cuckoo

Online time tracking system.

Major Project Report

Interactive Multimedia Design 2014

Thomas Clarke (B00557420)

Peer Group: 2 Mentor: Peter Nicholl

Acknowledgements

I would like to express my gratitude to Dr Peter Nicholl, for his assistance and flexibility as a mentor during the completion of my Major Project. I would also like to acknowledge his continued support throughout university in his role as Course Director.

Many thanks to the other members of teaching staff who have assisted me along the way, notable members of which include Giuseppe Trombino, Dr Ian Young and Paul McCormack.

I am also indebted to Stephen Reid and Gareth Dickey, for the opportunities and experience gained on placement.

Contents

1. Introduction

One of the many challenges facing students, particularly those in further education, is the ability to manage their time effectively; to ensure that they allocate an appropriate and balanced amount of time to each areas of their study. This real world challenge was the inspiration behind the creation of the site which will be discussed during this report.

1.1 Project Aims & Objectives

1.1.1 Project Aim

Having identified a problem, a project aim was drawn up so that those involved with the project would have a clear idea of what the long term goal would be.

"To create an online time tracking system to aid students with time management."

1.1.2 Project Objectives

With an aim in mind, various objectives were identified. These would be short to medium term actions that would ultimately lead to the completion of the final solution.

Conduct background research into related areas

This included finding statistics about student diversity in terms of age, gender etc. These statistics can be seen in Appendix A-

Researched existing competitors and similar sites, to determine features and functionality, as seen in Appendix B-

Draw up a list of requirements gathered from research

Use the information gathered from research to create a list of features for the system.

Create UI wireframes that address requirements listed in the previous stage.

Use the requirements list to determine what features were required on the site, as well as the most logical and user friendly way to display and structure these features. This objective will be regarded complete when all the feature requirements were included in the designs.

This stage's was dependent on the previous two stages being completed and could therefore be delayed. Its duration was dependent on the amount of requirements identified during research.

Research technologies needed to achieve the front end functionality

Before the back end could be designed, available technologies that could achieve the requirements needed had to be researched and identified. This was a potential bottleneck, as research may have struggled to provide viable solutions which would have delayed following stages.

Design back end structure needed to achieve front end functionality

Once research had been conducted and suitable technologies identified, a back end architecture was designed. This took into account the following areas-

Database structure, in regards to - the entities/tables needed to store the data, the nature of relationship between these entities.
Individual table structures, including - Primary and Foreign key assignment, Field names, Field data types.

Develop front end interface needed to achieve the functionality included in the wireframes

Once the front end had been designed, the next objective would be to develop and implement the front end interface of the site. Although this objective could be started in isolation from developing the back end, it would eventually have to be integrated together. Therefore it could potentially cause a bottleneck and delay in development.

Develop the back end system to achieve the functionality identified after research

With the back end researched and designed, the next stage would be to develop it. Again this could be done in isolation from the front end development to a degree. However after a certain point both would have to be integrated together in order to carry out integration testing. There each of the units would have to be completed and fully functioning. Therefore a potential bottleneck to the project was identified.

The remainder of this report will document the work that was undertaken in order to produce the final solution. This will include the objectives listed a above, and will touch on areas such as design, development and testing. The report will close with an evaluation and conclusion to the project.

2. Concept Definition & Testing

2.1 Idea Generation

As mentioned in the introduction, the concept was inspired by the real world problem of time management faced by students in higher education. Therefore a solution to this problem was proposed in the form of an online time tracking site. It was felt that that such as system would help students focus more time on academic studies and less on time management. Although time management is an important part or university life, it is not directly assessed in assignments. Therefore it is important that students spend as much time as possible developing their knowledge and skills, as ultimately these are the qualities that will be assessed to determine the outcome of their degree.

2.2 Anticipated Solution

The anticipated solution would be a responsive web app that allows students to register an account and begin tracking progress on their current assignments. The system would provide an online timer, present progress using user friendly data visualisation and provide weekly email updates.

2.3 Contextualisation

Research showed that there is a vast number of time tracking sites aimed at freelancers and other professionals, however there are fewer sites aimed at students. This showed that there was a gap in the market which could be exploited by the proposed site. Appendix B- shows a comparison table of similar existing sites that were identified during research.

2.4 Requirements Specification

Once the problem had been identified and a concept had been proposed, the next stage was to develop the concept further by drawing up a list of requirements the final solution would have to fulfil. These requirements are documented below, using the structure of the Volere Requirements Specification template.

2.4.1 Stakeholders

Client - Although no actual client was involved, the final solution would be assessed and reviewed by various teaching staff. Therefore it could be said that the clients for the project were Peter Nicholl (Mentor), Paul McCormack (COM 559 Coordinator) and George Moore (COM 553 Coordinator). These clients received progress indicators at various stages of the project, most notable with the creation of documentation at pre defined review checkpoints. As project mentor, Peter Nicholl received weekly progress updates during peer support meetings. Unlike real world clients, they would have no direct input into decisions making, however they would offer guidance where possible.

Customer - A potential customer for the final solution is the university itself. If the system is deemed successful, there is no reason why they would not encourage their own students to use it. No additional effort was required to meet the needs of this customer, as they would share the needs of the user, also wanting a system that would improve students' time management.

2.4.1.1 Other Stakeholders

Below are some of the other stakeholders identified to have an interest in the project.

Designer/Developer – Thomas Clarke

The design and development of the project was carried out by an individual possessing both the technical and design based knowledge required. This removed the potential for miscommunication, which could occur had the designer and developer been separate stakeholders. However it also increase the workload for the stakeholder, which led to various issues throughout the project.

Testers – Fellow students

It was decided that fellow students would be able to use the system and give feedback on their experience. This gave valuable knowledge about the system and how it met the needs of its users, especially as the tester were part of the target audience..

Usability Experts – Paul McCormack, Tim Potter, Gabriel Muldoon

Various members of the university design staff were identified as being able to offer their expertise in user experience design throughout the project. This would come in the form of

overseeing workshops, giving feedback on proposed designs and consultation during scheduled classes.

University Lecturers

Lecturers from universities may be interested in the project, as their students could potentially end up using this system. Therefore it may be beneficial to gather their views of the final solution via interviews or surveys. This knowledge could help better refine the user requirements and functionality needed in my system.

2.4.2 Hands-On Users

2.4.2.1 Category

The only group of hands on users identified for the system were students. However this group is very diverse, and so it was decided to carry out background research to get a better understanding of those who might use the system. As seen in Appendix A-

2.4.2.2 User Role

The only responsibilities identified for the users were creating a user account, and using the system itself. This would include adding their modules, assignments, deadlines and tracking their time using the online interface.

2.4.2.3 Subject Matter Experience

Users would not require any additional subject matter experience, other than the knowledge of their own course, modules and assignments. Therefore it was unable to rate the user's experience in terms of novice, journeyman or master, as this depended on which year they were in and how well they engaged in the course.

Technological Experience

It was identified that students have a wide range of technological experiences. Some students will be tech savvy and use computer everyday whereas others may use computers infrequently. Therefore it would be important that the system was accessible and user friendly for users with a novice experience level.

2.4.2.4 Other Characteristics

Below are some of the other user characteristics that were taken into account when considering the requirements of the system.

- **Physical Ability** – As shown by the background research, students are an extremely diverse group, this includes a wide range of physical abilities. Therefore the system would have to accommodate users with the lowest degree of physical ability such as motor control and visual impairment.

- **Age** – Despite the common misconception, the age range of students is quite broad. This includes students straight out of secondary education as well as mature and returning students. According to the Higher Education Statistics Agency, 36.1% of students in 2011-12 were aged 20 years and under. 25.6% were aged between 21 and 24 years, 12.1% were between 25 and 29 years, and the remaining 26.3% were 30 years and older. Obviously there is a wide age range of students. Therefore the site must accessible and easy to use for all age ranges that may use it. See Appendix A- for more information.

- **Gender** – The Higher Education Statistics Agency documented that in the year 2011-12 just over 57% of UK students were female, with EU students being 53% female and Non-EU students being 48% female. This shows a fairly even mix of male and female students, therefore the system would have to be accessible and usable to both genders. This would include creating a gender neutral user interface design. See Appendix A- for more details.

- **Ethnicity** - Students can be of any ethnicity, therefore it was important that no group felt excluded due to the design or functionality of the system. However this would not extend into creating versions for different languages, as this would be a major undertaking.

Requirements which are measurable, testable and traceable are known as Atomic requirements when using the Volere template. For a list of functional and non functional atomic requirements see Appendix C-

2.5 Feasibility Testing

Before the project could move forward it was important to identify the feasibility of the system proposed, this required identifying potential risks and outlining remedial actions that could be taken should difficulties arise.

2.5.1 Technical Feasibility

To begin the project was assessed in terms of technical feasibility, including a review of the use of software and hardware, as well as front-end and back-end technologies.

2.5.1.1 Feasibility of Software

As shown in Table 2.1 of Appendix C-, the potential issues regarding software were considered and it was decided that he project was feasible from a software point of view. The software needed to design and develop the system existed and was readily available to the designer/developer. There were also contingency plans in place should the designer's/developer's personal copies of the software become inaccessible.

2.5.1.2 Feasibility of Hardware

The feasibility of the hardware was discussed and is documented in Table 2.2 of Appendix C-. The security and stability of the development files was a major risk, as hardware failures were a possibility. However this risk is faced by all development projects, and with counter measures in place, it was not considered significant enough to render the project unfeasible.

2.5.1.3 Feasibility of Front End Technologies

Having decided that the hardware and software proposed were viable options, focus then moved to the front end technologies that would be used. Ultimately it was decided that the front end development would not pose any major risks to the project, which is evidenced by the contents of Table 2.3 in Appendix C-.

2.5.1.4 Feasibility of Back End Technologies

The feasibility of the back end technologies were then assessed. Again it was decided that the avoidance methods and remedial actions proposed, shown in Table 2.4 of Appendix C-, were sufficient to allow the project to continue despite potential risks.

2.5.2 Financial Feasibility

Although the project had been identified as being technically feasible, it was important to ensure it was also financially feasible. At this stage of the project few financial risks were identified, as the technologies needed were accessible via personal copies or university facilities, therefore it was decided that the project was financially feasible. One of the potential risks identified at this stage was domain registration and hosting costs; however this did not come to fruition as the final solution was hosted on university servers. Another potential risk was the license costs for frameworks or plugins used during development. However the chosen jQuery plugins and PHP framework (Codeigniter) were free, open source technologies and therefore did not cause any financial issues.

2.5.3 Schedule Feasibility

The biggest constraint that was anticipated was time, as it would be managed alongside the designer/developer's other modules. Table 2.5 of Appendix C- discusses some of the time risks identified. Although the designer/developer had never undertaken a project of this scale, they drew on what past experience they had to inform the creation of a Gantt chart shown in Appendix H-. Unfortunately some unforeseen complications outside of the designer/developer control arose during the project which caused delays; these will be discussed further in the Evaluation section.

2.6 Methodology Selection

The methodology chosen to implement during the design and development of the final solution was that of Prototyping approach. It was this was good compromise between the structure of the waterfall approach, and the flexibility of the Rapid Application Development model. Below are some reason explaining the choice of methodology-

- The developer had brief past experience using it.
- It would provide the flexibility needed, as the features/functionality were likely to evolve during the course of the project.
- Working prototypes would give a better understanding of the system.
- Missing functionality and bugs could be identified during the review stage, allowing the developer to refine the requirements during the next iteration.

3. User Experience Design

This chapter will document the evolution of the visual design of the system.

3.1 Navigation

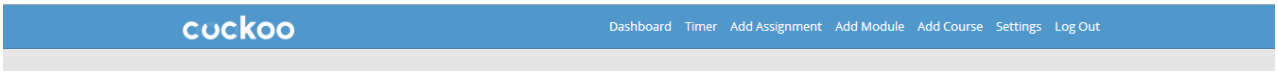
3.1.1 Primary

The primary navigation of the site will come in the form of a horizontal bar across the top of the page. As shown in the layouts in Appendix C-, there are two versions of the primary navigation; this is to accommodate the two persona main personas that will use the site.

Persona 1 - Visitor

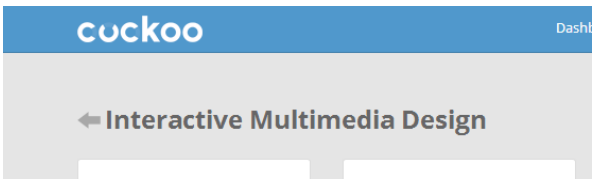
The first persona wishes to learn about the service and so the primary navigation must allow them to easily view the content that will aid their decision on whether to register for the service. This includes a link to the features list. If the content is successful a visitor may want to register, therefore a link to the registration page has been included. A login link has also been included for account holders who need to login. These action links have been positioned to the right of the screen, as this is where users have come to expect them. Reversing this design pattern would confuse users and impair the navigations usability.

Persona 2 – Account Holder



The second persona is an account holder who wishes to use the system and is likely logged in. As this persona has already registered an account the Features and Register links are no longer appropriate in this context and have been removed. The login link has been replaced with a logout button to notify the user that they are currently logged in. Lniks have been added to allow the user to start a timer, add new course/module/assignment and to edit their account settings.

3.1.2 Secondary

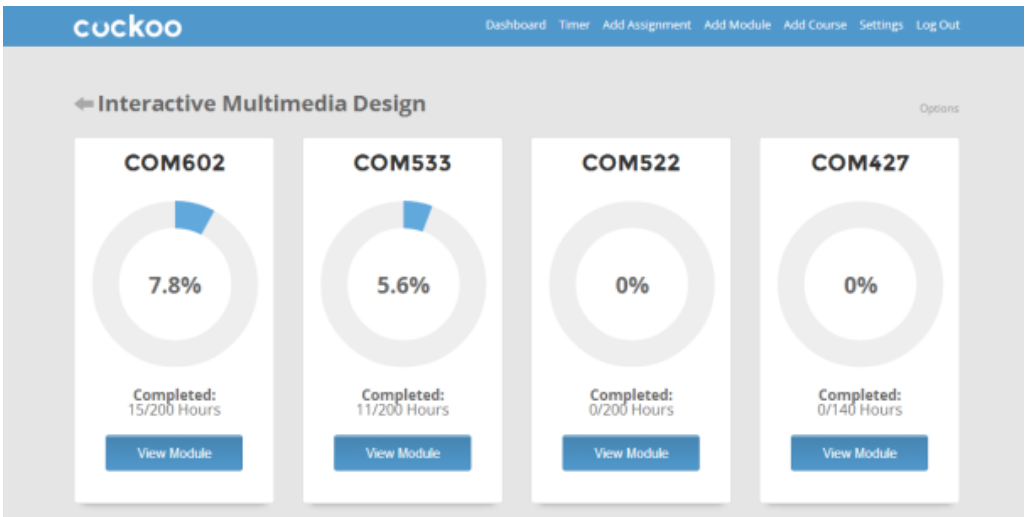


The main form of secondary navigation is breadcrumbs. This displays to the user where in the hierarchal structure of the site they currently are in relation to the Dashboard. For example when on an course page, the breadcrumbs will display a back button to Dashboard view. Although not strictly needed due to the site's shallow structure, breadcrumbs will offer the user a convenient visual representation of where there are and where they been. This allows them to quickly navigate to related higher level pages.

3.2 Page Layouts

It was decided to base the page layouts on a 960px 12 column grid, as this is a popular and standard approach in web design. It would also allow elements to be positioned and scaled proportionally to each other, which would help ensured the pages were balanced visually. This also allowed the different version for desktop, tablet and mobile to be designed easily using the 12 column grid as a reference. It was intended to use the responsive front end framework Foundation 5 by Zurb. This would allow the appropriate version of the site to be displayed to a device depending on it's screen width. This is achieved by in built media queries and css classes. Please refer to Appendix C- to see the evolution of the user interface from initial paper prototyping to updated sketches and mock-ups for various page desktop, tablet and mobile layouts.

3.2.1 Desktop

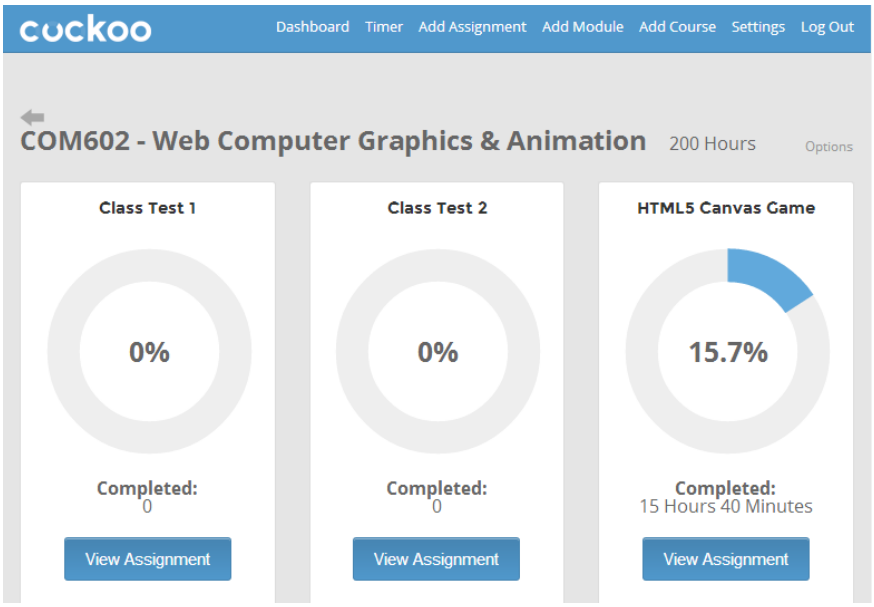


The desktop view of the site would display the tile elements on the dashboard view in four columns. I was decided that the desktop version of the site should offer all functionality available. As a desktop computer

is easier to work at due to the larger screen and access to peripherals such as keyboard and mouse, it was felt that adding and updating information would be best suited to the desktop. Therefore the ‘Add’ links are present in the navigation bar at the top of the page.

It was decided that the large table of work session shown on the assignment page was appropriate for the desktop version, as the user would have a large screen on which to read the information. This would ensure legibility and would be fairly user friendly on said large screen.

3.2.2 Tablet

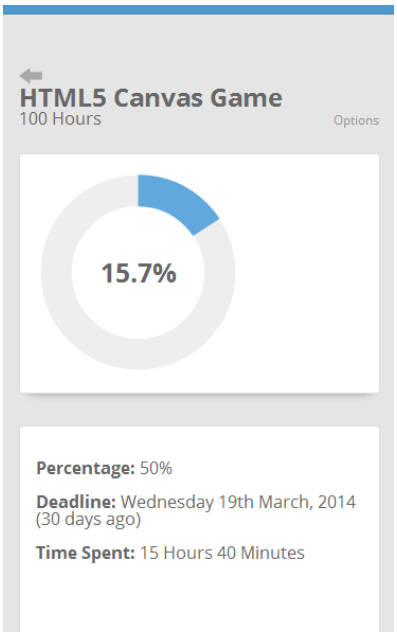


The tablet view of the site would be fairly similar to the desktop version, however it would display the tile elements on the dashboard view as three columns instead of four. It was decided that on smaller tablet screens the tiles should take up a larger percentage of the page, as this would make the text and donut charts larger in comparison. This would ensure the information was legible and easy to read for users. Making the information more accessible.

Again it was decided that most tablets offer a large enough screen to make adding and updating information using the various forms user friendly, without causing any issues realted to legibility. Therefore the ‘Add’ links are also supplied to the tablet view of the site.

It was also decided that the data table on assignment page was appropriate for the tablet view, as the screen would still be large enough to display the information in a legibale and easy to understand way. This would ensure that the user’s experience whilst reading the data in a tablet was user friendly and did not cause any issues in regards to the text not being legible.

3.2.3 Mobile



The mobile view would be tailored towards a different approach to functionality. It was decided that the mobile version of the site would be more focussed on tracking time, than adding courses, modules, assignments etc. Therefore the Add Course, Add Module, Add Assignment links would be removed from the navigation bar on mobile devices.

The features link on the homepage would also be removed on mobile devices, as it was non essential and would have wasted precious screen space otherwise.

It was decided that the image of the iPad and iPhone which appear on the tablet and desktop site was not necessary on the mobile site as it would be apparent that he site had a mobile version, as the user would be viewing it on a mobile device. Therefore it has been removed from the mobile version of the homepage.

As it is impractical and not user friendly to read large amounts of content on a small mobile screen, therefore it was decided that the jTable on the assignments page would not be displayed for mobile devices. It would not be a very user friendly experience reading such large amounts of tabular data on a small screen, and therefore it would only harm the site to provide such an experience to a user.

3.3 User Journeys

After deciding on page layouts, user journeys throughout the site were then taken into account. Below are breakdowns of the most frequent user journeys within the site. Please refer to Appendix C- for the actual user journey sketches.

3.3.1 Register & Login

This user journey relates to the visitor persona, and describes how a visitor would register an account from the homepage.

- The user clicks either the Register link in the top nav bar, or the TRY FOR FREE button to access the registration form.

- They enter their information into the form and click the submit button.
- If validation is unsuccessful the form is reloaded displaying validation errors. These will notify the user which input fields contained errors. This will ensure the user is never stuck not knowing what went wrong. Any fields which validated correctly will be repopulated with the inputted data, meaning the user does not have to re enter their details. This makes the form more user friendly and easier to use.
- If the registration from validates successfully the user is directed to the login form automatically. This seems appropriate as the logical next step after registration is logging in. This removes the need for the user to navigate the login page themselves, making the journey more user friendly. The email address they entered in the registration form is automatically passed into the email field on the login form. This prevents the user from having to retype their email, which speeds up the journey making it more user friendly.
- If the login form does not validate the user is presented with a validation error explaining what went wrong
- If the login form does validate, the user is logged in and directed to the Dashboard view.

3.3.2 Add Course

This user journey relates to the user persona, and shows the steps needed to add a course whilst on the dashboard view.

- The user clicks the Add Course button on top navigation bar, to access the Add Course form.
- They enter a course name and click the submit button.
- If the form does not validate correctly, the form is reloaded and the user is display an error message explaining what went wrong. This ensures the user is never confused about why the form did not submit.
- If the form does validate correctly, the course is added to the database and the user is redirected to the page of the course they just added. This was decided as best approach, as the course page would provide a link allowing the user to add a module to that course. This was seen as a logical progression through the site, allowing the user to add a course, then add a module, then add an assignment. This journey would lead the user step by step throught the site and would be easy to understand.

3.3.3 Add Module

This user journey relates to the user persona, and shows how a module would be added from the dashboard view

- The user clicks the Add Module link in the navigation bar, and directed to the Add Module form.
- They enter the appropriate information and click submit button.
- If the form does not validate, the user is displayed errors messages to ensure the understand what has happened. Their previously entered data is repopulated in the input fields to ensure they don't have to type it again.
- If the form does validate, the user is directed the module they just created. Again this seem a very logical way to step the user through the creation of courses, modules and assignments.

3.3.4 Add Assignment

This journey relates to the user persona and describes how they would add an assignment from the dashboard page.

- The user clicks the Add Assignment link, and is directed to the Add Assignment form.
- They enter the assignment information and click submit button.
- If the form validation fails, the user is presented with error messages explaining the cause of the problem. This helps the user figure out how to solve the problem.
- If the form validates, the assignment is added and they are directed to the appropriate assignment page.

3.3.5 Add Worksession/Track Time

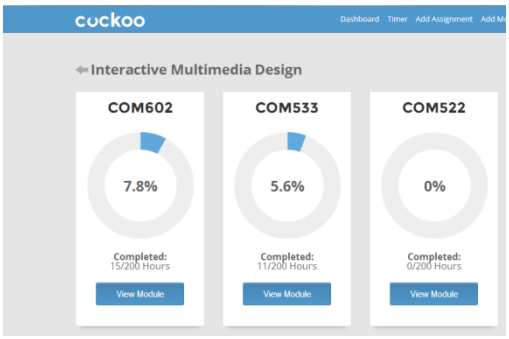
This user journey relates to the user persona and shows how they would add a work session from the dashboard view.

- The user clicks the Timer link in the navigation bar, and is directed to the timer page.
- The user selects an assignment from an auto populated select box. This is very user friendly as all their assignments are available to select. They then click the Start button.
- The timer starts and the interface changes to display the assignment name and the current elapsed time which updates in real time. Showing the current elapsed time to user makes the

timer accurate and therefore user friendly as the user is always aware of how long they have spent.

- If the user clicks the stop button a confirmation box appears. This ensures a user cannot accidently stop the timer, making the system more reliable and user friendly.
- If the user clicks cancel the confirmation box disappears to show the timer interface again. The elapsed time continues to update in realtime.
- If the user confirms by clicking ok, the timer stops and a textarea appears allowing the user to enter a description of the work carried out. The user can then click the Record Session button which refreshes the timer interface to the default appearance, allowing them to start another timer.

3.4 General Look & Feel



It was decided to use a minimal, clean, flat design as it would focus solely on the content over creative visuals. This stripped away any superfluous design features which would distract attention away from the content. The last several years have seen a rapid shift in website and application interface design to flat and minimal. This was also an influencing factor in the choice of the design, as it is important that the user associates the site on an equal footing as existing sites and potential competitors.

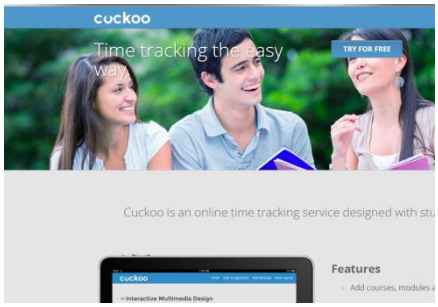
3.5 Brand Name & Logo

After previous research and experimentation, refer to Interim and Final UX reports, it was decided to use the approach of existing services such as Harvest and try an obscure word association for the brand name. After brain storming animals associated with time, the cuckoo bird was identified due to their loose associated with time keeping via cuckoo clocks. Appendix C- shows the initial sketches using the name Cuckoo.

A fairly minimal, clean design had been decided for the UI, therefore the logo had to compliment this. It was decided to stick with *Montserrat* as the font used in the brand logo/wordmark, as this would be consistent with the UI typography and strengthen

the brand. Several variations created during experimentation can be found in Appendix C-. The final wordmark is shown to the side, the beginning ‘cuc’ of the word were stylized to represent the donut charts used in the systems interface. This was achieved by duplicating the circular o from the end of the word and adjusting the transparency of certain sections. Incorporating the donut charts into the logo would create consistency and further strengthen the brand. It would also give a subtle suggestion of time and progress, as the charts also appear as highly simplify clock faces.

3.6 Imagery/Photography



It was decided that the public homepage would have a full width banner image. Research showed this as a popular choice among similar existing sites, something which may have been seen as a design trend and therefore avoided. However it was decided that it would be beneficial to associate the site with existing successful sites by using a similar design. The would reassure visitors that the site was comparable in quality and reliability to other sites using the approach. The use of blurred imagery also works very well with the proposed minimal, clean design.

3.7 Typography

3.7.1 Headings

Montserrat was chosen as a free alternative to Proxima Nova. Montserrat has the same legibility advantages as Proxima Nova, making it extremely easy to read and user friendly. Unfortunately it does not have the same range of variants as Proxima Nova, only having only Normal and Bold versions. However this was deemed sufficient for the needs of the project.

3.7.2 Body

Opens Sans was chosen to compliment *Montserrat* and continue the minimal approach.

Open Sans had a large number of variants, currently 5 weight variants including Light, Normal, Semi-Bold, Bold and Extra Bold. Each of which has an italic version, making 10 variants in total. This would allow a greater range of formatting and help with information hierarchy. The Open Sans family has been optimized for legibility, ensuring content is user friendly and easy to read.

