

---

# Lab 5

## Web Application Development 2

---

Over the next few weeks you are going to develop a guestbook application starting by designing a static interface then adding additional functionality over subsequent weeks.

Before starting to insert dynamic data we will first prepare a static version of the user interface. We will later use these as templates to insert application data from the database into the pages dynamically.

The example we are going to develop is a guest book web application.

We will need 2 pages:

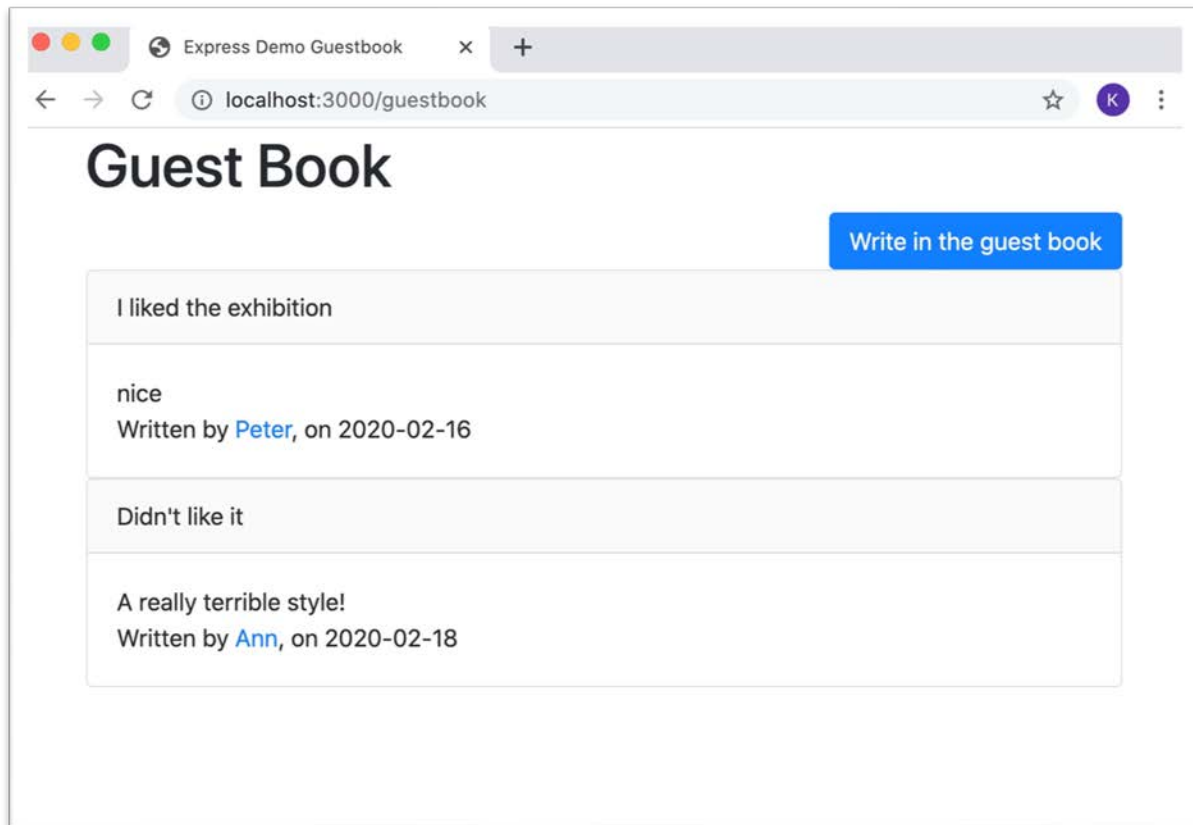
1. A page showing the guestbook entries
2. A page that allows a user to add a new entry to the guest book

Using your skills from the previous Web Application Development 1 module write the mark up for static versions of the user interface of a guest book web application.

Examples of such pages are shown below. At this point simply assume some entries for the guestbook and hard-code these into the pages.

We will later see how we can replace the hard-coded data with application data using templates.

## Exercise 1: The guest book page



Consider the information that is shown on the guestbook page. Also consider what such a page could look like once many users have left comments.

Your design should at least show:

- A page title
- A button to link to a page for new entries to the guest book.
- At least 2 guest book entries, each of which shows:
  - A subject line and
  - The actual guest book message. This should include
    - the name of the author
    - and the date.

