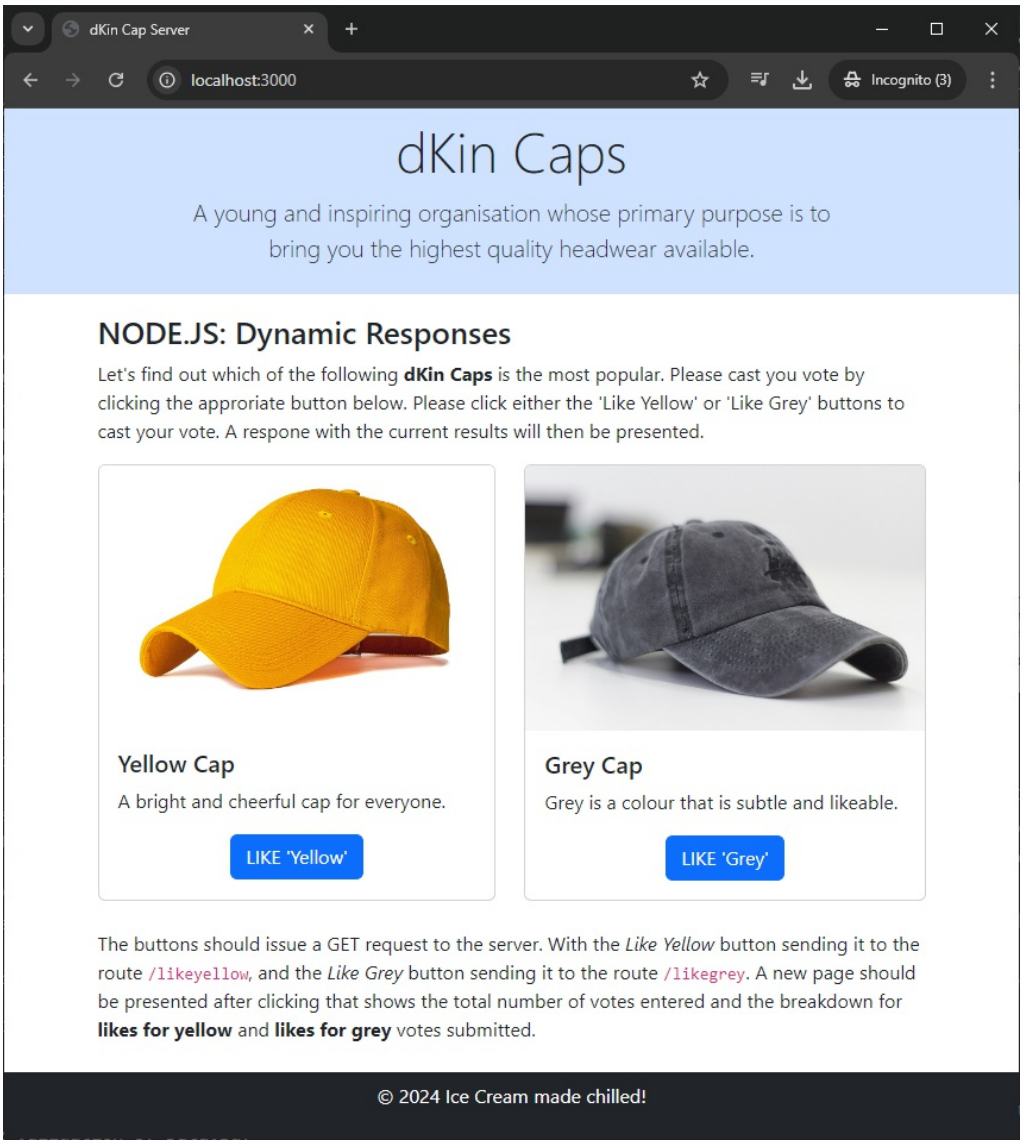
8.2C: Writing a Server Application

Tasks

In this task you are asked to demonstrate how a web server can be coded to serve *dynamic* pages.