3.7.3 Advantages of Hosted Fonts

Using Google Fonts would ensure consistency of the appearance and layout of the site, as users wouldn't need a local copy installed on their machine. This would strengthen brand recognition, and make the interface more user friendly as the consistency allows the user is able to earn and remember the layout of the site.

3.8 Colour

Blue was chosen as the main colour for the brand. Colour theory suggests that blue symbolises trust, loyalty, wisdom, intelligence, confidence and precision. These characteristics are important as the user has to have confidence in the site if they are to continue using it.

Blue is also associated with tranquility and calmness and has been known to benifit the human mind, by producing a calming effect. This is obviously a great charecteristic for the site to have as it will subconciously help to calm users who may be stressed out by Uni work. As students will log into the site in order to track their time, the colour choice may also help get users into a more appropriate state of mind before they commence their work. This will further increase their effeciency and productivity. Interstingly some believe that blue slows a viewer's motabolism and suppresses their appetite. Although this is not a deciding factor behind the choise, it would be a welcome side effect as it may allow users to work more productively on their Uni work without the need for frequent snack breaks.

Dark blue gives the impression of strength, power, professionalism and reliability; whilst light blue is more resfreshing and friendly, symbolising tranquility and understanding. Paler shades of blue are usually responsible for producing the calming effect on the viewer. Therefore there are benifits to incorporating any or all of these variations into the brand colours.

3.8.1 Colour Palette

Two main shades of blue were chosen, as shown below. #5098CC was chosen as the main brand colour and would emote a fresh, friendly personality. The darker #4685B2, was chosen as a secondary colour, it was used for subtle gradients used on buttons and for the footer on the homepage.

5098CC #4685B2

To compliment the blues chosen a secondary palette consisting of varying shades of grey was chosen. As a neutral colour grey works well alongside blue, when used on the white background needed for the minimal and clean design. These colours were mainly used for headings and body text.

#333333 #666666 #999999 #CCCCCC #FFFFFF

3.8.2 Colour Blindness

This was a potential concern regarding the colour choices of the branding. According to wearecolorblind.com around 8% to 10% of males have some form of colour blindness, with only 0.5% of females being colour blind. The most common forms of colour blindness are Deuteran (5.9%) and Protan (2.9%), fortunately these conditions are not affected too much by the chosen colour palette as they are more troubled by green and red respectively.

However the chosen colour scheme used blue and white throughout, which was a concern for those suffering from Tritan type colour blindness, as they have trouble distinguishing blue/yellow and yellow/white. Fortunately only 0.5% of those with colour blindness have the Tritan type. However given the target audience of the site is students, a wide demographic consisting of both genders and various ages, there is a possibility that some users may be affected. With that in mind it was important that the information displayed on the site did not rely solely on colour in order to be understood. Therefore the donut charts would be accompanied by a textual percentage.

3.9 Interaction

3.9.1 Time Pickers

When inputting times on the various forms of the site, users will be greeting by an HTML5 time input field. This causes the browser to serve a more appropriate input field which accepts a maximum of 4 digits separated in the middle be a colon. This makes it visually apparent to the user that a time is expected and makes completing the form more user friendly. Using HTML time input type also causes mobile iOS devices such as the iPhone and iPad to display the native time picker, this makes completing forms when using the device a lot more user friendly as the user can easily select a time using the process they are familiar with.

3.9.2 Date Pickers

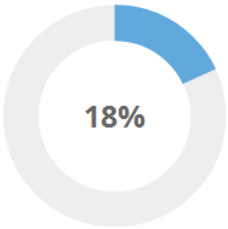
When inputting dates within the site, the use will be presented with a date picker calendar. This can be achieved with browser generated calendar in supporting browsers, or by using jQuery UI. Either way, the approach greatly simplifies the process of entering a date, as the user can now navigate to and select the appropriate date instead of having the manually type the date itself.

Using the HTML5 date input type also causes iOS devices to display the native date picker. Again this is beneficial as it allows the user to choose a date quickly and easily using a process they should be familiar with, as it is native to the device.

3.10 Information

The section below will discuss how information is presented throughout the site.

3.10.1 Donut Charts



Donut charts were used to visualize the total amount of time spent on a particular module or assignment, with the actual percentage displayed in the blank area in the centre. The circular shape of the chart subtlety implies a clock face, and ties in with the logo of the brand. The highly graphical approach also makes the information more accessible to users and extremely easy to understand compared to more textual approaches. It also takes into account the fact that users are impatient and insists on instant gratification; therefore an info graphic is much more instant in its delivery of information.

3.10.2 Data Tables

The assignment page presents a list of all work sessions tracked, and it was decided best to use a HTML table as the tabular data could be sorted logically into columns and rows. Using tables also improves accessibility of information for visually impaired users, as screen readers can easily interpret tabular data contained within a table.

As the user continues to use the system the amount of tracked work sessions would increase rapidly, and pagination was an essential feature. It was also important that the user could search or filter the table to locate the specific information they require. A jQuery plugin called jTable was used to provide this functionality. Allowing the user to filter the table gives them more control over how they interact with and view the information. This not only makes it more accessible but makes the user experience more personal. Zebra striping was used to colour alternate rows, allowing users to easily distinguish one row from another.

3.11 Tone & Voice

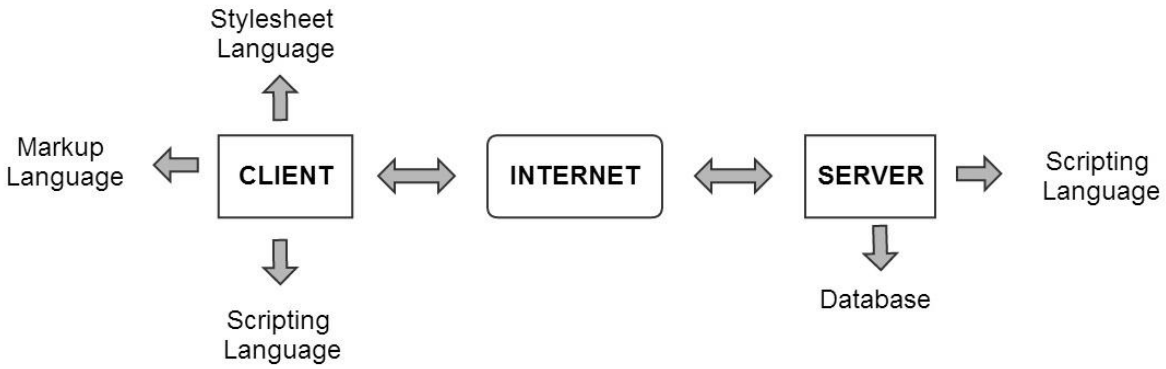
The tone and voice of the site had to reflect the values and characteristics wanted from the brand. As the target audience is diverse in terms of age and gender, the language used also have wide appeal so as not to alienate a particular group. Below are some of characteristics identified and a rationale for their choice-

Informative – as the site deals mainly with data visualisation it's important that the overall brand conveys a sense of being informative, so that users feel there is a benefit of using it.
Accurate – it was important that the brand promotes itself as accurate, as users would look for this trait in a time tracking service. Percentages were given to one decimal place to increase accuracy, and times where displayed to the exact hours, minutes and seconds where appropriate.
Informal/Friendly – It is important that users do not see the site as just another formal, corporate site associated with university. Instead they should see the site almost as a friend who will help them on a personal level, rather than a computer system which basically handles and displays data

4. System Design

The next stage of the project was to consider the various client and server side technologies that would be used to develop the site. Figure 4.1 shows a high level representation of how these technologies would relate/interact with each other.

Figure 4.1 – Representation of relationship between technologies



4.1 Client Side Technologies

4.1.1 Markup Language

HTML was the only viable option for the mark up language.. However it was decided that o specifically use HTML5. The new elements introduced in HTML5 will offers better semantics which will improve accessibility for impaired users, as well as improving search engine optimisation. Foundation by ZURB was chosen as a responsive front-end frame work for the project.

4.1.2 Style sheet Language

As HTML5 had been chosen as the markup language, CSS was the only real option of style sheet language. However there were several pre processors available which could have been used to extend the capabilities of CSS, a summary of these are found below-

SASS - Semantically Awesome Stylesheets is an open source language written in Ruby and released in 2007, a PHP version is also available. It enables the use of variables, nested rules and mixins (which allow blocks of code to be re used many times, much like calling a function). It is also the pre processor employed by Foundation.

LESS - Leaner CSS was influenced by SASS but adopted a more CSS like block formatting syntax. It also allows the use of variables, nested rules and mixins, but has the additional feature of

functions and operations. It was originally written in Ruby but has now uses JavaScript, giving it an advantage over other languages as it allows real-time compilation by the browser.

SCSS - Sassy CSS is a meta language of SASS, which was influenced by LESS and adopted it's more CSS like syntax. It offers the same capabilities as its parent language.

Stylus - Influenced by SASS and LESS, Stylus enables the use of variables, nested statements, mixins and in language functions. It also adopts a CSS like syntax structure, however this entirely optional as the code still works when characters such as braces and semi colons are omitted.

Chosen Technology

Although pre-processor languages are extremely powerful and have their advantages over basic CSS it was deemed unwise to implement one, as the developer had no experience using them and therefore setbacks could have been encountered. As the schedule was already rather tight, it was decided that basic CSS3 would be the best option, as the developer is experienced in its use.

4.1.3 Client Side Scripting

To achieve the dynamic client side interactions, form validation and asynchronous communication needed, the project would require JavaScript. Instead of low level JavaScript, it was decided that development could be simplified using a library. Below is a brief summary of the research conducted into JavaScript libraries and frameworks-

Dojo Toolkit - This is an open source modular JavaScript library, developed in 2004. Its features include widgets such as menus, tabs, sortable tables, dynamic charts and animated effects. It also enables both client and server side storage. However Dojo has been criticised for its poor documentation, steep learning curve for beginners compared to other libraries and the stability of the API.

Moo tools - My Object Orientated Tools is a lightweight, open source and as the name suggest object orientated JavaScript framework. It is a modular framework and allows developers to customize their own combinations of components depending on the project requirements.

JQuery - This is a free, open source JavaScript library which aims to it easier to navigate the DOM and select elements, handle events and create animations. It allows developers to create or install plugins on top of the library in order to increase functionality.

Chosen Technology

It was decided that jQuery would be the best framework to use, as the developer had past experience using it. It also has extensive documentation and a large support community which would aid with any issues arising during development. jQuery had also been noted as the fastest of the libraries available.

4.2 Server Side Technologies

4.2.1 Server Side Scripting

A large selection of server side languages were available to use including Perl, Cold Fusion, Java, Python and Ruby. However the developer had no experience of these languages and it was unadvisable to complicate the project using them. Due to the developers past experience using PHP, it was decided that this should be the scripting language used in development.
Having decided on a language, further research was carried out into various PHP frameworks that could be used to simplify development.

CakePHP - This is an open source framework for PHP which follows the Model View Controller design pattern, it was released in 2005 by polish programmer Michal Tatarynowicz.
Fusebox - This is a web application framework for PHP and ColdFusion, originally released in 1997. It does not require the user to use the MVC pattern or object orientated programming, but both can be used should the developer require these features.
Seagull - This is an object orientated PHP framework which uses the MVC pattern. The framework has an emphasis on modularity and allows the developer to add on functionality as required.
CodeIgnitor - This is an open source rapid development framework. It is loosely based on the MVC pattern, as models are optional in the framework. It is noted for it's speed compared to other frameworks with PHP creator Rasmus Lerdorf even stating he liked CodeIgnitor "because it is faster, lighter and the least like a framework."

Chosen Technology

Due to the developers past experience of the framework, it was decided that CodeIgnitor was the best solution for the server side scripting. Although the experience was quite brief, the developer was solely responsible for creating and maintaining a database management system using CodeIgnitor. This will offer valuable experience and reduce the learning curve needed to implement the framework into the project.

4.2.2 Server Side Storage

It was decided that a MySQL database using the InnoDB storage engine would be used as the server side storage method. This is due to the developer's knowledge and experience of this method. It was also an approach that was guaranteed to work on the university servers which would host the completed solution.

4.3 Architectural Pattern

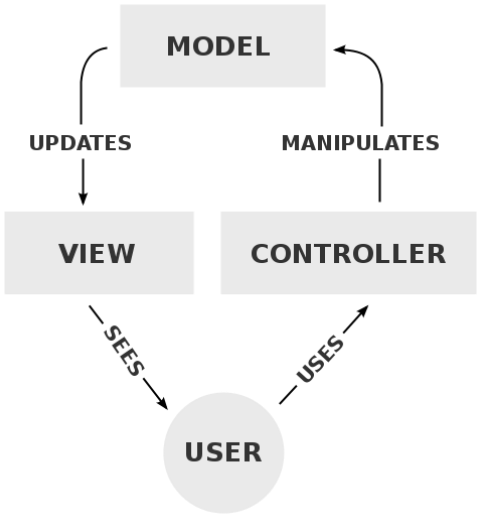
It was decided to the Model View Controller pattern when designing and developing the system architecture, as this approach would save time and effort in the long run. The MVC pattern focuses on separating data, logic and presentational code; a major advantage of this is the ability to easily maintain and reuse code throughout the project .

Controller - Sends commands to models and views to access and a display appropriate information.

Model – Quereies database and returns data.

View – Supplied content by controller, depending on what was returned by model.

Figure 4.2 – Overview of MVC architecture

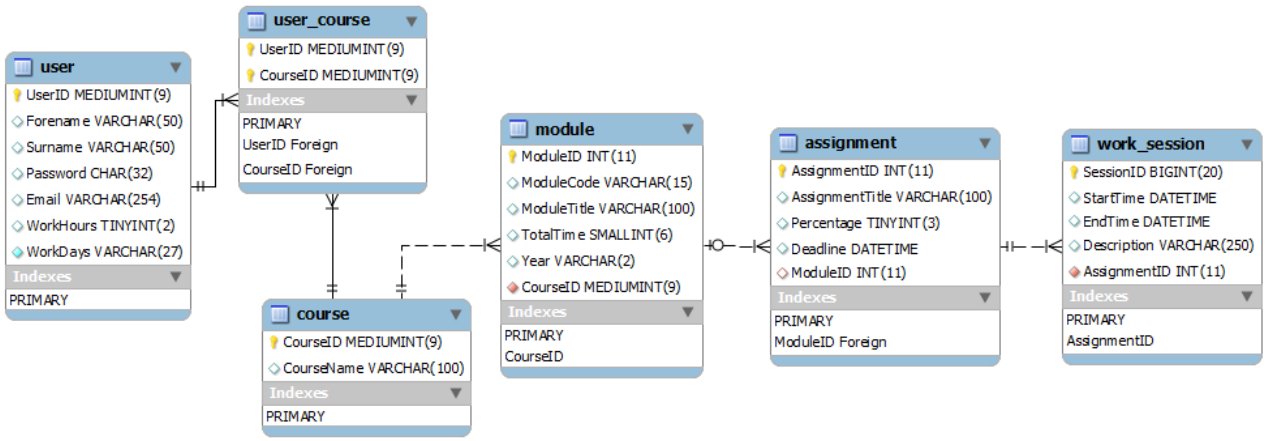


Examples of how the MVC pattern was used throughout the project can be found in the Implementation section.

4.4 Database Design

Having decided on the technologies and architecture the system would employ, the next stage was to design the database needed to facilitate the functionality of the site. Figure 4.2 shows the final database design including all entities and the relationships. The database was set up using the non-transactional innoDB engine. This approach is fast, requires lower disk space and requires less memory to perform updates.

Figure 4.2 – An enhanced Entity Relationship diagram for the site's database



4.4.1 Entity Relationship Modeling

At the beginning of the database design stage, ER modelling took place to identify the relationships needed in the database. This involved identifying the entities relating to the system, these were identified as users, courses, modules, assignments and work sessions. The real world relationships were identified, and these were used to determine the cardinality ratio between the entities and the actions required by such relationships. This ER modelling process is summarised in Table 4.1 below.

Table 4.1 Explanation of Relationships	Cardanality Ratio	Action Required
USER has many COURSE COURSE has many USER	The relationship between USER and COURSE is M:N	A link table (USER_COURSE) was created to handle the relationship between the USER table and COURSE table.
COURSE has many MODULE MODULE has one COURSE	The relationship between COURSE and MODULE is 1:M	Created CourseID as foreign key in MODULE table
MODULE has many ASSIGNMENT ASSIGNEMNT has one MODULE	The relationship between MODULE and ASSIGNMENT is 1:M	Created ModuleID as foreign key in ASSIGNMENT table.
ASSIGNMENT has many WORK_SESSION WORK_SESSION has one ASSIGNMENT	The relationship between ASSIGNMENT and WORK_SESSION is 1:M	Create foreign key in WORK_SESSION table.

4.4.2 Entity/Table Structures

After the relationships between the entities were identified and mapped, the next stage was to identify the characteristics or properties that needed to be stored in the database about each entity. Below is an in depth discussion of the individual tables within the database, including descriptions of the field's purpose, the chosen data type and a rationale for its choice.

4.4.2.1 User Table

The user table will contain all data relating to the users of the system.

Table 4.2 Column Name	Data Type	Description & Rationale
UserID	MEDIUMINT	This will store the unique user id number, which will be auto incremented. Using MEDIUMINT will allow for a maximum unsigned value of 16777215. This will allow for over 16.5 million users in the database, which is more than adequate for the system at this stage.
Forename Surname	VARCHAR(50) VARCHAR(50)	These fields will store the user's forename and surname. VARCHAR was chosen in favour of CHAR, due to the way data is stored. Both allow a maximum length to be entered, however VARCHAR uses less memory as only the characters being used are stored. Although some user may have double barrel names, it is extremely unlikely that these would exceed 50 characters.
Password	CHAR(32)	This will store an MD5 encrypted version of the user's password, as a 32 digit hexadecimal number. PHP has a built in function to generate the MD5 hash of the inputted string, which will be used when a user is registering or logging into the system. Therefore the data type CHAR with a max length of 32 characters was used.
Email	VARCHAR(254)	A maximum length of 254 characters has been decided for the email address field. According to a technical report by Dominic Sayers, a valid email address including the username, at symbol and domain cannot exceed 254 characters. Although it is unlikely for user to have email addresses of this length, the system must accommodate for all valid inputs possible.
Work Hours	TINYINT(2)	This will store the number of hours a day the user has specified they are able to work. As this will be used in calculations, an integer datatype will be needed. Although unlikely, the highest possible value for this field will be 24. Therefore TINYINT will be more than adequate as it can accept a maximum value of 255. The length of this field will be limited to 2 digits.
WorkDays	VARCHAR(27)	This will store a comma separated list of all the days of the week the user is willing to work. VARCHAR with a maximum length of 27 has been chosen as it has been decided to store the days as three letter shortened versions. Therefore the longest possible string would include 7 three letter strings and 6 commas, totalling 27 characters in total. The PHP functions implode and explode would be used within the model to access and insert this data. However at this stage it appears as though this functionality may be left out due to time restraints.

4.4.2.2 User_Course Table

The many-to-many relationship between users and courses required a link table to be created. Table 4.3 describes the chosen structure of the user_course database entity.

Table 4.3 Column Name	Data Type	Description & Rationale
UserID	MEDIUMINT	This is a foreign key linked to the User table, and makes up one half of the unique composite key. Therefore it must match the MEDIUMINT data type used in the user table.
CourseID	INT	This is a foreign key linked to the Course table, it makes up the other half of the unique composite key. It too must match the data type of the parent table, and therefore is set to INT.

4.4.2.3 Course Table

The course table will store information relating to the courses entered by the user. This would allow users to group their modules by course; allow them to track time on multiple courses using the same profile. Table 4.4 documents the chosen structure of the course entity.

Table 4.4 Column Name	Data Type	Description & Rationale
CourseID	INT	This will store the auto incremented unique course identification number. INT is required to accommodate the potential number of entries, using the smaller MEDIUMINT would only allow each user to add a single course. Although unlikely, using INT allows the scenario of each user adding 256 courses each.
CourseName	VARCHAR(100)	This will store the name of the course, and will be used in the user interface of the system to identify the course. Using VARCHAR with a max length of 100 characters should accommodate all possible course titles.

4.4.2.4 Module Table

The module table stores information relating to the individual modules being taken by the user, and its structure is shown in Table 4.4 below.

Table 4.4 Column Name	Data Type	Description & Rationale
ModuleID	INT	This will store a unique auto-incremented id number for the module. The data type of INT will allow for 4294967295 consecutive id numbers to be generated using auto increment. This should be more than adequate for the number of modules which the system should have to contain.
ModuleCode	VARCHAR(15)	This field will store the Module Code relating to the module. It is used simple for display purposes within the user interface, and not for uniquely identifying the module within the system. Using VARCHAR allows alphanumeric strings, which is ideal for the module codes as they are generally a combination of letter codes followed by numbers. It is difficult to identify the longest module code used by

		universities; however a max length of 15 characters should accommodate even the longest of codes.
ModuleTitle	VARCHAR(100)	This field will store the title of the module. It will also be used in the interface to identify the current module. Using VARCHAR with a max length of 100 characters should allow for all possible module titles and will allow both alphabetic and numeric characters to be used.
TotalTime	SMALLINT	This will contain the numeric value of the total number of hours to be spent on the Assignment. It was decided to use SMALLINT instead of TINYINT, as the latter only allows a maximum value of 255. It is possible that some assignments require more than this value and so SMALLINT was chosen.
Year	VARCHAR(1)	This will store which year of their course the module will be taken on. It will allow modules to be grouped together accordingly. As this value does not need to be used in calculations it has been decided to use VARCHAR with a max length of 1, instead of one of the integer data types. This will save space in the database.
CourseID	INT	This is a foreign key linking the module to a course within the COURSE table. Therefore its data type of MEDIUMINT matches that of the parent table.

4.4.2.5 Assignment Table

The assignment table contains all information relating to the individual assignments being undertaken by a user, the chosen structure is shown in Table 4.5.

Table 4.5 Column Name	Data Type	Description & Rationale
AssignmentID	INT	This will store the unique assignment id number, which will be auto incremented each time a new record is created. Using INT accommodates the unlikely event of each user adding an average of 256 assignments each.
AssignmentTitle	VARCHAR(100)	This will store the title of the assignment, and will be used in the interface as a display name for each assignment. VARCHAR with a max length of 100 characters should be more than sufficient for all potential inputs.
Percentage	TINYINT(3)	This will store what percentage of the parent module the assignment is worth. It will be used by the system in back end logic and calculations. As the maximum value will not exceed 100, a data type of TINYINT will accommodate any possible value.
Deadline	DATETIME	This will store the date and time of the deadline for the Assignment. Using DATETIME allows the day, month, year and time to be stored. This is essential for the functionality of the system. Using DATETIME also allows the data to be calculated and accessed using various PHP functions.
ModuleID	INT	This is a foreign key linking the Assignment record to a Module record. Therefore the data type matches the INT of the parent table.

4.4.2.6 Work Session Table

This table stores the individual work sessions tracked by a user each time they use the online timer, the chosen entity structure can be seen below in Table 4.6.

Table 4.6 Column Name	Data Type	Description & Rationale
SessionID	BIGINT	This will store the unique id number for each record. BIGINT is required to allow a user to record a suitable number of work sessions. Using INT would only allow for an average of 256 work sessions to be recorded by each user, whereas BIGINT allows well over 4000 million work sessions to be recorded. This is more than sufficient.
StartTime EndTime	DATETIME DATETIME	This will store the date and time that the Work Session was started and ended. Using DATETIME allows the day, month, year and time to be stored. This is essential for the functionality of the system. Using DATETIME also allows the data to calculated and access using various PHP functions.
Description	VARCHAR(250)	This is an optional field that will store a brief description of the work session. VARCHAR allows alphanumeric strings to be inserted, and a max length of 250 should be sufficient to describe the session.
AssignmentID	INT	This is a foreign key that will link the work session to the appropriate Assignment, as such the data type matches the parent table.

5. Implementation

During development there were many iterations of the project, for the simplicity sake these have been summarised into three main iterations accompanied by the notable achievements and challenges in each. Due to page limits code snippets have not been included, however each subheading lists appropriate source files. For further detail please see code comments.

5.1 Iteration 1 – Static Page Templates

The first iteration was to create static versions of the page layouts in HTML5. It was decided to use Foundation 5 by ZURB to implement the 12 column responsive design. Foundation has in built media queries and classes that allowed the developer to easily specify the number of columns an element should span depending on the current device width. The development made use of HTML5 tags such as header, footer and nav to ensure more semantic mark-up.

Datepicker – views/add_assignment_view.php, js/my_scripts.js

Although modern browsers can provide native datepicker for HTML5 date inputs , not all are user friendly. Therefore it was decided to use text inputs and supply a jQuery UI datepicker for all browsers. This would ensure a user friendly experience and consistency among all browsers. Inside my_scripts.js, the datepicker method was called and the dateFormat parameter was set to 'dd/mm/yy' to make the interface more user friendly.

Donut Charts – views/course_view.php, js/my_scripts.js

To create the donut chart visualisations, the plugin circliful.js was used. The main rationale was that each chart was created by assigning attributes to an html div element. These attributes would be used within the plugin to generate the appropriate Canvas drawing, and allowed a percentage to be set for the chart as well as a textual label. This plugin seemed most appropriate, as the dynamic versions of the charts were anticipated to be generated by a foreach loop. Therefore this div elements could easily be constructed using the array values. The plugin is called inside my_scripts on document ready, for all divs with class donut. To provide wider browser support for the HTML5 Canvas donut charts, a polyfill created by Paul Irish had to be included in the project.

Timeago.js – views/assignment_view.php, js/my_scripts.js, js/jquery.timeago.js

A plugin called timeago.js was used on the assignment view, to convert a supplied deadline date into a natural string describing the difference between the deadline and the current time. Within

the jquery.timeago script the allowFuture parameter was set to true, as this would allow the plugin to handle future dates.

Device Specific Visibility – views/assignment_view.php

It was decided that the mobile version of the site would prioritise tracking time over other functionality offered by the desktop site. This meant certain features were hidden; an example of this is the data table on the assignment view. This was achieved by adding the foundation class hide-for-small-only to the table container.

Fade In Tile divs

It was decided to fade in the tile elements on the various pages, as this would make the transition between views less jarring than the elements simply appearing. This would enhance the user experience and make the interface more aesthetically pleasing. This was achieved the jQuery code shown below.

```
// Fade in tile elements
var delay = 0;                          // Initialise delay variable as 0
$('.tile').each(function() {            // For each tile element
    $(this).hide();                     // Hide initially
    $(this).delay(delay).fadeIn(1000);  // Fade element in after delay
    delay += 250;                       // Increase delay by 500 milliseconds
});
```

To begin a variable called delay was set to zero. The jQuery each() function was then used to loop through each .tile element in turn. Each element was hidden initially, and then shown using the fadeIn() function with a duration of 1 second. A delay was added before each fadeIn(), which was incremented by a quarter of second each time. This causes the tiles to fade in sequentially on after the other.

