Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2016

Project Title:	**A Novel People-aware Assessment Management System**
Student:	**C.Y. LEOW**
CID:	**00730596**
Course:	**EIE4**
Project Supervisor:	**Dr Thomas J. W. Clarke**
Second Marker:	**Dr Christos Papavassiliou**

ABSTRACT

Assessment workflow management is a significant aspect of any educational system, and the administrative aspects, for both students and staff, can be tiresome and repetitive. While traditional Virtual Learning Environments (VLEs), such as BlackBoard, are powerful tools and can provide a significant range of functionality, there is interest within the Electrical & Electronic Engineering department at Imperial College for a modern, simpler solution. This project aims to demonstrate the viability of such a system and make explicit the underlying principles with which it should be built. Through analysis of existing systems, user interview and the identification of design patterns and engineering principles that encourage usability, a proof of concept demonstration platform was developed. In addition to providing demonstrations of functionality, the platform highlights the importance of usability and trust fostered by a logical and intuitive design. The platform is additionally enhanced with the inclusion of external data interfaces that provide configuration based integration with existing data sources and systems.

ACKNOWLEDGEMENTS

My first thank you goes to my project supervisor, Dr Clarke. Without his support and guidance throughout the project, I would have been far more stressed and lost. His interest in the project and subsequent thoughts were always helpful and generated significant insight throughout. Dr Thomas, Dr Perea, Dr Papavassiliou and Dr Tate must also be thanked for taking the time to be interviewed during the requirement capture stage. Similarly, members of the EIE4 year group and friends who tested my application must be thanked for their comments, advice and support. Finally, Xiao Xiangming and Vic Boddy deserve a special thank you for their help provisioning and locating a development server in the department.

TABLE OF CONTENTS

1 INTRODUCTION

Assessment management is a key aspect of any education institute and a digital solution is necessary in the 21st century. Any system should facilitate communication between students and educators, manage workflows effectively and, most importantly, be intuitive and easy to use.

A further requirement of such a system is the adoption of modern technologies and application design. Web browsers are more powerful than ever and website functionality is being expanded at a constant rate. With this added power, websites are being transformed into single page applications (SPA) that provide a smoother and unified user experience (Schneider, 2016).

The Electrical & Electronic Engineering (EEE) department at Imperial College London is interested in the development of an internally developed assessment management platform. This platform, nicknamed FACTS (Feedback and Assessment Continuous Tracking System) (Clarke, 2015), and designed in a principled manner, will address the issues faced by existing assessment management implementations.

This project encompassed the development of a proof of concept assessment management application in order to demonstrate the viability of such a system. Following analysis of existing assessment management solutions and identification of guiding design concepts, improvements were developed in a principled manner. Subsequent testing was then used to verify these improvements and validate their effectiveness. Finally, the underlying design principles and their justifications were detailed for future reference.

The content of this report details the progress of the project during application implementation. Initial user requirements are established in the Requirement Capture chapter and the underlying design guidelines codified in the Analysis & Design chapter. The Implementation chapter ties the specific implementation details back to the design guidelines and highlights the salient application features. Validation of design principles and verification of the developed functionality is detailed and evaluated in the Testing chapter, with a subsequent holistic analysis of the project in the Evaluation chapter.

2 BACKGROUND

While Imperial College has a full assessment management system in Blackboard, there are difficulties and quirks that can cause confusion, increase workload and mislead students. The Electrical & Electronic Engineering Department is interested in developing a modern solution that will allow easy and intuitive management of the various assessment workflows. To aid development, existing systems were investigated and analysed. These results are presented below and several high level conclusions are drawn.

2.1 EXISTING IMPLEMENTATION

BlackBoard software is used globally across 20,000 organizations (Empson, 2014), and it is a large and diverse piece of software. Unfortunately, this diversity and complexity does not seem to have been handled in a structured manner and has led to difficulties completing administrative tasks. Anecdotal evidence from staff at Imperial indicates that the complex navigation structure and multiple views for the same role make it difficult to use the system without a significant learning curve. In addition, the bulk management capabilities of the platform are limited to a specific program that must be downloaded. This limits the effectiveness of a cloud platform and increases the administrative workload for staff.

Moreover, students complain about the use of user guides or tricky navigation to complete simple tasks, such as uploading coursework or viewing grades. An example of this tricky navigation that students must regularly use to view pages is the concept of 'course jumping'. To access a view that exists, but is not linked to in a course, the view must be accessed in another course, and then 'jumped to' through a drop down menu. The circuitous route to access desired pages is a burden to students and administrators, and should not be a design feature of a critical piece of software.