An example of mark up for such a page is shown on the next page.

---

```
<html>
  <head>
    <title>Express Demo Guestbook</title>
  </head>
  <body>
    <h1>Guest Book</h1>
    <div>
      <button type="button">
        Write in the guest book
      </button>
    </div>
    <div>
      <div>
        I liked the exhibition
      </div>
      <div>
        <div>
          nice
        </div>
        <div>
          Written by <a href=#>Peter</a>, on 2020-02-16
        </div>
      </div>
    </div>
    <div>
      <div>
        Didn't like it
      </div>
      <div>
        <div>
          A really terrible style!
        </div>
        <div>
          Written by <a href=#>Ann</a>, on 2020-02-18
        </div>
      </div>
    </div>
  </body>
</html>
```

You can format the page and the html elements using a CSS framework such as Bootstrap.

If you want to use the Bootstrap CSS then you need to include links to the framework in the head section of the page.

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">
```

and then use the pre-written styles supplied in the Bootstrap CSS to style elements

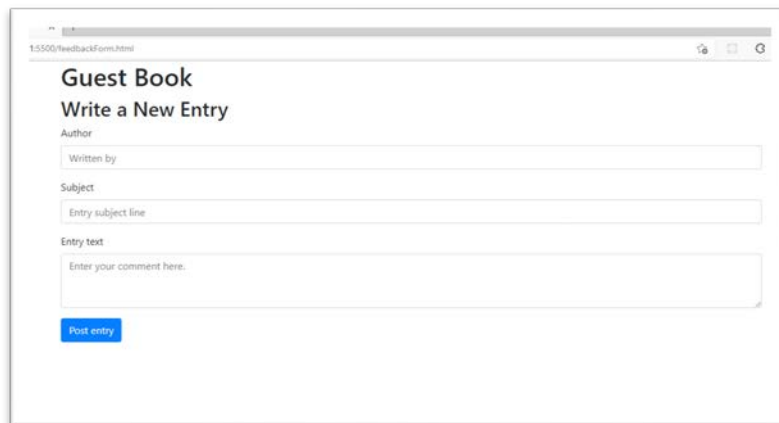


---

## Exercise 2: The new entries page

This page should allow the user to submit a new entry. An HTML form can be used for this purpose.

An example of such a page is shown below.



The screenshot shows a web browser window with the address bar displaying '15500/feedbackForm.html'. The page content is titled 'Guest Book' and 'Write a New Entry'. It features three input fields: 'Author' with the placeholder 'Written by', 'Subject' with the placeholder 'Entry subject line', and 'Entry text' with the placeholder 'Enter your comment here.'. A blue button labeled 'Post entry' is positioned below the text area.

On the page there should be form input fields for all required information:

- the name of the author
- the subject line and
- the actual message.
- there should also be a submit button.

Create and style the page. You can use bootstrap CCS framework to style your pages. Alternatively, you can develop a CSS yourself and link it to the page.

An example of a markup of such a page is shown on the next page.



---

We are going to develop a guestbook application which will use a NeDB database to store messages from visitors and some html template pages to display these messages on screen.

We will implement the model component of the MVC architecture using a NeDB database in Lab 6 and the views using Mustache templates in Lab 7. This lab sets up a controller.

### Exercise 3: Add a controller to the application

Set up a new web application using Node and Express as follows:

Create a new folder called `guestbook`. Open the folder in Visual Studio Code, then open a terminal window (menu bar | Terminal | New Terminal).

Initialise the new project using npm by typing: **npm init** in the terminal window and setting up a `package.json` file.

Install the express module by typing: **npm install express** in the terminal.

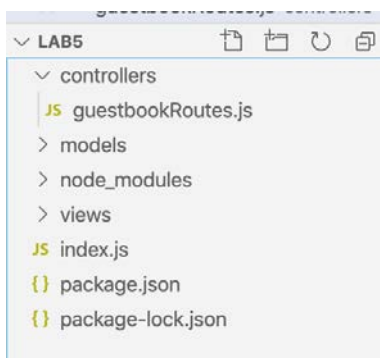
Create new file called `index.js` in the `guestbook` folder at the root of your application and then set up three folders called

- `controllers`,
- `models` and
- `views`.

to hold the code for the respective elements of the application.

In the `controllers` sub folder create a file called `guestbookRoutes.js` which we will use hold the controller functionality.

The application should now have the following folder structure:



Open `index.js` and add code to handle requests to the root.

---

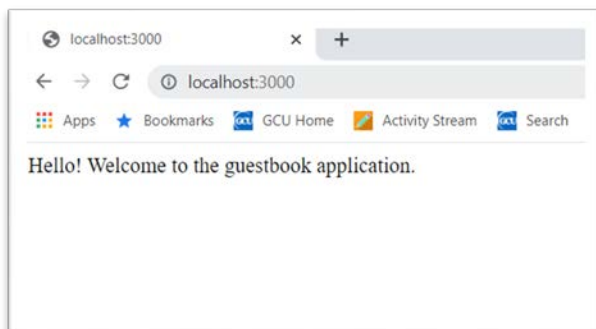
```
const express = require('express');
const app = express();