Timer Form

Below will explain the how the functionality of the online timer was achieved.

```
$('#start').click(function(){

    $('#start, #timer_form #assignment_id').hide(); // Hide elements
    $('#stop').show();  // Show Stop button

    var startDateTime = getDateTime(); // Get constructed datetime string of user's current time
    $('#timer_form').append("<input type='hidden' name='start_time' value='" + startDateTime + "' />");

    var assignment_name = $("#assignment_id").find(":selected").text();
    $('#timer_top_div').prepend("<span id='timer_display'>00:00:00</span>");
    $('#timer_top_div').prepend("<span id='assignment_name'>" + assignment_name + "</span>");
```

When the user clicks the start button, jQuery is used to hide the selectbox in the timer form as well as the start button itself. Another button #stop is then made to appear.
A function getDateTime is run and the value return is assigned to a variable. This function use javascript date functions to construct a datetime string for the current time.
This datetime value is prepended into the timer form in a hidden input. A span is prepended to the timer form to act as the timer display on screen. The assignment name which was chosen inthe selectbox is then displayed inside a span element above the timer dispay.

```
var start = new Date().getTime();   // Get users current time

    timer = window.setInterval(function() // Timer function
    {
        var now = new Date().getTime();

        elapsed = Math.round( (now - start) / 1000 );

        var hours = Math.floor(elapsed / 3600);         // Calculate whole hours
        var minutes = Math.floor(elapsed % 3600 / 60);  // Calculate minutes past whole hours
        var seconds = Math.floor(elapsed % 3600 % 60);  // Calsulate seconds past whole minutes
```

A variable start is created and the current time is assigned to it using getTime(). A setInterval() is then set to run a function every second. This function begins by setting a variable called now as the current time using getTime(). The elapsed time is then calculated by comparing the start variable and the now variable. The value of the elapsed variable is the calculated into hours, minutes and seconds.

```
if(hours > 0 && minutes == 0 && seconds == 0)   // On the hour, every hour
{
   $("#sound").html("<embed src='<?php print base_url(); ?>cuckoo.mp3' hidden=true autostart=true loop=false>");
}
```

It was decided that a cuckoo sound would be played each hour to give the user an indication of how long they had been working. This was achieved using a conditional statement that ran when $hours was greater than 1, $minutes equalled zero and $seconds equalled zero. jQuery was then used to embed an mp3 file in the document that would autostart.

```
/* Prepare values for display */
if(hours.toString().length == 1){ hours = "0" + hours; }
if(minutes.toString().length == 1){ minutes = "0" + minutes; }
if(seconds.toString().length == 1){ seconds = "0" + seconds; }

elapsed = hours + ":" + minutes + ":" + seconds;

$('#timer_display').html(elapsed);
$(document).attr('title', 'Cuckoo | ' + elapsed);
```

The hours, minutes and seconds variables are then prepared for display by adding leading to them if required. The variables are then concatenated with a set of colons to create display in the HH:MM:SS format. This string is then displayed inside the #timer_display element, as well as in the page title which is display on the browser tab.

```
$('#stop').click(function(){

    var check = confirm("Are you sure you want to stop the current session?"); // Return result of confirm dialogue box

    if(check == true)   // If user confirmed stopping the timer
    {
    $('#timer_display').html(elapsed);                  // Update timer display
    $(document).attr('title', 'Cuckoo | ' + elapsed);   // Update page title
    clearInterval(timer);                               // Stop timer
    timer = false;                                      // Set timer to false, used in conditional check
```

When the #stop button is clicked a confirmation box is displayed checking if the user wants to stop the timer. If the user agrees the confirm will return true. This causes an if statement to run which updates the page title and timer_display element one more time to ensure to displays accurately to the second. The timer setInterval is then cleared, stopping the timer, and the variable timer is set to false. This well be used in conditional logic later.

If the user cancels the confirmation box it simply disappears to display the timer again.

```
$(window).on('beforeunload', function(e){ // Triggered when a user navigates away from the page

    if(timer) // If timer has a value, if timer is running
    {
        e.preventDefault();
      return "Warning! The current work session details will be lost if you leave the page.";
    }

});
```

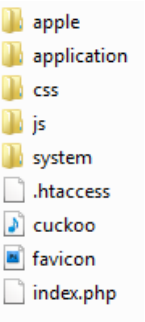
The on('beforeunload') event was used to display a warning message to the user. This event would fire when the browser tab or window attempted to close. An if statement checked if the timer was running and if so preventDefault() was used to prevent the tab from closing, a message then warns the user that closing the window will cause the current work session to be lost. This would prevent a user from accidentally closing their browser and losing data.

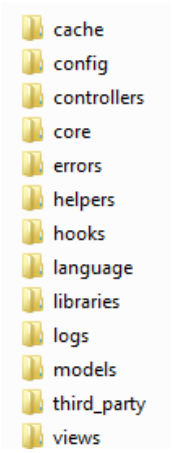
This feature will be explained with better clarity during the demonstration.

5.2 Iteration 2 - CodeIgniter Integration

5.2.1 General Information

The second iteration involved integrating the static pages into the PHP framework CodeIgniter. This involved downloading a copy of the framework from ellislab.com. The root file structure includes folders for core system files, such as helpers and libraries; as well as a folder for all custom application code specific to the project. The application folder contains subfolders for the controllers, models and views, as well as configuration files.





The first step was to customize several files inside the config subfolder. An appropriate site_url and base url was set inside the config.php, as well as editing the database.php to match the database configuration details. These config file values could be referred to through the project using functions site_url() and base_url(). This would allow paths for stylesheets, scripts and other files to be generated easily. It would also allow links to be updated easily when moving the site.

It was decided to create several controllers for the various aspects of the site, as this would make development and maintenance easier. The **visitor** controller controls interactions with the public homepage, the **login** controller handles login and sessions, the **site** controller handles interactions made by logged in users and the **email** controller handles email functionality.

It was also decided to separate the model files logically depending on what functionality they related to. As such the models are as follows- user, course, module, assignment, work and email.

CodeIgniter recognises the controller and model files as classes that can be accessed throughout the project. It takes an object orientated approach to loading and using these classes, and this approach was adopted by the developer when interacting with CodeIgniter's core functionality. However the developer also used a procedural approach when writing logic and calculations within controller and model functions, this was due to the majority of the developers past experience using a procedural approach.

Passing Data Between Controllers and Views

CodeIgniter allows data to be passed from controllers to views as a multidimensional array called $data. Arrays are added to $data using a textual string as the key for the entry. For example

$data['example'] = array(...) would add the new array into the $data array. The $data array is then passed into a view, and the individual arrays inside can be accessed by using their index/key name as a variable. For example using $example inside would access the contents of that array.

CodeIgniter also allows parameters to be passed to controller functions from a view using URI segments and the GET protocol. An earlier iteration of the system used this approach to pass id values to the controller; however it was realised that this would allow a user to simply edit the URL in order to access other users data. Therefore it was decided to use form submit buttons instead of links, as this would allow parameters to be passed using POST. This involved storing unique id values in hidden input fields, which was more secure as parameters were not visible to users.

CodeIgniter Helpers & Libraries

During the project several of CodeIgniters native helpers and libraries were used to speed up and simplify development. This included using the form helper to create forms and input fields, sa well as using the form validation library to validate forms and return error messages.

Header & Footer Templates – views/header.php, views/footer.php

The MVC approach allowed the header and footer sections of the page layouts to be created as template files. These views were then loaded by CodeIgniter, with page content view nested between, each time a controller needed to display a page. This prevented duplicate code having to be written and maintained.

As mentioned, two versions of the top navigation bar would be used. This was accomplished inside the header template itself using conditional logic. The CodeIgniter function fetch_class() was used to get the name of the current controller, if the visitor controller was in use the public navigation was shown, if the site controller was in use the 'logged in' navigation was shown. Although the MVC framework encourages logic is kept out of the view, it was justifiable in this scenario as the logic itself is minimal and the alternative would require separate views to be created for each of the navigations. This would complicate maintenance, as the designer would have three view files to edit if changes were needed in the header. The alternative PHP syntax was used for the if statement, as this allows HTML content to be nested inside. This would allow future designers to maintain the html without need of PHP knowledge.

Custom Helper Functions – helpers/my_functions_helper.php, config/autoload.php
During development it was realised that multiple models had to convert seconds into hours and minutes for display purposes. Therefore it was decided to create a custom function to be reused. A custom CodeIgniter helper file was created, inside which the function was placed. Basically the function accepted a value of seconds, calculated the hours, minutes and seconds, and output an appropriate natural language string. For more detail please refer the comments inside my_functions_helper.php. The autoload.php file was edited to automatically load the my_functions helper, making it available to all models.

The following subsections will describe notable aspects of the system functionality and how they were achieved using the MVC approach. This should clarify how the controllers, models and views interact with each other in order to manipulate and pass data throughout the site.

5.2.2 Registration Form

This subsection will describe the process of submitting the registration form, as such the controller function described is the one called when the registration form is submitted.

Controller- visitor/registration()
The controller function loads CodeIgniter's form validation library. It then uses the set_rules() function to set valiadation rules for each of the input fields in the form. These rules check that all fields have been filled in and enforces maxlength limits.

A notable aspect of the validation occurs on the email input field. The first of which is the use of CodeIgnitor's valid_email rule, which checks for an appropriate format of characters and the inclusion of an @ symbol. The second notable aspect is the use of a custom callback function called callback_unique_email. This allowed a custom function to be created to check that the email entered was not already in use by another user.

This custom function accepted the email address entered as a parameter, an automatic feature of being a CodeIgniter callback function. It calls the check_email() function in the user_model passing the email value in, this would return an array containing values if a match for the email was found in the database. If the array was empty, and no matches were found, the unique_email callback function would return TRUE. If not the function would return FALSE.

A custom error message was also created for the unique_email rule, that would notify users that the email was already in use.

A third notable aspect of validation was the use of the CodeIgniter's matches rule, this ensured that the second password field matched the first password field.

The controller then checks if the form validated correctly using form_validation->run(). If it doesn't validate the registration page is loaded again which will display the error messages to the user.

If the form does validate, the controller then begins the process of creating the account. It begins by creating an array $data containing all the data POSTed from the form, using CodeIgniter's input->post() function. By default the values for WorkHours are set to 10 and the WorkDays is set to all week days. The controller then calls the create_user() user_model function, and passes in the $data array. It then loads the views needed to display the Login form.

As the email entered by the user is still present in the POST data, this is displayed on the login form. This enhances the user experience, as the user doesn't have to retype their email again to login.

Model- user-model/check_email(), user_model/create_user()
The check_email() function accepts the $email value passed from the callback function. It then uses this value in an SQL query to select any user records that have a matching email address. The result is returned as an array to the unique_email callback function.

The create_user() function accepts the $data array passed in from the controller function, and performs a database insert using CodeIgniter's form helper insert() function. This accepts two parameters, the first is the table name and the second is an array of data to insert.

View- views/registration_view.php
The registration view displays registration form to the user. This is achieved using CodeIgniter's form helper to open a form which directs to the registration() controller function. It also uses the form_input() function to create HTML input fields. An array is supplied to the function to specifiy the attributes for the input field. This includes the use of CodeIgniter's set_value() function, which prints the previous value of the input field stated. This is used when the form validation fails and the user is redirected back to the form, it repopulates the fields that validated correctly with the previous value. This improves user experience as they do not have to re type their details. If a previous value was not entered, the field simply appear blank.

Although CodeIgniter provides a form_label() function for creating labels, it was decided to hard code these in PHP as this would allow the input fields to be wrapped inside the label. This would make the space between label text and input field clickable, making the site more accessible and easier to use specifically on mobile devices.

CodeIgniter's validation_errors() function is used to print out the error messages returned by the controller. This notifies the user of their mistakes, ensuring they know where they went wrong.

5.2.3 Login Form

This subsection will discuss the process of submitting the login form.

Controller – login/index()

The controller loads the form_validation library and sets validation rules. Again a custom callback function is used, called email_exists. This passes the inputted email to the check_email() model function, which queries the database, if the email exists in the database the callback function will return TRUE.

Another callback function rule called password_match() is applied to the password field. This takes the inputted password and combines it with the CodeIgniter encryption key, before hashing the result with MD5. This is then checked against the hashed password in the database. If they match the callback function returns TRUE.

If the form validation is not successful, the controller sends the user back to the login view with validation errors displayed.

If the form validates successfully, the controller calls the get_user_details() function in the user_model, passing in the $email. This returns the appropriate user details from the database. These are then stored in a SESSION variable and the user is redirected to their dashboard view.

Model – user_model/check_password(), user_model/check_email()

The check_password() function accepts the salted and hashed password passed from the callback function, and queries the database to see if it matches the existing password. It returns an array containing the record that matched, or an empty array if no matches were found.

The check_email() function is the same one mentioned previously in Registration Form subsection, please refer above for an explanation of the code. Reusing this function made development faster, and would simplify future maintenance of the system.

View- views/login.php

The login view displays the login form to the user. It uses CodeIgniter's form helper to create the form and input fields. Using the set_value() function allows the email entered by the user in the registration form to be PSOTed into the field. This means the user does not have to retype their email, making the experience more users friendly. The validation_errrors() function is used to display validation error messages to the user.

5.2.4 Dashboard Page

The dashboard is the default view when a user logs in. It displays to the user a list of all the courses they have added to their account.

Controller – site/dashboard()

The controller function begins by calling the get_user_courses() function in the course_model passing in the user_id stored in the SESSION. This returns an array called courses containing all user's courses, which is added to CodeIgniter's $data array. An if statement then uses the count() function to check if the number of courses in the courses array equals one.

If only one course is found there is no need to display the dashboard view, as it would be more user friendly to automatically display the course view for that particular course. The controller gets CourseID value from the courses array and assigns it to a POST data variable, as this is what the course() function expects. It then calls the course() function.

If the course array contained multiple values, the controller simply loads the header, dashboard_view and footer views; it also passes in the $data array which will be used in the view to display the appropriate content.

Models – course_model/get_user_courses()

The get_user_courses function accepts the user_id passed from the controller and constructs and SQL query to select the CourseID and CourseName for all courses related to the user. It returns these results as an array to the dashboard() controller function.

Views – views/dashboard_view.php

The dashboard view file checks if the $courses array passed in by the controller is empty. If so, the user has not added any courses. Therefore it prints out a tile element notifying the user they have

not entered any courses, and provided a link to the Add Course form using CodeIgniter's site_url() function to construct the appropriate path.

If the $courses array is not empty, the user has added at one course. Therefore a foreach loop is used to print out a tile element containing the course name as a form button. This form submits to the course() function in the site controller and POST the appropriate course_id via a hidden input field.

5.2.5 Course Page

The course page displays to the user a list of tile elements containing the modules related to the current course, with information about their progress for each module.

Controller- site/course()

The course() function creates a variable $course_id to store the value POSTed to it (either from a form on the dashboard page or from the dashboard() function, as mentioned above). It then calls the get_course_data() model function, passing in the course_id. This returns an array of information for that course. It then calls the get_course_modules() model function, again passing in the course_id. This returns an multidimensional array of all the modules for that course, as well as information about the progress of the related assignments. These arrays are added to CodeIgniter's $data array, and passed in when loading the appropriate views.

Models – course_model/get_course_data(), module_model/get_course_modules()

Unfortunately page limitations do not permit this aspect of the system to be discussed as it is quite complex and involves the creation of a multidimensional array of data being created using nested foreach loops and database queries. However it is anticipated that this aspect will be explained in full during the demonstration of the site.

Views – views/course_view.php

The empty() function is used to check if the $course_modules array is empty. If so a tile element is printed to the screen notifying the user that they haven't added any modules to the course, and supplies a link to the Add Module form using site_url() to construct the href.

If the $course_modules array contains values, a foreach loop prints out a tile elements for each of the modules. The module code and title are printed out by accessing the ModuleCode and ModuleTitle inside the $course_modules array.

A nested if statement then checks if the module has any assignments by checking if the ModuleAssignments index of the $course_modules array has any values.

If the ModuleAssignments index does not have values, the module has no assignments. Therefore a tile element is printed notifying the user of this and provides a button linking to the Add Assignment form using site_url() to construct the href.

5.2.6 Module Page

Again the length of this document does not permit this aspect of the project to be discussed in sufficient detail, therefore it has been decided to explain this functionality during the site demonstration.

5.2.7 Assignment Page

The controller and model functions involved with presenting the assignment page are quite similar to those discussed for the Course and Module pages, therefore it has been decided to discuss aspects of the assignment view itself and how they relate to controller and models. Each subheading list related source files.

jTable Integration – views/assignment_view.php

The assignment view includes a dynamic table showing the work sessions recorded for that assignment, which is achieved using the AJAX based jTable plugin. The jtable method was called on the #session_table div, was would cause the plugin to generate an html table inside the container. Various option parameters were then set in order to achieve the functionality needed. Paging was set to true to allow pagination, and a default page size of 10 was set. Sorting was set to true to allow users to sort columns in ascending or descending order. defaultSorting was set to StartDateTime DESC as this would display most recent work sesson at the top of the table. saveUserPreferences was set to false to stop jTable storing user preferences as a cookie, as this was causing problems with CodeIgniter. Refer to code comments for further explanation.

jTable listAction – views/assignment_view.php, controllers/site.php, models/work_model.php

CodeIgniter's site_url() function was used to print out the appropriate controller function path. The current assignment id was appended to the end of the listAction path, passing it to the controller function jtable_list() via the GET method. The controller then called the get_jtable() function inside the work_model, passing in the assignment id, as well as three more parameters

automatically passed to the controller function via GET by the jTable plugin. These parameters are-
jtSorting - this identifies which column to sort by and whether ascending or descending.
jtStartIndex – this specifies which row to currently show as the first in the table, and is needed to allow pagination and filtering.
jtPageSize – this specifies the number of results to return for display on a single page.
A fourth parameter jtTextFilter is passed in which was POSTed to the controller from the view, which contains the text string entered by the user to filter the table results by Description.

The get_jtable() function inside the work_model, queries the database to get a record count of how many work sessions there are for the current assignment with Descriptions that match the text filter string. Using the SQL rule LIKE with % wildcards mean the filter string can appear anywhere in the Description even as a substring of another word. If the text string is empty, all appropriate assignments are returned regardless of Description as the LIKE rule is basically ignored if an empty string is supplied. The record count value is then assigned to variable.

Another database query is made to select the actual results for the table. This involves a rather complex SQL query. Some notable features of which are the use of SQL's DATE_FORMAT function to separate the date and time from datetime fields and assign as separate result values. The TIMESTAMPDIFF function is used to calculate the difference between the StartTime and EndTime, giving the duration of each work session in seconds.
The WHERE clause of the query is constructed using the assignment_id and text filter parameters passed from the controller, the constructed string is then appended to the existing query string.

An ORDER BY rule is constructed using the sorting variable, and a LIMIT rule is constructed using the start_index and page_size variables. This is also appended to the existing query string. These rules enable the pagination and sorting of the jTable displayed to the user. The results of the query are then returned as an array. A foreach then loops through the array to stripslashes() from the Description, which were added by the insert/edit forms. The custom hours_minutes() helper function is used to convert the TimeSpent value into a human readable string.

Another array is then constructed, to hold the record count value calculated above, as well as an array of the records returned from the database. This array is returned to the controller function, which encodes it into a JSON object as required by the jTable plugin.

jTable Search Filter – views/assignment_view.php,

The assignment view includes a text input allowing the user to filter the table results using a keyword search of the Description column. This is achieved by binding jQuery's keyup() event to the input field using the id #jtable_filter as the selector, when the event fires the getTable() function is called. The getTable() function contains the jtable() method call which initialisation and generates the jtable on the page. Inside the jtable() method a parameter called jtTextFilter is set with the current value of the text input field, this is achieved using jQuery's val() method. This parameter is automatically POSTed by the jTable plugin when the listAction action is called.

jTable deleteAction – views/assignment_view.php, controllers/site.php, models/work_model.php

The deleteAction path was constructed using CodeIgniters site_url() function. jTable automatically adds a delete button to each row of the table, and when clicked it POSTs the assignment id of the row selected to the controller function. This id is passed into the delete_work_session function inside the work_model. This then constructs an SQL query to delete the appropriate work session from the database. A JSON encoded object is returned, as required by the jTable plugin.

jTable fields – views/assignment_view.php

The fields option allowed the columns to be specified for the jTable. The SessionID column key parameter was set to true, as this would enable it to be used as the unique identifying used when deleting a row from the table. A notable achievement in regards to the jTable fields was the ability to specify one column name to be used for sorting the table but displaying another column value in the actual table. This was achieved using the display option. An html string was constructed into which values would be inserted. These values were set using data.record then the name of teh column which was to be displayed. Data.record would access the JSON object supplied from the database, and the column name would specify which column value to use.

5.2.8 Settings Page

Populated Settings Checkboxes – controller/site.php, user_model.php, settings_view.php

The settings page is populated with the current user settings from the database, which includes a set of checkboxes showing the working days selected by the user. The settings() function in the site controller assesses a model which returns an array of the current user's settings. It then uses PHP explode function to convert the comma separated list of days into an array. Another array was created containing the shortened version of the day name as the key, and the long version as the value. This would be used inside the view to assign appropriate values and labels to the

checkboxes. The settings view was then loaded and the data passed in. Inside the view a foreach loop printed out a checkbox for each of the days, it checked whether the exploded array values matched those in the other array, and if so set the checkbox to checked. The second array was also used to supply the appropriate long version of the day name to be displayed as a label, by matching it against the short version key with the values in the exploded array.

5.2.9 Email Functionality

Weekly Email Summary - controller/email.php, model/email _model.php

A new controller named email was created to handle the email summary functionality using a function called weekly_summary. The function identifies the date of the Monday and Friday of the current week, using PHP's date() and strtotime() functions. These are set as variables and passed to the email model to query the database. The email model function constructs an SQL query using the variables passed in, to select all users who worked that week and return the results as an array. A foreach loop accesses this array, and queries the database again to select information for modules worked on by each individual user. A nested foreach loop then selects information of for the assignments within each module.The result of these queries and nested foreach loops is the creation of a multidimensional array containing a list of all users who worked that week, including information on each module and each assignment within that module. The multidimensional array is return to the controller. A set of nested foreach loops within the controller then access the array to construct the html message that would be used in an email.

Unfortunately CodeIgniter's email library was not compatable with the uni server, despite working locally and so development was halted and time prioritised elsewhere. However if the feature was to operate as expected a cron job would have to set up on the uni server, to call the controller function. This would likely involve technicians when hosting on the uni server, however if hosted by an outside company an interface like cPanel could be used instead. It was decided to edit the controller to simply print out each email message to the screen, so that the functionality could be seen during the demonstration.

5.3 Iteration 3 - Securing & Optimizing Code

The third major phase of iterations involved optimizing the existing code and making certain features more secure.

Cross Site Scripting Filtering – config/config.php

As the site included lots of forms data being sent via POST it was important that they were protected against cross site scripting. CodeIgniter has inbuilt XSS filtering and this was enabled by setting $config['global_xss_filtering'] to TRUE inside the config.php file.

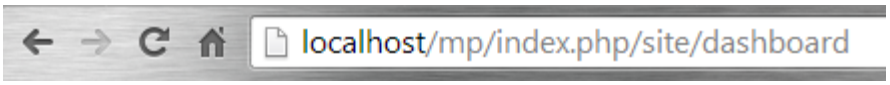
Password Encryption

Earlier iterations simply took the user's password string and stored an MD5 hashed version in the database. However MD5 hash has be cracked and can easily be decoded to return the original string, therefore a more secure method was needed. It was decided to add a salt to the password before hashing the combined string, as this would prevent a hacker from determinin the original password. The salt string was generated using an encryption key stored in CodIgniter's config.php file. The use of a salt protects against Rainbow Table attacks which check a list of pre calculated hashes against the passwords stored in a database. The rainbow table also contains the original string for each hash, and can therefore reveal the unencrypted password. However this approach will not work unless the creator on the rainbow table also knows the salt string. Salting also provides protection against Dictionary attacks, which systematically try an exhaustive list of common words or passwords.

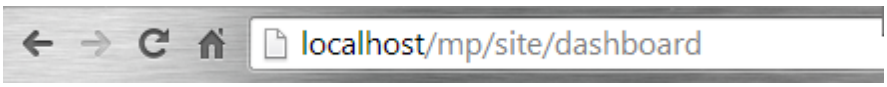
As evidenced by the recent Heartbleed bug, a security breach of one website can potentially leave other site's user accounts at risk, some of which may contain sensitive personal information. Therefore it is important that the user's password is safely encrypted, in case a hacker gains access to the database.

Tidying URLs - mp/.htaccess, config/config.php, config/routes.php

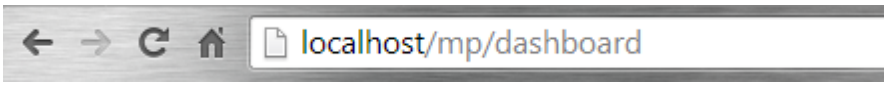
By default CodeIgniter URLs contain index.php followed by the current controller name and function name. This is quite untidy and not exactly user friendly.



The config.php file was edited to set the index_page value to a blank string; an htaccess was then created in the root folder of the site to remove the index.php from the URL.



The routes.php file was then used to remove the controller name from the URL, this involved rerouting the shortened display URL to the original URL with controller and function name.



Unfortunately the university servers prevented the htaccess file from working, and therefore the index.php was reinserted on the live version. However the clean URL will be visible during the site demonstration.

6. Testing

6.1 White Box Testing

As the developer had a working knowledge of the code, they employed white box testing during each iteration of the system. This would ensure that the code being written behaved as intended.

6.1.1 Unit Testing

The developer employed unit testing during development to ensure that the current section of code was working in isolation before attempting to integrate it with the rest of the system. This would help catch errors and bugs in the code early on, and would prevent faulty units from causing other parts of the system to fail. This would prevent time being wasted trying to debug the entire system after integration. This mostly came in the form of developing and testing the controllers and models in isolation.

Unit Testing a Controller function

Figure 6.1 shows a code snippet demonstrating how unit testing was employed when creating the insert_update_course() controller function.

Figure 6.1 – Code showing unit testing in controller function

```
$form_type = "insert"
$form_validation = TRUE;

if ($form_validation !== false ) // If the form validates
{
    $data = array(
    'CourseID' => NULL,
    'CourseName' => "Hello World"),
    );

    if($form_type == "insert") // If the action was called buy an insert form
    {
        print "Insert Query";
        var_dump($data);
        var_dump($_SESSION['username']);
    }

    if($form_type == "update") // If the action was called buy an update form
    {
        print "Update Query";
        var_dump($data);
        var_dump($_SESSION['username']);
    }

} // Close if form validated
```

The same function would be used for both insert and update forms (this will be discussed further during the demonstration), when integrated into the system the form type would be passed from the form itself. However during unit testing this was simulated using a variable with an appropriate string value. When integrated with the rest of the system, form validation would be applied to the inputted form data, however for unit testing this was simulated using a variable set to either TRUE or FALSE.

If the $form_validation variable was set to TRUE an if statement would run and simulate data being passed to the course_model. This consisted of hard coding the values into the $data array. A set of nested if statements then check the value of the $form_type variable. An appropriate message is printed to the screen to signify the type of query that would be run. The $data array and SESSION variable values are also dumped to the screen to ensure that the correct data is being passed. A similar approach was used to simulate the form validation being successful.

Unit Testing a Model

Figure 6.2 shows how unit testing was implemented when developing the insert_course() function in the course_model. When integrated with the controller, the function would accept two parameters. The first would be the $data array containing the inputted form data, and the second

would be the user's id accessed from the SESSION variable. However during unit testing these were simulated using a hard coded array and variable.

Figure 6.2 – Code showing unit testing in a model function

```
function insert_course()    // THIS FUNCTION INSERTS ASSIGNMENT
    {

        $data = array(
            'CourseID' => NULL,
            'CourseName' => "Hello World"),
            );

        $username = "clarke-t6@email.ulster.ac.uk"

        $query = $this->db->insert('course', $data);    // ENTER COURSE DATA INTO TABLE
        $course_id = $this->db->insert_id();            // GET ID OF THE LAST ENTERED RECORD

        $query = $this->db->query("SELECT UserID FROM user WHERE Email = '" . $username . "'");
        $user = $query->result_array();

        $this->db->query("INSERT INTO user_course VALUES (" . $user[0]['UserID'] . ", " . $course_id . ")"); // INSERT INTO LINK TABLE
    }
```

This approach was repeated when creating all other MVC interactions throughout the system, to ensure each entity was working correctly in isolation before attempting to integrate them with the rest of the system.