The lessons learnt from BlackBoard are twofold: 1) the navigation hierarchy must be well defined and offer clear views for each role; and 2) platform services should be integrated into the browser or existing software, not pushed to a proprietary program.

2.2 OTHER IMPLEMENTATIONS

Several other systems were examined during the research; however, none of these was identified as a close enough fit to the requirements listed and as such, these were only used for functionality inspiration and as guidelines for avoiding pitfalls.

2.2.1 MOODLE

Moodle is powered by one of the largest open-source teams in the world (Moodle Org, 2002). It provides a highly configurable and free VLE that can provide all of the functionality that

BlackBoard does. It also provides a student centric experience, unlike BlackBoard, which is instructor focused (Lambda Solutions, n.d.). While Moodle has too much functionality to offer – it offers a fully integrated VLE – and would require too much effort to customize and implement, it does have several aspects that can be reflected in the departmental VLE. The idea of enrolment methods, and in particular the 'Cohort Sync' functionality, is a powerful functionality (Moodle Org, 2002). Instructors can set up their course to enrol a cohort, or group of students, such that any updates to the cohort will be reflected in the individual student's enrolment in the course. This functionality can be easily used to synchronize with the departmental database, or even replace the departmental option selection service in the future.

Another desirable feature that Moodle has implemented is their bulk user upload with column naming to provide customizability. User information that is being uploaded has to have a column definition row, which has led to a system that can accept any data through configuration, not code, changes (Moodle Org, 2002).

2.2.2 Geenio

Geenio is an alternative VLE that operates on the cloud in a Software as a Service (SaaS) model. This means that it is a paid service for large installations, but benefits from increased support and a newer user interface (Geenio, n.d.). Apart from the cost of the service, Geenio is not an ideal as it is a solution focused at businesses, aimed at purely virtual learning environments and not used to support classroom based learning. In a similar vein to Moodle, it also provides a significant amount of functionality, such as course planning, test creation and automatic marking, which would not be used by this project. The inspiration taken from Geenio is the evidence that a browser only, HTML5 approach is a positive direction for VLEs to move in (Business Wire, 2014).

2.2.3 Kornukopia

This last VLE examined is similar to Geenio as it is a cloud based service, but is different because it is completely free. Kornukopia is an education sector focused VLE, with integrated functionality for managing general student information (Kornukopia, n.d.). It offers a similar level of functionality to Moodle and Geenio, making it unideal for this platform. However, a feature of note is the reporting module and framework that allows course owners to generate custom reports. It is a simple yet powerful idea, as it allows for the manipulation of data in a manner that is defined by the user, and not the programmer (Kornukopia, n.d.). Kornukopia only allows basic data manipulation and queries to retrieve information, but a flexible reporting framework could be built into the core platform, allowing for more complex and customized reports to be created.

2.2.4 Conclusions

The overwhelming conclusion that can be drawn from examination of these systems is that these VLEs are over specified for the requirements of this project. Prebuilt systems provide a large amount of functionality, more often than not, at the expense of ease-of-use and intuitiveness in the user interface. In addition, as older web technology is used, most likely for widespread compatibility, the examined systems felt slower and less responsive.

While not ideal for the project, the above systems did highlight useful functionalities and design patterns that were useful in the specification of the project.

3 REQUIREMENTS CAPTURE

This project aims to deliver a web interface for the administration and management of assessments within the Electronics & Electrical Engineering department. In a general sense, the platform will allow the interested parties to interact with assessments in a managed workflow that aims to reduce administration costs, shorten feedback time and increase communication between students and course administrators. The requirement capture for this project was done over a two-month period, with input from course owners and the departmental administrators. The list of requirements is detailed below, split into those that are generally applicable and those that are specific technical components.

3.1 General Requirements

In order to make the system as adoptable as possible, and create a highly usable design, there is a requirement for the web interface to be as intuitive as possible. This will be managed in two major ways: firstly, views should be unique and navigation between them done in a clear and deterministic manner; and secondly, design components on the web interface should be reused to maximize familiarity.

