Web Report

James Edmeads (je16232), Josef Valvoda (jv16618)

Solar Builder: Overview

Our site allows the user to build and create a solar system of five planets which are then displayed in an interactive 3D model. The user can zoom into the planets or move the camera to change their viewing aspect. The website consists of three pages, the initial landing page, a page to choose each planets distance, speed, size and colour and the main solar viewing page.

We have chosen to use no frameworks throughout the development so we could take a low level approach to aid our own learning of all aspects of this course. This decision was taken as we feel that once we understood each aspect the frameworks, and any issues with them, would be much easier to learn and implement. We have used two libraries as described below which provide javascript sliders and the planets, both of which we have ammended.

Throughout development we have used firefox, safari and chrome to develop so that we could see any differences in the browsers and adjust these as we went. This also aided our own learning by having to understand these differences and what caused them.

HTML - A

We have created three main HTML pages for the site. XHTML delivery has been used for these to ensure the pages are correct.

We have used HTML5 tags throughout such as em and section rather than i or div. However, we still use div tags in some places where it makes sense to do so, i.e within a section on an element we simply want to attach style or javascipt to, the tagged line is already in a section, does not constitue a section in itself but requires an id or class. We use span for inline tags.

We use class where appropriate to style groups of similar objects and id tags for individual objects which allows us to attach style to whole groups and also individualise specific elements. We have used tags to collect relevant groups of sections together so that these can be moved together with styling. This grouping also allows us in the javascript to search via parent or child nodes which enables us to write one function for multiple objects, such as the modal content and sliders on the choice page.

All css and javascript is removed from the html files and the relevant css / js files referenced only in the header sections. On the server we check whether the browser accepts HTML or XHTML and deliver pages back in the correct style.

In terms if specific types used, we have implemented drop down menus, buttons, images in addition to standard tags such as h and p. We also make use of hidden forms on the page where we can collect data to send to the server. The send buttons are shown but the forms data is filled out by javascript as this allows us to use the javascript sliders on choice.html and send the values in a secure way to the server. The hidden element is also used on the first page to hide the buttons that are not triggered until the user has built a solar system allowing us to create dynamic pages, this will be discussed further below.

CSS - A

All of our styling is carried out without the use of a framework. All style is placed in the css files rather than the html file. We have added multiple stylesheets for the first two pages to allow for a responsive website which will change the position of elements on the page to allow for desktop, tablet and phone sized screens. This is particularly important for phone sized screens as all elements need to change position and size to look correct. The index page is a good example where we reduce the size of the main header, align the text and definitions one below the other and the buttons below this. This ensures the user can use the page in line with modern phone styling.

The elements on the page are reactive to the screen size with a mixture of percentages for positions and pixel placement used. The percentages allow the larger groups of elements to adjust to the screen size whereas the pixel placement is used for individual elements within these to maintain their ratios and not overlap. This means that scrolling is enabled on some pages so that the elements may go off screen and the user would scroll to find them. This is particularly important on phone sized screens as it ensures the elements remain are a visible size and do not overlap, and it is usual for scrolling to take place on phone sized web applications.

As described above class tags are used for groups of similar objects to attach styling to all elements, and the id of elements to change individual aspects such as position. This has ensured that repetition is kept to a minimum in the stylesheets, and that they are easy to read. This also ensures that groups can be adjusted on the screen together by using the class attribute and maintain their relative positions declared by the ids.

All font sizes use em so that the browser will display them correctly except the modal content which uses pixels as we wanted to specify how these appear. Also colors are declared in rgb format which also allows us to use transparency on some elements, the main one being the warning about local storage on the index page.

Styling is used briefly on the solar page to adjust the position of the canvas on which the javascript runs and so avoids having an edge to the screen, and the title which links back the main page uses a z index to ensure it stays on top.

We have styled the buttons on the choice page to signficantly change their appearance, with circular borders and transparent backgrounds. The noUiSliders are styled to create the clean, slim look that they have. Other buttons have been formatted to change their appearance. Modal content is triggered by user selection (captured in js) and the style is changed to block or hidden. Opaque style is used on the first page local storage info.

