

Liberty Information Technology

Work Experience Programme

Developer Workshop – Student Guide

Contents

Introduction	3
Overview of Resources	3
So what's the challenge?	4
Getting Setup	5
Create a GitHub Account	5
GitHub Set-up:.....	6
Set-up GitHub on your machine.....	6
Config GitHub to use proxy	6
Create Repository	7
Project Challenge.....	11
Let's get started.....	11
Creating the Front End.....	12
Creating the Details Screen	16
Exercise 1	16
JavaScript + JQuery	17
Changing from the Personal Details page to Car Details page	18
Exercise 2	19
Exercise 3	20
Car Details Section	21
Exercise 4	22
More Validation	23
Exercise 5	23
Exercise 6	23
Quote Details.....	23
Exercise 7	24
Creating the Back-End	24
JSON	24
Exercise 8	26
Ajax	26
Exercise 9	27

Taking a look at the Server..... 29

 Exercise 10..... 31

 Exercise 11..... 32

What’s next? 32

Liberty Information Technology

Developer Workshop Student Guide

Introduction

Within this document you, the student are provided with a list of tasks to follow in order to complete the developer workshop part of the work experience programme.

This document acts as a guide that you can use to help navigate through the development tasks you are required to complete. The aim of the developer workshop, is to provide all students with an opportunity to learn and understand the basics of what software development is all about and how you can create your very own personalized web application.

By the end of this workshop, you will have created a responsive web app using some of the latest technology out there available to developers such as JavaScript, HTML, CSS and node.js to name a few.

Throughout this workshop, as you work through this guide, remember to work collaboratively with both your facilitators and other students around you. Although this is a personal project, which this guide will help you through, IT has always been a very interactive environment. There are lots of great resources available to enable you to learn how to do certain tasks, but none more so than the people around you in your group. Ask questions and work with others to create the best possible project that you can.

Overview of Resources

As we have already mentioned above, there are plenty of resources to help you to complete your project. For each development language that we will be using, there is a corresponding cheat sheet/ reference guide which will give you an overview of the language and some tips and

hints to get you started. There are also external resources which can be used to help further your knowledge. Remember though, the facilitators are here to help you learn so if you can't find what you're looking for or you're unsure, then make sure to ask them.

The cheat sheet/ reference guide will be provided on the day of the workshop. It is provided as a series of html documents, the first of which is referenceGuide.html which can be opened in any browser.

So what's the challenge?

All computer applications are generated from code which has been written by a developer somewhere, including the software that you are using to read this document. Increasingly we are seeing that the future of software development is leading to Web Applications and Websites, where the code and therefore the application can be accessed and used anywhere in the world as long as the user can connect the Internet.

Your challenge over the next couple of days is to create a car insurance quotation tool. Using the base template provided, expand the existing HTML and CSS to create a single page web application. The application should take in personal details and vehicle information which will be used to produce a quote by querying an API (application programming interface). The development languages which we will be using are:

- HTML & CSS for front end (i.e. user's view) display and styling
- JavaScript which will provide front end functionality
- JSON to model our data
- AJAX to transfer our data between client and server
- Node.js to act as a server

These will be explained in more depth later, so don't panic if you're not sure what they are yet.

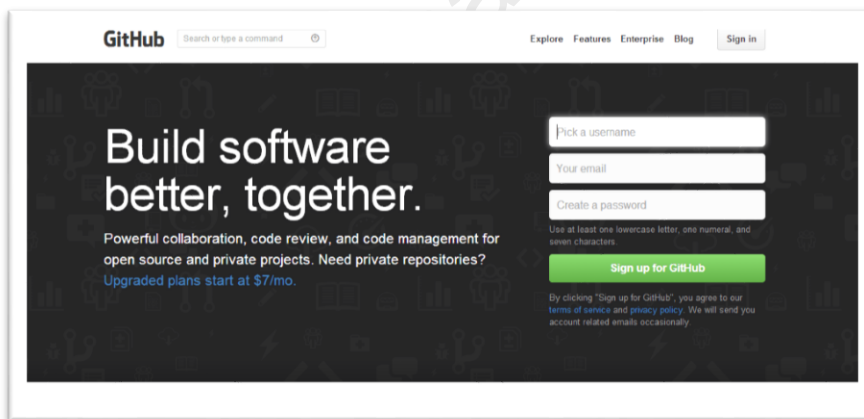
Getting Setup

Create a GitHub Account

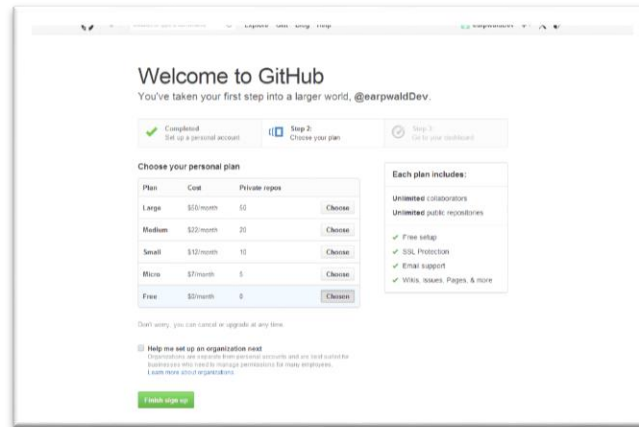
Before we start coding there is one very important thing we have to do; we need to get somewhere to save our code to. For the purposes of this workshop, we will be running our code from a locally hosted server, but in order to make sure that you get to keep a copy of your work, we are going to use a service called [Github](https://github.com). Github allows us to create places to store our code online which, similar to the web app you are about to create, can be accessed from anywhere over the internet. The other benefit of storing our code here is that we can keep backups of our code and should any issues occur, we can revert back to an earlier version. **This is known as source control.**

In order to use *GitHub* you need to create an account so let's get started:

1. Open up Chrome and go to www.github.com, you should see something like the following:



2. Pick a username for your *GitHub* account and enter your email and a password for the new account and click the "Sign up" button.
3. On the next screen you'll be offered payment options, but we don't need to bother with that as we are going to be creating a free account.
4. Click Finish "**Sign up**" to create your account.