Additionally, to assist server administrators and bulk users of the platform more effectively, several factors were considered to help to reduce administration time. Primarily, the user interface should be designed to default to the most common workflow. While all functionality will have reasonable customizability, the use of common defaults should reduce overheads. Furthermore, bulk administration, uploading and processing should be available for all repetitive tasks, further reducing administration costs.

Finally, to ensure effective communication between course owners, markers and students, a robust notification and alert system should be implemented. This system will consist of visual cues and alerts on the web interface itself, and email notifications. Online, a dashboard will feature unfinished tasks, alerts and messages in obvious locations, with links to their location within the system. As well as the online alerts, certain events should generate email communications for more timely notification.

3.2 Specific Technical Requirements

Technical specifications of this project are best discussed with respect to several criteria that led to their selection. Specifically, during the requirement capture, a list of components was made and each was evaluated with respect to four criteria:

- Generalizability, or whether or not the module was created for a general enough purpose to be globally helpful
- Uniformity, which determines whether the component will cause confusion by being too different from the rest of the design.
- Implementation effort, which concerns the amount of effort required to design and implement functionality and a discussion of its worth.
- Graphical user interface (GUI) cost, which identifies the complexity, or negative cost, that the component would add to the GUI.

Most of the components below were influenced by discussions and interviews with the various interested parties.

3.2.1 Feedback/Mark Split

This component would provide the ability for course owners to return marks and feedback at two separate times. In turn, this would allow markers to provide feedback much more quickly and directly than previously.

While this functionality is not directly generalizable, the design paradigms that base it – dividing an element to its base elements and increasing flexibility – are important for future proofing an application. Implementation costs would be low as this was identified early and will be an additional state within a workflow, rather than a new workflow; therefore, this would not strongly affect the uniformity of the application. The only potential cost would be to the GUI, but that can be easily mitigated with clear indication of state and state-change notifications.

3.2.2 Group Administration

Students should be able to create, modify and be marked by groups. Administrators would therefore not have to undertake manual group tracking and would be reassured of group integrity.

This functionality would be quite generalizable as grouping is a basic concept and can be sub-classed to an individual (group size of one). There would be a medium level of implementation effort for this as it only involves adding to a previous workflow (single user submission). Consequently, the adaptation of this workflow could have a high cost to the GUI in terms of complexity. This would be effectively managed and kept uniform by matching the group and single user flow for all steps except for group creation and group signing.

3.2.3 Marker Trade/Swap

A framework could be put in place to allow markers to trade or give away marking (work that they have). This would be course owner or administrator controlled and updated and reflected throughout the student workflow. Hopefully, this would increase marking compliance, as unfinished work could be redistributed more easily.

Trading/transferring of responsibilities or assets is a very general idea, which could be applied to other aspects of the project. The implementation of this would be at a medium level, as it would mean the introduction of a new workflow and the potential alteration of another one. However, in terms of GUI cost, it would be quite low, as it requires a new interface that does not interact with other views.

3.2.4 API Access

This would create the system as an application program interface (API), exposed through representational state transfer (REST) endpoints that could be accessed programmatically by other applications. Fundamentally, this is a strong idea, as it means that other communication between machines could be allowed and customizations would not have to be developed personally.

An API of sorts is a necessity for any single page web application – information must be accessed remotely and then composed at the browser. This means that an API already exists, and as a design decision can be easily composed as a REST API. As the API is being adapted to the same information in a slightly different format, the costs to the GUI are non-existent.

3.2.5 External Data Connection

While an external data connection is a more general requirement, it is still a specific area of technical development. To ensure the continued integration and usefulness of the service, there have to be interfaces to allow information to flow in and out. The best way to design and implement this would be to create an implementable interface for a connection that would allow information exchange.

These connections would be easily customizable because of their plugin like nature, representing connections that can be plugged in, rather than core architectural blocks. Additionally, a plugin system for external interfaces would allow for easy expansion and provide for a fall back if necessary. As this functionality should be quite generalizable, the implementation effort would be quite high, but the benefits gained would be much greater than the costs. Separately, the GUI costs may increase slightly because of the complexity of interfacing a variety of plugin modules, but this should only affect a small subset of users (system administrators) and not the bulk of users.

4 ANALYSIS & DESIGN

To deliver a well-engineered product, the analysis and design stage addressed the ways in which complexity would be handled effectively. Key design and workflow patterns from the requirements stage were identified and followed throughout development to ensure consistency and provide an intuitive user interface. Similarly, complex workflows were analysed and optimized for the most common operations. These aspects were analysed for both the front and back end systems and are detailed below.