6.1.2 Integration Testing

Once all units had been tested successfully, they were integrated with the rest of the related units. Figure 6.3 shows how the insert_update_course() controller function was integrated with the various model functions.

Figure 6.3 – Code showing how controller function was integrated with model and view

```
/* Form Validation */
$this->load->library('form_validation');
$this->form_validation->set_rules('course_name', 'Course Name', 'required|max_length[100]');

if ( $this->form_validation->run() !== false ) // If the form validates
{
    $data = array(
    'CourseID' => NULL,
    'CourseName' => htmlspecialchars($this->input->post('course_name'), ENT_QUOTES),
    );

    if($this->input->post('form_type') == "insert") // If the action was called buy an insert form
    {
        $this->load->model('course_model');                              // Load course model
        $this->course_model->insert_course($data, $_SESSION['username']);   // Insert course into database via model
    }

    if($this->input->post('form_type') == "update") // If the action was called buy an update form
    {
        $data['CourseID'] = $this->input->post('course_id');
        $this->load->model('course_model');                              // Load course model
        $this->course_model->update_course($data);  // Insert course into database via model
    }

    $this->index();                                                  // Return to dashboard view

} // Close if form validated
```

The variable used in the unit testing to simulate form validation has been removed. Instead the actual data POSTed from the view form is validated using CodeIgniter's form_validation library.

The variable used in unit testing to simulate the form type, has been removed and instead the if statement uses data POSted from a hidden input field in the view to determine the form type value. The nested if statements now integrate with the course_model. The first now passes the $data array and username value from the SESSION into the insert_course() model function to add a new course to the database. The second nested if statement now passes the POSTed course_id into the update_course() model function in order to update the appropriate database entry.

6.2 Usability Testing

Once the site was live, users were asked to complete an online survey about their experience using the site. This also acted as a form of black box testing as the users were asked to perform certain test cases, without knowledge of the underlying code, and record any problems they encountered. See Appendix I- for screenshots showing the results of the survey, or view online results at surveymonkey.com/results/SM-SJG3SRJ/. These results will be discussed in the Evaluation chapter.

6.3 Requirements Fit Criterion

As mentioned in the Requirements Specification, atomic requirements were recorded in the form of Volere Snow Cards, see 0. This included Fit Criterion which were could be used in order to test whether the requirement was met. Table 6.1 shows the results of this testing.

Table 6.1 ID#	Fit Criterion	Requirement Met?
01	Either a paper based or digital based ER diagram has been produced	Yes
02	A MySQL database has been created using PHPMyAdmin.	Yes
03	Use interface to register account using the email clarke-t6@email.ulster.ac.uk.	Yes
04	View user account settings created in #3 on settings page.	Yes
05	Use settings page form to change user account work hours and work days.	Yes
06	Use Add Course form to insert Interactive Multimedia Design as course into database.	Yes
07	Use the site interface to view course details entered in #6	Yes
08	Use interface to edit the name of the course added in #6 to IMD.	Yes
09	Use the interface to delete the course added in #6	Yes
10	Use interface form to add COM601 to course added in #6	Yes
11	Use interface to view information related to the module added in #10	Yes
12	Use interface form to update information for COM601 module added in #10	Yes

13	Use interface to delete COM601 module added in #10	Yes
14	Use interface to add Individual Assignment to module added in #10	Yes
15	Use interface to view information for Individual Assignment added in #14.	Yes
16	Use interface to change name of assignment added #14 to Assignment One	Yes
17	Use interface to delete assignment added in #14	Yes
18	Use interface timer to record a work session for assignment added in #14	Yes
19	Use interface to view work session recorded in #18.	Yes
20	Use interface to delete the work session recorded in #18	Yes
21	A user should be able to tell at a glance how long their current work session has been running.	Yes
22	The user account clarke-t6@email.ulster.ac.uk created in #03 will show the COM 601 module added in #10.	Yes
23	A user should be able to create an account and add at least one module within 5 minutes of using the system.	Yes
24	This would be basically impossible to test on my own. Limited testing could possibly be carried out by granting access to a beta site to fellow members of the IMD course.	Unknown
25	Try accessing an account with incorrect login details, form validation should fail.	Yes

6.4 Browser Testing

See Appendix I-for results of browser testing.

7. Evaluation

7.1 Evaluation of test results

As of writing this 18 users had completed the survey, with the majority of them giving positive feedback. 72% of users reported that at least one element on the homepage would convince them into registering an account. Therefore the homepage design can be seen as a success, as the main aim of the page was to inform users and persuade them to use the site. Only 20% of users said it wasn't apparent how to proceed in regards to setting up courses, modules, assignments etc. Although this is relatively small, it may be beneficial to redesign the top navigation bar to emphasize the links more. This is backed up by the fact one user had trouble recording a work session, mentioning how they struggled to locate the timer link and suggested making the text bigger. Some of the main compliments given by users included the visual design, the easy to use interface and the easy to understand data visualization. Therefore it can be concluded that these aspect were successful and met the requirements identified at the beginning of the project.

As the system met all of the fit criterion identified in the atomic requirements, it is safe to say that the site is a viable and robust system which achieves all that it set out to do.

As shown in Appendix I- the site appear consistently in all major browsers. This is a great success as it will ensure the brand of the site is reinforced via consistency across all mediums. It also makes the site more user friendly and accessible as the layout of the navigation and other page elements will be the same no matter which browser is used

7.2 Evaluation of methodology

Using the prototyping methodology proved to be a good decision. The iterative approach allowed the developer to constantly assess each working version of the system in terms of how it met the requirements. It was also flexible enough to allow new requirements to be identified and incorporated into the next iteration. Overall this approach allowed the system to be refined via the various iterations to produce a system which would better meet the requirements identified.

7.3 Evaluation of Plan

At the beginning of the project the designer/developer devised a schedule for the design and development of the system. Unfortunately throughout the project there were complications in

another module, outside of the designer/developer's control, which prevented this schedule from being followed as planned. However the counter measures identified in the feasibility testing allowed the designer/developer to overcome these difficulties and continue forward with the project.

8. Conclusion

Overall it is felt that the project was a success. Various forms of testing have showed that the system meets the requirements specified at the start of the project. The designer/developer also succeeded in meeting the objectives set out,. Ultimately this led to the project aim of creating *"an online time tracking system to aid students with time management"* being achieved. This is verified by the positive feedback received via the online usability surveys.

The designer/developer is pleased with the role they played in the management of the project throughout. Despite setbacks along the way, they were able to design and develop a system which met the requirements specified on time. Although some minor styling issues persist, the designer/developer is pleased with the functionality they were able to achieve. Some notable achievements the developer is particularly pleased with is the use of nested foreach loops and complex SQL queries to create a multidimensional array of data.
Another notable aspect of the project was the use of the CodeIgniter framework, this allowed the developer to gain valuable skills and experience with a PHP framework which could be employed in the future career. The developer was also pleased with how they were able to integrate several jQuery plugins into CodeIgniter, most notably the AJAX based jTable.

The use of the MVC pattern is another valuable experience which the developer could employ in future projects. This especially reinforced the benefits of reusing code where possible as it sped up development and made the project easier to maintain.

8.1.1 Future Work

One aspect of the system that the developer would like to improve is the process of how form selectboxes were populated from the database. Currently selectboxes are populated with all available options from the database, for example on the timer page the selectbox is populated with assignments for all modules. This could potentially become a very large list. It would be better to implement a set of AJAX powered selectboxes that would allow a user to first select the

module they want to work on, then select a related assignment from a second dropdown that is populated using an AJAX call.

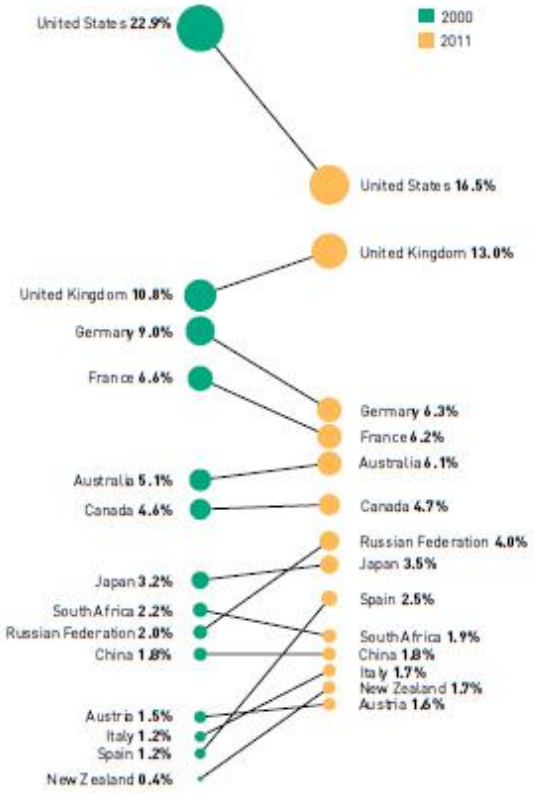
Unfortunately CodeIgniters email functionality was not compatable with the university servers. Therefore it would be desired to rectify this in some way in order to get the weekly email function working. This could involve using a service such as MailChimp to handle the sending of email. It could even be as simple as moving the site from university servers to a commercial hosting company, as their configuration may allow codeIgniter to handle email as it was doing when tested locally. If this was achieved, the developer would also like to implement a confirmation email feature when registering an account.
On e unwanted aspect of the site was the fact that POSTing data between controller, models and views meant that clicking the browser back button caused form data to be resubmitted. Some browsers allow this without question, whilst other ask for confirmation of resubmitting data. This is obviously not very user friendly. A potential fix to the problem would be to use CodeIgniter's flash data to store the data instead of POST, however further research would be required. Another potential fix would be to use AJAX calls on all forms, as this might negate the effects to actually submitting a form synchronously.

Another area which would be looked at is the storage engine of the databae, which is currently using a non transactional approach. Although this is faster, there is the potential of data being lost if hardware or software fails during an update. Therefore research would be carried out into storage engines that use a transaction safe approach, as these would prevent data being lost upon failure. Obviously it is extremely frustrating is a user's data is lost, especially if tracking a work session.

9. Appendices

Appendix A- Student Diversity Research

Figure 10: Shares of international student market, 2000 and 2011



Source: OECD (2013) *Education at a Glance*

1.5 Gender and age profiles

In 2011–12, 56% of students were female. Within this, there was some variation by level of study: at first degree level around 55% of students were female, while at postgraduate research level the proportion of women was around 47% (this had increased from 44% in 2003–04). At the same time, there was variation by domicile (Table 2). The gender split varies widely by country: 79% of students from Pakistan were male, but just 30% of students from Taiwan were male.

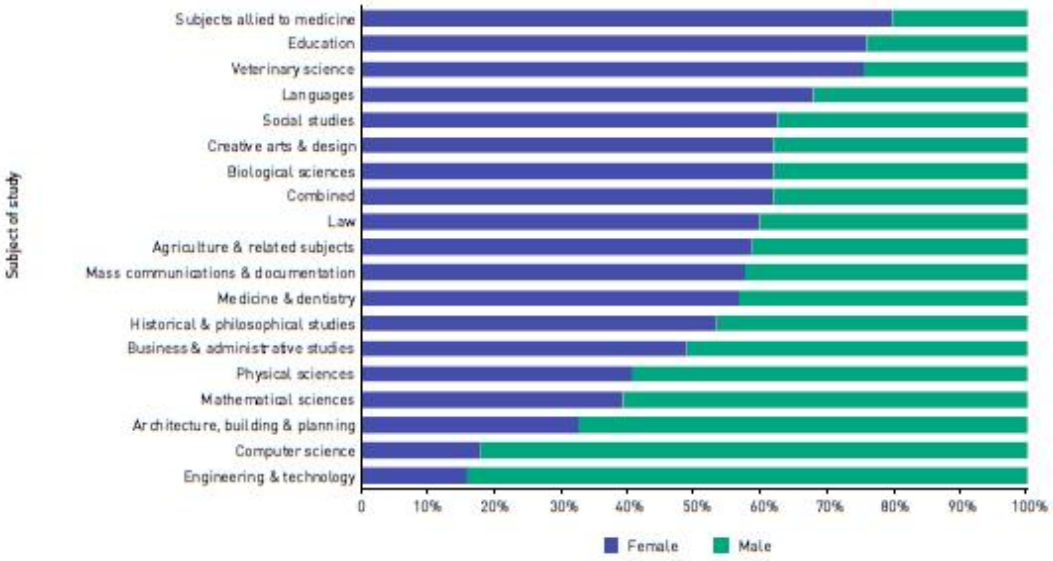
In addition, there was substantial variation in the proportions by gender between different subjects of study (Figure 11). It is beyond the scope of this report to explore the reasons for these differences, but the wide variation by gender between subjects is a striking feature of the student population.

Table 2: Gender of students by domicile, 2011–12

	Percentage of female students
UK students	57.7%
Other EU students	53.7%
Non-EU students	48.0%

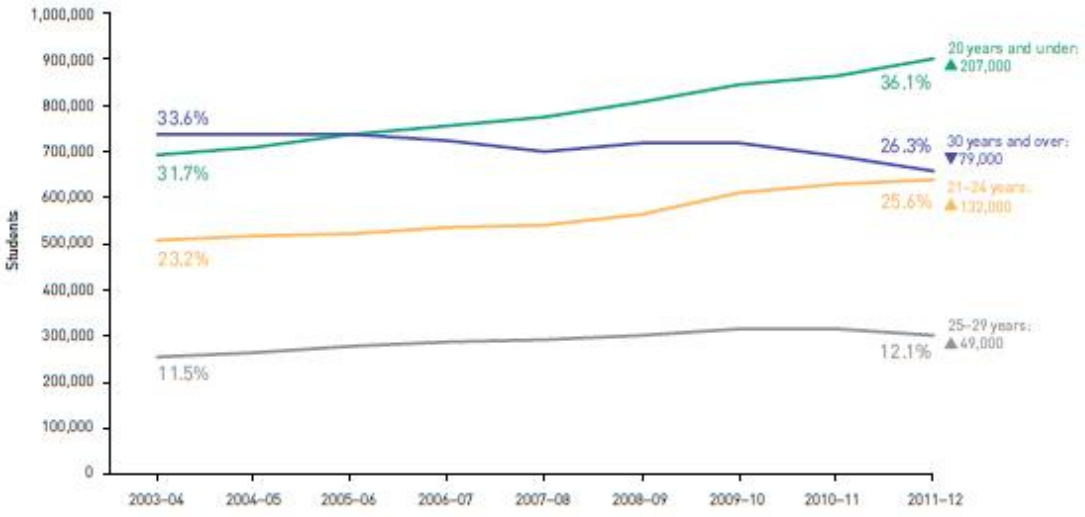
The number of students aged under 30 increased by 388,000 between 2003–04 and 2011–12, while the number aged 30 and over decreased by 79,000. Figure 12 shows changes in the age of students in further detail, and Figure 13 shows year-on-year percentage change. The latter illustrates that while the number of students under 30 increased every year between 2003–04 and 2011–12, the number of students over 30 did not always decrease each year despite the overall trend noted above.

Figure 11: Students by subject of study and gender, 2011–12



Source: HESA

Figure 12: Students by age group, 2003–04 to 2011–12



Source: HESA

Appendix B- Similar Sites Research

Shown overleaf.

Included

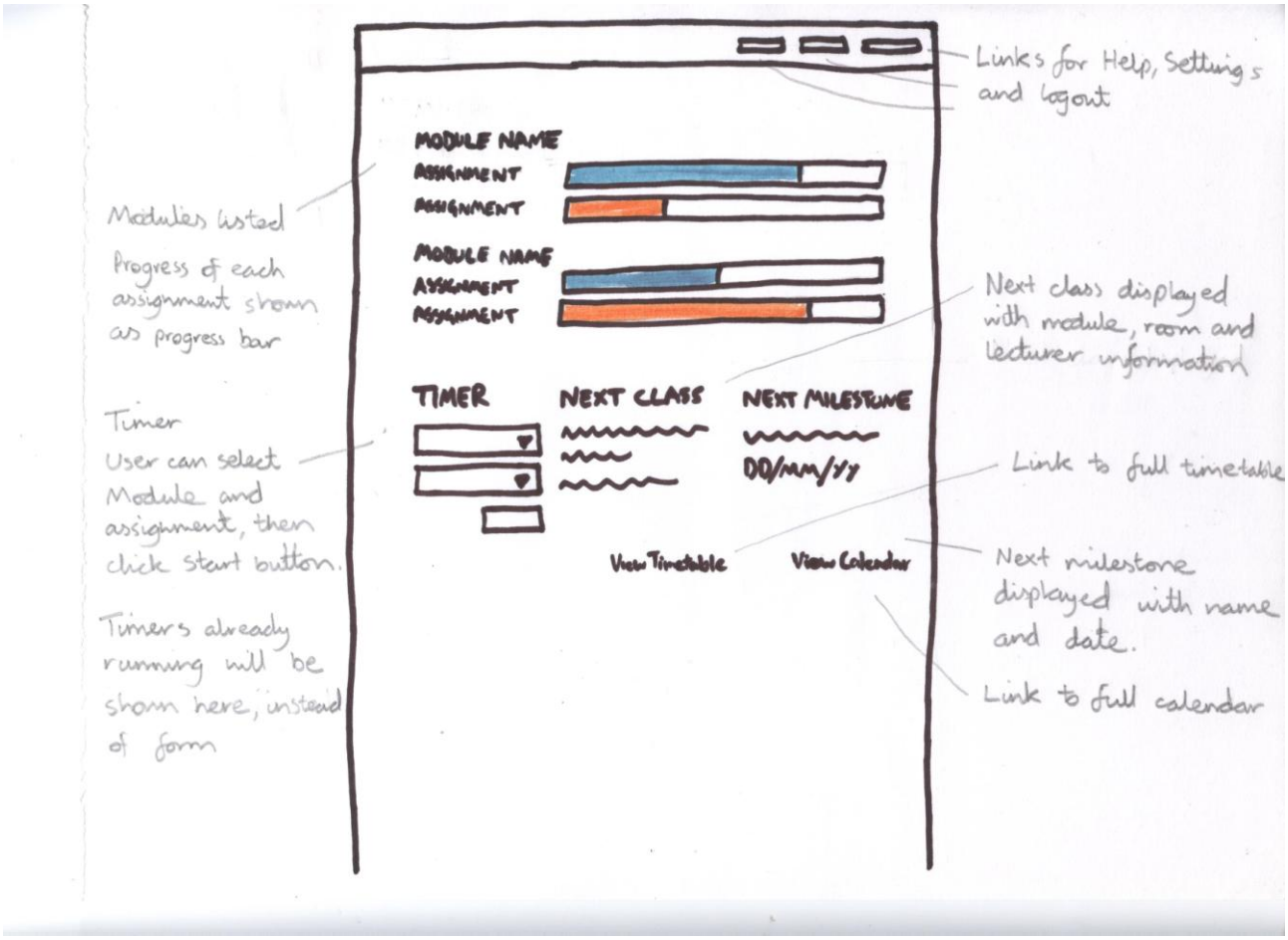
Either included with limited functionality or Available via third part add on.

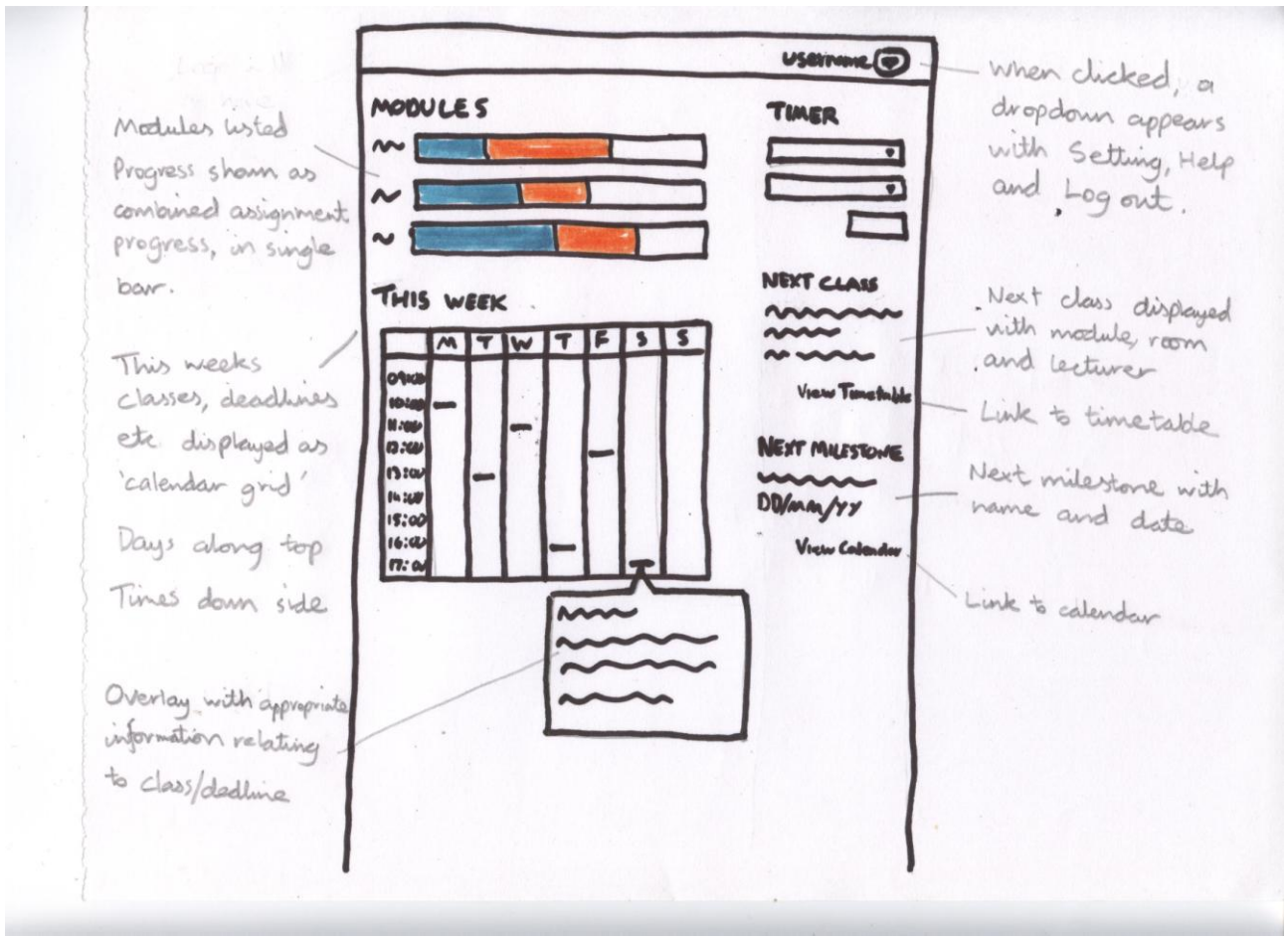
Not Included – features I was not sure about were marked Not Included as the website did not make them clear.

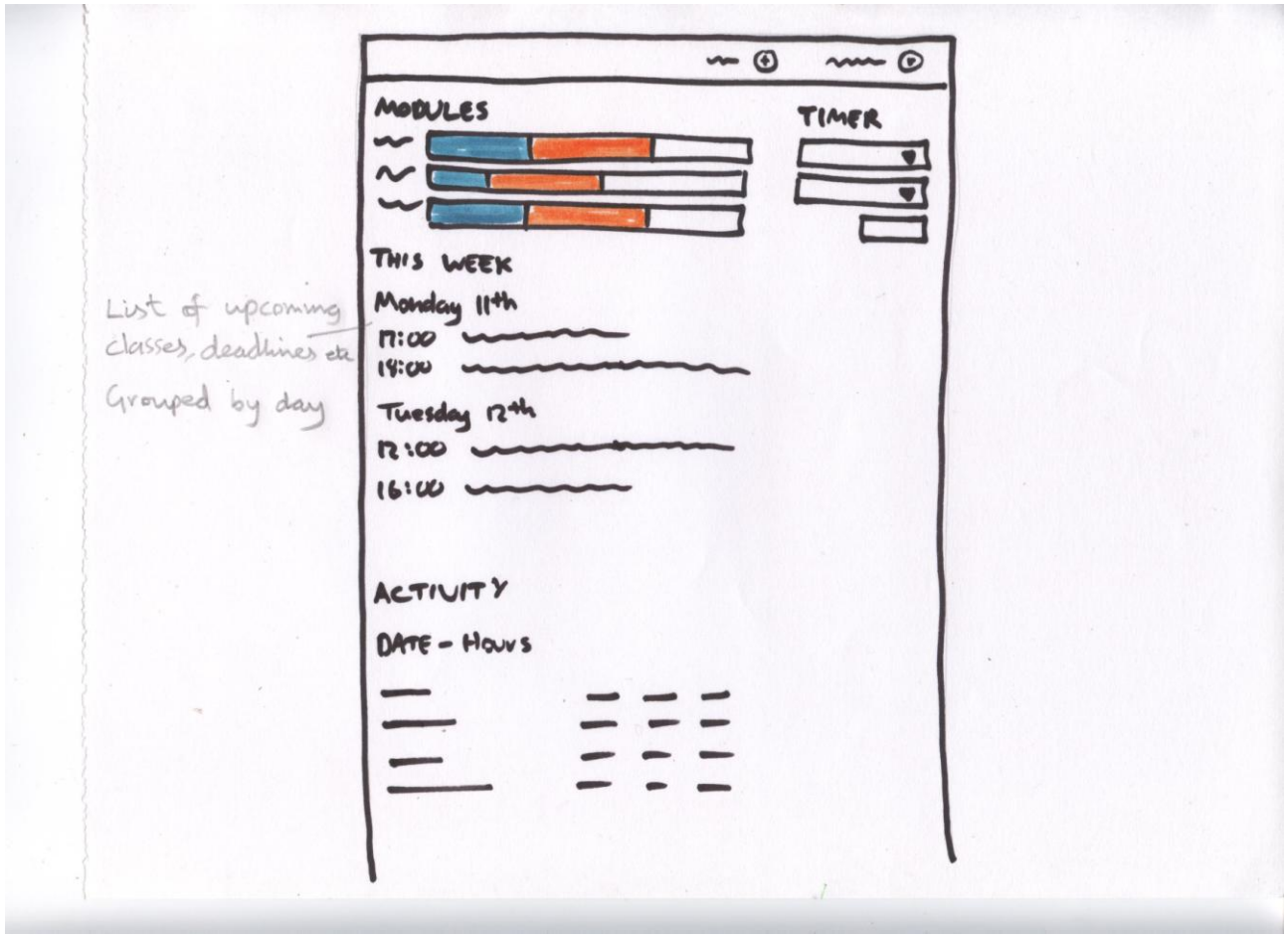
Planned for future release.