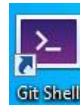


GitHub Set-up:

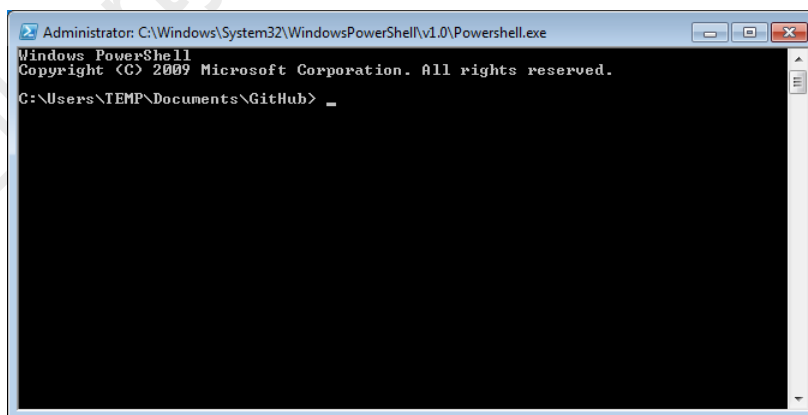
In order to create your repository, we have carry out a few set-up tasks. First of all, we will configure GitHub so that it can connect to the internet. Once that is done, we will then set up a local copy of code which you can use as part of this workshop.

Set-up GitHub on your machine

Config GitHub to use proxy



1. Locate the “Git Shell” shortcut on your desktop
2. Open up “Git Shell”
3. A command (cmd) window should appear similar to this:



4. Type the following two commands into the the command line, making sure to replace the values for USERNAME and PASSWORD below with the username and password you have been allocated. Your facilitator will give you the PROXY value. Type one command in at a time remembering to hit enter after each one.

- ✓ `git config --global http.proxy http://USERNAME:PASSWORD@PROXY:PORTNUMBER`
- ✓ `git config --global https.proxy https://USERNAME:PASSWORD@PROXY:PORTNUMBER`



The reason we have to do this is that your computer is currently behind a secure firewall to prevent all sorts of nasty things happening. So in order to allow GitHub access to your machine you have to pass in details to the proxy in order to all authentication.

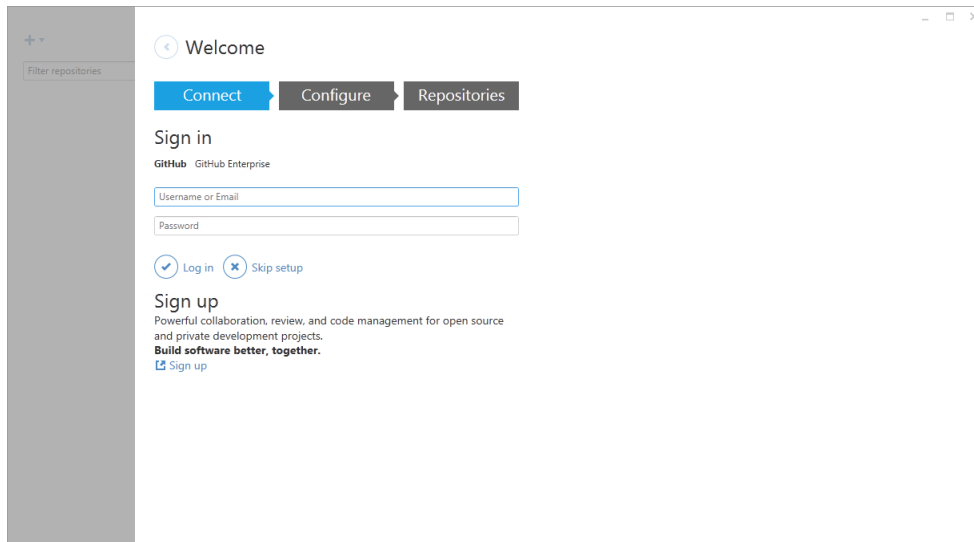
5. If entered successfully there should be no errors. If that are any errors ask your facilitator for help.
6. Once completed you can close the “*Git Shell*” window.

Create Repository

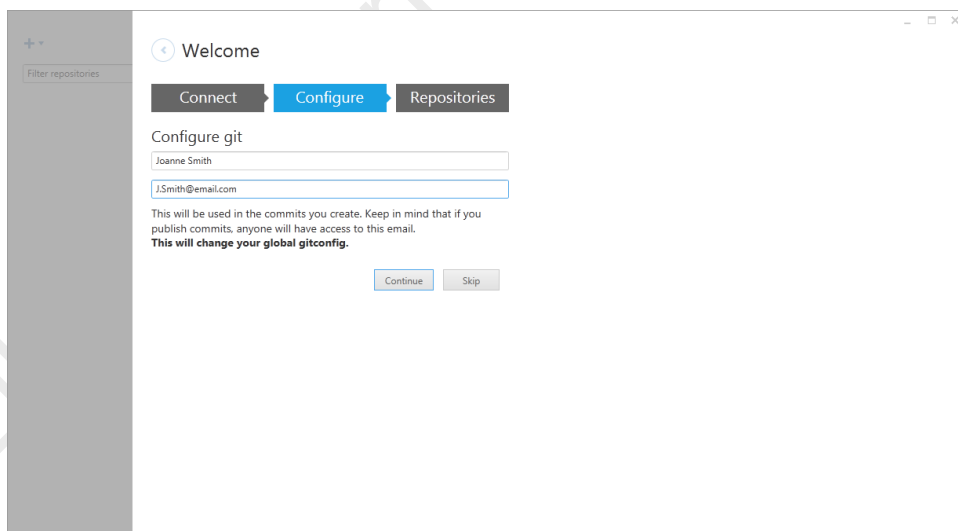
To create the repository where we will save all our files too, we are going to start up the GitHub client and download a local copy of the code.