4.1 ANALYSIS

As the requirements capture has already highlighted many of the specific challenges that will be encountered by each aspect of the project, this analysis will be accordingly shorter. Analysis of the major individual requirements revealed that the majority of design complexity would revolve around the separation of workflows between the front and back ends. With a loosely coupled user interface and server application, it is necessary to repeat certain tasks in both places, such as validation, and permission data much more stringently. However, explicit and logical choices must be made to clearly separate responsibility for data representation, manipulation and filtering between the front and back ends. This was addressed throughout the design process by designing and working on both aspects in conjunction with each other.

4.2 BACKEND DESIGN

The sole purpose of the backend is to serve information and persist data changes. As defined during the requirements capture, this is handled by a REST API. This choice ultimately simplified the workflow balance issues identified during the analysis stage. By ensuring that API endpoints have small, specific functionalities, workflows are naturally dictated and managed by the front-end interface. Apart from data validation and stage checking, the individual endpoints are as stateless as possible and do not dictate system workflow.

In spite of this fundamental benefit, several challenges did arise during development. Chief among these was the representation and relationship management of complex data. Additional challenges were faced with the design of system security.

Data representation in a REST API is hierarchical by definition. Any request can be encoded in the uniform resource identifier (URI) and HTTP method used. Additionally, the URI path format of URIs encodes the relationship between data in a parent-child manner. While this ensures that data relationships are self-explanatory, it adds complexity to the representation of data in client applications. Instead of receiving a fully decorated object, several requests must be made to recreate an object on the client side. While this ensures that hidden attributes are only exposed to the correct users, it splits the data relationship and manipulation responsibility between the front and back ends and increases the chance of error.

A commonly offered solution to this piece-meal data access problem is to develop a REST API with 'hypermedia as the engine of application state' (HATEOS). This type of architecture informs users of data relationships and availability of actions by dynamically defining and including them in server responses. However, it also requires a significantly higher level of development and only guides the creation of data relationships, and does not remove the responsibility completely.

HATEOS was not used and the eventual solution was to define endpoints that expose fully decorated objects. This meant that the requirement to rebuild objects, and therefore the data relationship complexity, could be removed from the client side. Unfortunately, this also sacrificed the specificity of both content and information visibility that is offered by a fully REST API. As the full object is decorated and exposed with each request, the ability to hide attributes with protected endpoints is lost.

Fortunately, the attribute visibility issue can be mitigated with the concept of encoding views. To allow for varying levels of information exposure, object attributes could be annotated with a view level. This level is part of a hierarchy that allows information exposure to be built up in line with user access levels. Each endpoint within the API can similarly be annotated with a specific view level and therefore expose a different amount of information. The only down side of this approach is the requirement to define multiple endpoints for different views of the same information.

Tied to the definition of these endpoints is the security and access component of the back end. To allow for these various privileged views, the system must support individual endpoint security configuration. This calls for the system to be able to authenticate and authorize a user's access to information.

Authentication is the process that verifies the identity of a user, and can be used to establish a user's permissions. As the application aims to be integrated within existing systems, the authentication interface is designed in a loosely coupled manner. This means that the interface can be implemented to authenticate users and retrieve user details from an external service.

Once authenticated, these details are then used to provide user context and look up authorization permissions. The decision to locate user permissions locally has to do with the nature of external authentication services. External services generally have a much larger user base, many of whom will not need access, and the service will not necessarily have mechanisms to provide custom permission and authorization schemes. With local user permissions, the adaptability of an external authentication system can be coupled with specific endpoint permissions.

A further requirement of the system, to provide scalability and reduce server complexity, is that authentication and authorization should be done in a stateless manner. With a stateless security mechanism, a user session is authenticated exclusively with user provided information. This removes the requirement for any server-based session and supports expansion and rapid growth with scale.

Finally, the backend is publicly available and already authenticated user permissions could be exploited by external parties. The system must be designed to mitigate these traditional security threats and should be developed with cross-site request forgery (CSRF) and SQL injection protection.

4.3 FRONTEND DESIGN

The frontend graphical interface was designed with two major areas of focus: aesthetics and functionality. Aesthetics covers the 'beauty' of the application, including the purely visual choices and information layout. On the other hand, the functionality of the graphical design covers the behaviour of actions within the application.