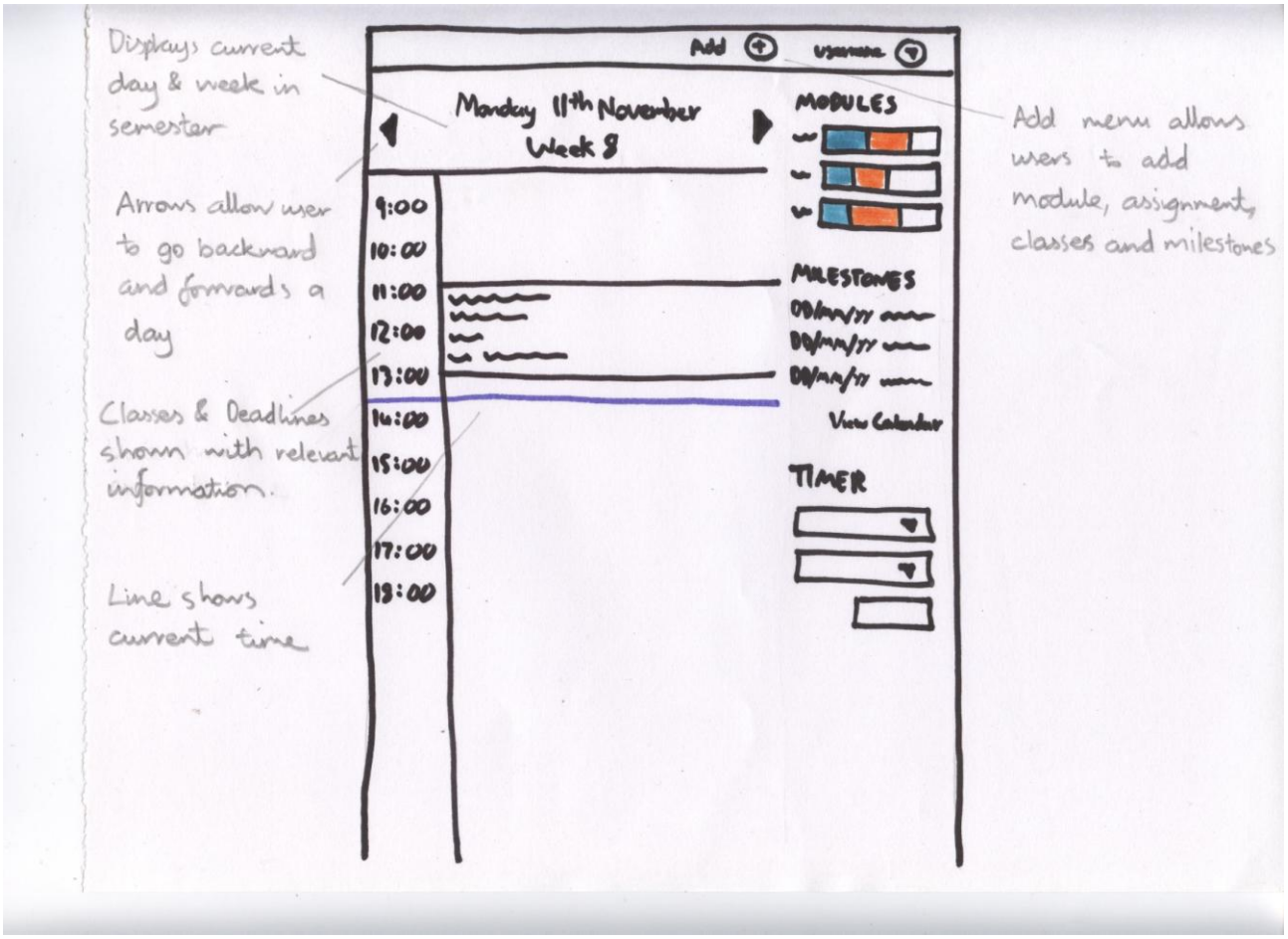
Appendix C- Page Layouts

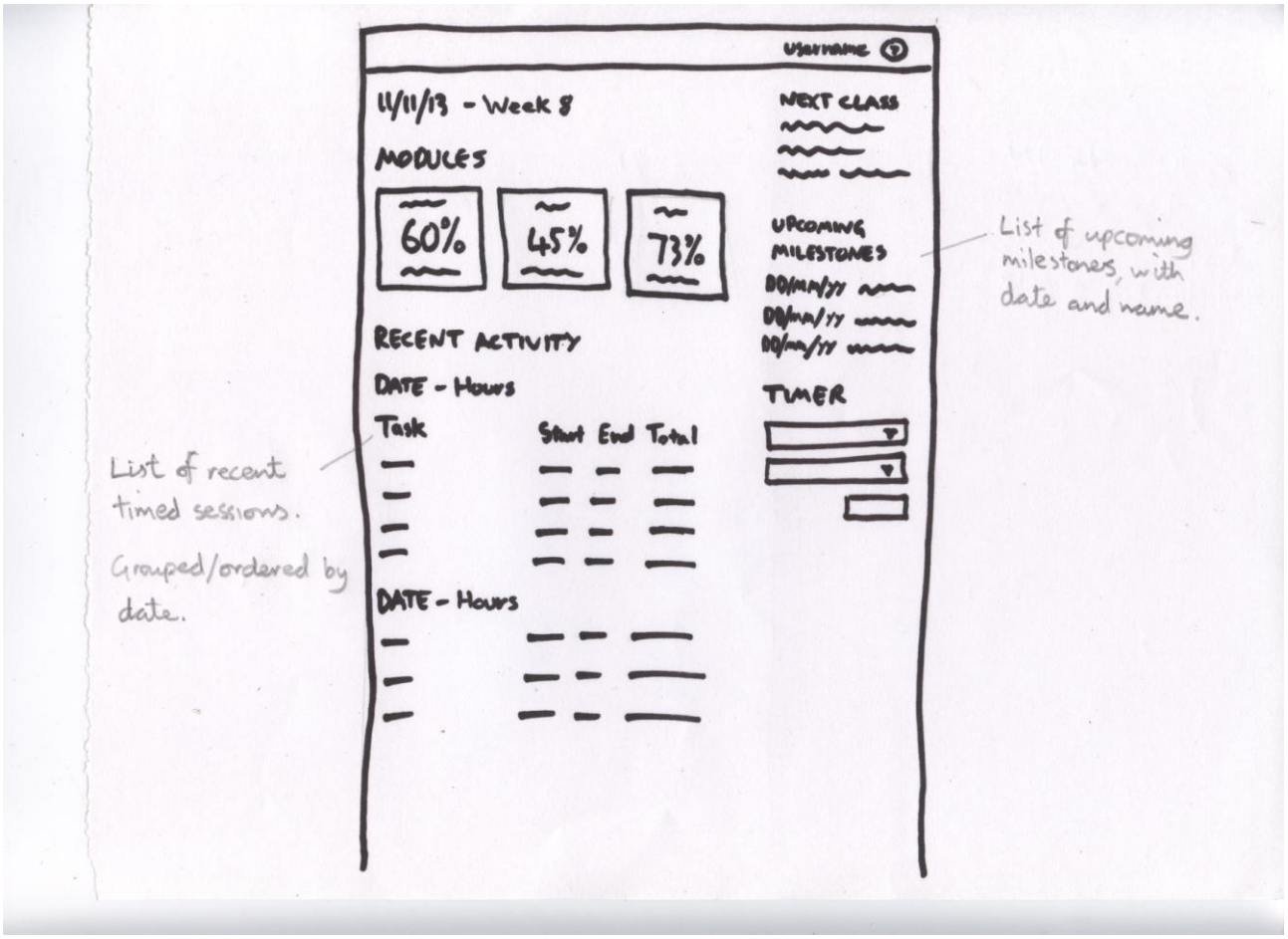
Appendix C.1- Initial Paper Prototyping

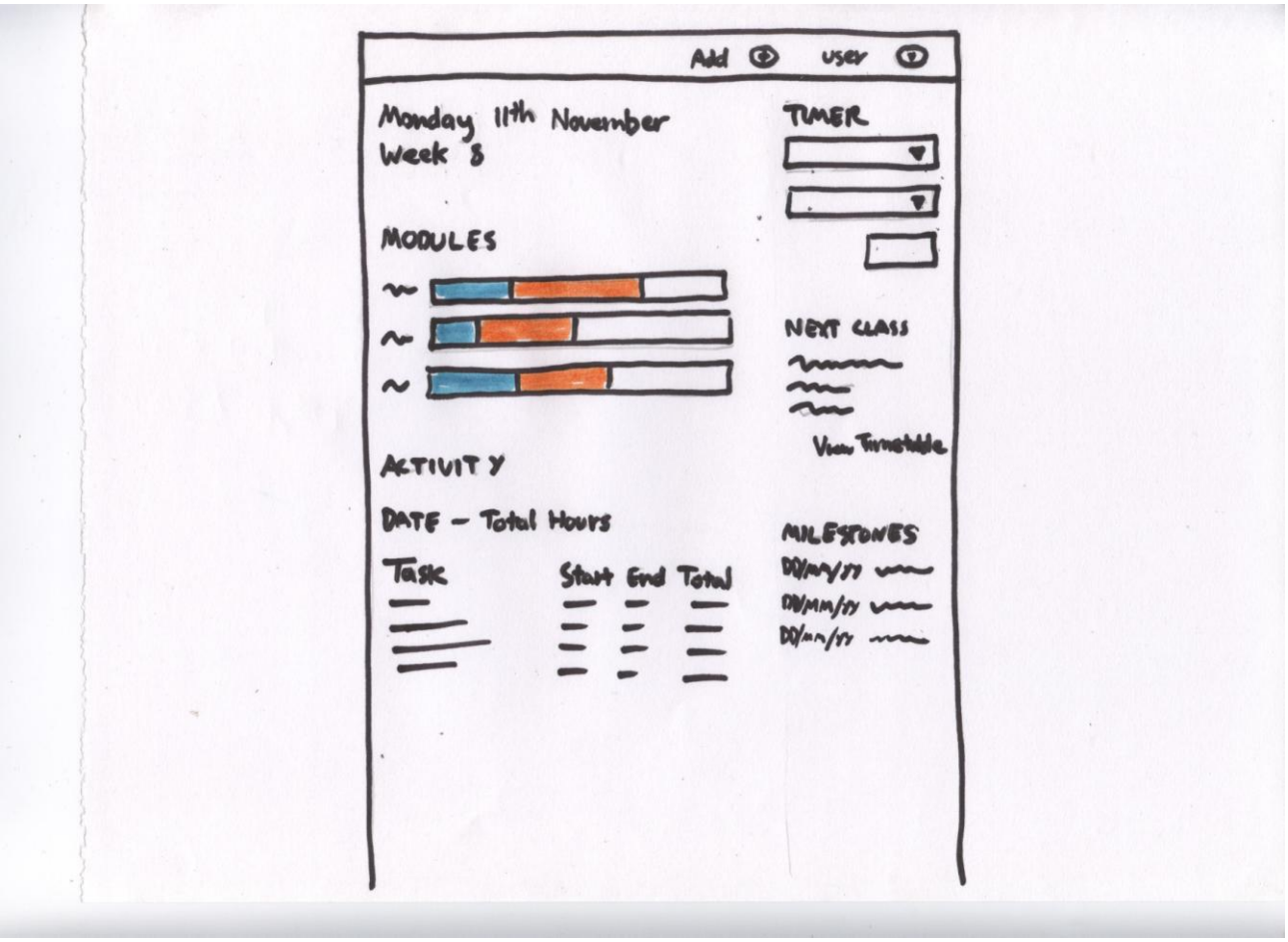



USERNAME
MODULES
TIMER
THIS WEEK
M T W T F S S
NEXT CLASS
View Timetable
NEXT MILESTONE
DD/mm/YY
View Calendar
Modules listed
Progress shown as combined assignment progress, in single bar.
This weeks classes, deadlines etc displayed as 'calendar grid'
Days along top
Times down side
Overlay with appropriate information relating to class/deadline
When clicked, a dropdown appears with Setting, Help and Log out.
Next class displayed with module, room and lecturer
Link to timetable
Next milestone with name and date
Link to calendar

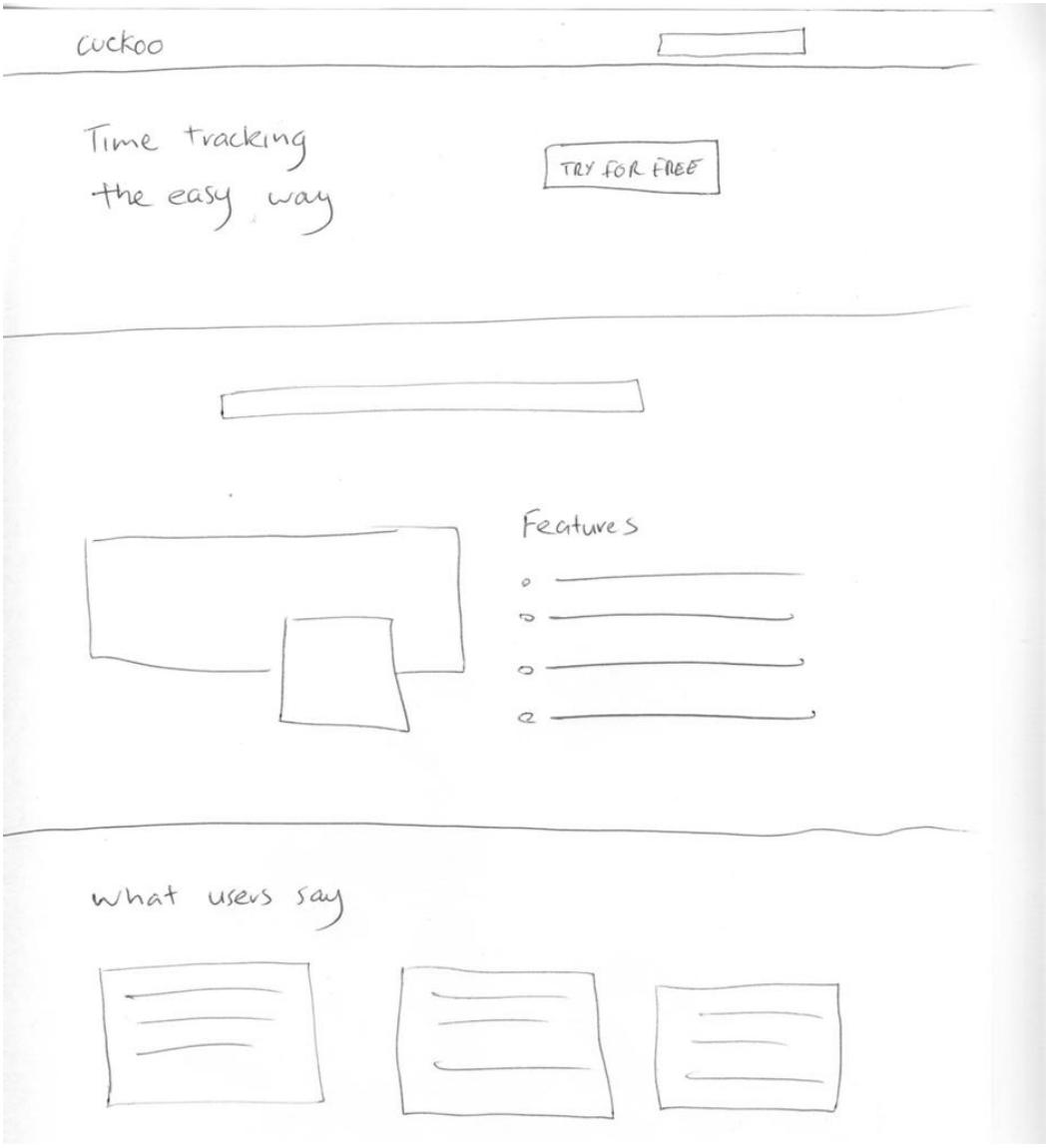

MODULES
TIMER
THIS WEEK
Monday 11th
17:00
19:00
Tuesday 12th
12:00
16:00
ACTIVITY
DATE - HOURS
List of upcoming classes, deadlines etc
Grouped by day

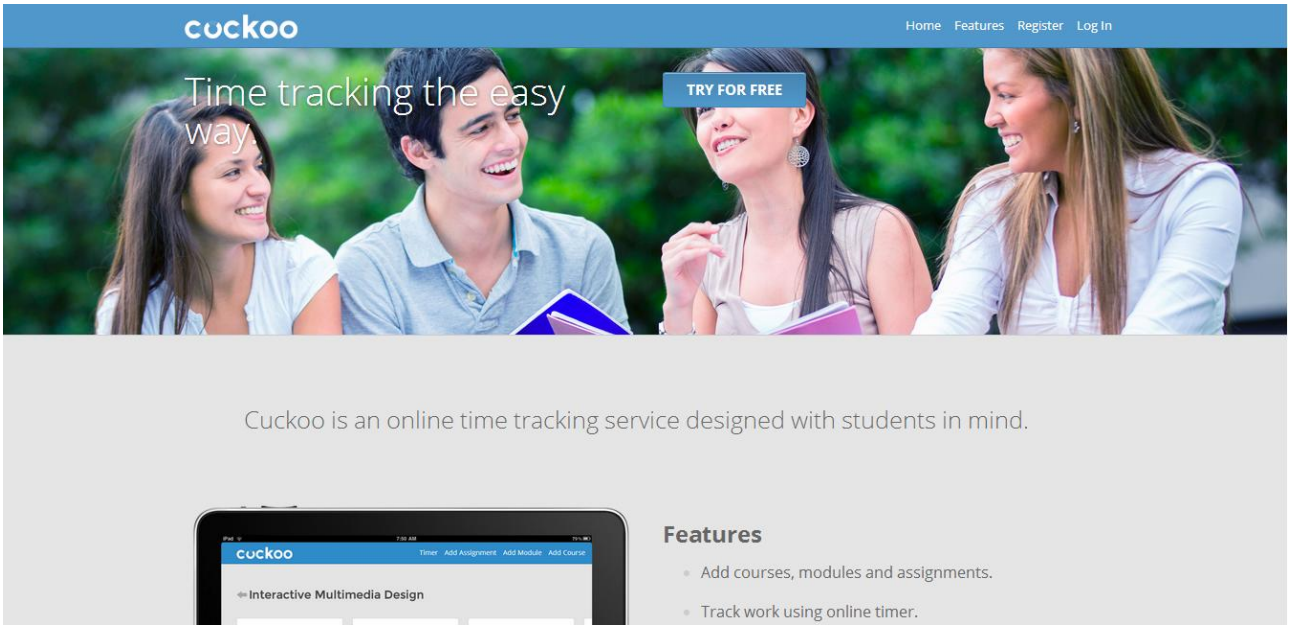

Displays current day & week in semester
Arrows allow user to go backward and forwards a day
Classes & Deadlines shown with relevant information.
Line shows current time
Add
Username
Monday 11th November
Week 8
9:00
10:00
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
MODULES
MILESTONES
DD/MM/YY
DD/MM/YY
DD/MM/YY
View Calendar
TIMER
Add menu allows users to add module, assignments, classes and milestones


Username
11/11/13 - Week 8
MODULES
60%
45%
73%
RECENT ACTIVITY
DATE - Hours
Task
Start
End
Total
DATE - Hours
NEXT CLASS
UPCOMING MILESTONES
DD/MM/YY
DD/MM/YY
DD/MM/YY
TIMER
List of recent timed sessions.
Grouped/ordered by date.
List of upcoming milestones, with date and name.

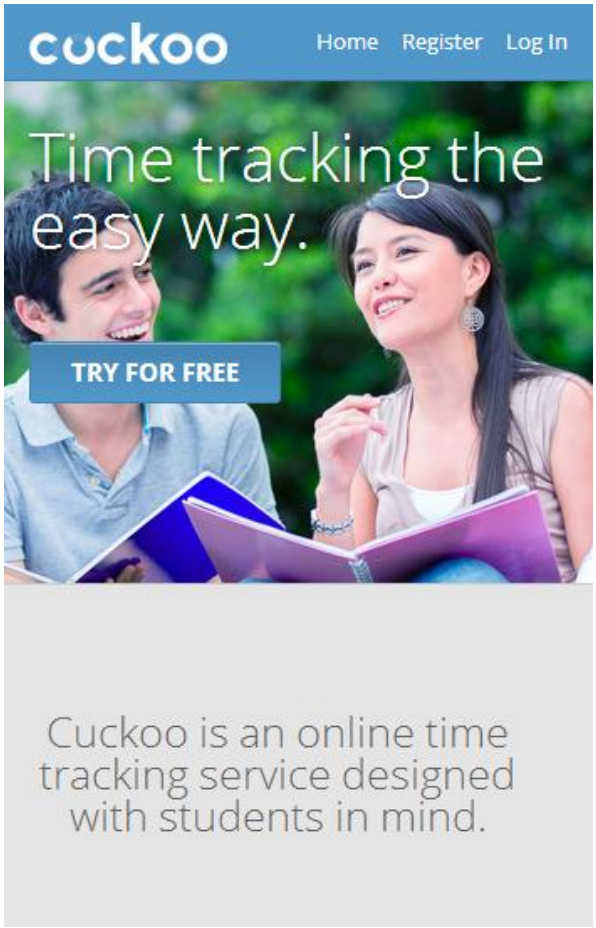

Add
user
Monday 11th November
Week 8
TIMER
MODULES
NEXT CLASS
View Timetable
ACTIVITY
DATE – Total Hours
MILESTONES
Task
Start
End
Total

Appendix C.2- Homepage (Desktop)

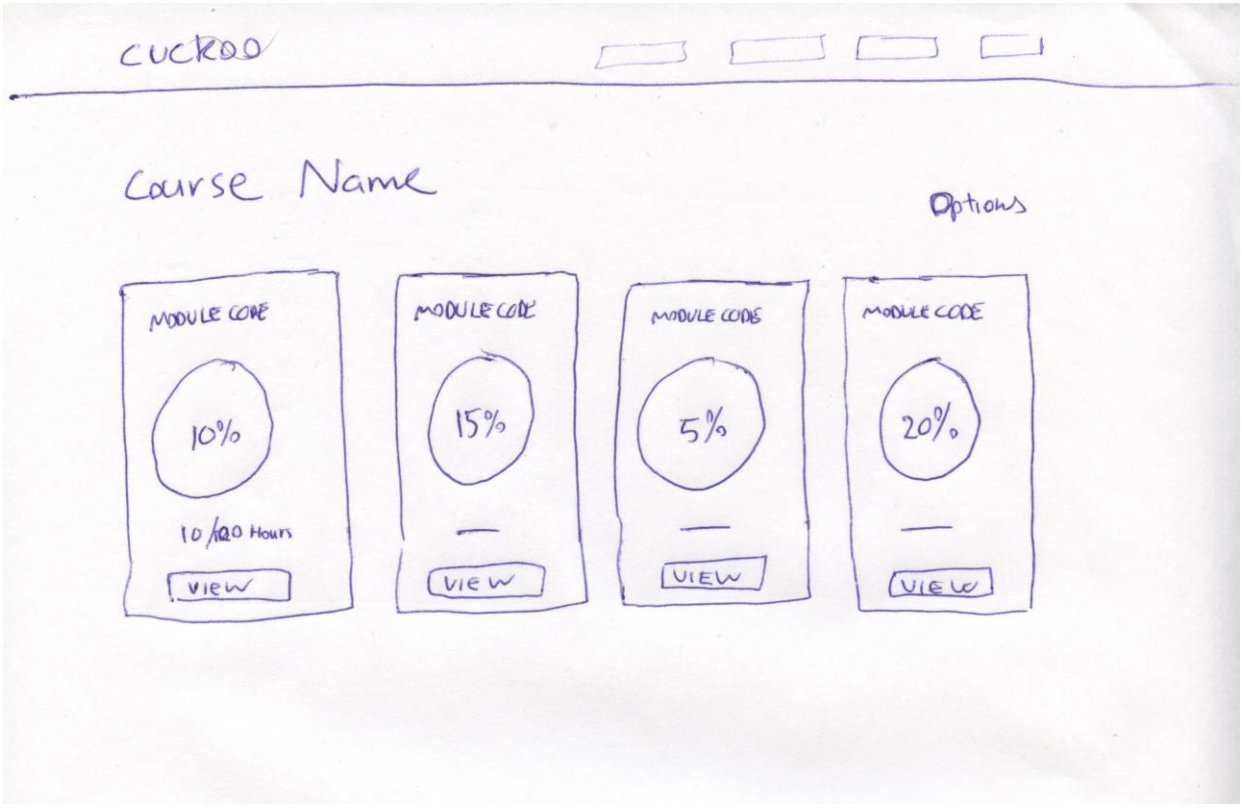


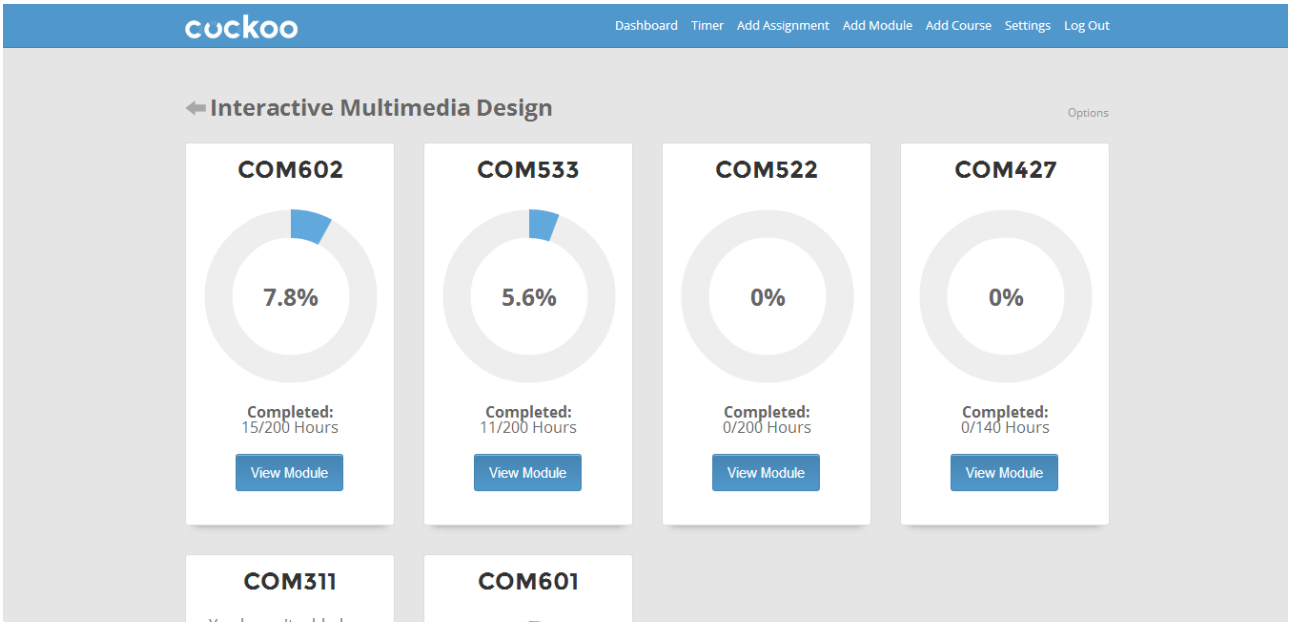


Appendix C.3- Homepage (Mobile)

