Mobile#	Cap Type	Number of Caps	Comment
List is empty							

[Click here to return to the feedback form.](#)

© 2024 A webpage footer with important info!

Task10.1.2 Membership database 'empty'



When adding further membership details, these should be stored in the database, and when the *Retrieve Membership Details From Database* button is then clicked, the details should be displayed in a table:

You will need to create a web server that implements a simple *Membership Server*, with an interface to a simple database (file) and accepts requests from a user (client browser). These requests include a **POST** to store a *new* record into the database and a **GET** to display the current/updated contents of the database.

The database structure (schema) should support a record with the following fields:

Name	Type
id	INTEGER PRIMARY KEY AUTOINCREMENT
fname	TEXT
sname	TEXT
email	TEXT
mobile	TEXT
fname	TEXT
numcaps	TEXT
favourite	TEXT
comment	TEXT

NOTE: All the fields, except for the `id` field, are of type `TEXT` (i.e., the mobile number and the number of caps are treated as strings)

Membership List

localhost3000/membershipdetails

Incognito (2)

dKin Caps

A young and inspiring organisation whose primary purpose is to bring you the highest quality headwear available.

Membership List

The table below lists all the registered members of the dKin Caps community. Accessing this data, stored in a persistant database, is performed on the server.

ID	Firstname	Surname	Email	Mobile#	Cap Type	Number of Caps	Comment
1	Mick	Hobbs	mick@deakin.edu.au	0499000111	Between 1 and 10 caps	Snapback Cap	I wash my cap 3 times a week!
2	Sam	Smith	samsmith@deakin.edu.au	0411987987	More than 30 caps	Sun Cap	Never go outside in the sun with out my trusty cap.
3	Alexander	Nguyen	aln@deakin.edu.au	0424366221	Between 1 and 10 caps	Runners Cap	I wash my cap 3 times a week!
4	Sophia	Taylor	sophie@deakin.edu.au	0433775775	0 caps	Bucket Hat	Yet to get a cap... But when I do the first will be a Blue Bucket Hap
5	Noah	Rodriguez	noah@deakin.edu.au	0462421133	30+ caps	Trucker Cap	Sometimes, wearing a trucker's cap gives me a feeling of anonymity or privacy, helping me blend in or go unnoticed in a crowd...

[Click here to return to the feedback form.](#)

© 2024 A webpage footer with important info!



# Hints and Submission

## Steps

To complete this task, you are required to:

1. Make a standalone *Node.js* program (i.e., a createdB.js file) and execute it on the *Node.js* console to create and initialise a server side *SQLite3* file database in the server folder. The database contains a table that is used to store the data sent from a form page of your own website (as described in the table above).
2. Make another *Node.js* program (i.e., a index.js file) in your *Node.js* server folder. This program is able to launch the server, accept the data sent from a form page of your website (in a **POST** request on the route /submitmembership), save the received data into the database table, and display the table data upon the request from the client (i.e., a **GET** request on the route /membershipdetails).
3. Make necessary changes to a form page of your website (can be stored in the template file index.ejs), so that it is able to:
  - send the form data to the server using a **POST** message linked to a **Submit** button
  - send a data retrieval request (e.g., a **GET** request message linked to a **Retrieve Membership Details From Database** button) to the server using a web browser.
4. Launch the server by executing your *Node.js* program you made.
5. Visit the form web page of your website via the local *Node.js* server (e.g., <http://localhost:3000/> which will retrieve the template view index.ejs page)
6. Enter data into the form and submit the form to the server (three or more times, by clicking a **Submit** button). The data should be saved in the server database table.
7. Within the form page, send a data retrieval request (e.g., by clicking a **Retrieve Membership Details From Database** button) to the server. After receiving the request, the server retrieves data from the database table and displays the retrieved data in the browser.

## Hints

The code snippet for the **Retrieve Membership Details From Database** button could be:

```
<h5 class="mt-4">List Feedback</h5>
<p>
  The following button will issue a <code>GET</code> request to the <code>/feedback</code>
  route to retrieve feedback stored in the DB.
</p>
<div class="d-grid gap-5d-md-flex justify-content-md-center mb-4">
  <a href="/feedback">
    <button class="btn btn btn-success" id="getfeedbackBtn">
      Retrieve Membership Details From Database
    </button>
  </a>
</div>
```

This task is similar to the example provided in the unit site. You will note the structure of the database table will be different (slightly) and there will need to be a change to the form inputs to match the 'fields' expected in the database.