1. Locate the “*GitHub*” shortcut on your desktop
2. Open up “*GitHub*”
3. You should see the welcome screen:

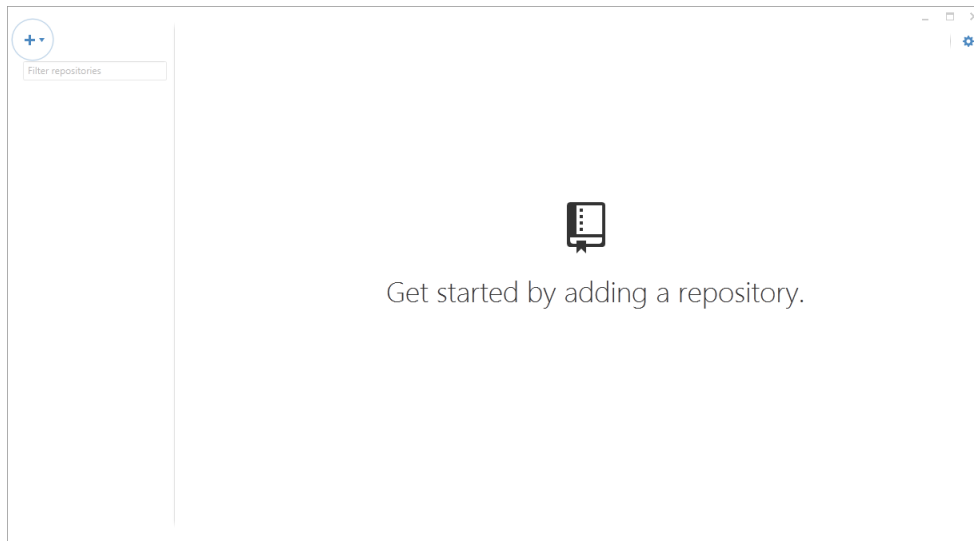


4. Using the details of the account you just created from above in the *Create a GitHub Account* section login into GitHub.
5. Click “Log in”
6. This should bring up the configure screen which should have your name and email already populated (based on the details you provided when creating your account).



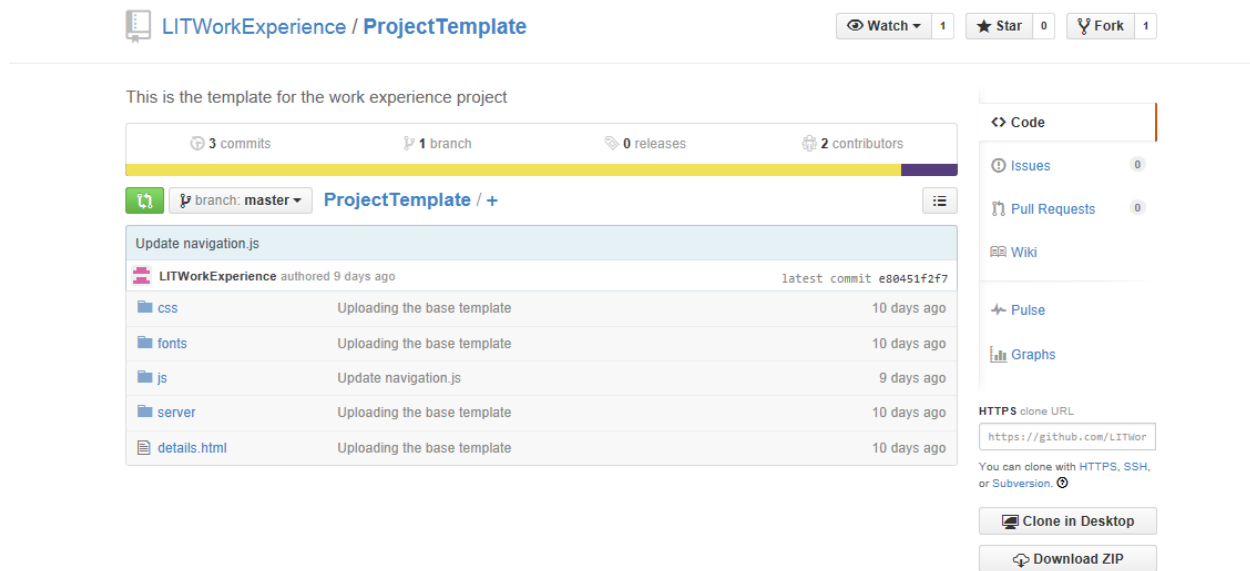
7. Make sure the details are correct and click “continue”.
8. You will then be asked to find local repositories. You can just click “Skip”.

9. You should now be presented with this screen:




10. Visit the LIT Work Experience GitHub page -

<https://github.com/LITWorkExperience/ProjectTemplate>. You may need to log back in using your details if you have logged out. It should look something similar to this:



11. In order to get a copy of the source code you need to *Fork* from the LIT master copy. To

do this click on the  **Fork** option in the top right hand corner.

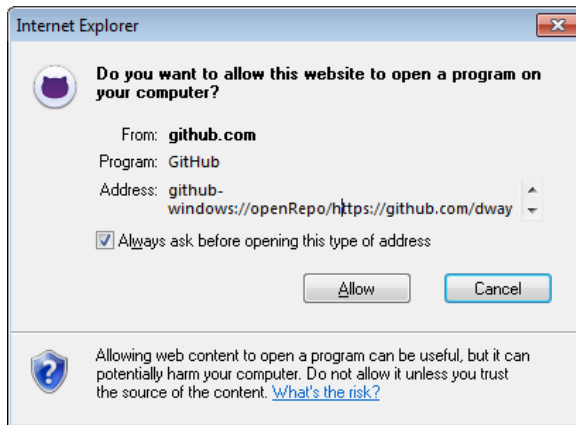


A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

12. Now that you have a forked version of the source code you need to clone it locally so that you can make local updates and sync them to your account. In order to do this click

on the  button on the right hand side.