4.3.1 AESTHETICS

While the functionality of any application defines the basic user experience, the aesthetics are important for users' trust and confidence in the application. A study found that 94% of users mistrusted websites because of design elements, highlighting the importance of an aesthetically pleasing application design (Harris, et al., 2004).

To ensure the application design was aesthetically pleasing, and to avoid unnecessary visual design work, a design framework was used to construct the visual elements throughout the application. Material Design, a relatively new design methodology by Google, was chosen to underpin this application design (Google, 2016). The use of the Angular Material Design framework, emulating the design of Google applications, offers increased recognition and usability from design elements already widely used in existing systems. Additionally, it benefits from a principled design specification and a naturalness that guides and encourages simplified user interaction.

More specifically, Material Design focuses on providing a "visual language" that allows users to interact with applications in a "tactile reality". Lighting and motion provide visual cues, spatial relations and subtle interaction feedback. Examples of this are the 'button ripple' when a user interacts with a button and the use of white-frames, that provide drop shadows to highlight the spatial hierarchy within a page.

Design principles also naturally encourage other design practices important for user interfaces. Separation of concerns, ideal information density and adaptation to device size are all side effects of the design choices outlined in the Material Design specification and implemented in the Angular Material Design framework.

A major feature of this framework is the card-based content architecture. Within the main application context, multiple cards can be organized within a blank space to provide information in a separated, yet cohesive, manner. Using these cards, content must be logically divided to fit into the individual physical spaces. The developer must think carefully about how information is isolated and displayed to ensure a logical context within a card. If done correctly, many textual labels can be removed and data displayed in a simpler manner.

Page spacing also encourages this efficient representation of information and ensures that the application, while uncluttered, is ideally saturated with information. Additionally, to avoid excessive scrolling or loss of context, the majority of content should be displayed within a one-screen view. With comfortable margins and spacing between elements, the Material Design framework assists readability, but forces the developer to consider how information is represented efficiently. These considerations lead to visual cues, such as icons, tool tips and drop down menus that reduce clutter and to the inventive reuse of physical spaces on the page.

Advanced data binding and element visibility control built into the core Angular framework allow the page to be dynamically adapted for different situations. Throughout use, spaces can be reused to reflect changes made and provide status or context for actions. Reusing space is additionally useful as it reduces the actual screen real estate required for an action to be carried out and simultaneously allows invalid actions to be temporarily removed, disabled or overwritten.

In addition to the reuse of physical spaces, the responsive nature of the Angular implementation of Material Design allows information density to be altered based on screen size. Built in controls allow elements within the application to be hidden or minimized on smaller screens. Conversely, more information and detail is automatically expanded and displayed on larger displays. This ability to adapt to various screen dimensions ensures that space is used effectively and allows any invalid functionality to be removed on certain devices.

4.3.2 Functionality

As identified in the requirement capture and analysis stages, the functionality of the application is paramount. The existing VLE solutions are not ideal because of difficult, excessive or poorly designed functionality. To avoid these same mistakes, workflow and layout patterns were examined and followed throughout development. These patterns were chosen to ensure that desired functionality was laid out in a logical manner, and followed to ensure consistency throughout and to allow for an intuitive interface. Many design and workflow patterns were identified and followed throughout the application, but one major theme draws them together: convenience.

While software can be incredibly powerful, it is the convenience that it provides over existing implementations will influence adoption. An example of this is internet banking, where consumers have adopted this convenient trend, despite concerns over security and trust (Lichtenstein & Williamson, 2006). Convenience can be thought of as a digital implementation of common sense. In essence, it is the culmination of minor improvements and functionality that are individually trivial, but together encourage user interaction and adoption.

This focus on convenience has driven the design decisions throughout development of this project and led to three main design patterns that are aimed at easing user interaction. These patterns are default activity, contextualization of information and explicit state.

Default activity is the idea that all functionality and actions have a default state that can encompass the supplementary states without special cases. This concept was highlighted by Linus Torvalds, when he described “good taste” as the ability to “see a problem and rewrite it so that the special case goes away and becomes the normal case” (Torvalds, 2016). Developing with this ideology in mind is very convenient for the user. It provides a default workflow that can eliminate rote work and automate certain bulk actions. Additionally, it ensures that extra functionality is consistently implemented and easily adoptable.