Don't forget to install sqlite3!  
-npm install sqlite3

## What will you submit?

You should submit:

- Source code of the template web page of your main page *form* (i.e., the `index.ejs` file)
- Source code of the template file that renders the contents of the database into a table (i.e., the `members.ejs` file)
- Source code of the *Node.js* file (i.e., the first `createdB.js` file) that creates a server **file** database with a table in it.
- Source code of the *Node.js* server program file (i.e., the second `index.js` file).
- Screenshot of the browser window showing the the **empty** database table.
- Screenshot of the browser window showing the form web page with entered data.
- Screenshot of the browser window showing the retrieved data from the database table after the data retrieval request is sent to the server.

sit774-task10-1p

db

views

.gitignore

index.js

package-lock.json

package.json

```
let sqlite3 = require('sqlite3').verbose();
let db = new sqlite3.Database('myMembersDB'); // file database

db.serialize(function() {
  sqlite3.verbose();

  db.run("CREATE TABLE IF NOT EXISTS Members (id INTEGER PRIMARY KEY AUTOINCREMENT, fname TEXT, sname TEXT, email TEXT, mobile TEXT, numcaps TEXT, favourite TEXT, comment TEXT)");
  db.run("DELETE FROM Members");

  // The SELECT operation is performed on the DB one row at a time and the function
  // is called for each row 'selected'
  console.log('Display all content from all rows of the DB');
  db.each("SELECT * FROM Members DESC", function(err, row) {
    console.log(`[all] (${row.id}) Name: ${row.fname} ${row.sname}, Email: ${row.email}, Mobile: ${row.mobile}, numcaps: ${row.numcaps}, favourite: ${row.favourite}, numcaps: ${r
  });
  // Or you can select 'specific' fields from a data row
  // console.log('Display only the name and option fields from all rows of the DB');
  // db.each("SELECT name, option FROM User", function(err, row) {
  //   console.log(`[subset] Name: " + row.name + " Option: " + row.option);
  // });
});
db.close();
```

sit774-task10-1p

db

comments.txt

1 createDB.js

createFilledDB.js

deleteDB.js

displayDB.js

myMembersDB

sit774-task10-1p

db

views

404.ejs

error.ejs

footer.ejs

header.ejs

3 index.ejs

invalid-form.ejs

members.ejs

thankyou.ejs

.gitignore

index.js

package-lock.json

package.json



4

```
PS C:\Users\new\OneDrive - Deakin University\Desktop -MDk\SIT774\workshop 2024\2024-t1-git\sit774-tasks\wk10\sit774-task10-1p> node .
Web server running at: http://localhost:3000
Type Ctrl+C to shut down the web server
Connected to the file SQLite database 'myIceCreamDB'
::1 - - [15/Sep/2024:05:42:31 +0000] "GET / HTTP/1.1" 200 8719
::1 - - [15/Sep/2024:05:42:32 +0000] "GET /favicon.ico HTTP/1.1" 404 1515
```

5

# dKin Caps

A young and inspiring organisation whose primary purpose is to bring you the highest quality headwear available.

## dKin Caps: Membership

Signing up to become a member of the **dKin Caps** community is very rewarding. Please use the *web-based form* below to give us your details and you will receive some great offers from us!

### Membership

Please complete the form below to provide some valuable feedback on your favourite ice cream.

Firstname:

Surname:

Email:

Mobile:

04xxxxxxxx

Number of caps owned:

No caps yet

Between 1 and 10 caps

Between 11 and 29 caps

More than 30 caps

Favourite Cap(s):

Select a cap style...

Comments:

Please provide a few words to describe your favourite cap...

Submit

Reset

### List Feedback

The following button will issue a **GET** request to the `/membershipdetails` route to Retrieve all the members details stored in the DB.

Retrieve Membership Details From Database

6

# dKin Caps

A young and inspiring organisation whose primary purpose is to bring you the highest quality headwear available.

## Membership List

The table below lists all the registered members of the **dKin Caps** community. Accessing this data, stored in a persistant database, is performed on the server.