JavaScript: client side - A

We have started with an idea for a 3d solar system, where every planet would have properties defined by a user, and so we knew we will rely heavily on JavaScript to implement this 3d interactive simulation. The library we have chosen for this is Three.js, because it has multitude of examples and tutorials online and cross-browser compatibility. In our initial testing, it also performed very well displaying good enough animation on even a cheap notebook, and smooth animation on a bottom-line MacBook.

We have also found and used an extension for Three.js called threex.planets.js, which models individual planets in space. We had to master the Three.js library to understand how to write code to display multiple planets at the same time, have them rotate around the sun and allow the camera to move freely around them, zooming in and out, so that the user would be able to view the solar system from every angle. We learned about and adjusted the lighting to get single source of light, but since we decided to model sun in the centre as one of the planets, we have set the light source outside of our solar system. This would be one area where we would go further if the time would permit.

We have also made sure that we are generating our planets in a DRY way, looping instead of setting each planet individually. We have later adapted the threex.planets.js so that we could generate any planet with just one method call, instead of having each planet as an individual method.

We have also implemented NoUiSliders a library for creating javascript sliders on the page. These stop us being able to "use strict" on this while page as the way they are created does not comply with strict and without re-writing parts of the library this cannot be overcome. Instead in this js file "use strict" is declared for all other functions.

By setting up our HTML correctly we are able to use parent, child and next Element nodes to use one function for events on different objects on the page, such as the modal help content, ensuring no repetition in the code.

We also avoid using gloabal variables with the one exception being when they are needed to interact with the threex planets library.

We have checked all code for compatibility with older browsers and now that IE8 is 'dead' do not have any issues. This could have caused an issue with the XMLHttpRequest calls which we would have solved with an if/else statement and if it was not usable would have used a ActiveXObject instead. The other issue that old browsers may have had was with eventlisteners, to solve this we would have again used a conditional statement and an attachEvent() if it could not be used. The browsers that these would have affected though are now dead so we did not implement these.

All of our JavaScript code is written with "use strict" throughout to prevent bugs, global variables and immediate code. We never use vertical {} blocks inside a horizontal () function calls, we also use closures so we can create private fields without using 'this' notation.

PNG - A

We have tried out Gimp, but decided to use Photoshop and Procreate for the majority of the work.

We have used Photoshop to resize and blur the background images for our website and Procreate to design all the planets (With an exception of the real planets where the texture is from the threex.planets.js and the author and bubble planet which were designed in Illustrator.) We hand-drew each planet on an iPad in Procreate app, layering different brush strokes and brush widths on top of each other. Later we have imported them in Photoshop to adjust their size and format and exported them as .jpeg to be used as textures for the 3D spheres.

SVG - A

We tried out working with Inkscape, but as long time Adobe addicts we have switched to Illustrator later on. We have designed some of our planets (bubble and author) and our progress bar, indicating how many planets remain to be set up, in vector graphics.

First, we learned how to freehand draw curved lines, editing the anchors and their handles to get desired shape of a wave. We have then created a duplicate of this wave, transformed it by reflecting it along the x axis, and joined it on its end to create an object which could be filled with colour. We layered these objects on top of each other, selecting different colour for each to create the layered look of the bubble planet.

Later we drew our faces as paths, tracing a picture of us, edited the paths to fit the underlining photo better, grouped the ears to the face, adjusted the colours. We created polygons and filled them up with gradients, experimenting both with radial and linear gradients. We drew again wave patterns to create psychedelic clouds over our author planet, and filled them in with patterns. We placed a photo of motherboard as an underlining texture.

Both planets are demonstrate our ability to create vector graphics, we tried to engage with variety of illustrator tools and while the final planets do not fit in the overall theme of the solar system, they illustrate our ability to engage with path editing, transformations, gradients, patterns, freehand drawing etc. The majority of our planets were created in pixel graphics, because that style suited them better as per the section on PNG.

