


Contacts Application Design Document

In our design document, we omitted a lot of what we said we were going to add because to get the main functions working was the main priority and that took a fair bit of time. We split the name field into two – the first name and last name, and took out address and city fields since we felt that the post-code field was enough. We changed the sex, marital status and group's field to a drop-down box and radio heads. Originally, the group field were tick-boxes and was later changed to radio heads since we felt that a person being in multiple groups was unnecessary. To be able to upload pictures however required FTP access and knowledge of PHP and we had neither, however we left the “choose file” and “Submit” buttons in – even though they don't do anything – for aesthetic purposes,

The Contacts application changed a bit, this is what was originally submitted:

Add New Contact

First Name	<input type="text" value="Insert First name"/>
Last Name	<input type="text" value="Insert Last name"/>
Sex	<input type="radio"/> Male <input type="radio"/> Female <input checked="" type="radio"/> Other
Birthday	<input type="text" value="dd/mm/yyyy"/>
Marital Status	<input type="text" value="Single"/>
Personal Email	<input type="text" value="Insert a valid email address"/>
Work Email	<input type="text" value="Insert a valid email address"/>
Mobile Number	<input type="text" value="Insert phone number"/>
Work Number	<input type="text" value="Insert phone number"/>
Address	<input type="text" value="Insert home address"/>
City	<input type="text" value="Insert City"/>
Postcode	<input type="text" value="Insert postcode"/>
Organisation	<input type="text" value="Insert organisation"/>
Group	<input type="checkbox"/> Family <input type="checkbox"/> Friends <input type="checkbox"/> Work <input type="checkbox"/> Other
Contact Picture	<input type="button" value="Choose file"/> No file chosen <input type="button" value="Submit"/>
	<div>Notes</div> <div></div>

And this is what we have now:

Page 1

[Add a Contact](#) [Go to Map's page](#)

First Name	Last Name	Sex	Birthday	Marital Status	Email	Work Email	Mobile	Work	Postcode	Organisation	Group	Notes
------------	-----------	-----	----------	----------------	-------	------------	--------	------	----------	--------------	-------	-------

Page 2

Add New Contact

First Name

Last Name

Sex

☐ Male
 ☐ Female
 ☒ Other

Birthday

Marital Status

Personal Email

Work Email

Mobile Number

Work Number

Postcode

Organisation

Group

☐ Family
 ☐ Friends
 ☐ Work
 ☒ Other

Contact Picture

No file chosen

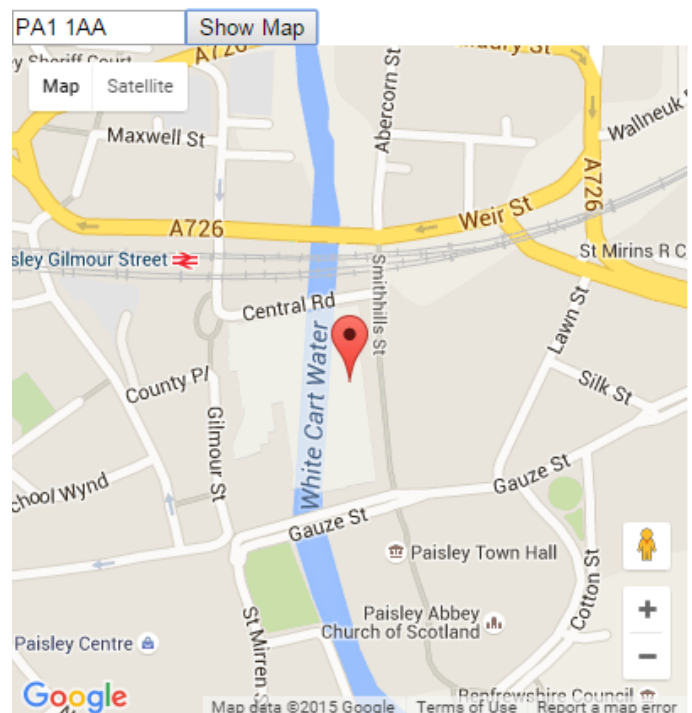
Notes

Page 3: Map

[Back to Contacts Page](#)

Please enter the post code in proper format.

e.g. PA1 1AA



While working together, we both learned from each other; we bounced ideas off each other, discussions and arguments ensued on the general layout, what looked good or not, problem-solved and worked really well as a team.

Pseudo code was written to see what functions were needed, and we came up with this list: saveContact, deleteContact, printContacts and exportContacts.