ID	Firstname	Surname	Email	Mobile#	Cap Type	Number of Caps	Comment
List is empty							

[Click here to return to the feedback form.](#)

7

# dKin Caps

A young and inspiring organisation whose primary purpose is to bring you the highest quality headwear available.

## Membership List

The table below lists all the registered members of the **dKin Caps** community. Accessing this data, stored in a persistant database, is performed on the server.

ID	Firstname	Surname	Email	Mobile#	Cap Type	Number of Caps	Comment
7	Ran	Zhang	ran.zhang@deakin.edu.au	1234567899	1 - 10 caps	Snapback Cap	I like Snapback Cap!
8	Jone	Dee	Jone.dee@deakin.edu.au	1234567899	11 - 29 caps	Trucker Cap	Trucker Cap is good !

[Click here to return to the feedback form.](#)



# Ontrack task- 10.2C: Search a Database

In this task, you are required to extend your web server code from **Task 10-1P** to allow a user to search the **dKin Membership** database for entries matching a particular input from one of the database fields.

You will need to extend the main feedback form page (from **Task 10.1P**) by adding a **new form**. This will have two input fields, one for the *Search Term* and the other with a **pulldown select** for the *database field* to search in. A sample of a very simple *form & submit button* (added to the end of the main page) is shown in the screenshot below:

Mobile:

Number of caps owned: ☐ No caps yet ☐ Between 1 and 10 caps ☐ Between 11 and 29 caps ☐ More than 30 caps

Favourite Cap(s):

Comments:

**List Feedback**  
The following button will issue a *GET* request to the */membershpdetails* route to Retrieve all the members details stored in the DB.

**Search Icecream Type**  
The following button will issue a *POST* request to the */search* route to retrieve feedback on the ice cream type inputted.

Search term:

Field:

© 2024 A webpage footer with important info!

Task10.2.1 Form page with search field

In the example used here, the database holds the records of 9 members.

In the example used here, the database holds the records of 9 members.

**dKin Caps**  
A young and inspiring organisation whose primary purpose is to bring you the highest quality headwear available.

**Membership List**  
The table below lists all the registered members of the **dKin Caps** community. Accessing this data, stored in a persistent database, is performed on the server.

ID	Firstname	Surname	Email	Mobile#	Cap Type	Number of Caps	Comment
1	Mick	Hobbs	mick@dakin.edu.au	0499000111	Between 1 and 10 caps	Snapback Cap	I wash my cap 2 times a week!
2	Sam	Smith	samsmith@dakin.edu.au	0411987987	More than 30 caps	Sun Cap	Never go outside in the sun with out my trusty cap.
3	Aksander	Nguyen	ahn@dakin.edu.au	0474366221	Between 1 and 10 caps	Runners Cap	I wash my cap at least 7 times a week.. I don't like dirty caps.
4	Sophia	Taylor	sophia@dakin.edu.au	0433775775	0 caps	Bucket Hat	Yet to get a cap.. But when I do the first will be a Blue Bucket Hat
5	Nash	Rodriguez	nash@dakin.edu.au	0462421133	10+ caps	Trucker Cap	Sometimes, wearing a trucker's cap gives me a feeling of anonymity or privacy, helping me blend in or go unnoticed in a crowd..
6	Derick	Long	dericklong@dakin.edu.au	0499111000	1 - 10 caps	Trucker Cap	I love my trucker cap, I wear it in the morning when I go to work as a barista.
7	Petra	Smithton	pppss@dakin.edu.au	0400000001	30+ caps	Beanie	My beanie is warm and fluffy I love my beanie.
8	Astrid	Lawson	stasz@dakin.edu.au	0155668999	Between 1 and 10 caps	Sun Cap	I spend lots of time outside, so my trusty sun cap is always on my head. Gotta be sun smart!
9	Bredie	Hammesmith	hammes@dakin.edu.au	04123545678	1 - 10 caps	Snapback Cap	One caps.. I love it.. but I never, ever wash it.

[Click here to return to the feedback form.](#)

© 2024 A webpage footer with important info!

Task10.2.2 Database containing 9 members records





The example below shows a search on term `smith` within the *Surname* field:

Search Members Database

The following button will issue a **POST** request to the `/search` route to retrieve feedback on the ice cream type inputted.