13. You may get a warning message. Just click allow:



14. The “*GitHub*” desktop application should now open up and ask you to select a directory to save your code too. You can just use the default *GitHub* directory selected and click OK.

15. You should now have a locally copy of the base template ready for development.

Project Challenge

To start off our project we are going to work from the base template which you just downloaded and sync'd from GitHub.

Let's get started

The next sections are going to be the actual coding steps that will create your web application and the server. Before diving straight into the code, let's have a look at the structure of our overall application so that it is easy to see what it is that we are trying to achieve.

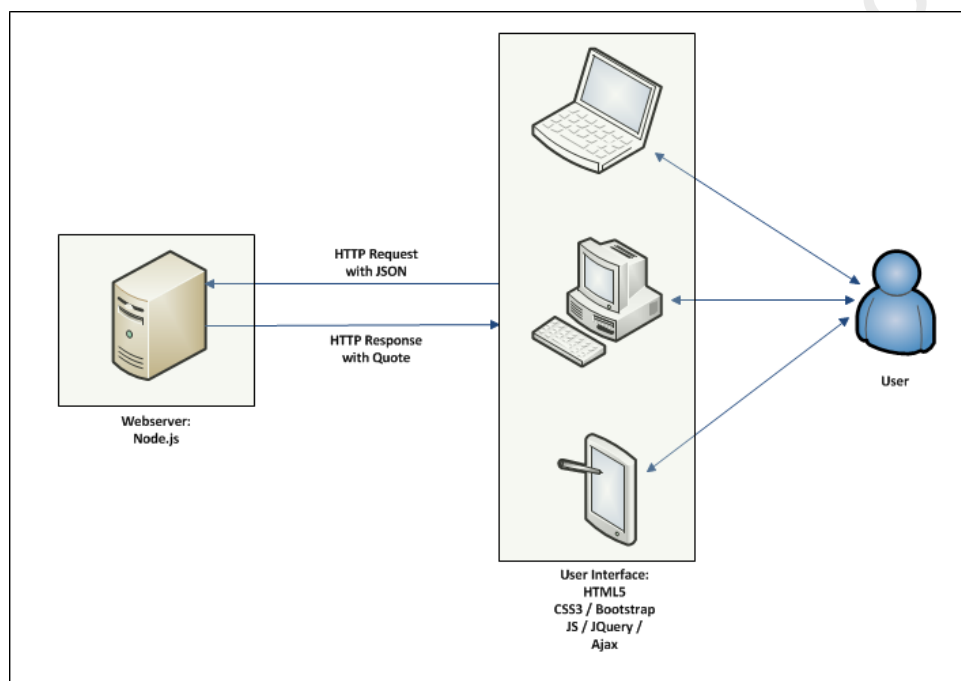


Figure 1 - System Diagram for Car Insurance Application

As can be seen by our system diagram, the system can be broken down into separate distinct parts. Within web development there are two distinct sections, one is known as the front-end and the other the back-end, which are just different ways of describing the user interface (UI). The front-end is what the user actually sees and can use, meanwhile, back-end is the server code which works out of sight on the back of the system. These two components are

completely separate from each other but can communicate and talk to each other through HTTP requests and responses (*we'll see more on those later on*).

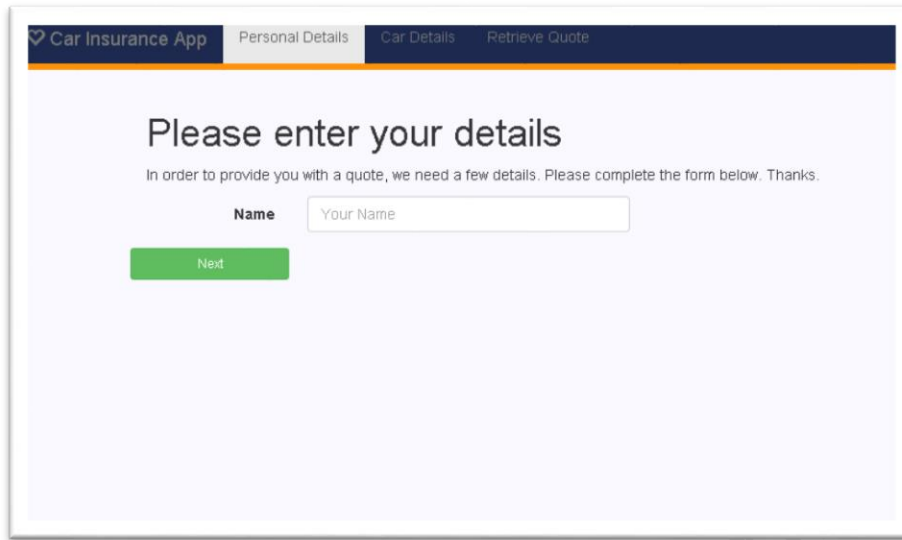
For our purposes, we are going to start building the front-end first. This means creating the HTML and CSS code which will be used by a browser, such as Chrome, to generate the user interface.

Creating the Front End

As mentioned, the front end is what the user gets to see and touch. Every browser takes in code files which are written by developers. These code files are used to generate a UI which displays the user (i.e. us). Using the template already provided as part of the base template we are going to add some new code which will add some functionality to our web app.

But first, let's see what we are starting from:

1. Open up the Chrome browser
2. Open the webpage "details.HTML" from your repository (*you can do this by clicking and dragging the file into the browser window*). You should see something like this:



This is a relatively basic webpage that currently just asks for a name and provides the user with a next button to move to the next section.

Note: *The navigation and the styling of the site have been provided for you. We are using twitters bootstrap in order to remove a lot of the need to use CSS to style the site, which in itself can take a long time to get it right. There is more about bootstrap in our developer guide.*

Our first challenge is to add the rest of the UI elements to the page, so that more fields are displayed.