Aside from convenient functionality, the format in which information is displayed can increase the ease of the application. Raw data can be complex and well-designed applications create easily digested representations of that data. However, this can be taken a step further and the information can be contextualized. Instead of requiring the user to understand how information relates to the current state of the system, it is calculated and displayed by the application. A simple example of this is ‘relative time’, or a coarse textual representation of time (‘just now’, ‘5 minutes ago’) as a comparison to a fixed reference point, usually the current time. This contextualization allows users to understand the information quickly and can reduce misinterpretation.

Contextualization does not just apply to the representation of information; it also affects navigation of the application. In general, individual views within the application are well designed with a logical layout, using consistent visual cues and contextual relations to ensure easy comprehension. However, the user’s location within the application and their current workflow are harder to display and must be explicitly highlighted for awareness. This is done by creating a spatial and visual hierarchy.

For navigation within the application, a grid-based spatial hierarchy was established to indicate the user's current location. The horizontal axis is represented by the breadcrumb trail. This commonly found trail of links across the top of the screen indicates the user's current 'depth' within a category and provides easy intra-category movement. Similarly, the vertical axis is represented by the side navigation panel. This panel highlights the currently selected category for easy contextual awareness. These two navigation 'axes' combine to give the user a clear representation of their location and create a comprehensive navigation framework.

This spatial hierarchy is expanded to workflow management along the z-axis. Workflow actions are able to project above the default plane of content and occupy the user's attention completely. This can be coupled with a visual dimming of the lower content, which serves to highlight the raised content and grab the users' attention. In general, these actions are ones that directly alter the current page or the model state and need explicit attention and confirmation.

Finally, the theme of convenience also extends to the user adoption process. Users should be able to adopt the new application without significant relearning of existing workflows and layouts. By identifying and designing to common existing patterns, the user effort required to switch becomes lower and allows for wider adoption.

5 IMPLEMENTATION

The implementation of the application can be discussed with two major focuses. Firstly, the most interesting aspects of the implementation languages and frameworks should be identified. Secondly, the functionality identified in the requirement capture should be highlighted and linked back to the design patterns and engineering principles previously discussed.

5.1 IMPLEMENTATION LANGUAGES & FRAMEWORKS

Although the choice of implementation language and framework was a major component of the interim report, it merits further discussion here. Throughout development, many important language features and specific design patterns emerged or dictated the programming style.

5.1.1 FRONT END

As identified in the requirements capture phase, the application front-end development focused on producing a single page application (SPA) platform. This was achieved with a JavaScript framework, called Angular, that provided powerful, dynamic content manipulation functionality, and the design framework (Material Design), described in the aesthetics design section of this report. Together, these two frameworks focused development on user functionality and usability and reduced basic technical development. Several salient features of these frameworks and interesting developed functionality are highlighted in this section.

The most important functionality provided by the Angular framework is the fundamental design shift to a model-view-controller (MVC) paradigm. Developers are then given the freedom to separate their visual and functional design, resulting in a cleaner and better-encapsulated implementation. This is provided through several core building blocks: directives, filters, services and controllers.

Directives and filters are both graphical elements that translate the data model into a visual representation.

Throughout the developed application, filters were used to refine and clarify data representation. In general, they have been used to contextualize information to make it more easily understood and to extract relevant data elements from longer lists. Specific examples are filters that were built to provide a relative representation of time and others to separate a list of assignments based on their due date.

In a similar, but more powerful, manner to filters, directives are used to provide a refined view of information or to offer specific behaviour. Directives are custom-built HTML elements that provide a developer-defined user experience. These document object model (DOM) elements can be developed both as independent visual elements and as integrations into existing elements. This flexibility encourages code re-use and provides a standardized way that code can be shared. Externally developed directives have been heavily used within this project and were added with low configuration and development costs.

With the capability for more advanced functionality than filters, the use of directives within the project has been to simplify development and offer consistent UI behaviour. Many examples of this are available in the numerous directives offered by the Angular Material Design framework that provide the core UI components. However, a custom-built directive example is the file download directive used to negotiate the file download workflow in a single page application. This workflow is required because of the security restrictions on background content transfer with JavaScript processes. This directive handles the file-download workflow by initiating a download request for a specific file. Then, if authenticated, it receives an anonymous download link that can be opened in a new browser tab and received through a traditional file download prompt.

While directives and filters are used for visual data representation, Angular services and controllers provide data manipulation and organization functionality. More specifically, controllers provide the data manipulation tools for a specific view. Controllers can be specified to a directive level and provide an isolated scope with function definitions and local data storage. These functions and data are dynamically bound to their view and are provided directly to the contained directives and DOM elements. This level of isolation and specificity encourages modular development with a strict separation of concerns.