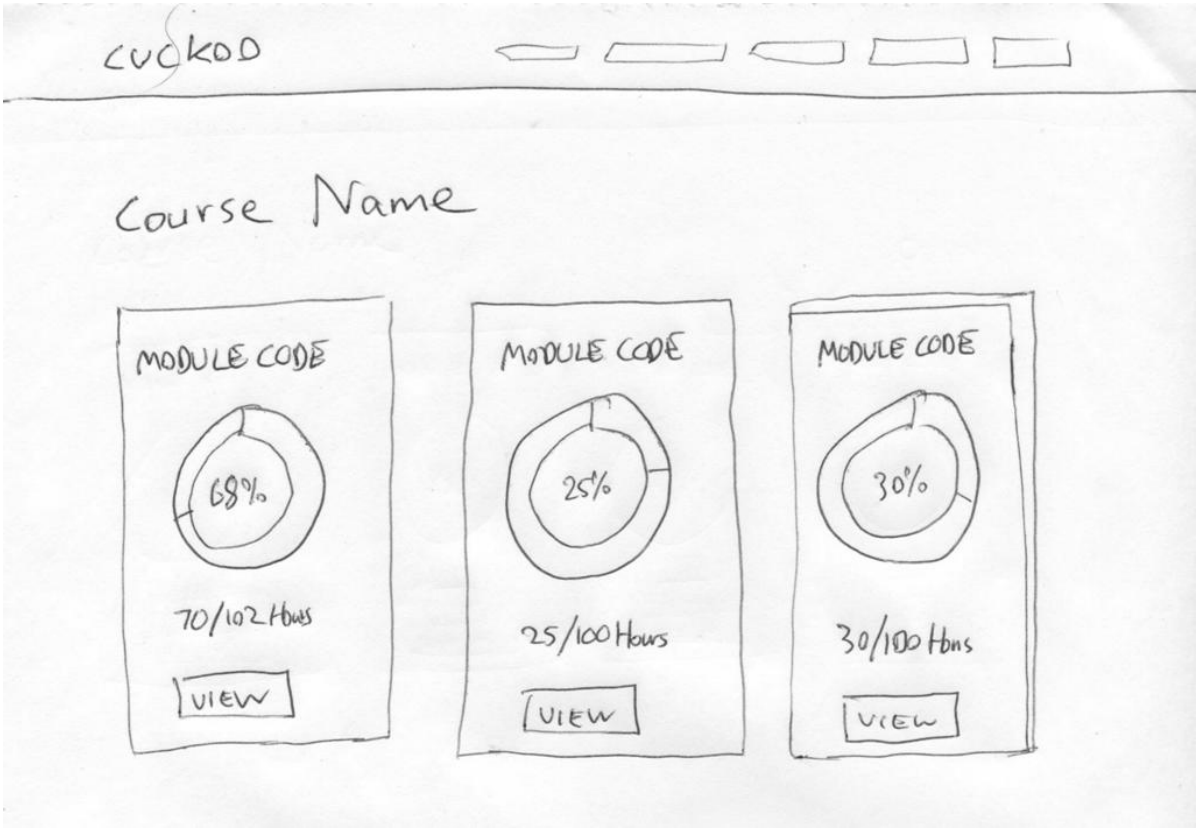


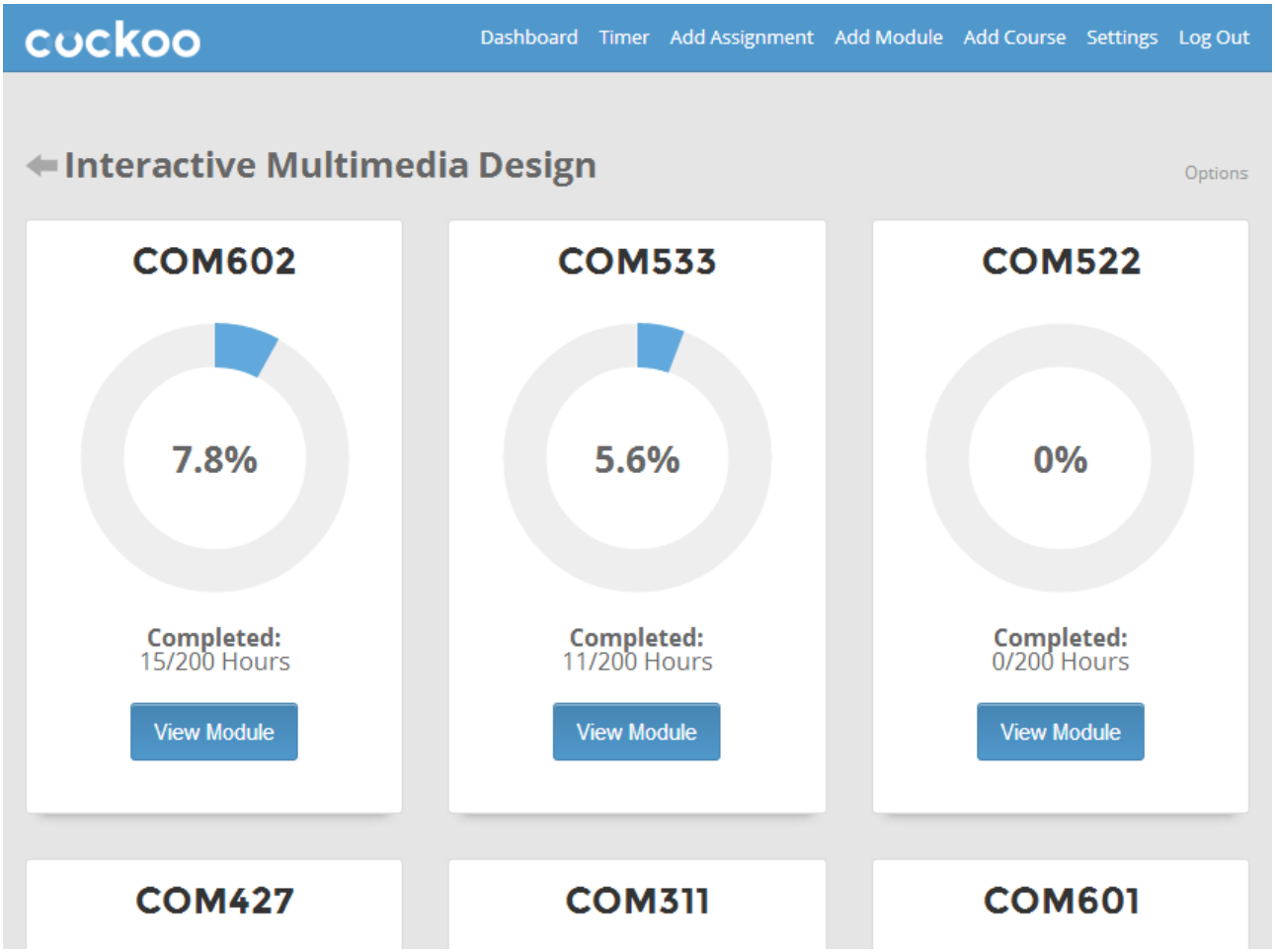
Appendix C.4- Course Page (Desktop)



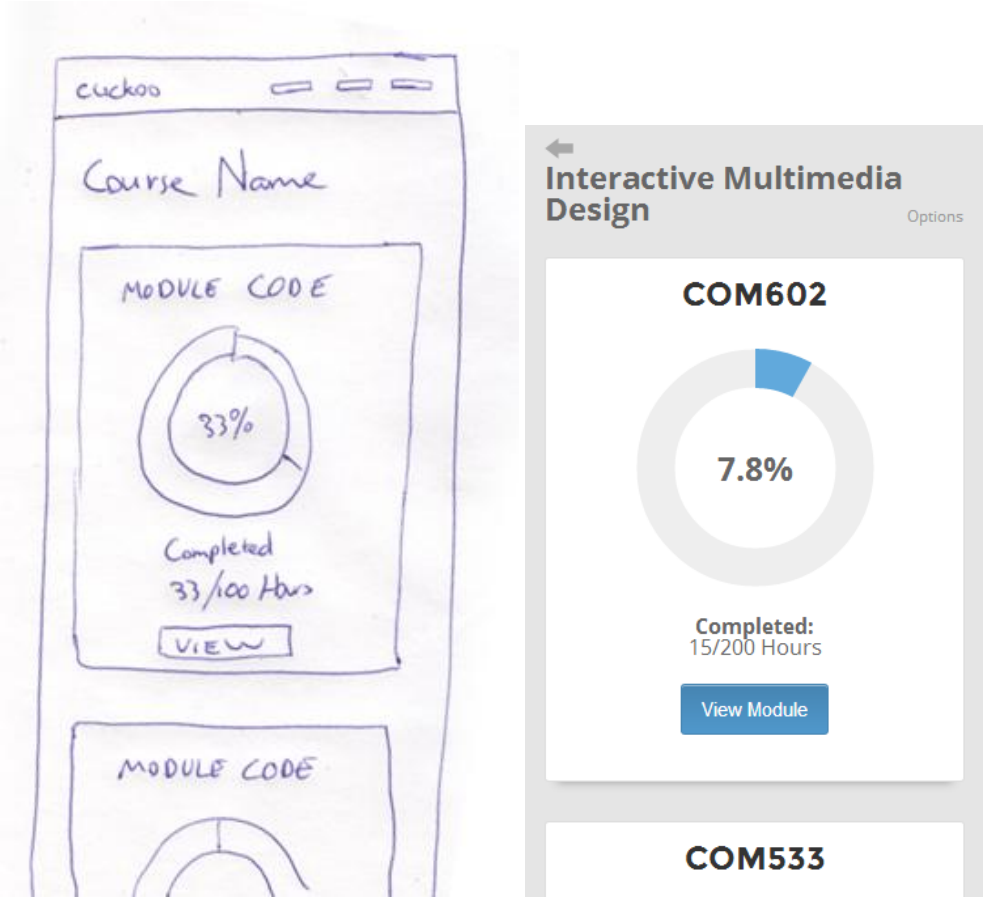


Appendix C.5- Course Page (Tablet)

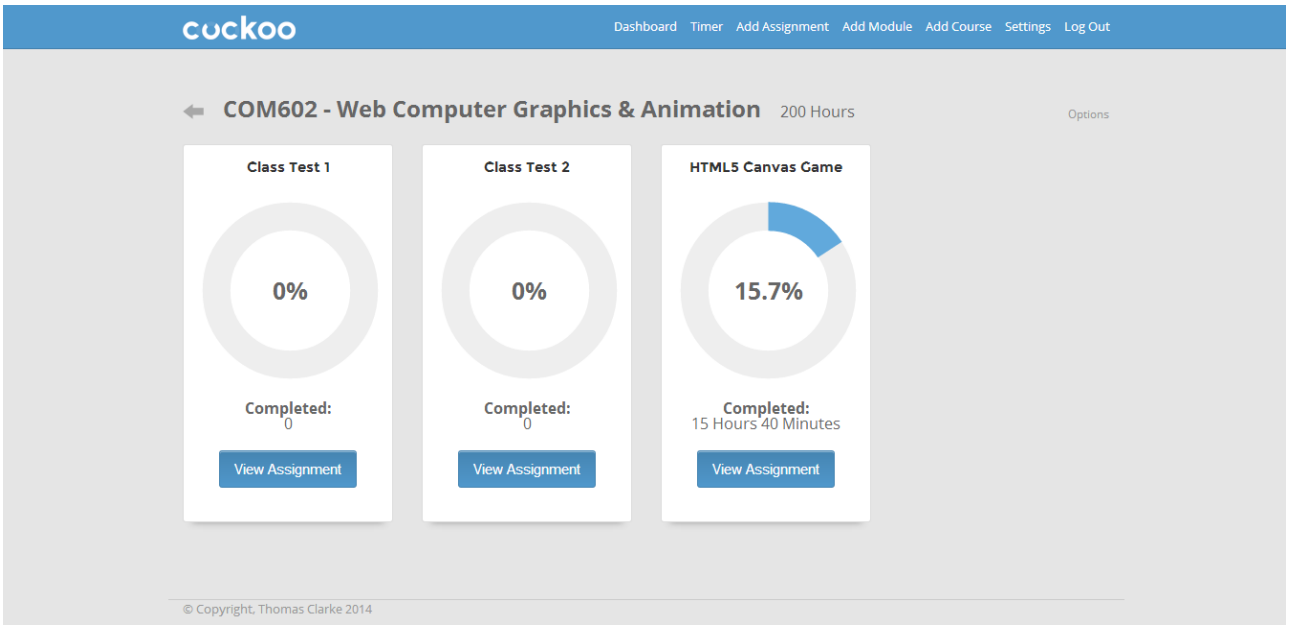


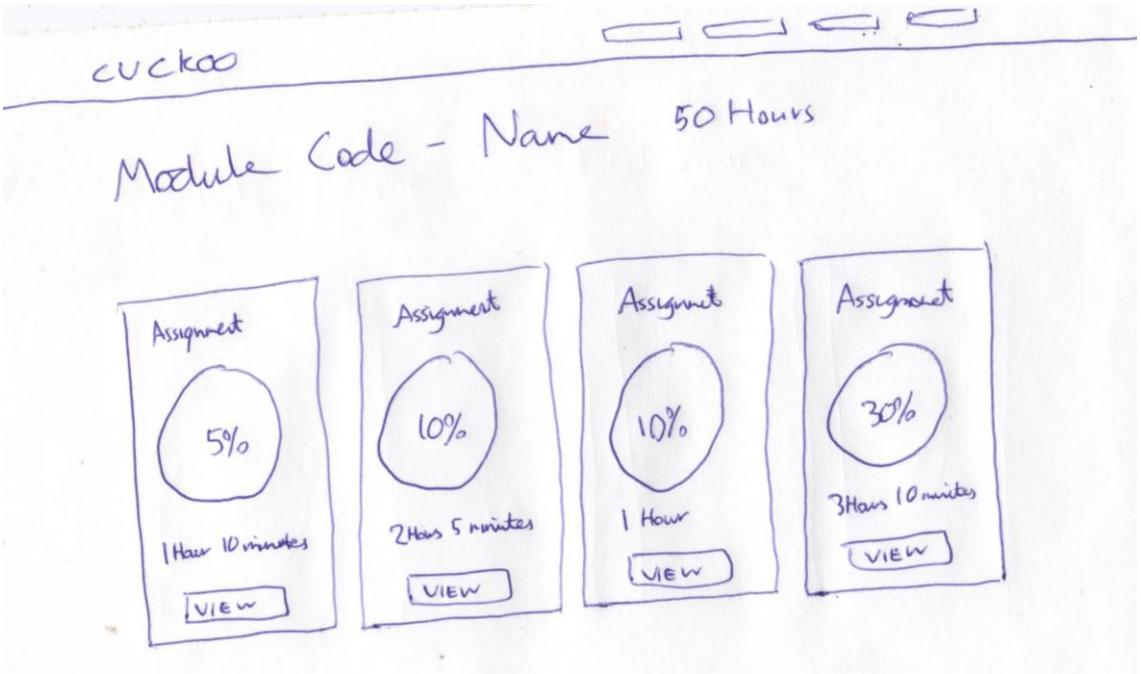


Appendix C.6- Course Page (Mobile)

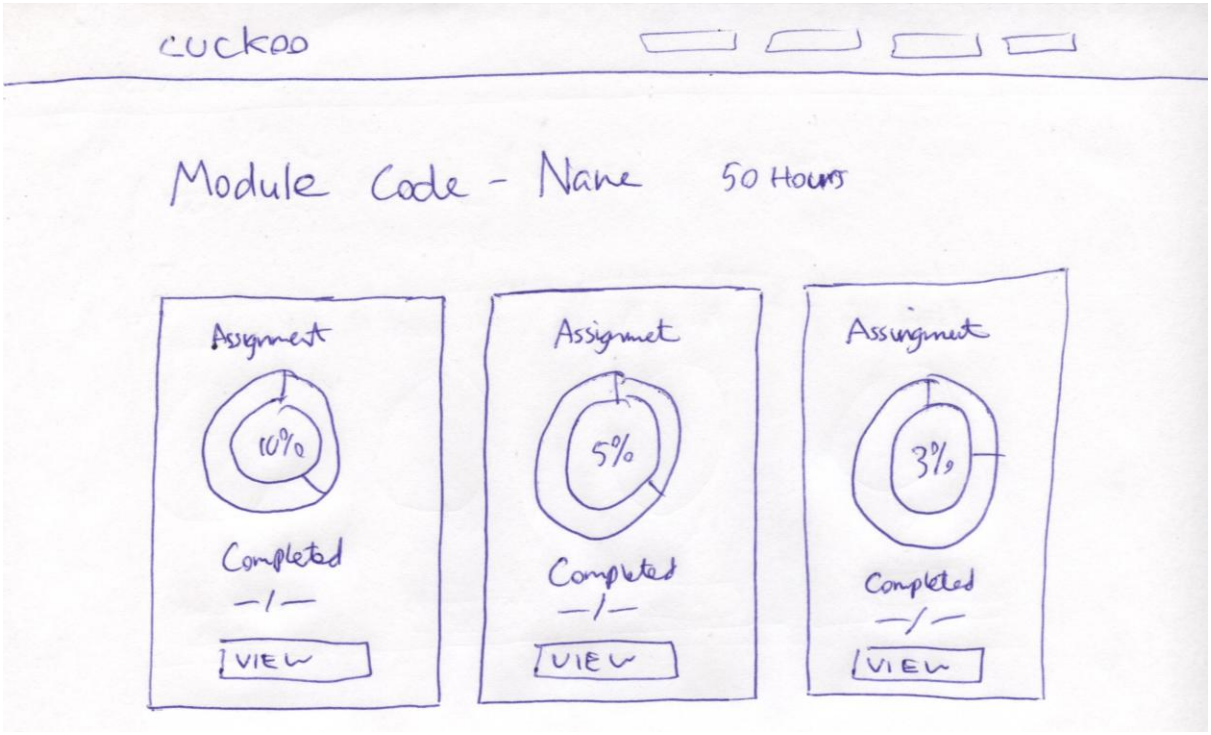


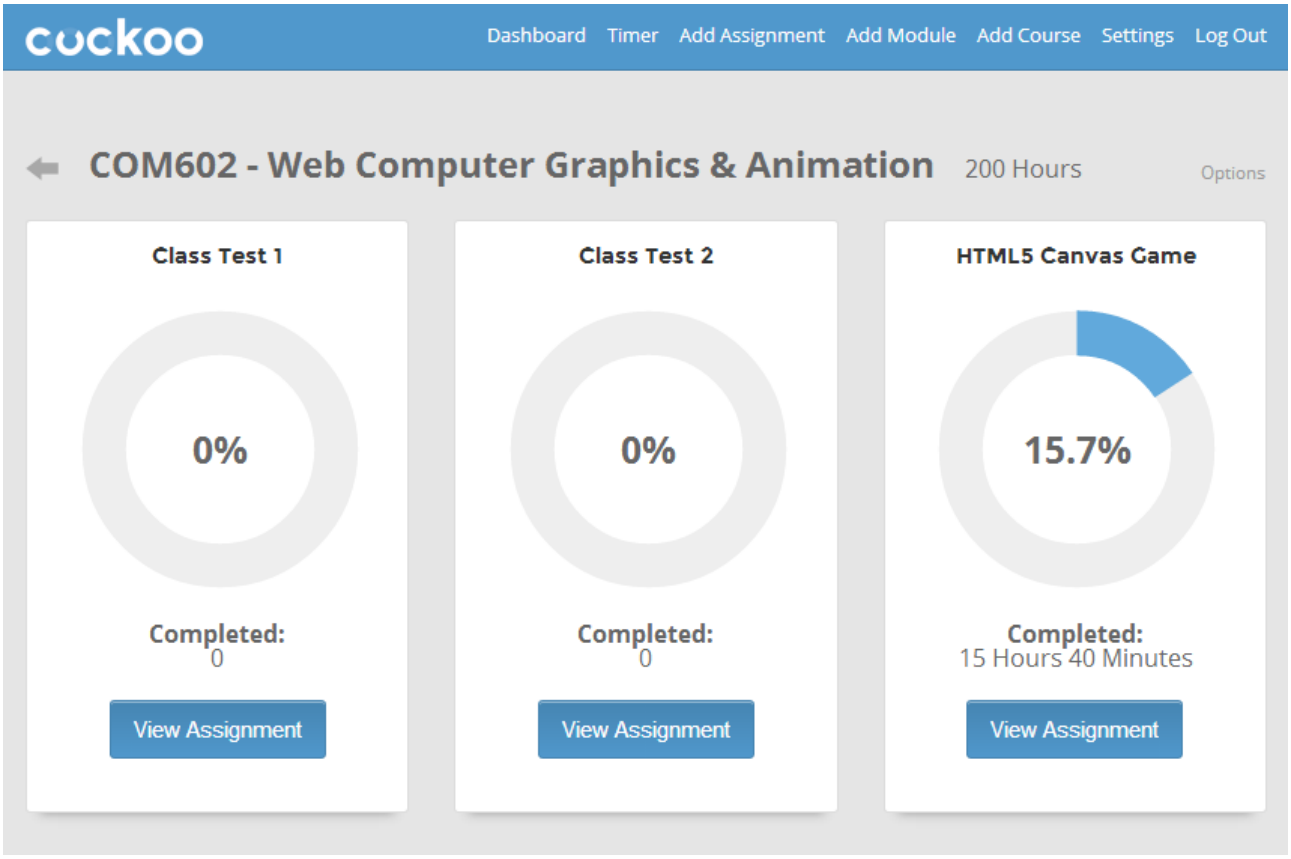
Module Page (Desktop)



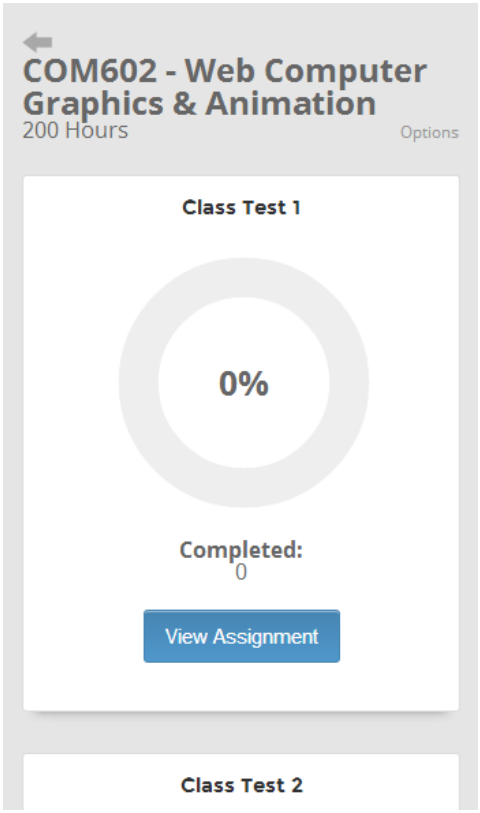


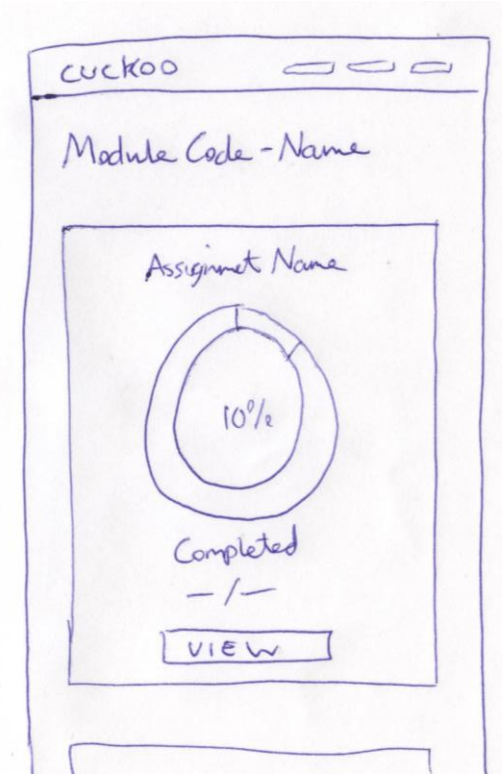
Appendix C.7- Module Page (Tablet)



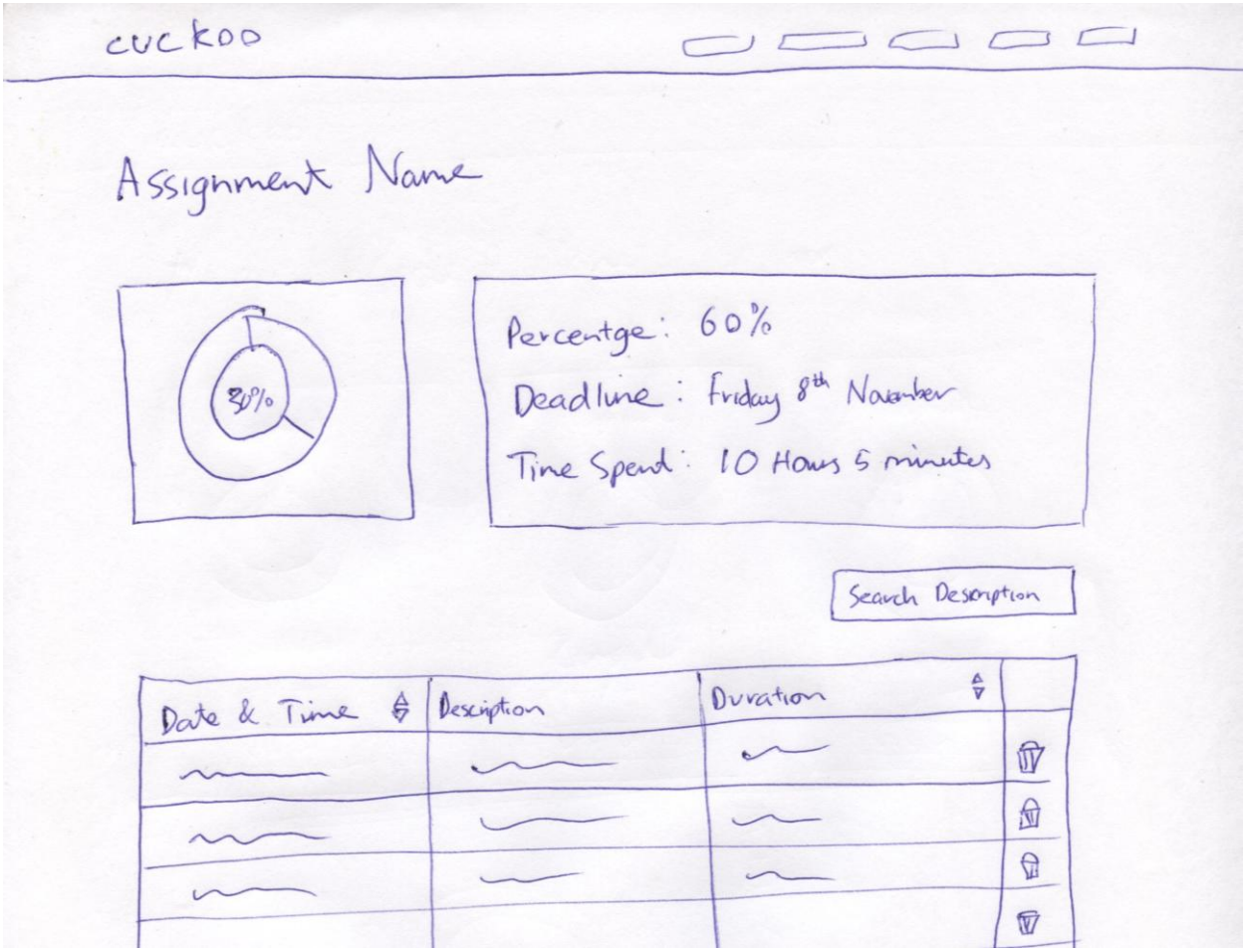


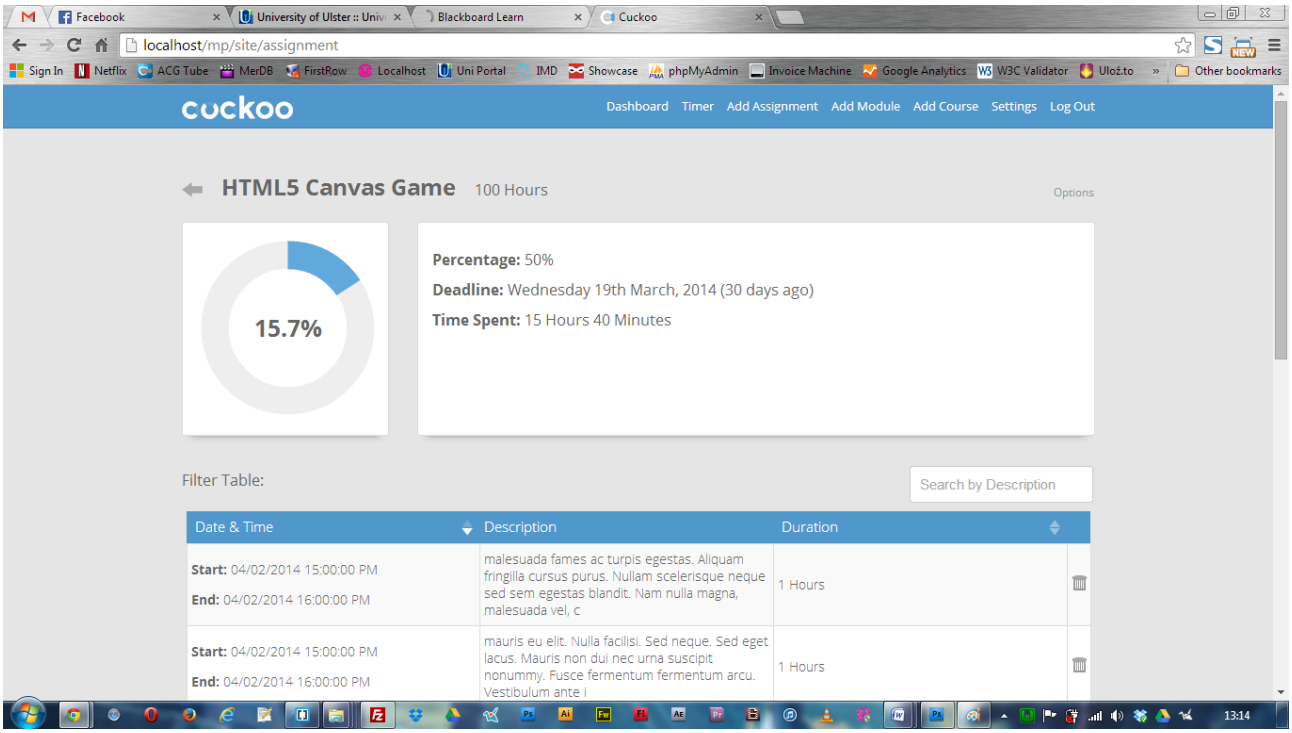
Appendix C.8- Module Page (Mobile)