Let's take a look at the code we are going to be modifying. For editing all the files today, we are going to be using Sublime Text Editor; however, any plain text editor can be used but some will not provide the same functionality as sublime does i.e. syntax highlighting:

1. Open up Sublime and open the **details.HTML** page within it.

This is the HTML code which is the basis of all websites or web applications. Don't be worried if you have never seen this before, it can look very complicated when you first see it but it is relatively easy to understand.

HTML is a collection of tags and attributes from a preset list which are enclosed in angle brackets (<>). Each tag has a specific function and is tailored to the need of the developer by the use of the attributes. It is these tags which the browser reads and then displays, in a more user friendly format. For more information on HTML and the tags available, check out our developer guide.

For now though, let's have a look at what's in our template, you should find the following code in the template:

```
<div id="dvPersonalDetails">
  <div id="dvPersonalDetailsAlert" class="alert alert-warning personalDetailsAlert">Error Message</div>
  <h1>Please enter your details</h1>
  <p>In order to provide you with a quote, we need a few details. Please complete the form below. Thanks.</p>

  <div class="row">
    <form class="form-horizontal">

      <div class="form-group">
        <label for="nameInput" class="control-label col-md-2">Name</label>
        <div class="col-md-4">
          <input id="txtName" type="text" name="nameInput" class="form-control personalDetails" placeholder="Your Name" />
        </div>
      </div>

      <!-- The code for the remainder of the personal fields will be added here -->

      <div class="form-group">
        <div class="col-md-4">
          <input id="btnNext" value="Next" class="btn btn-default btn-sm btn-success" onclick="showCarDetails()" />
        </div>
      </div>

    </form>
  </div>
</div>
```

In our code, we use the <div></div> tags to specify sections or to group portions of the code together. All the code inside the outer div in the above picture is all one section on the page.

For our project there are 3 distinct sections:

- Personal Details
- Car Details
- Quote

Each section will be contained inside separate <div> tags. Throughout the code we will be using <div> to enable us to apply CSS styles using the class attribute and to make easily

identifiable sections in our code. There is also a header for the section and a description for it which will be displayed to the user (<h1> and <p>).

The first task that we need to do is to expand the HTML to add the remainder of the controls that we need the user to be able to see and edit. These are:

```
<div class="form-group">
  <label for="nameInput" class="control-label col-md-2">Name</label>
  <div class="col-md-4">
    <input id="txtName" type="text" name="nameInput" class="form-control personalDetails" placeholder="Your Name" />
  </div>
</div>
```

To add these elements to the page, we are going to use the same structure that already exists, as shown in the picture above. This means that we will have a label which will print the text between the tags, to the screen to tell the user what we want entered and the input tag, with the correct type attribute, which is what we use to create the textbox or other elements on the page. These are created inside the <div> tags as shown in order to apply the relevant bootstrap CSS styling to the page.

Creating the Details Screen

Exercise 1


Using the structure above, create the following elements on the page using the appropriate tags to create textboxes, dropdown menu's and radio buttons (for help/examples of these check out our self-help dev. guides).

- Name - Textbox
- Age – Number Textbox
- Gender – Radio Button (Male / Female)
- Town/City - Textbox
- Email Address – Email Textbox
- Years no claims bonus – Dropdown (Select,0,1,2,3,4,5+)
- Next – Button (validate, hide personal and show car details)

Once you get these added, your page should look something like:

The screenshot shows a web application interface for a 'Car Insurance App'. At the top, there is a navigation bar with four tabs: 'Personal Details' (active), 'Car Details', and 'Retrieve Quote'. Below the navigation bar, the main heading is 'Please enter your details'. A sub-heading reads: 'In order to provide you with a quote, we need a few details. Please complete the form below. Thanks.' The form contains the following elements:

- Name:** A text input field with placeholder text 'Your Name'.
- Age:** A number input field with placeholder text 'Your Age'.
- Gender:** Two radio buttons labeled 'Male' and 'Female'.
- Town/City:** A text input field with placeholder text 'Your Town/City'.
- Email Address:** A text input field with placeholder text 'Your Email Address'.
- No Claims Bonus(in years):** A dropdown menu with 'Select' as the current selection.
- Next:** A green button labeled 'Next'.

Remember to sync your changes with your GitHub account everytime you make a change. This can be done by opening up your GitHub desktop application and clicking the  icon in the top right hand corner.

Now that you have the basic view for the page created, we need to start making it do something.

If you open your updated version in Chrome, you will be able to enter details into the fields that you have created and which are now being displayed by the browser. However, when you click the button, nothing much will happen. That's because we haven't told the browser what we want to happen when the button is clicked. For this we are going to introduce **JavaScript**.

JavaScript + JQuery

JavaScript (JS) is another development language which developers use to control their front end applications.

It is different from HTML and CSS in that with JS you tell it what you want to do, whereas the others are languages to specify how things should look.

HTML and CSS are a list of instructions which tell the browser how the page should look and the browser just translates these to provide a static page.

JS allows us to make things happen: to hide or show parts of the page, to do calculations or to take data entered on the page and use it in some other way.

Changing from the Personal Details page to Car Details page

The first thing that we are going to use JS for is to change the users view from the *Personal Details* page, to the *Car Details* page. To do this, we will add code to the *showCarDetails* function which can be found in *navigation.js*. Open up *navigation.js* in sublime and you will see the following code:

```
function showCarDetails() {  
    // Hide the personal details section (dvPersonalDetails)  
    // Show the car details section (dvSectionDetails)  
}
```

A function in JS is a small section of code statements within the { }, which will run in order when the function is called.

In our case, the JS function name is *showCarDetails* which is called in the *details.HTML* page as an onclick event on the *btnNext* button. Currently, the only thing within our function is two comments to specify what the next tasks and comments are proceeded by the // symbols.