Search term:

Field:

© 2024 A webpage footer with important info!

Task10.2.3 Input search field "smith"

### dKin Caps

A young and inspiring organisation whose primary purpose is to bring you the highest quality headwear available.

#### Search Results

The table below lists all the registered members of the **dKin Caps** community. Accessing this data, stored in a persistent database, is performed on the server.

Thank you for your request to search the **dKin Caps** members database.

Below are those entries from the database using the search term `'smith'` on the field `'sname'`.

ID	Firstname	Surname	Email	Mobile#	Cap Type	Number of Caps	Comment
2	Sam	Smith	samsmith@deakin.edu.au	0411987987	More than 30 caps	Sun Cap	Never go outside in the sun with out my trusty cap.
7	Petra	Smithton	pppsss@deakin.edu.au	0400000001	30+ caps	Beanie	My beanie is warm and fluffy! I love my beanie.
9	Brodie	Hammersmith	hammer@deakin.edu.au	04123545678	1 - 10 caps	Snapback Cap	One cap... I love it... but I never, ever wash it.

[Click here to return to the feedback form.](#)

© 2024 A webpage footer with important info!

Task10.2.3a Results of field "smith"



While the next example shows a search on term wash within the *Comment* field:

Search Members Database

The following button will issue a **POST** request to the **/search** route to retrieve feedback on the ice cream type inputted.

Search term: wash

Field: Comment

Search Membership Database

© 2024 A webpage footer with important info!

Task10.2.4 Input searching for "wash"

dKin Caps

A young and inspiring organisation whose primary purpose is to bring you the highest quality headwear available.

### Search Results

The table below lists all the registered members of the **dKin Caps** community. Accessing this data, stored in a persistent database, is performed on the server.

Thank you for your request to search the **dKin Caps** members database.

Below are those entries from the database using the search term 'wash' on the field 'comment'.

ID	Firstname	Surname	Email	Mobile#	Cap Type	Number of Caps	Comment
1	Mick	Hobbs	mick@deakin.edu.au	0499000111	Between 1 and 10 caps	Snapback Cap	I wash my cap 3 times a week!
3	Alexander	Nguyen	ain@deakin.edu.au	0424366221	Between 1 and 10 caps	Runners Cap	I wash my cap at least 7 times a week... I don't like dirty caps.
9	Brodie	Hammersmith	hammer@deakin.edu.au	04123545678	1 - 10 caps	Snapback Cap	One cap... I love it... but I never, ever wash it.

[Click here to return to the feedback form.](#)

© 2024 A webpage footer with important info!

Task10.2.4a Results searching for "wash"

Note that in these examples the 'partial' word is used in the search. This is implemented using SQL commands, and as such it is possible to use wild cards. Here the search term is wrapped up in the % symbols, such as a search for smith in the *surname* would be %smith% and would return 3 members details.

Also, if a search term doesn't return any rows, a message like "Nothing found" or "List is empty" should be displayed.



## Steps

To complete this task, you are required to:

1. Extend your `index.ejs` template to hold another form for the **Search Term** and **Field to Search**. The input fields should have a unique/relevant name such that the server can extract this from the request message. The form should `POST` the request to the route `/search`.
2. Extend your web server code (`index.js`) to handle a new `POST` request on the route `/search`. In this handler, the `search` data field should be extracted.
3. Issue multiple database request to find all rows that have a given `search term` (or substring) in various `fields` of the members database, like those examples provided above.
4. Create a new template file (`search.ejs`) that is called (`response.render()`) with the parameters: `title`, `search term` and `rows`. It should display the list of matching `rows` (as returned by the database request in step 3). **NOTE:** This template is very similar to the `members.ejs` template used in Task 10.1P.

## Hints

The SQL code snippet for **Searching** the database on a field *like* a given term could be:

```
const query = `SELECT * FROM Comments WHERE ${searchfield} LIKE '${searchtype}'`;
```

Where the variables `searchfield` and `searchtype` is extracted from the *form* data passed into the handler.