app.get('/', function(req, res) {
  res.send('Hello! Welcome to the guestbook application.');
```

}}

```
app.listen(3000, () => {
  console.log('Server started on port 3000. Ctrl^c to quit.');
```

});



We will now add a controller for this request handler.

Implement the request handler for requests to the root of the application in `guestbookRoutes.js`

In `guestbookRoutes.js` import the Express module so that we can use the `express.Router()` class.

```
const express = require('express');
```

Then create an instance of the Router class.

```
const router = express.Router();
```

Using the router instance to implement the request handler for HTTP GET requests to the root of the application.

```
router.get("/", function(req, res) {
  res.send('Hello and welcome to the guestbook application.');
```

});

Make the new router code accessible in `index.js` by exporting it from the new router module. Do this by adding the following code to the end of `guestbookRoutes.js`



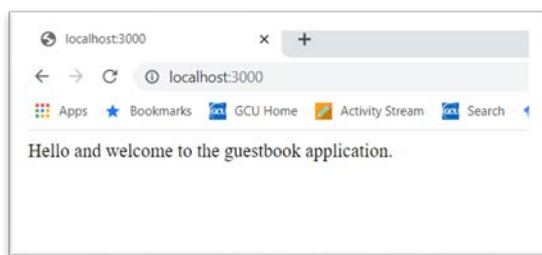
---

```
module.exports = router;
```

In index.js import the new router and map it to all requests starting from the root of the application. Then remove the old request handler for requests to the root.

```
const router = require('./controllers/guestbookRoutes');  
app.use('/', router);
```

Change the message to “Hello and welcome to the guestbook application”. Test the application to make sure it is working correctly.



#### Exercise 4: Add request handlers to guestbookRouter.js

Add a request handler for /guestbook to guestbookRouter.js.

```
router.get('/guestbook', function(req, res) {  
    res.send('<h1>Guestbook Messages</h1>');  
})
```

Next, we want to add a handler to guestbookRouter.js. to serve a static page called about.html when /about is requested. Add a folder called public to your application, create a short About us page called about.html.

Remember that in order to serve static files we need use the path module to access the file system.

Install the path module using terminal: **npm install path**

Import the module and add the code to handle static paths to index.js:

```
const path = require('path');  
const public = path.join(__dirname, 'public');  
app.use(express.static(public));
```

Note that the code that identifies the folder ‘public’ as the folder for static resources in this application is added to index.js (and not guestbook.js) as it belongs to the application setup and not the routing.

---

While you are editing the index.js file you should also add the body-parser middleware which parses the incoming request bodies before you handle them

```
const bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({extended: false}));
```

Then write a handler in guestbook.js to re-direct any requests to /about to the static file.

```
router.get('/about', function(req, res) {
  res.redirect('/about.html');
})
```

Add functionality for the 404 - Not found response from index and re-create this functionality in guestbookRoutes.js We use the Router.use() method to implement the functionality for returning a response.

```
router.use(function(req, res) {
  res.status(404);
  res.type('text/plain');
  res.send('404 Not found.');
```

```
});
```

Add functionality for internal server errors to guestbookRoutes.js. Here we want to provide a response in case the request processing has caused an internal error. We again use the router.use() method, but this time with the all arguments, including the error object.

```
router.use(function(err, req, res, next) {
  res.status(500);
  res.type('text/plain');
  res.send('Internal Server Error.');
```

```
});
```

---

### Exercise 5: Separation of router and controller

In order to modularise the application further we can separate the definition of the routes, i.e. the URLs that are supported by the application (the routes) from the logic that produces the response (the controller) when these routes are called.

Create a new folder called routes in the application.

Move the guestbookRoutes.js file into the new folder.

Ensure that you update the import statement in index.js to reflect the new location of guestbookRoutes.js.