We are going to use a library called JQuery to help us find the page elements and then to make changes with them. An example of what a jQuery selector looks like is shown below:


```
$("#dvPersonalDetails")
```

What this will do is search the HTML to find the tags where the ID is *dvPersonalDetails* and will return it, and all information about that element to the JavaScript that calls it, including any

HTML inside of the tags. We know it is an ID as the identifier is preceded by a # symbol, the other main way is by using a fullstop (.) to match a class, but this can return multiple results.

Exercise 2

Check out the JS/jQuery self help guide to find a way to hide and show the two sections.

Remember to sync your changes with your GitHub account everytime you make a change. This can be done by opening up your GitHub desktop application and clicking the  icon in the top right hand corner.


Once you have added the code and tested it to make sure it works, you will notice that when you click the button the screen will go blank. This is because we do not have any HTML in the new section which is being displayed, so there is nothing to show. Also, you may have noticed that you can move ahead of the *personal details* section without adding any information. We need to validate this to ensure the user can't proceed without fully completing the form. Again, we are going to use jQuery to find the elements that you have added and then test to see if they have any data entered. Below is an example of how the name textbox would be validated. While other textboxes will be similar, the other input types such as radio buttons and dropdowns will be different.

```
if ($("#txtName").val() != "")
{
    // Hide Personal details section
    // Show Car Deatils section
}
else
{
    // Show personal details error message
}
```

This is the validation within an *if...else* statement. What this block of code is saying is that if the expression between `()` evaluates to true, then the computer will run the first code block `{ }`, if it evaluates to false, then it will run the code within the second code block `{ }`.

Exercise 3

Try writing the code needed to validate the first section using jQuery and JS in the *showCarDetails()* function. You will want to validate that each of the fields in the *personal details* section has a value entered, only revealing the *car details* section when then form is completed and hiding the *personal details* section.

Remember to sync your changes with your GitHub account everytime you make a change. This can be done by opening up your GitHub desktop application and clicking the  **Sync** icon in the top right hand corner.

Car Details Section

When you complete the validation for the first section you should be seeing a blank form. Now it is time to give you a bit more free rein with the project. The *car details* section now needs to be created in HTML in the same way that you have already created the *personal details* section.


The following fields need to be added:

- **Manufacturer** – Dropdown (Select, Audi, BMW, Citroen, Mini, Toyota, Volkswagen)
- **Model** - Textbox
- **Car Age** – Number Textbox
- **Engine Size** - Textbox
- **Storage** – Dropdown (Select, Garage, Driveway, Cul-de-sac, Public Road)
- **Estimated Value** - Number Textbox
- **Back** – Button (returns to personal details)
- **Next** – Button (validate, hide car and show quote)

When you are finished, the page should look something like the screenshot below:

Exercise 4

Using the personal details code as a template, add HTML and CSS within the `dvCarDetails` `<div>` tags to create the above elements in the `details.HTML` file.

Remember to sync your changes with your GitHub account everytime you make a change. This can be done by opening up your GitHub desktop application and clicking the  **Sync** icon in the top right hand corner.

More Validation

Now that the UI for this section is in place, we need to validate it and move to the final section of our form which will display the quote.

Exercise 5


Use the knowledge that you gained in the previous use of jQuery and JS to validate the elements of a section, to validate that all the *car details* have a value entered for them. You will also need to hide *dvCarDetails* and show *dvQuoteDetails*. Use the *showQuoteDetails()* function.

Exercise 6

Additionally you will need to reverse the hiding and showing of person details and car details sections for the back button. Do this using:

```
function showPersonalDetails() {  
    // Show the personal details section (dvPersonalDetails)  
    // Hide the car details section (dvSectionDetails)  
}
```

Remember to sync your changes with your GitHub account everytime you make a change. This

can be done by opening up your GitHub desktop application and clicking the  **Sync** icon in the top right hand corner.

Quote Details


We're ready to start the last UI section of the page. This is to display the quote which has been found by the user's details. For now, we just want to display the following:

- Title for the section
- Label to display "Quote: £"
- Label to display the API quote response value
- Back Button – returns to car details section

Exercise 7

Go ahead and create these elements in the details.HTML file and add the JS to make the screen go back to car details on the Back click event.

Remember to sync your changes with your GitHub account everytime you make a change. This

can be done by opening up your GitHub desktop application and clicking the  icon in the top right hand corner.

Creating the Back-End

Now we have a working UI, complete with page transitions between the sections and validation to check that all our fields have data in them, we are going to move into working with the back end. This is the part of our system which a user would not get a chance to see and is where the data that we have gathered in the front end is processed. We are going to use NodeJS to create an API which we can make single transaction calls to, using a specified URL, passing data, to get an expected response. We will make our calls using JSON and Ajax.

JSON

JSON stands for JavaScript Object Notation, which is the medium for which we will use to transfer our data between the front-end browser and the back-end server.

Variables within JavaScript are stored as an Object. An Object is a collection of information all stored in the same place, for example you could store yourself as a Person Object:

Person:

- Name
- Address
- Age
- Gender
- Etc.

To access a specific value within an Object, you must specify both the object that you want referenced and the value within it. For example, using our person object, if I wanted to find out the name of a particular person I would have to say *Person.Name*.

JSON builds upon these objects by specifying a default format to transfer data. Find the package, package.json, within the server folder of the template that you were given and open it in sublime.

```
{
  "name": "CarInsuranceAPI",
  "description": "API to return rating values for the car insurance application",
  "version": "1.0.0",
  "dependencies": {
    "express": "~4.0.0",
    "cors": "~2.4.1"
  }
}
```

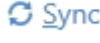
You should see something similar to the above. This is a piece of JSON which specifies details about the project. All of JSON objects are made up of text “*name : value*” pairs, separated by commas. A name can be anything that we want as long as it is a string of characters. A value must be either a text value (such as “Name”) or another object (seen by dependencies) which is then made up of further name - value pairs.

We are going to create our own JSON to pass to the server using the data that the user has entered into the UI. Firstly, we will need to get the values from the front end and store them as variables (just a place to store data dynamically as the programme runs). The next part of this challenge is to create the JSON using these variables, using the format that we have seen above.