We have also realised that since we are dynamically taking data from the same looking website for five planets, we need some sort of progress indicator. We have therefore designed one, comprising of five .png exports of vector graphic created circles, where the large empty circle indicates the progress through the set-up of the solar system.

Server

We have built a https server on top of the provided server in node.js, not using the express. We have generated self-signed certificate using openSSH, to encrypt the communication between the browser and the server. We have also build another simple http server which only redirects the users from http to https. Our https server listens on port 443, while our http server listens on port 80, each therefore to the port where its respective communication would come through.

We have done URL validation, checking for all the edge cases: /, /,/, /../, // and rejecting any symbols which are not a valid ascii character. We also limit the length of our allowed URLs.

We communicate with the server by both sending the data via forms and at the end of the URL, we also use AJAX on our pages to check for example which user is creating a solar system by sending /check? followed by an ID which is check by the database. If the user already exists we let them edit or display their existing system, if not we generate a new user ID which is send back to be stored in local storage. This ID is later passed every time to the server via a hiden from to insert the planet data under the correct user. Ajax transactions also happen on our choice and solar pages.

Most time working on the server was spend on making sure all the edge scenarios with storing the data in the database and retrieving it back, would not break the server. We use call backs extensively to communicate with the database and have ensured we try and catch any errors and handle them either by redirecting to the home page, or displaying the solar system without planets if none were selected by the user.

We have limited the use of global variables, keeping only the necessary ones. For example we moved the keys, originaly kept globaly to a https call, as thats the only place where they are called.

Database- A

We use an SQLite database for the web site to record the current user and the planet parameters for each chosen planet. These are stored in two seperate tables - planet and system. The system id is a foreign key in the planet so that the user will only be able to access planets that they have created. The planet table has an auto-increment id column so that each planet has a unique id. The planet also has a number column which when combined with the system id provides a unique key. We can use the users id to access only their planets as they can only ever make 5 planets. This is controlled by the choice page.

We have controlled database access by having the database in a seperate server-side module which cannot be accessed by the user. We stop this by banning any url that tries to access the db folder, or combinations of .. or / that may allow user the access.

We form the userId's in the database module. The client side will store these locally using session storage, chosen as we want each new visit to the site to act as a new user. On arrival at the site the client side javascript will send any locally stored id to the server which will check against the database. If the database cannot find this user it will create a random key which is 16 characters long, this will be stored in the database and sent back to the user via the server for storage.

When the user selects each planet the planets variables and the user id is sent via the server to the database which will then insert these in the correct table.

When the solar page is loaded the client side will send the userid to the server which will get all the planets from the database that correspond to this user and send back to client side to build each planet.

All database queries and insertion are carried out within try / catch loops to ensure any issues are caught and dealt with by the server so that internal workings are not obvious to the user.

Database functions that are needed to be used by the server are written as module exports so that they can be called on the server side. Other functions for internal db.js use are not exported.

All function calls are completed via callbacks to ensure that items are not returned before the database can provide them.

We have taken the decision to clear the database on server start. We are aware that in a real world application this would lead to a loss of data and would remove this function but we have kept it in place for development purposes and to stop unnecessary data being collected.

Dynamic Pages - A

The index and solar pages on our site are dynamic. The index pages shows different buttons and the info bar about local storage dependent on data it recieves from the server. It sends the id stored in local storage. The server will then query the database to see if this user exists and whether they have chosen five planets. If so then the page will display the button options to go straight the their solar system or to ammend their planets. If the user has not been to the site before it will display the warning about local storage use. If they have been but not selected 5 planets it will not display the warning but only give them the choice to choose their planets.

The solar page is fully dynamic in that it request data from the server for the parameters for each planet and will build each planet object according to this data and display them on the page. Without the server data it will only display a sun. Everytime the user lands on this page it will request the data again to check for chages before displaying the new system

Conclusion

In conclusion we feel that we have made a fun and very interactive website that users will enjoy. We have learned a lot about all the aspects described above and explored different options of how to complete these and we hope you have fun playing with our website.