```
const router = require('./routes/guestbookRoutes');
```

Now create a new file guestbookControllers.js in the controllers folder. This file will process the responses.

In guestbookControllers.js set up and export a callback function that is going to produce the response for requests to /guestbook. This will currently just be a stub with no real functionality. Over the next few labs we will develop it into a dynamic page.

```
exports.entries_list = function(req, res) {  
  res.send('<h1>Guestbook Messages</h1><p>Not yet implemented: will show a list  
of guest book entries.</p>');  
}
```

Replace the callback functions in guestbookRoutes with the name of the function you have just written and exported from guestbookController.js . Ensure access to guestbookController.js by importing the module into guestbookRoutes.js.

```
const controller = require('../controllers/guestbookControllers.js');  
...           ...           ...  
...           ...           ...  
  
router.get('/guestbook', controller.entries_list);
```

Then create similar functions for responses to the root '/'

In guestbookControllers.js export the callback function that produces the response.

```
exports.landing_page = function(req, res) {  
  res.send('<h1>Welcome to the guestbook application.</h1>');  
}
```

In guestbookRoutes replace the callback functions with the name of function you have just written and exported from guestbookController.js

```
router.get("/", controller.landing_page);
```

---

## Exercise 6: Add functionality for requests to an additional URL /new

This page will eventually show a form that allows users to add new entries.

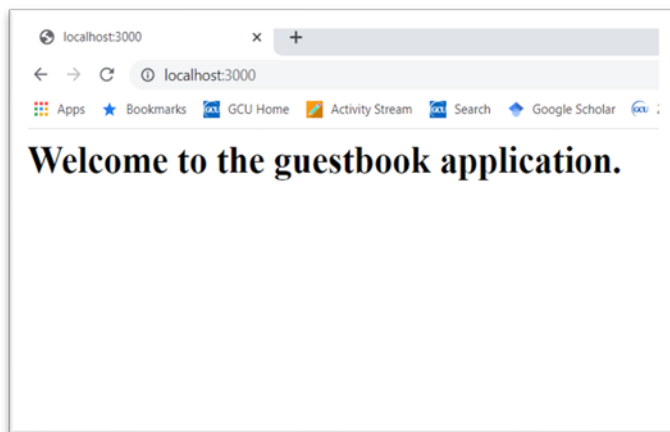
Add the new route `new_entry` to `guestbookRoutes.js` :