Exercise 8

Follow these two steps to create your own JSON using the correct format and have it printed out to the screen so you can see its format.

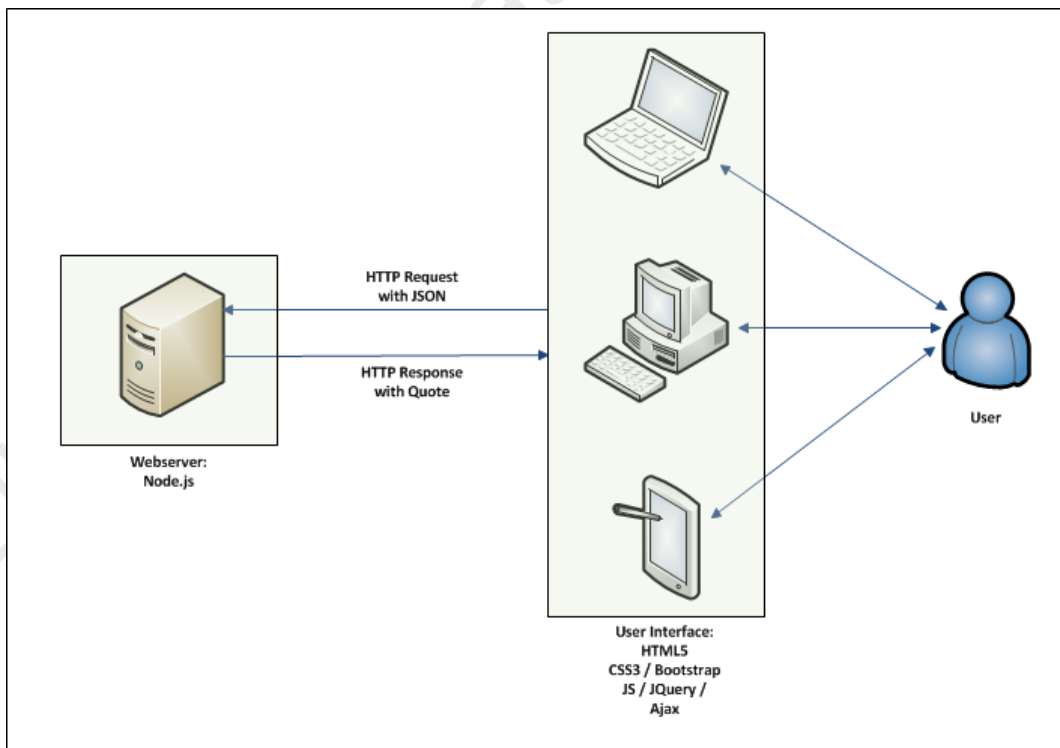
Remember to sync your changes with your GitHub account everytime you make a change. This

can be done by opening up your GitHub desktop application and clicking the  icon in the top right hand corner.

Ajax

Ajax is another component that can be found in JavaScript. It is a way to asynchronously (a fancy word for saying that it allows other tasks to complete while the Ajax request is being completed) send a request to a service or server, such as our API.

Ajax is the component that actually deals with making and receiving these requests. Let's take a look at our overall diagram from earlier on. You will see that there are Http Request and Responses between the Front and Back-end.



Ajax is the tool which generates the request, using the data that we give it and then waits for a response from the server and decides what to do based on the type of response it gets. It does this by taking a URL which is provided and sends a message to it, with the data provided. Let's take a look at an example:

```
$.ajax({
  type: "POST",
  url: "http://localhost:53753/api/functionToCall",
  data: { /* JSON data is added here */ }
}).done(function(msg) {
  // JS Actions to complete whenever a response is recieved
});
```

From the code snippet above, you can see that it works by passing a series of values to the Ajax function. In our example, it provides the type of Request to create the URL that the request should be sent too, the data which we want to send and the URL that the request should be sent to in JSON format.

Note: A request type can be of type GET/ POST/ PUT/ DELETE.

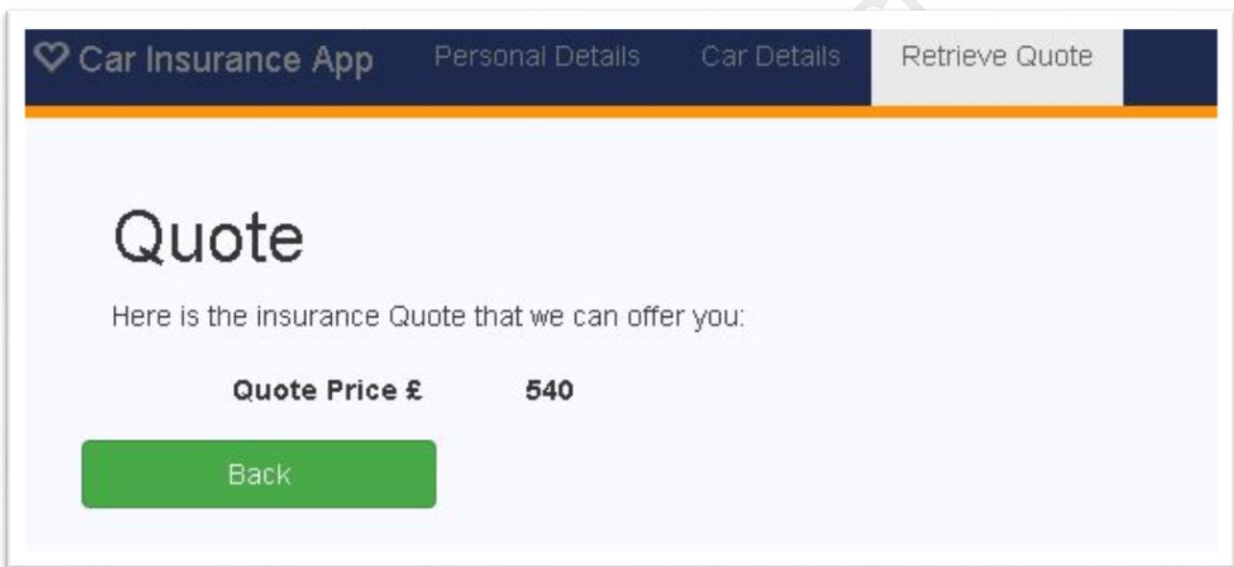
Exercise 9


Try now to create your own server request to the existing global server which will be called when you click the *Get Quote* button.