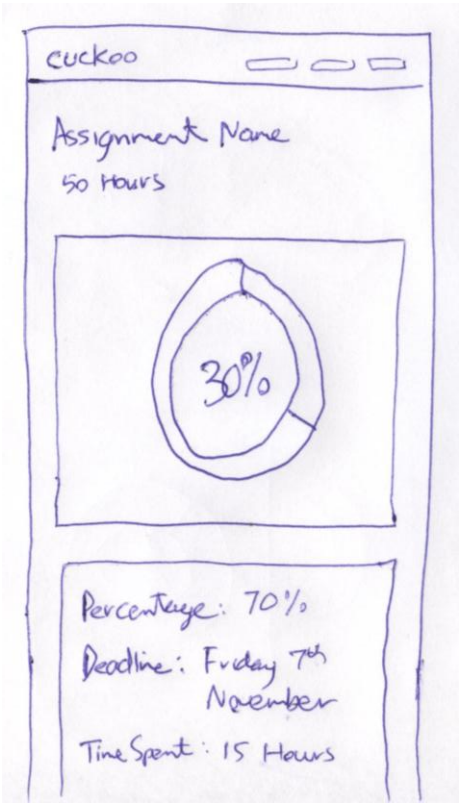


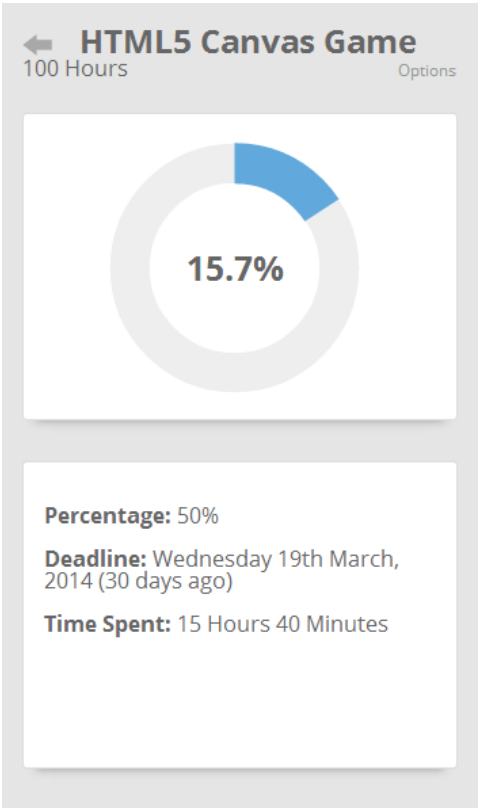
Appendix C.9- Assignment Page (Desktop)





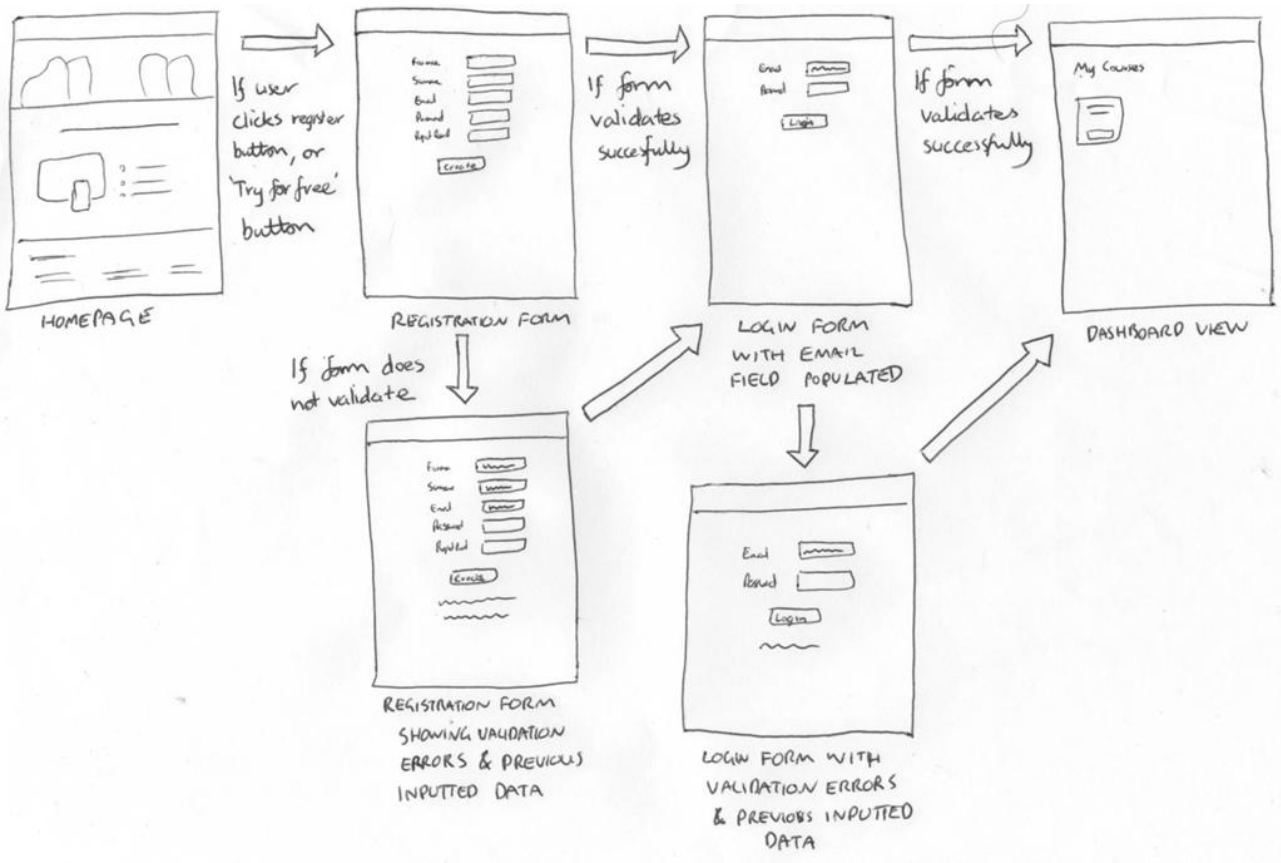
Appendix C.10- Assignment Page (Mobile)



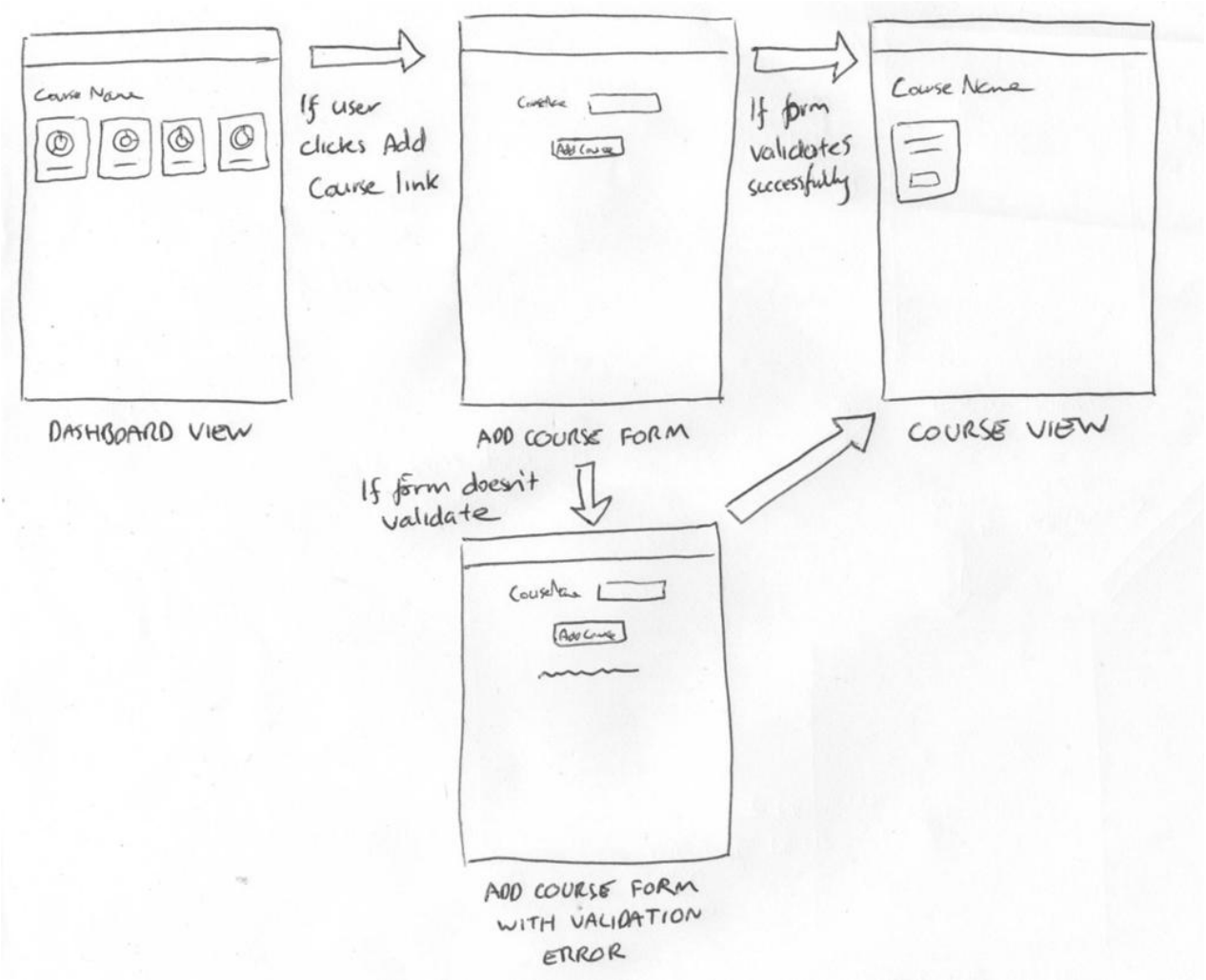


Appendix D- User Journeys

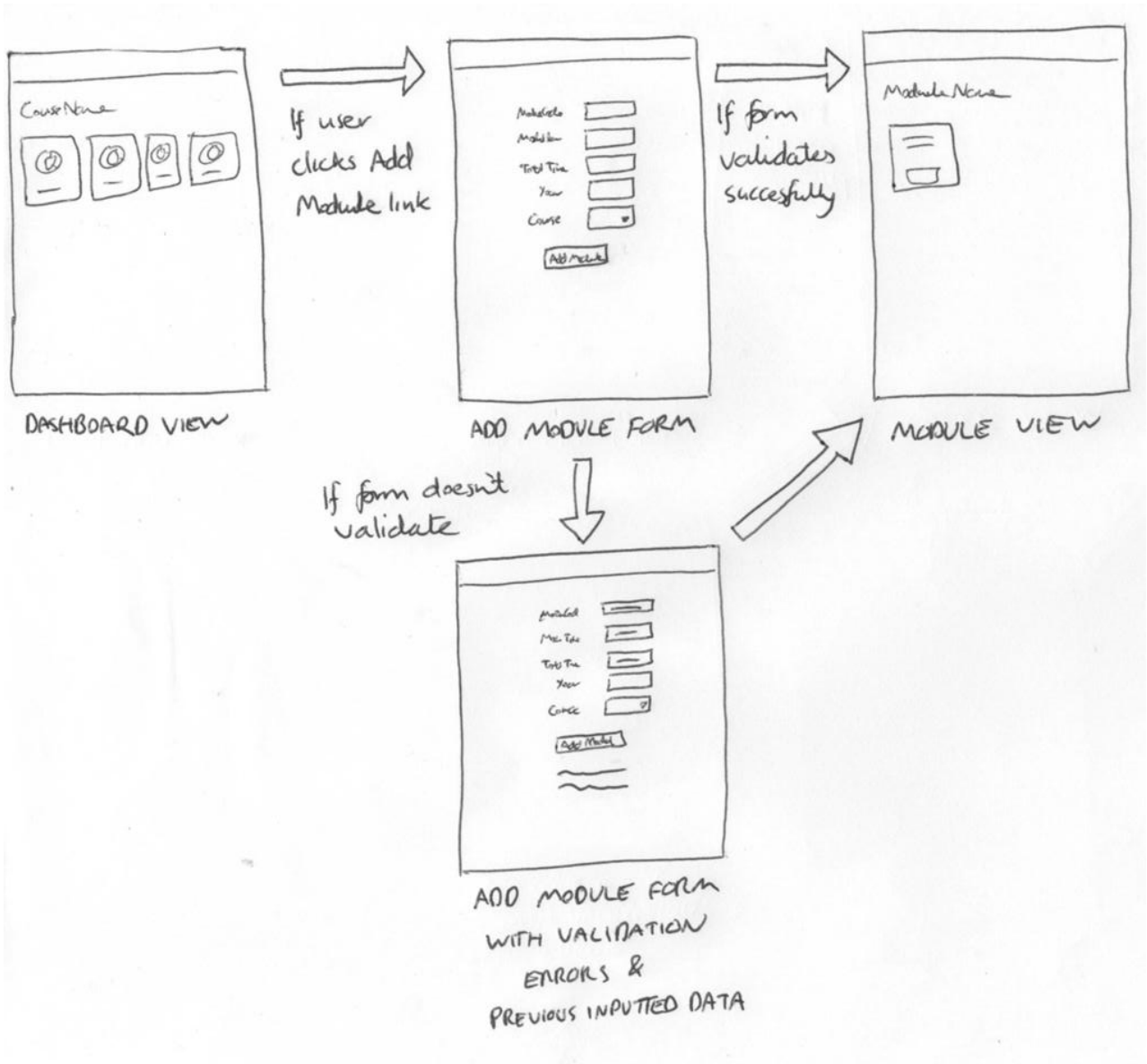
Appendix D.1- Registration Form



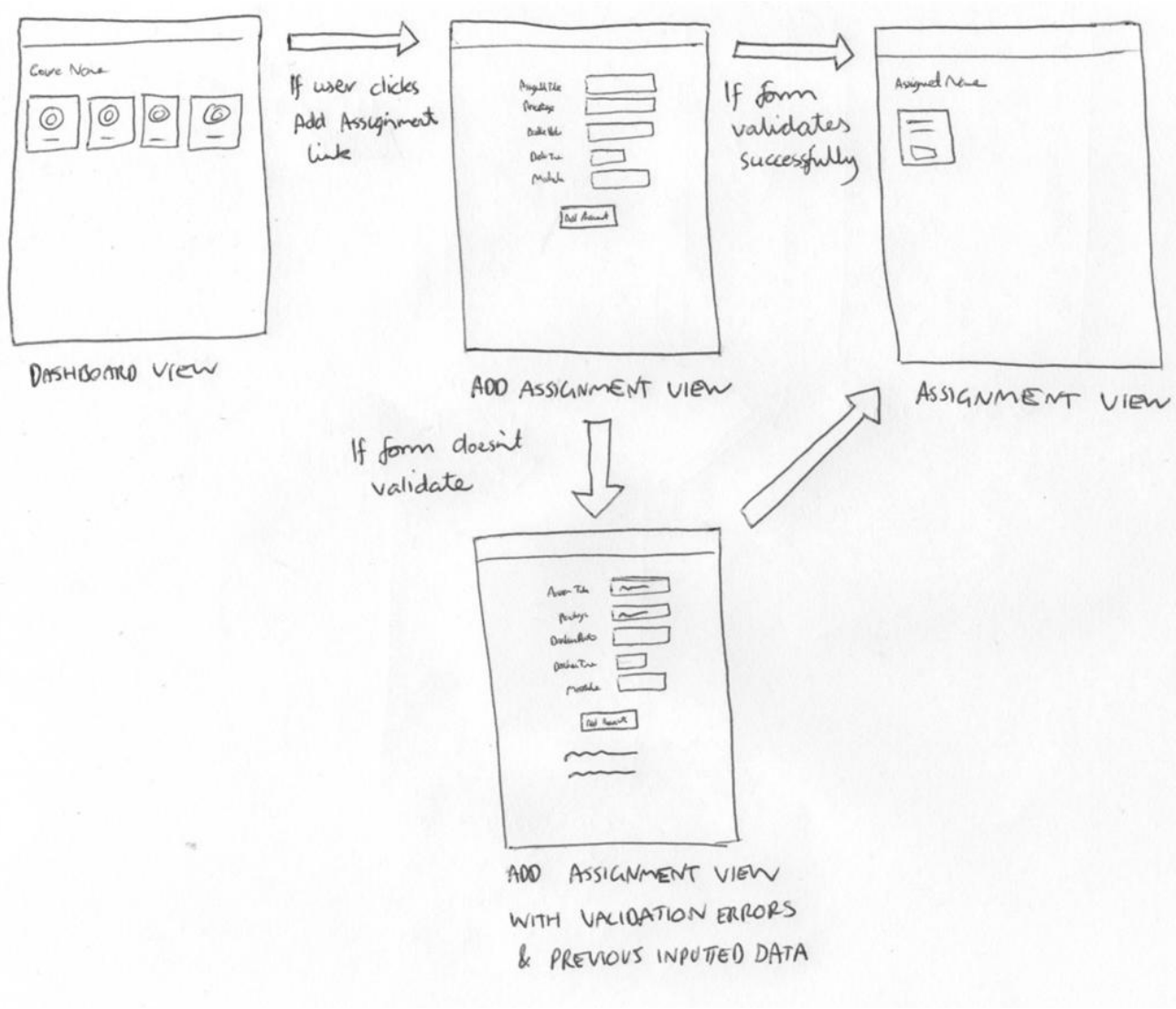
Appendix D.2- Add Course



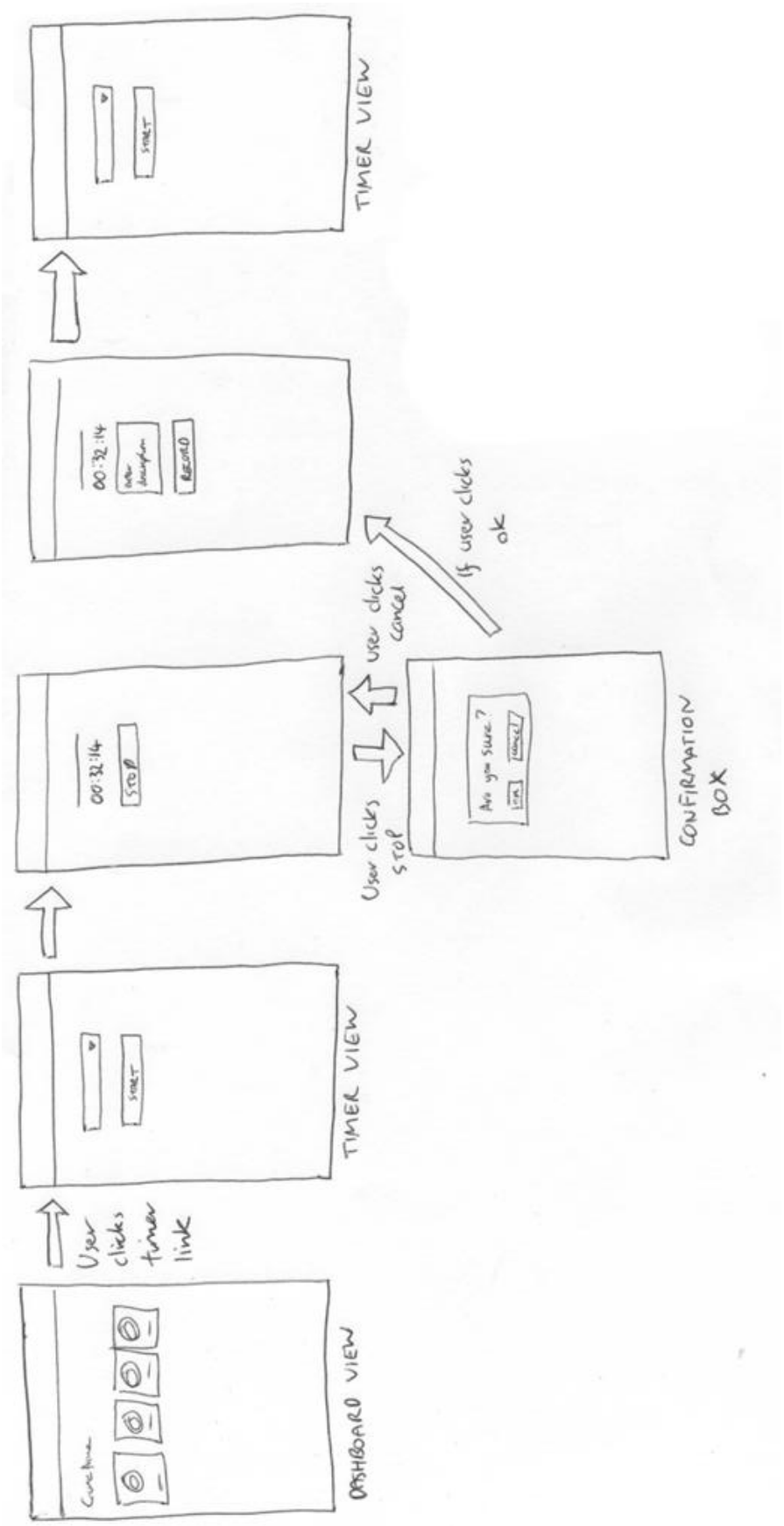
Appendix D.3- Add Module



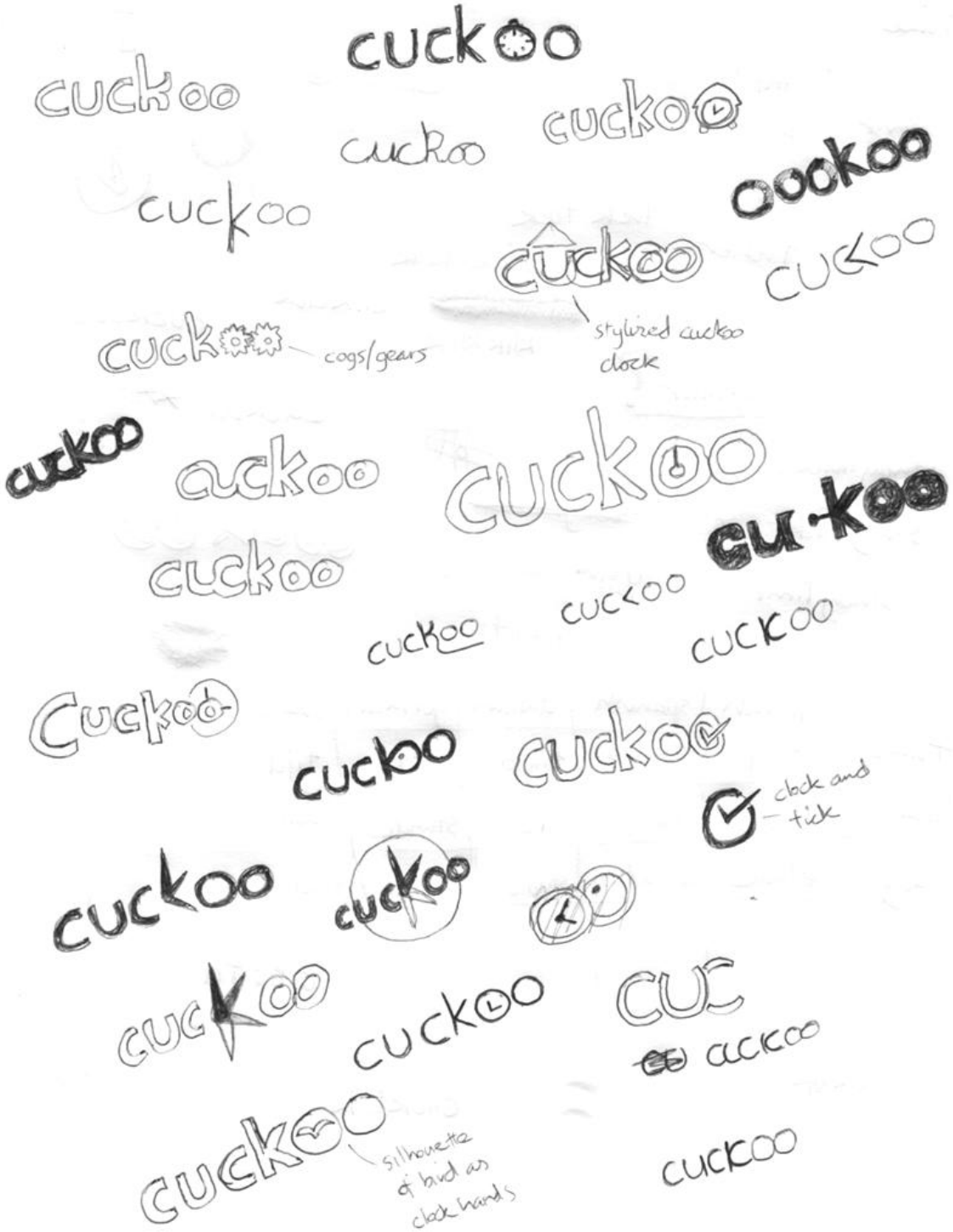
Appendix D.4- Add Assignment



Appendix D.5- Add work session



Appendix E- Logo Experimentation




cuckoo
cuckoo
cuckoo
cuckoo
cuckoo
cuckoo
cuckoo
cuckoo

Appendix F- Atomic Requirements

Only high priority requirements have been included to save space.

Appendix F.1- Functional Requirements

Requirement ID: #01 **Type:** Functional **Use Case ID:** #

Description: Create Entity Relationship diagram for the system.

Rationale: In order to create an effective and efficient relational database, it is important to ensure the relationships between the entities involved in the system are fully understood.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Either a paper based or digital based ER diagram has been produced.

Priority: High **Dependencies:** **Conflicts:** None Identified

Requirement ID: #02 **Type:** Functional **Use Case ID:** #

Description: Create a database with the required tables, columns and relationship

Rationale: A database is fundamental to the system as it will store all the information relating to a user's account.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: A MySQL database using PHP MyAdmin has been created.

Priority: High **Dependencies:** #01 **Conflicts:** None Identified

Requirement ID: #03 **Type:** Functional **Use Case ID:** #

Description: Create PHP scripting needed to add user account in the database.