## What to submit

You should submit:

- Source code of the template web page of your extended main page *form* (i.e., the `index.ejs` file)
- Source code of the template file that renders the contents of the results of the database field search into a table (i.e., the `search.ejs` file)
- Source code of the *Node.js* server program file with the new `POST` handler for the `/search` route (i.e., the second `index.js` file).
- A document with **3 Screenshots** of the browser window showing at least 3 search terms:
  1. A specific *surname* that does exist
  2. A specific *comment term* that does NOT exist
  3. A search term that returns more than one row, from either the surname or comments fields

# Ontrack task- 10.3D: Website project (Part 3 of 3): Add Database Access to Your Website

## Tasks

In the final part of your website project, you are asked to incorporate database access into your site.

It is left open to you what form that database access will be but it must meet the key requirement that your database should involve **more than one** data table.

## Database Access Options

Some possible options could include, but not limited to, **combinations** of the following items (or similar that relate to your website):

- User login
  - Note: this can be a simplistic *demonstration* implementation and does not need to include full encryption/security
- Saving contents of a *Feedback Form*
- Processing contents of an *Item/Product Search*
- Product lists/tables created from stored item information

## What will you submit?

You should submit:

- Combined source code of your website.
- Source code of the *JavaScript* file that initialises the database tables
- Source code of the *JavaScript* file that handles the requests/interactions with the database tables (i.e., the `index.js` or other file)
- Screenshot of the browser window showing the **input** of data.
- Screenshot of the browser window showing the **result/output** of database interaction.

# Ontrack task 10.4HD: Complete Awesome Website Feature(s)

In this task you are required to research and implement one or more features into a web page. Using your knowledge gained in this unit so far combined with your research and experimentation, the feature(s) implemented should demonstrate an **extension to your knowledge** and the ability to **communicate this to others**. In doing so you should be able to highlight the **awesomeness** of the feature(s).

Examples of features you could consider to research/develop/implement can include (*but in no means limited to these*):

- Application of extended user interface styling/interaction effects (i.e., background image scrolling, more involved or tailored *Bootstrap components*) to implement/enable a *positive user interaction* with your webpage.
- Implementation of more involved routing methods and error handling in the server side, within the *express* code.
- Implementation of (more) secure user authentication mechanisms - such that the *username* and *password* are **not** sent or stored in *plain text*.
- Dynamic content within a web page that is produced from JSON data, which could be obtained from a local or remote webserver.
- Alternate permanent storage (database) approaches.

**NOTE:** Your web page should also include information/descriptions of the feature(s) you **have** implemented and **how** they were implemented in your page. Your website should be a small but clear **wiki / tutorial** on the how these specific feature(s) were implemented, allowing another developer easily implement similar features.