Remember the steps are:

- Get the user's input from the fields that you created
- Create the JSON using these key names:
 - gender
 - age

- noClaimsBonus
- costOfCar
- carStorage
- Create your Ajax call to send using the following URL:
 - <http://lit-wrkexp-01.lit.lmig.com:8080/api/calculateRates>
- When a response is received it will be in the following format { "result" : "value" }
 - Display this result to the label that you created earlier within the quote section (see screenshot below)



Remember to sync your changes with your GitHub account everytime you make a change. This can be done by opening up your GitHub desktop application and clicking the  icon in the top right hand corner.

Taking a look at the Server

The server is the final component of our overall system that we have not looked at yet. We choose to use NodeJS for the server to provide a simple lightweight way to make our API. Let's take a look at what is going on whenever we use Ajax to make a call and see what is happening inside it. Open up server.js in Sublime. The first part of the server is just setup. It's calling in all the references to other libraries that it needs to run, similar to HTML when we call in CSS or JS files.

```
var express    = require('express');
var cors       = require('cors');
var modifiers  = require('./modifiers');
var api        = express();           // Need to call to generate the API server

var port = 53753;
```

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

router.get('/', function(req, res) {
  res.json({message: 'Welcome to the Car Insurance API. Provide the correct URL and JSON to calculate the rates'}});
});

router.post('/calculateRates', function(req, res) {
  // This is the JSON object that we passed as a parameter. Values can be accessed by model.ValueName
  // i.e. model.age

  var model = req.query;

  // Setup basic values
  var basePrice = 500;

  // Call each of our built functions to get the specific modifier needed to make the calculation
  var totalCost = basePrice * modifiers.Personal(model.age, model.gender);

  // Return the calculated value as a JSON object
  res.json({ result: totalCost });
});
```

The main part of the server is shown above. This is where we declare our routes, in other words, where we define what we want to happen when an Ajax call is made to a particular URL (in our case <http://localhost:8080/api/calculateRates>).

In the template there are two calls already created. These are the base call for `/api` and `/api/calculateRates`, the first will deal with GET requests, and the second will work only for POST requests.

The function declared within the setup tells us what code is going to run when the router is hit with that request. This function takes in two parameters, the HTTP request (req) and the HTTP response (res).

In the `calculateRates` route, we can see that a variable called `model` is being created. This is where all of our inputs from the user will be stored. You can see them being accessed when the `modifiers.Personal` function is being called using the `model.age` syntax.

Finally it adds a JSON result to the response to return the calculated total cost.

Let's take a look at what is going on in the `modifiers.Personal` function. Open up `modifiers.js` in Sublime, you will see the following function in this file:

```
Personal: function (age, gender) {  
  //Set a default value for the modifier  
  var modifier = 1.08;  
  
  /* Applies following rules:  
  *   - If male and between 17 and 21 then rate is 1.5  
  *   - If male and between 21 and 25 then rate is 1.15  
  *   - If male and over 25 then rate is 1.12  
  *   - If female and under 25 then rate is 1.1  
  *   - If female and over 25 then rate is 1.08  
  */  
  
  return modifier;  
},
```

Here, the function takes in an age and a gender and returns a modifier which can be used in the total cost calculation within the server. Currently it just returns a default value, however the rules are specified to return a default value.

Exercise 10

Change the AJAX call to use your locally running server and use your knowledge of JS to use the values in age and gender to allow the sever modifier to be changed based on the values entered by the user.


Local Server:

- Url for local server: <http://localhost:8080/api/calculateRates>

note: To test this you will need to have your own local server running so that you can send it requests and get responses. To start the server follow the following instructions:

1. Open up cmd prompt (Click Start > type *cmd* > Hit enter)
2. Using the *cd* command, navigate to the directory where the server.js file is located
3. Install the NodeJS components (do this only once) by typing the following command
 - a. *npm install*
4. Run the node server using the following command
 - a. *Node server.js*
5. To stop the server after testing hit *Ctrl + C*

Remember to sync your changes with your GitHub account everytime you make a change. This

can be done by opening up your GitHub desktop application and clicking the  icon in the top right hand corner.


Once you have that coded, test it in your browser to see it working (**note:** you will need to stop and start your server via the *command prompt (cmd)* so that the changes you have made can be detected). You should see that the price of the quote will change depending on if it is a man or a woman making a quote and by their age.

Finally, there are three other functions in the modifiers class which need to be coded.

Exercise 11

Write the JS code to complete the modifiers class so that the correct value is returned depending on the inputs received. You will also need to edit server.js to call these new functions when calculating total cost.

The quote price should now change depending on the inputs provided.

Remember to sync your changes with your GitHub account everytime you make a change. This can be done by opening up your GitHub desktop application and clicking the  icon in the top right hand corner.

What's next?

Congratulations on finishing the challenge that we set for you. You've worked right the way through from the front-end to the back-end and got a basic web application working.

The great thing about your system is that it can be further extended. Try and extend your site further. Using the skills you have developed over the last few days, extend either the UI so that it looks nicer and has your personal touch or if you think you are up to the challenge try and make a more complex back-end which offers the customers more options when it comes to receiving a quote.

If you feel capable of creating your own routes, try and extend the API to provide more tools which can be used in the front-end. It's your chance to get creative and see what you are capable of developing.

Don't forget you have a good selection of resources to choose from, but more importantly you have others in the room, including a few professional developers so don't be afraid to ask for help!

Liberty Information Technology