Specifically, here you are asked to build a very simple web server application that enables users to vote on if the *like* or *don't like* ice cream. The user selects their vote by clicking one of two buttons on a static web page (which is provided in the resources link of the Task8-2C ontrack site), as shown in the screenshot below:

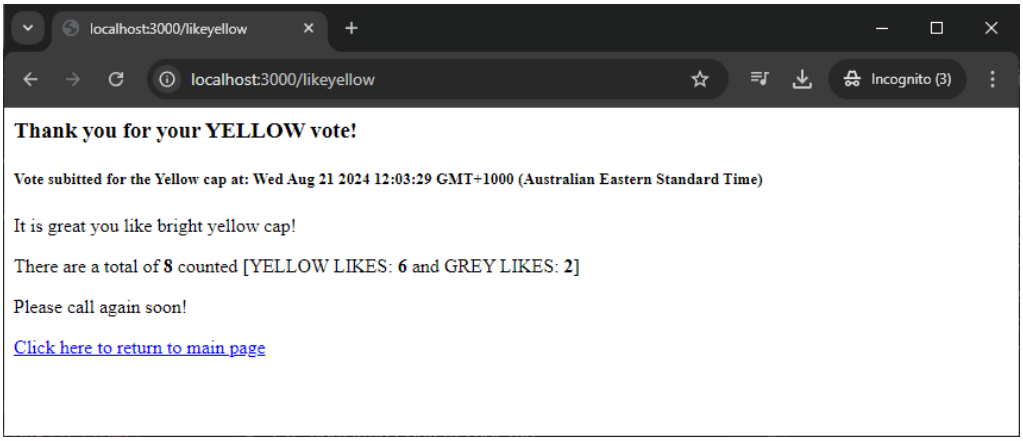


Task8.2.1 Node.js application serving the `index.html` Ice Cream Voting Home Page

The web server should handle **two** GET requests, one for a *LIKE YELLOW* vote and the other for a *LIKE GREY* vote. On receipt of the vote (respective GET request), the server should keep count of the number of votes submitted for *LIKES YELLOW*

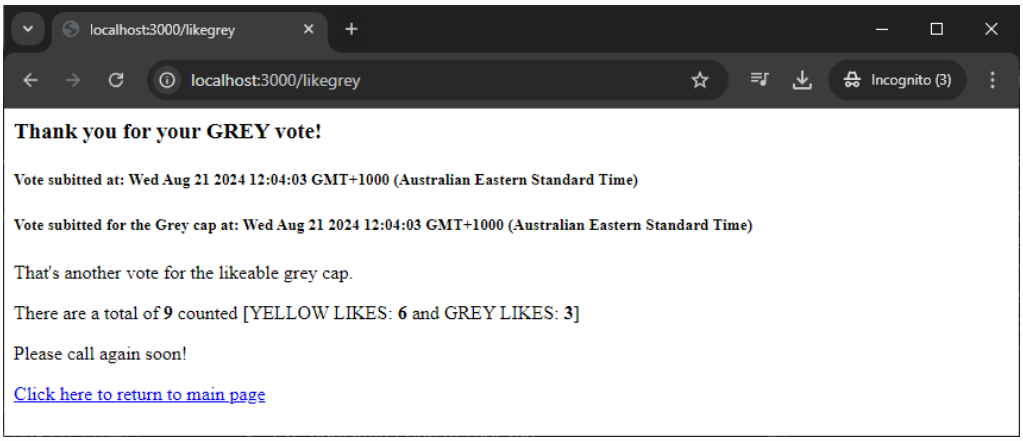
and *LIKES GREY*, then return a response page that contains the date/time the vote was submitted and the tally of total votes, # yellow likes, and # of grey likes. A hyperlink back to the root/main page should also be given.