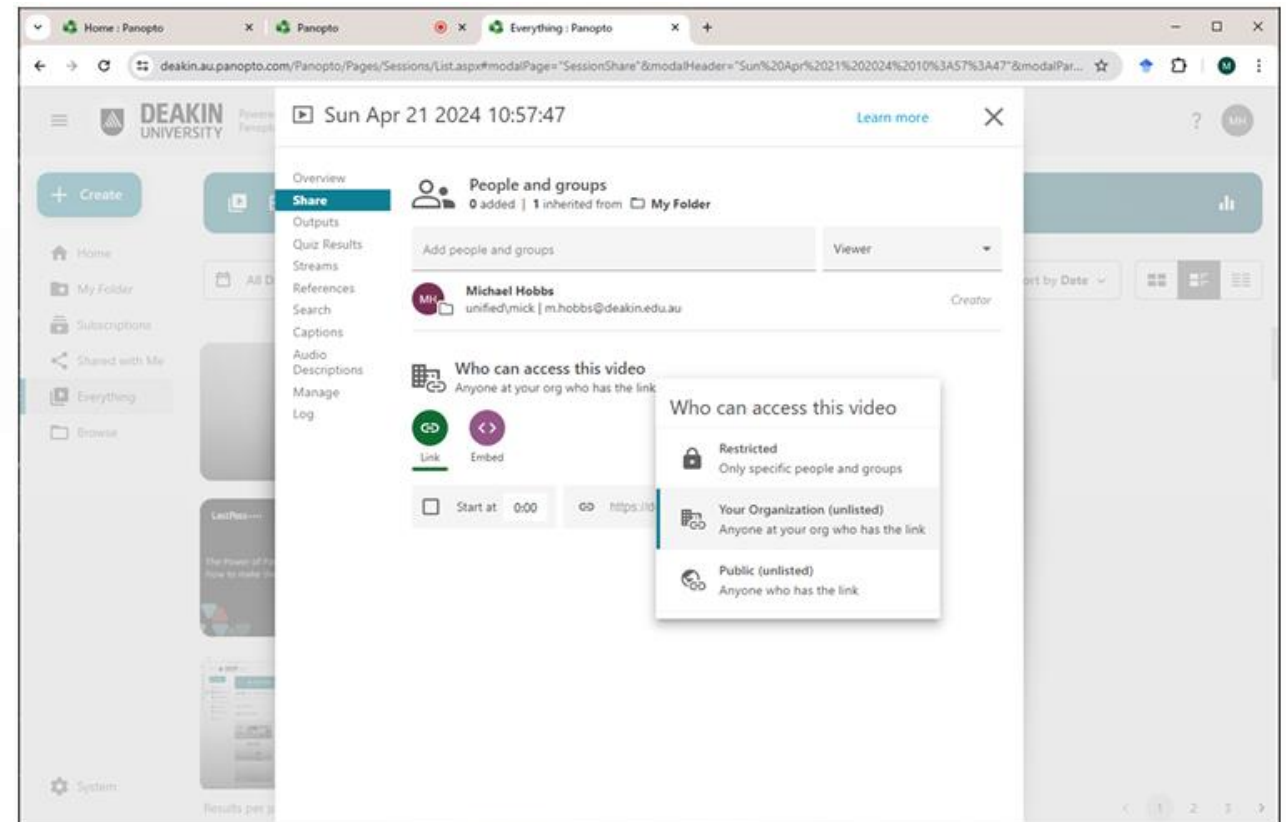
## IMPORTANT: BEFORE YOU START

Please discuss your ideas for the topic of this task with the Unit Chair (in an online class/workshop session or be email directly) before starting. This is to ensure you have selected a topic/area that has sufficient depth needed to meet the requirements of the task.

## What will you submit?

You should submit a **single PDF document** that includes:

- A brief introduction and overview of the topic/feature(s) you have selected for this task (that are expanded further in the webpage itself).
- A link to a **source code repository** containing your project (i.e., a github repo).
- A link to a narrated **video** of your project walkthrough (**10 mins max**) that highlights/demonstrates the awesome features your have developed and explains how these were implemented. The video should be uploaded to *Deakin Panopto* (<https://deakin.au.panopto.com/>) – ensuring this uploaded file is visible to people from this organisation (Deakin) with the link.



Task10.4.1 Panopto Settings



# Attention!

1. Please discuss your ideas for the topic of this task with your Unit Chair Via Ontrack chat area!
2. This is to ensure you have selected a topic/area that has sufficient depth needed to meet the requirements of the task.
3. For assignment feedback, our teaching team will provide them ASAP.
4. For 10.4HD, please pay attention that the plan should be discussed with unit chair beforehand (otherwise, there will be penalty-you could fail the task and not get HD).



# Ontrack task 10.5P: Draft Learning Summary

## Overview

This task provides you an opportunity to get **feedback** on your Learning Summary Report. The Learning Summary Report outlines how the work you have completed demonstrates that you have met all of the unit's learning outcomes and indicates the grade you believe you have demonstrated in your portfolio. In this report you can justify why you should be awarded this grade based on the work you have completed, what you have learnt, and the unit's assessment criteria.

## Submission Details

Submit a PDF print out of your Learning Summary Report for feedback.

## Instructions

Download the template for the Learning Summary Report from the task resources, and prepare a *first draft*.

**Remember:** The Learning Summary Report has a large impact on your final result. This is the document where you make a case that you should be awarded a certain grade. The more clearly you can demonstrate good quality work in this document the better grade you are likely to receive. Aim to be clear and concise, and refer to the work you have included to demonstrate your understanding.