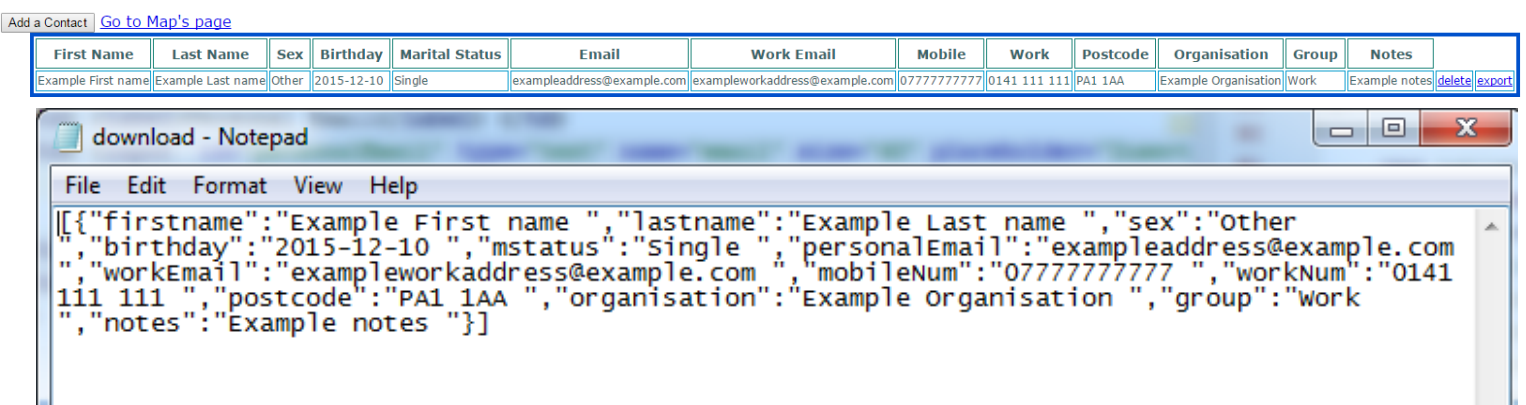
The coding was difficult, mostly being in jQuery and referencing a lot to websites like Stack Overflow. The function “Contact” was declared and variables were declared inside the function. To save form information by discussing it and remembering the labs done so far, we decided on local storage; however that required using “JSON.stringify” and “JSON.parse” and looking them up on Stack Overflow made it clear. We also referenced a lot to Labs 4 and 7, using new methods like push(), splice() and .bind(). We created two JavaScript files – Contacts.js handles the JavaScript/jQuery events while Models.js defines the variables and objects and handles them.

Firstly, we had to link the buttons in the contact form and after some searching, we found the `function.prototype.bind()` and figured out how to attach an event handler onto elements for example `#saveButton` is bound to `saveContact()`; we put that in `Contacts.js`.

In `Models.js`, we defined the local storage so it read `localStorage["contacts"]` and packed the local storage into an array by using `JSON.parse`. To do that however, we needed to read the text that was put into the fields so we created a function called `getContacts()` with a new variable `storedContacts[]`; and since it was an array, we used a for-loop with `push()` which made it possible to add all contacts into the array.

To save the contact form, we had to make a new function because `saveContact()` was to save the form, however the new function, `saveContacts()`, converted the local storage which was an array into a string. To then export the contacts into a text file, the variable `storedContacts()` was called again into a new function: `exportContacts()` and the two lines of code underneath was taken and modified from Stack Overflow.

When exporting the data onto a notepad, the download file popped up and exported like this:



We figured out where it was taken the data from, and it was from `saveContact()`. However when we tried to add a new line by using `"\n"` or `"\r\n"`, it would print them out like `*"firstname": "Example First name\n"`. After some debating, we decided on instead of putting `"\n"`, a space would be put at the end instead and so it would be spaced out more.

For `deleteContact()` we looked up on how to remove items from an array and found `splice()`; we then moved onto `printContacts()`. We made table variable and used `document.getElementById` to link `"contactTable"` from the HTML page and used a for-loop while linking each row appropriately. It is called when the user clicks the appropriate button and the html generated by the function `printContacts()`, includes the delete function on the `"onclick"` event with the associated contact id.

Although we had saved the contacts into the local server, we needed a function to actually show them thus we created the functions `showForm()` and `showList()`. `showForm()` was created to hide the list of contacts and show the form. `showList()` then hides the form, calls `printContacts()` which gets the list element and appends all contact to it by generating html and appending it to the table using JavaScript inserts (`insertrow`, `insertcell`).

Finally, we added the map page by taking the code from the "Simple GeoTag & Map application" in Moodle Demo's. Since we couldn't figure out how to put the input field and button where the post-code field would be or making it so the text inside the post-code field

would show up in the input field, in the end, we decided to link the 2 pages together – the page where our contacts printed and Map.html – by using ``.

As mentioned before, this code was taken from the Demo’s section in Moodle. When the original demo was tried, the post-code had to be entered twice before the map showed up. We thought that there was a bug in the code but after studying it, there was no avail in fixing it.