Add the controller which sends the following message to `guestbookControllers.js`:

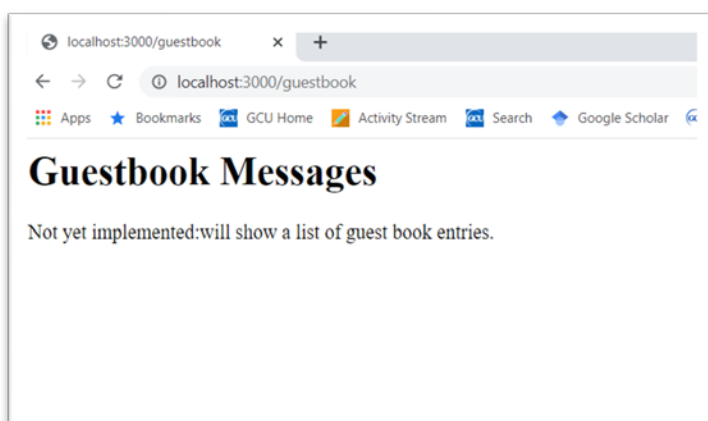
```
<h1>Not yet implemented: show a new entry page.</h1>
```

**Test all application routes.**

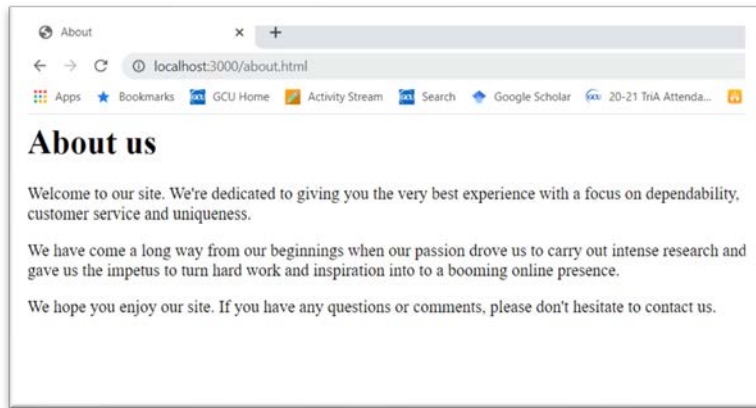
`http://localhost:3000/`



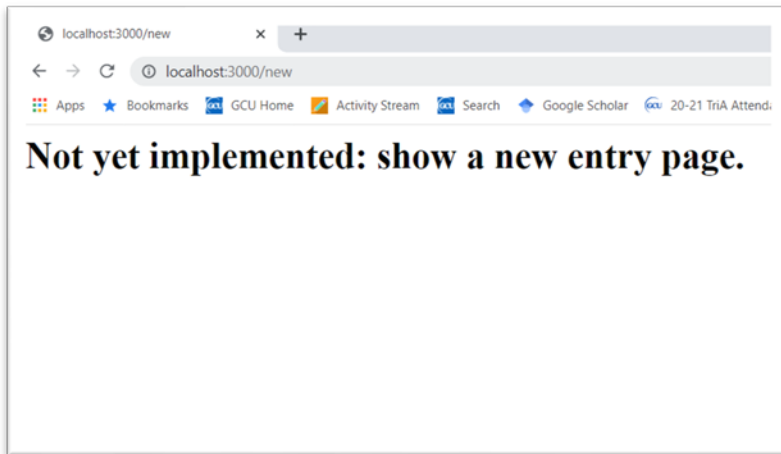
`http://localhost:3000/guestbook`



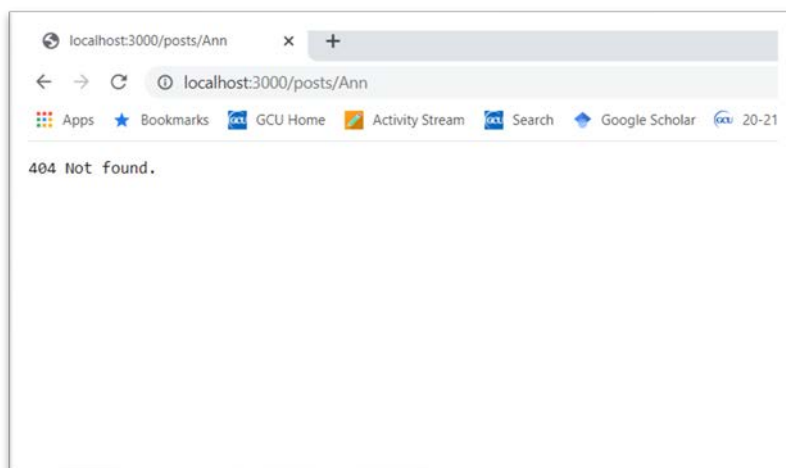
`http://localhost:3000/about` (is redirected to `about.html`)



`http://localhost:3000/new`



`http://localhost:3000/something` (testing unhandled requests)



---

## Appendix A    ./routes/guestbookRoutes.js

```
const express = require('express');
const router = express.Router();
const controller = require('../controllers/guestbookControllers.js');

router.get("/", controller.landing_page);

router.get('/guestbook', controller.entries_list);

router.get('/new', controller.new_entry);

router.get('/about', function(req, res) {
    res.redirect('/about.html');
})
router.use(function(req, res) {
    res.status(404);
    res.type('text/plain');
    res.send('404 Not found.');
```

```
    })
    router.use(function(err, req, res, next) {
        res.status(500);
        res.type('text/plain');
        res.send('Internal Server Error.');
```

```
    })
    module.exports = router;
```

---

## Appendix B     ./controllers/guestbookControllers.js

```
exports.entries_list = function(req, res) {  
    res.send('<h1>Guestbook Messages</h1><p>Not yet implemented:will show a list of  
guest book entries.</p>');  
}  
  
exports.landing_page = function(req, res) {  
    res.send('<h1>Welcome to the guestbook application.</h1>');  
}  
  
exports.new_entry = function(req, res) {  
    res.send('<h1>Not yet implemented: show a new entry page.</h1>');  
}
```