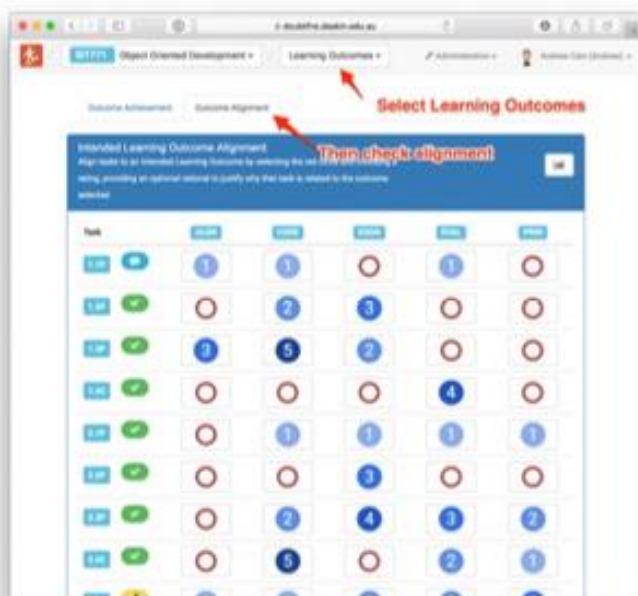
Also keep in mind that at this stage you are working on a **draft**, you will get feedback on this and can use that feedback to improve your report for inclusion in your **portfolio**.

The Learning Summary Report will go at the start of your portfolio. It will be followed by all of your tasks, report, diagrams, etc. In this summary just refer to the other pieces included in your portfolio rather than re-writing them here. Think of this like an index referring the reader to other pieces that demonstrate your achievements.

Comments: For the learning summary, only you can summaries what you have learnt in the unit, and where else you can improve. ChatGPT can only rewrite the requirements for you!

Try to include some of the following aspects:

- Use the alignment tool in OnTrack to check and adjust the links between the tasks you have completed and the unit learning outcomes. Make sure that you have a range of evidence for each of the unit learning outcomes.



Check how you have aligned things with the Alignment Tool

- Indicate the grade you think you have achieved (or will achieve by the end of the unit).
- Write something up for the reflections section of the Learning Summary Report, or list some dot points you can expand later.

Submit the *draft* for feedback, and it will be signed off as complete.

## What next?

Your work on the unit tasks should have helped you to achieve the unit learning outcomes. In completing these tasks you will have created outputs that help you demonstrate this achievement. So what now?

The assessment for this unit is by **portfolio**, so now you should be starting to

think about putting all of your tasks together into your portfolio for final submission. This is where all of your tasks are finally assessed in order to determine your final grade!



Create your portfolio using OnTrack

Use the **Portfolio Creation** tool in OnTrack to guide you through the steps needed to create your portfolio. This will combine together all of your work related to the tasks with the *final* version of your learning summary report. Once it is created you can download it and review it to make sure that you have included all of the aspects you think show off what you have achieved.

# Draft Learning Summary example

### Self-Assessment Details

The following checklists provide an overview of my self-assessment for this unit.

	Pass (D)	Credit (C)	Distinction (B)	High Distinction (A)
Self-Assessment				✓

Self-Assessment Statement

### Declaration

I declare that this portfolio is my individual work. I have not copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part of this submission been written for me by another person.

Signature: **Sam Baird**

### Portfolio Overview

Start with something like "This portfolio includes work that demonstrates that I have achieve all Unit Learning Outcomes for SIT774 Unit Title to a **High Distinction** level."

This learning journey started, like many other IT units, with the most daunting part... installing the required software!

Once everything was downloaded the real work could begin. I enjoyed a lot of these tasks, I especially enjoyed than many allowed for creativity as well as simply mastering the programming techniques. When I say creativity, I don't just mean my awesome sense of humor... but also the creativity to research beyond the task requirements to include additional features in your web pages.

I admit I did enjoy, and felt a huge sense of achievement when I managed to get encrypted passwords in the form of a "cypher-key" json (created by an additional plugin which I downloaded) and managed to send them via https(another plug-in).

I was disappointed that I didn't get a chance to set up features I found such as a 'bootstrap boilerplate' which enables you to do things like enable negative margins.

All in all, I think you'll agree that I went beyond task requirements in many cases. I admit my commenting wasn't always great, but this was mainly a time issue.

I also think that most of my code was pretty neat... once id had more practice with bootstrap (had to go back and correct earlier work). Java script was a bit easier as its more like languages I'm used to such as C++ and C#

### Reflections

The most important things I learnt:

HTML and Bootstrap defiantly! Most web pages I look at now I would probably be able to build with an html text document... well a close approximation anyway... I have always been a believer in learning from the ground up and I think this would be a skill useful for solving many business problems.