Rationale: The system needs to handle multiple user account for individual students

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Use interface to register account using the email clarke-t6@email.ulster.ac.uk.

Priority: High	**Dependencies:** #01, #02	**Conflicts:** None Identified

Requirement ID: #04 **Type:** Functional **Use Case ID:** #

Description: Create PHP scripting needed to display user account details from database.

Rationale: In order to edit their account settings, the user will first have to view the current settings.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: View user account settings created in #3 on settings page.

Priority: High	**Dependencies:** #01, #02, #03	**Conflicts:** None Identified

Requirement ID: #05 **Type:** Functional **Use Case ID:** #

Description: PHP scripting needed to update user settings in database.

Rationale: The user may want to change their account email address, work hours or work days.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Use settings page form to change user account work hours and work days.

Priority: High	**Dependencies:** #01, #02, #03, #04	**Conflicts:** None Identified

Requirement ID: #06 **Type:** Functional **Use Case ID:** #

Description: PHP scripting needed to add course to the current user account.

Rationale: The site will organise work done in terms of courses, modules and assignments.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Use Add Course form to insert Interactive Multimedia Design as course into database.

Priority: High	**Dependencies:** #01, #02	**Conflicts:** None Identified.

Requirement ID: #07 **Type:** Functional **Use Case ID:** #

Description: PHP scripting needed to display course details from database.

Rationale: The site will organise work done in terms of courses, modules and assignments.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Use the site interface to view course details entered in #6

Priority: High **Dependencies:** #01, #02, #06 **Conflicts:** None Identified.

Requirement ID: #08 **Type:** Functional **Use Case ID:** #

Description: PHP scripting needed to update course details in database.

Rationale: The user may want to change the display name for a course.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Use interface to edit the name of the course added in #6 to IMD. Check the data has been updated using PHPMyAdmin.

Priority: High **Dependencies:** #01, #02, #6, #7 **Conflicts:** None Identified

Requirement ID: #09 **Type:** Functional **Use Case ID:** #

Description: PHP scripting needed to delete a course entry from database.

Rationale: A user may start a new course and new longer need previous ones.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Use the interface to delete the course added in #6

Priority: High **Dependencies:** #01, #02, #06, #07 **Conflicts:** None Identified

Requirement ID: #10	**Type:** Functional	**Use Case ID:** #
Description: PHP scripting needed to added a module to a course.		
Rationale: The site will organise work done in terms of courses, modules and assignments.		
Originator: Thomas Clarke – Designer/Developer		
Fit Criterion: Use interface form to add COM601 to course added in #6.		
Priority: High	**Dependencies:** #01, #02, #06	**Conflicts:** None Identified

Requirement ID: #11	**Type:** Functional	**Use Case ID:** #
Description: PHP scripting needed to read existing module information from database.		
Rationale: The system needs to display module information at various points throughout the sit.		
Originator: Thomas Clarke – Designer/Developer		
Fit Criterion: Use interface to view information related to the module added in #10.		
Priority: High	**Dependencies:** #01, #02, #06, #10	**Conflicts:** None Identified

Requirement ID: #12	**Type:** Functional	**Use Case ID:** #
Description: PHP scripting needed to update existing module details in database.		
Rationale: Changes may be made to a module, which the user would have to update on the system.		
Originator: Thomas Clarke – Designer/Developer		
Fit Criterion: Use interface form to update information for COM601 module added in #10.		
Priority: High	**Dependencies:** #01, #02, #06, #10, #11	**Conflicts:** None Identified

Requirement ID: #13	**Type:** Functional	**Use Case ID:** #

Description: PHP scripting needed to delete a module from the database.

Rationale: User may no longer be taking a module and needs to delete it.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Use interface to delete COM601 module added in #10.

Priority: High **Dependencies:** #01, #02, #06, #10, #11 **Conflicts:** None identified.

Requirement ID: #14 **Type:** Functional **Use Case ID:** #

Description: PHP scripting needed to add an assignment to a module.

Rationale: The site will organise work done in terms of courses, modules and assignments.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Use interface to add Individual Assignment to module added in #10

Priority: High **Dependencies:** #01, #02, #06, #10 **Conflicts:**

Requirement ID: #15 **Type:** Functional **Use Case ID:** #

Description: PHP scripting needed to read information data from the database.

Rationale: Assignment information will be used at various points throughout the site.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Use interface to view information for Individual Assignment added in #14.

Priority: High **Dependencies:** #01, #02, #06, #10, #14 **Conflicts:**

Requirement ID: #16 **Type:** Functional **Use Case ID:** #

Description: PHP scripting needed to update assignment information in database.

Rationale: The user may have entered incorrect information and need to change it.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Use interface to change name of assignment added #14 to Assignment One.

Priority: High **Dependencies:** #01, #02, #06, #10, #14, #15 **Conflicts:**

Requirement ID: #17 **Type:** Functional **Use Case ID:** #

Description: PHP scripting needed to delete an assignment from the database.

Rationale: The user may have added an assignment accidentally and need to remove it.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Use interface to delete assignment added in #14.

Priority: High **Dependencies:** #01, #02, #06, #10, #14, #15 **Conflicts:**

Requirement ID: #18 **Type:** Functional **Use Case ID:** #

Description: PHP scripting needed to add an work session to an assignment.

Rationale: The time tracked using the online timer will need to be recorded

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Use interface timer to record a work session for assignment added in #14

Priority: High **Dependencies:** #01, #02, #06, #10, #14 **Conflicts:**

Requirement ID: #19 **Type:** Functional **Use Case ID:** #

Description: PHP scripting needed to read work session information from database.

Rationale: The user will have to be able to view previously recorded sessions.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Use interface to view work session recorded in #18.

Priority: High | **Dependencies:** #01, #02, #06, #10, #14, #18 | **Conflicts:**

Requirement ID: #20 | **Type:** Functional | **Use Case ID:** #

Description: PHP scripting needed to delete a work session from the database.

Rationale: A user may have accidentally recorded a work session, and need to delete it.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Use interface to delete the work session recorded in #18.

Priority: High | **Dependencies:** #01, #02, #06, #10, #14 | **Conflicts:**

Appendix F.2- Non Functional Requirements

Requirement ID: #21 | **Type:** Non Functional Usability – Ease of Use | **Use Case ID:** #

Description: The system must give feedback to the user when an action occurs.

Rationale: It is important that the user is aware of what the system is doing. This helps reassure the user that what they expected to happen is happening. For example when tracking time, the system should display a dynamic timer to show that tracking has started, as well as the current duration.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: A user should be able to tell at a glance how long their current work session has been running.

Priority: High | **Dependencies:** Variable. | **Conflicts:** None Identified.

Requirement ID: #22 **Type:** Non Functional
Usability – Personalization
Use Case ID: #

Description: The system will need to be personal to the current user, showing the modules they are taking, the assignments related to that module etc

Rationale: It is important that the system is personalised to the user as by its very concept the system aims to help users with their personal time management.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: The user account clarke-t6@email.ulster.ac.uk created in #03 will show the COM 601 module added in #10.

Priority: High **Dependencies:** #01, #02, #03, #10 **Conflicts:** None Identified.

Requirement ID: #23 **Type:** Non Functional
Usability – Learning Requirements
Use Case ID: #

Description: Although it may not be possible for the system to be intuitive to the user. The system will need to guide the user through the process of setting up and using their account. Although this will require a small amount of learning, the user will not be overloaded with information they have to retain.

Rationale: It is important that the system is easy to use, as potential users would quickly move away and try an existing competitor's product.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: A user should be able to create an account and add at least one module within 5 minutes of using the system.

Priority: High **Dependencies:** Variable. **Conflicts:** None Identified.

Requirement ID: #24 **Type:** Non Functional
Performance - Capacity
Use Case ID: #

Description: The system needs to be capable of handling thousands of user accounts.

Rationale: If the system is to be a viable solution used by university students, it will have to handle thousands of student accounts.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: This would be basically impossible to test on my own. Limited testing could possibly be carried out by granting access to a beta site to fellow members of the IMD course.

Priority: High **Dependencies:** #01, #02, #03 **Conflicts:** None Identified.

Requirement ID: #25 **Type:** Non Functional
Operational – Interfacing with Adjacent Systems **Use Case ID:** #

Description: The system should have a responsive design, making it accessible and user friendly on any device or platform.

Rationale: Due to the target audience the system is likely to be used on both desktop and mobile devices. Therefore it is crucial that the system is consistent and usable no matter which platform is being used.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Add the module COM 602, Web Computer Graphics & Animation to the system using a mobile device.

Priority: High **Dependencies:** Variable, hard to define. **Conflicts:**
None Identified.

Requirement ID: #26 **Type:** Non Functional
Security - Access **Use Case ID:** #

Description: Only users with a valid username and password can access an account.

Rationale: As the information stored in the user accounts is personal to that user it is important to ensure it secure and only accessible by them.

Originator: Thomas Clarke – Designer/Developer

Fit Criterion: Try accessing an account with incorrect login details, form validation should fail.

Priority: High **Dependencies:** #01, #02, #03, #04 **Conflicts:** None Identified.

Appendix G- Feasibility Testing Discussions

Table 2.1 Potential Risk	Methods of Avoidance	Remedial Action
Access to image/graphics editing software needed for creating the visual mock ups and graphical content of the site, is essential.	The designer could run personal copies of Adobe Photoshop and Adobe Illustrator on their local machine.	If personal copies of these programs were not accessible, it would be possible to use the facilities and software provided by the university to complete the work.
Access to a text editor or web development application is essential for developing the various pages and scripts needed for the site.	The developer could download and run personal copies of a code editor such as Brackets or Notepad ++ on their local machine.	
Access to a virtual server to allow local development of PHP and MySQL files is essential.	The developer could download and run WAMP Server on their personal laptop machine.	
University server settings may not offer the required functionality required for the finished site to run properly.	Research the functionality offered by the university servers, and avoid where possible using technologies or techniques which conflict with this. Consult with lecturers, peer mentor or technical staff about university server capabilities . Use past experience to inform what technologies are supported on the servers.	Hosting could be acquired via a commercial hosting company. This would ensure all the functionality needed to run the site is available. The developer has past experience using the hosting services of Big Wet Fish, therefore they would be an ideal alternative to the university server should the need arise.

Table 2.2 Potential Risk	Methods of Avoidance	Remedial Action
The development files may have been lost or corrupted due to issues with the developer's machine. This could include hard drive or CPU problems.	Backup files regularly on cloud storage services such as Dropbox, Google Drive or Copy. Creating back up on another physical machine or external hard drive, to further reduce risks.	Restore or replace the corrupted/damaged hardware. Recover the files from the most up to date back up available.

Table 2.3 Potential Risk	Methods of Avoidance	Remedial Action
Difficulties may arise regarding front end development, such as using jQuery to create the dynamic and interactive features needed.	Studying COM601 material such as lecture notes and practical tutorials. Reading books such as Head First jQuery.	Discuss issues with experts such as module coordinator Peter Nicholl, or other teaching staff with expertise. Get in touch with placement colleagues for their advice or

Difficulties may arise regarding development when using AJAX and JSON to make the client and server interact asynchronously as needed.		experience with the situation. Discuss the issue on online forums such as StackOverflow, granting access to the combined knowledge and experience of many experts.

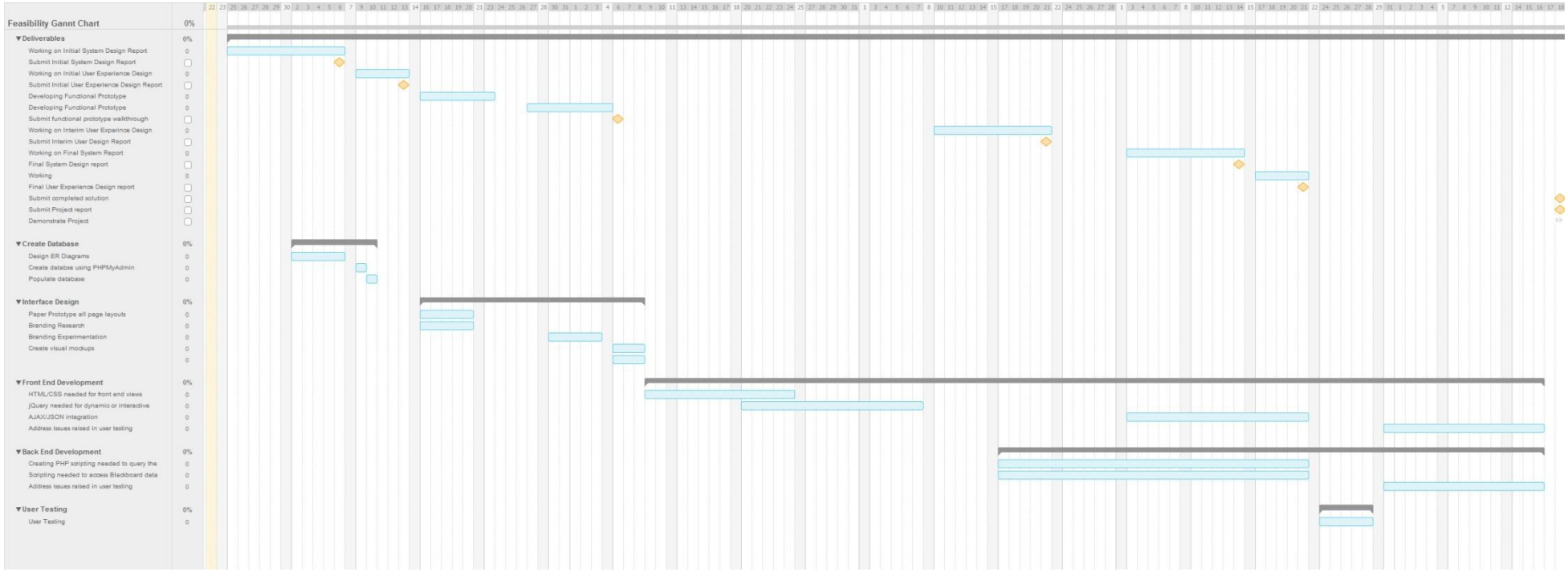
Table 2.4

Potential Risk	Methods of Avoidance	Remedial Action
Problems in regards to the designing the database in regards to the entities and relationships needed.	Study and reflect on the teaching material received in Year 2 COM427 regarding database design and server side scripting using MySQL and PHP. Attend Programming Help classes scheduled throughout the semester.	Discuss the issues with experts such as Dr Ian Young, Giuseppe Trombino or other experience teaching staff. Make use of online forums such as StackOverflow to discuss the issue other professionals and experts.
Difficulties may arise during the creation of the database, tables and relationships needed for the system to function.		
Problems may arise with the back end scripting needed to query the database and perform other procedures.		

Table 2.5

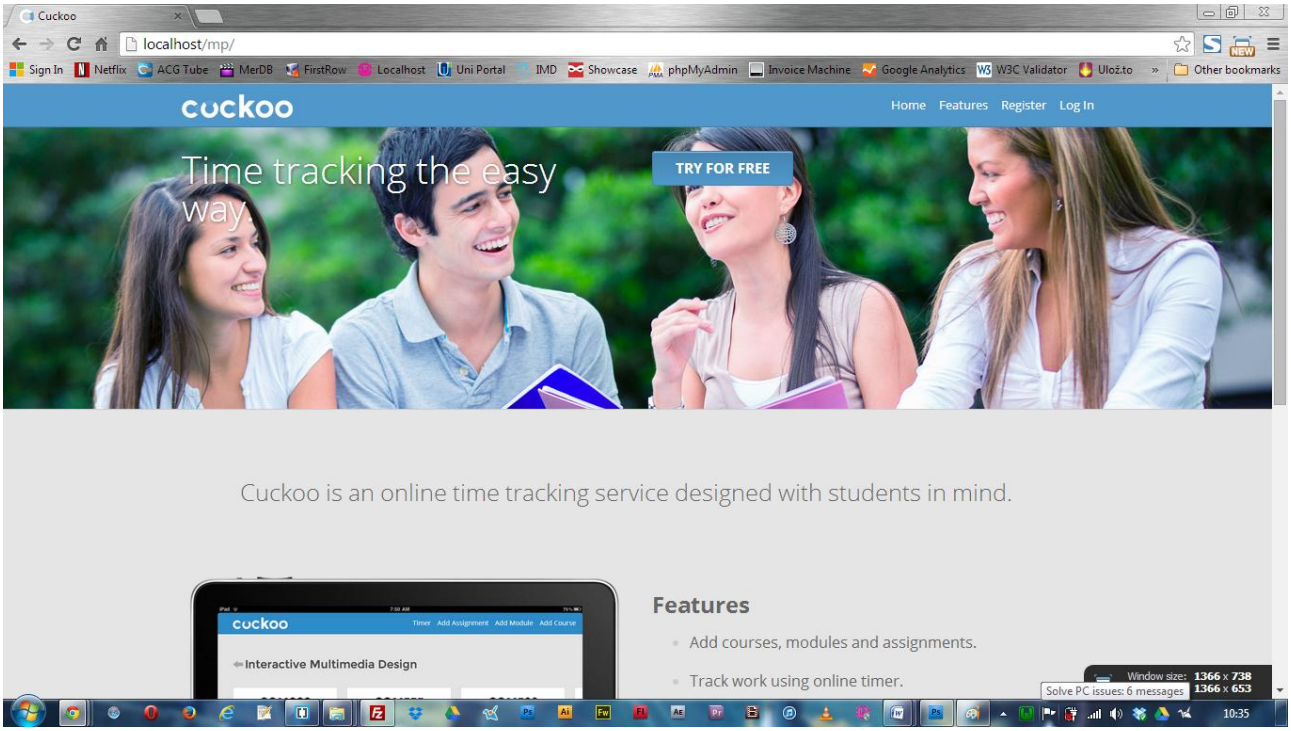
Potential Risk	Methods of Avoidance	Remedial Action
Front end development may drag on and delayed dependant stages.	Creating and working to a detailed schedule for each stage of the project's design, development, testing and implementation. Meeting regularly with module staff, peer support group and personal mentor to discuss the progress of the project.	If the particular area causing the project schedule to be delayed is non-essential or has no other dependant features, it may have to postponed until the end or discarded completely.
Back end development may be drag on and delay dependant stages.		
Populating the database with data may take longer than anticipated, and delay the development and testing of the system.		
Testing the system could take longer than anticipated. Thus reducing the time needed to address the issues found.		
All of the above risks could be caused by time being spent on other university modules, or other personal and social commitments.		

Appendix H- Schedule Gantt Chart

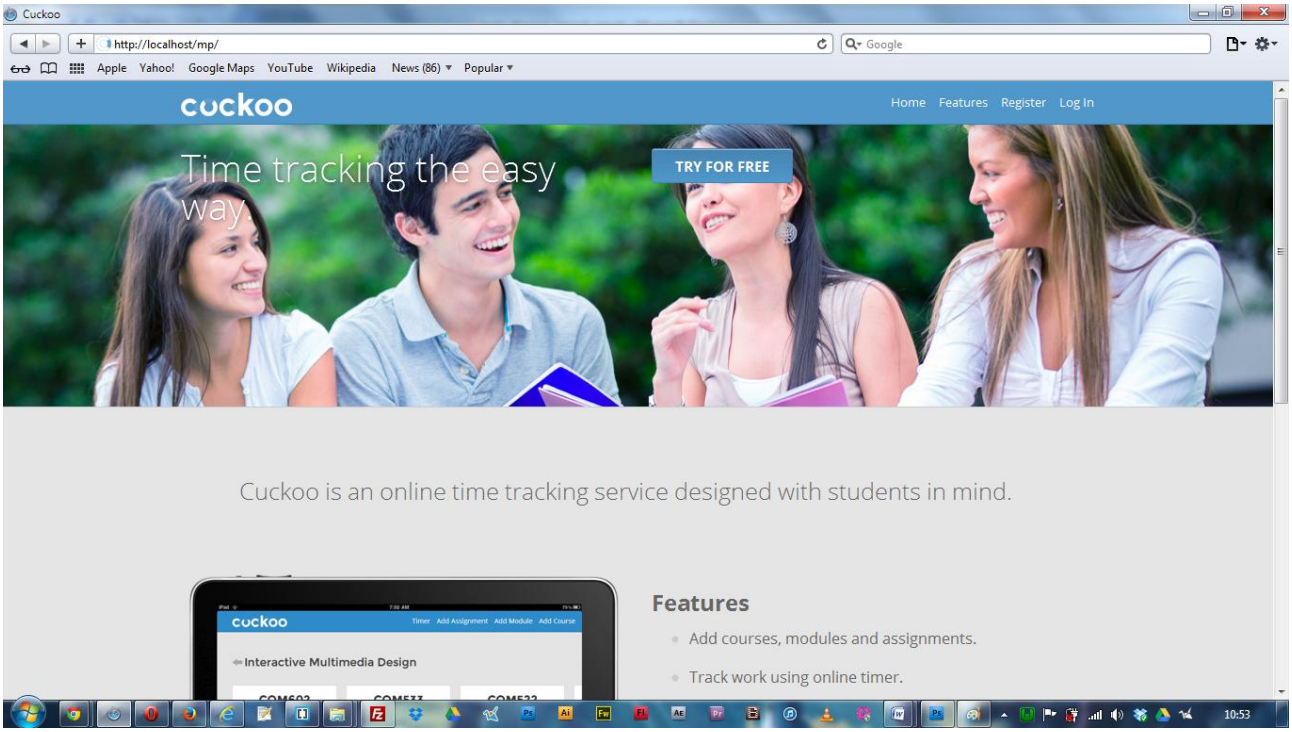


Appendix I- Browser Testing Screenshots

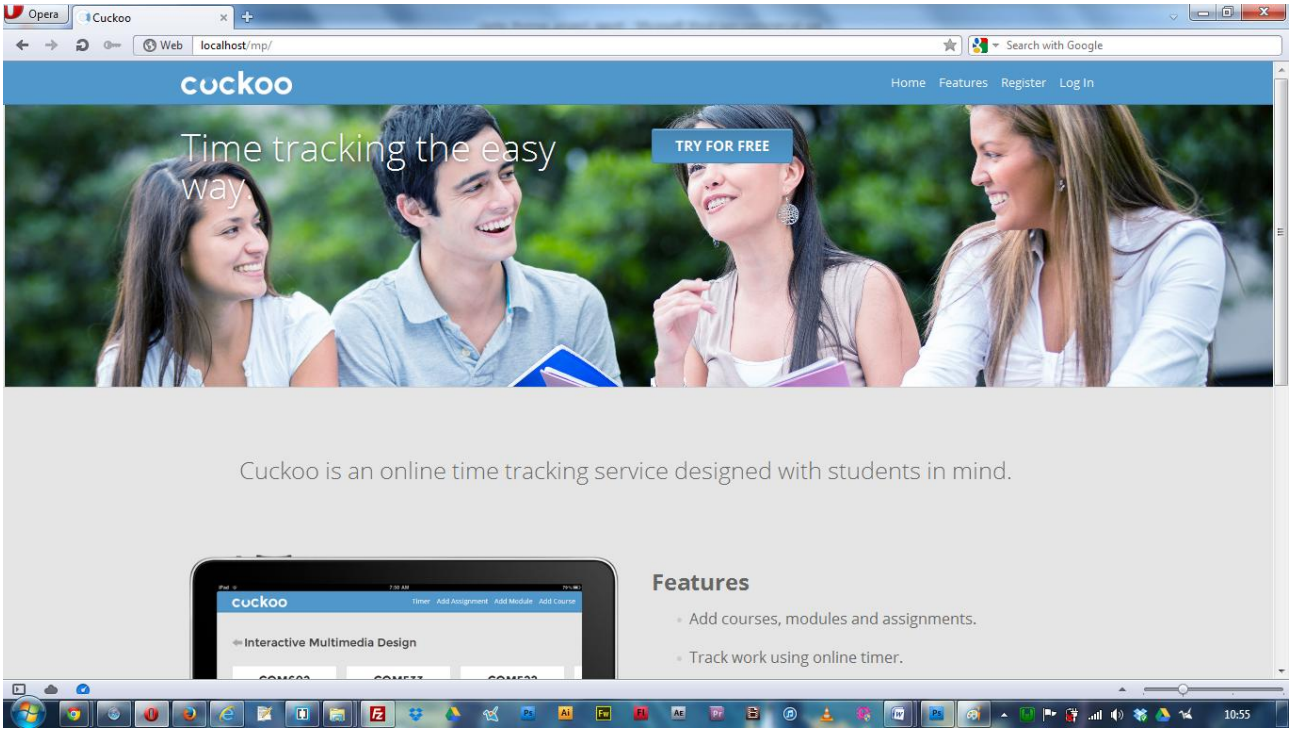
Google Chrome



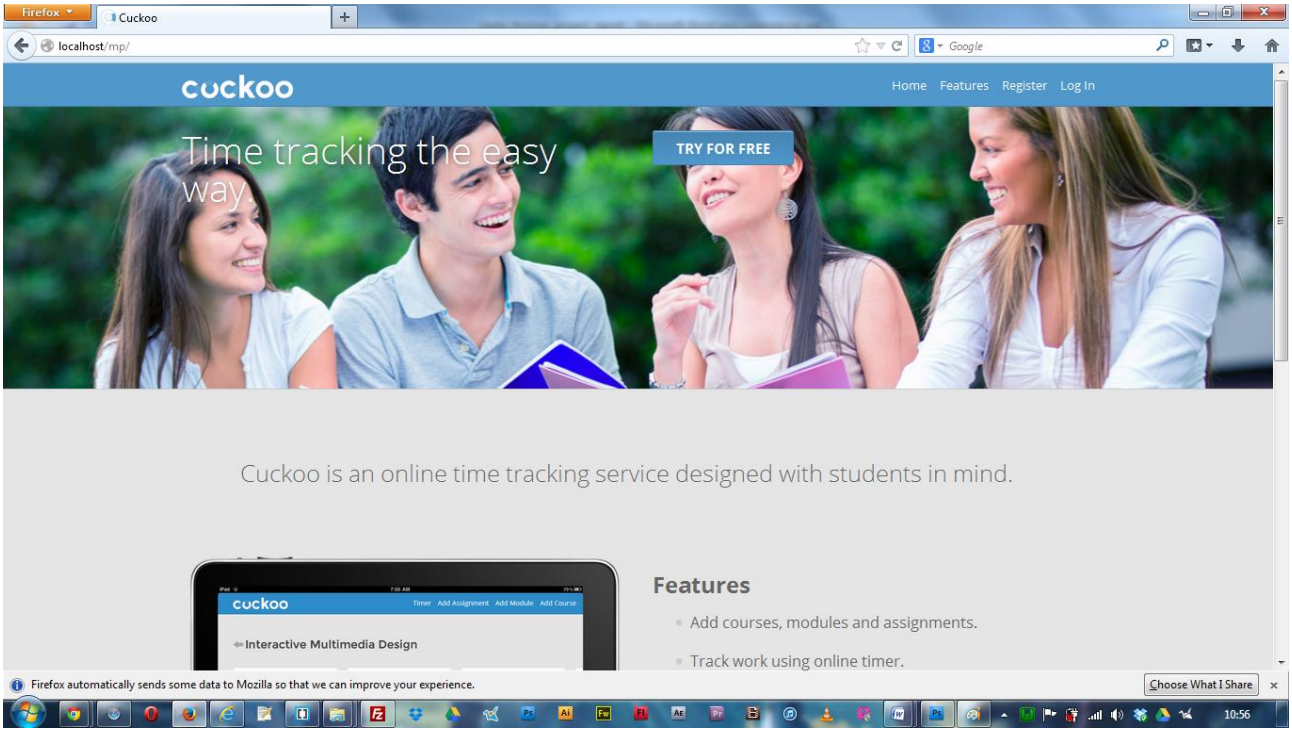
Appendix I.1- Safari



Appendix I.2- Opera



Appendix I.3- FireFox



Appendix I.4- Internet Explorer