A screenshots for a sample of a **LIKES YELLOW** vote response is shown below:



Task8.2.2 Node.js application serving the dynamically created LIKES YELLOW vote

Similarly, the page showing a **LIKES GREY** vote response could look like:



Task8.2.3 Node.js application serving the dynamically created LIKES GREY vote

> **NOTE:** The counting of votes can be held in a simple global variable in the server… there is **no need** for this data to be stored **permanently** (e.g., in a database). Here it is fine for the data (vote) values would be reset back to zero when the server is restarted.

Steps

Follow the steps below to complete this task:

1. Create your own local directory to hold the new Task 8.2C project.
2. Use express module to create a *Node.js application* (i.e., using the `index.js` from your Task 8.1) that will set a local web server. The server listens to the port **3000** and will have handlers for *404* and *500* errors.

3. Modify your `index.js` file to:
    - remove the hardcoded response for the `GET` on the `/` route (as done in Task 8.1P) and change it to serve pages form the `public_html` folder, as done in previous weeks (sample of this code is shown in the snippet below)
    - Define some global variables that will hold the total number of `likesyellow` and `likesgrey` votes.
4. Add a route handler for a `GET` request on `/likeyellow`. This would enable access to `http://localhost:3000/likeyellow` using the respective button on the main page or via a web browser.
    - Increase the number of `likes yellow` votes.
    - Prepare a response message that includes the vote data, and a hyperlink back to the main page.
    - Send the response back to the client.
5. Add a route handler for a `GET` request on `/likegrey`. This would enable access to `http://localhost:3000/likegrey` using the respective button on the main page or via a web browser.
    - Increase the number of `likes grey` votes.
    - Prepare a response message that includes the vote data, and a hyperlink back to the main page.
    - Send the response back to the client.
6. Run the command `npm run start:dev` (to run the `index.js`) in a Command Prompt (Windows) or Terminal (Mac OS) within your local directory to make the server work.
7. Use the provided `index.html` file to access your newly created `/likeyellow` and `/likegrey` route handlers (or access them directly through `http://localhost:3000/likeyellow` and `http://localhost:3000/likegrey` URLs)
8. Go back to the Command Prompt or Terminal, and shutdown the server.

Hints

- Visit https://expressjs.com/en/guide/writing-middleware.html website to see how to write code for *express functions*.
- The examples pages shown in the screenshots above have the dynamically created webpages using raw HTML fragments.
    - OPTIONAL STEP: How could you get the server to send back a `bootstrap` formatted response?
- The structure of the file `index.js` with the *new added code* (as highlighted below) could be:

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

const app = express();
const port = 3000;

// Have the logging code
app.use(morgan('common'));

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


// << ADD GLOABL VARIABLES HERE >>
// ....

// << ADD CODE HERE >>
// The handler for a GET request on route '/likeyellow' and return a dynamic page (fragment)
// ...

// << ADD CODE HERE >>
// The handler for a GET request on route '/likegrey' and return a dynamic page (fragment)
// ...


// The last route handler can be used to return your own error messages
// it is expecting an 'Error' object as the first parameter, which is generated
// internally from Express when an error is detected.
app.use( (error, request, response, next) => {
    // we may use properties of the error object
    // here and next(err) appropriately, or if
    // we possibly recovered from the error, simply next().
    let errorStatus = error.status || 500;
    response.status(errorStatus);
    response.send(`<h3>${errorStatus}: ${error.toString()}</h3>`);
});


// Tell our application to listen to requests at port 3000 on the localhost
app.listen(port, ()=> {
    // When the application starts, print to the console that our app is
    // running at http://localhost:3000. 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`)
})
```

What will you submit?

You should submit:

- Source code of the file `index.js` .
- Screenshot of the browser window showing an example of the *Likes Yellow vote* with statistics, when visiting `http://localhost:3000/likeyellow`.
- Screenshot of the browser window showing an example of the *Likes Grey vote* with statistics, when visiting `http://localhost:3000/likegrey`.