It is amazing just how many websites use bootstrap components such as cards and collapsibles.

I also predict that JavaScript gets used a lot to spit out html code, I think this is a basic skill that will be very useful.

I would love to add server side here but I don't think there would be as much use for it unless I get into backend web app programming... which I honestly wouldn't mind.

I feel I learnt these topics, concepts, and/or tools really well:

HTML I think I'm pretty confident with, JavaScript too, I could have used some more time/material on server side as well as actually publishing the website.

Bootstrap/css, I think there was enough time devoted to these but there's really no limit to the learning curve. I think you would generally need a lot of practice to become efficient at designing in depth css/bootstrap components.

I found the following topics particularly challenging:

The most challenging thing, I would say, was learning to position and format everything how you wanted it with bootstrap AND CSS I had to research a lot of stuff in order to trouble shoot.

Learning server side took me a while as well, it was basically a matter of figuring out one small challenge at a time.

I found the following topics particularly interesting:

As already mentioned I enjoyed the HD task... I think my favorite part was when server side was introduced. That said. I think I did a pretty neat job designing some of the earlier web pages. I really just liked seeing the page come together as I enabled more features on it. It was very satisfying originally designing like a wire frame and then adding increasingly cool features.

As you can probably tell, I also fancy myself as something of a comedian when given the chance. I appreciate you putting up with my humor.

Part of me wishes I had more time for designing webpage content. I know this is an optional extra so I left it out when pressed for time.

I still need to work on the following areas:

Just generally increasing my knowledge base, I enjoy learning new features, if there's one thing id like to get better at its using CSS and bootstrap. But I guess that's just practice, there where times I spent ages trying to center a button or line of text, or adjust the boarders/backgrounds/margins of divs and paragraphs. Margins where the worst!

I would also like to learn how to manipulate features in static web pages, so you don't need to write out an entire, complex html, with styles, scripts ect, as a whole lot of strings with no formatting. I found a plugin that could supposedly do this by allowing java script commands which allow the programmer to send data packets in messages from the server that are seen by the client's browser and can be defined as variables in the html, but I didn't have time to play around with it and I didn't bookmark it.

The things that helped me most were:

As with most units, the weekly workshops where awesome! By far one of the best learning resources in my entire student career! I also made use of the help hub, I appreciate that they where willing to look into the tasks further. This is where I found the bootstrap boilerplate... even though I didn't get to use it.

It was also helpful how the tasks where structured, I am a fan of OnTrack, I love the interface (I think there's a bit of what you call gamification). I also love that the tasks are well spaced out and have a solid increase in difficulty towards the end of the unit. There was only 1 week where there was a total of around 10 tasks due at once which put on a bit too much stress when you factor in work as well.

My progress in this unit was ...:

I think I did pretty well and got most tasks in on time, I did need a couple of extensions with the website but that was mainly due to the volume of other work due at the same time.

I also think the HD should be left to be due in last, once you've learned how to use databases, as there's a lot of ideas you can get from this.



If I did this unit again, I would do the following things differently:

Use more design tasks as templates for the website, this would have saved a lot of time... that said... the tree pages where some of my best jokes!

I would have liked to put more on my website... but I didn't have the time. I admit I probably spent a lot of unnecessary time designing the initial pages, but it was fun writing them.

SUGGESTED IMPROVEMENT:




Get us to publish the website so you can actually look through them... either that or use videos... zoom is really easy, you don't even have to switch on your camera, I just found there where some things that where much easier to show on screen than to write about... but that might just be me.

Al in all I enjoyed each new challenge in this unit. Due dates where stressful at times but you where pretty good with extensions. Thanks for some awesome tasks.



# Tips

For learning summary, please look through all your tasks, if there :

1.  Time exceed, our teaching team will mark it after you submit your portfolio.
- 2  Resubmit, make sure all resubmit tasks are fixed and completed, otherwise you will be downgrade to current target level.
- 3  Only full of green ticks and time exceed tasks, (wen do not accept Resubmit tasks), then you can submit your portfolio.

With the completion of this week, we also reach the end of this unit.

Hopefully you had a good learning experience and feel that you have learned a lot in this unit as a whole.

At week 11 workshop, we will provide you a **Q&A session**.