Within these controllers, much of the general functionality is abstracted into services. These provide a centralized manner in which data can be retrieved, manipulated and stored. Services, like controllers, help with the separation of concerns, and consequently simplify the development process. Additionally, the abstract nature of services encourages comprehensive testing and allows them to be mocked during testing of other software components.

A specific, and interesting, service that was developed in the application is the token management service. This service manages the authenticated token that is transmitted to the server with each API transaction and the information stored within this token. Management covers both translation of token content and status of the token. The status of the token is tracked to ensure that an awareness of local token persistence and token validity are given, as these dictate the existence of an authenticated token. Once persisted and validated however, the content of the token is interpreted and made available to the application generally. This is done in a centralized manner to ensure a consistent user model representation regardless of API or token changes.

5.1.2 *Back End*

To take advantage of a greater level of pre-existing knowledge and a robust enterprise user base, the Java Spring Boot framework was used to develop the back end API. This choice offered a robust Inversion of Control (IOC) 'plumbing' framework, as well as the numerous well-developed and customizable Spring Boot libraries. However, the high level of specification and excessive customizability did lead to several difficulties during development.

A major functionality offered by the Spring framework is the ability to 'plumb' together objects using a configuration based approach. This inversion of control separates the definition of desired functionality from the actual implementation. It allows developers to focus on individual concerns and abstract away complexity through interfaces. Additionally, this level of isolation offers increased flexibility for both testing and altering implementation.

This 'plumbing' and interface based development is especially important for the incorporation and development of external data interfaces. By ensuring that the definition and implementation of the data access and authentication mechanisms are separate, it only requires configuration changes to integrate a new data source or change the authentication provider. Apart from being a project requirement, data source and authentication adaptability, powered by inversion of control, are fundamentally valuable functionalities of a complementary system.

Another important functionality provided by the Spring framework and associated libraries is the ability to annotate code. These annotations, shown in Figure 1, are used by the Spring framework to apply specific properties and behaviours to functions and attributes. Not only does this provide a convenient manner with which functionality can be added, it also ensures cleaner and contextual code. A specific example of this is the ability to annotate individual attributes within a domain model and limit the amount of information encoded on an individual endpoint basis.

```
@JsonView(Views.CourseOwner.class)
@PreAuthorize("hasRole('ROLE_COURSE_OWNER')")
@RequestMapping(path = "/courses/{courseId}/admin", method = RequestMethod.GET)
Public Course getAdminCourse(@PathVariable("courseId") Integer courseId) {
	// Method implementation
}
```

Figure 1 An example of the annotations used by the Spring framework to simplify development. This method is annotated to display a configurable level of information to a pre-authorized group of users. Additionally, the specific address that the content is available at and the access methods are further described by another annotation.

Equally useful is the use of annotations to define and secure endpoints within the REST API. Developers are able to annotate which function responds to an API request and additionally annotate the rule that authorizes a user's access. This specific functionality, shown in Figure 1, allows the organization of endpoints by general topic and abstracts the request routing and authorization process away from general development.

While the abstraction of many processes within the Spring framework reduces clutter and offers conveniences, it is also a source of difficulty for both debugging and customization of very specific behaviour. As the framework 'plumbs' together files in an indirect manner, following code execution can be confusing, and at times impossible, while debugging. Additionally, if a component within the framework fails or does not act as expected, there is little recourse, and refactoring or redesigning is usually required.

Luckily, the difficulties faced by this framework were generally followed by large stages of refactoring that led to surprising, yet logical, solutions. A major example of this is the error handling and management aspect within the Java environment of the API. Initial development of the API incorporated traditional Java development patterns, with exceptions and errors handled and the continuation of execution. However, this workflow neglected the disjointedness of an API and it cluttered endpoint definitions with multiple responses for each error type returned.

Eventually, a shocking solution presented itself: don't handle the errors, let them reach the API and provide feedback to the client service. Instead of handling the errors in an incomplete environment, the API could provide feedback to a user that an action had failed and defer responsibility to the client. This solution presented itself in the form of more annotations and a global handler that 'hinted' at what response to give for each error type. Immediately, code became clearer as exceptions were directed to the endpoints and then assigned a HTTP response code and descriptive message, ensuring a fully informed response by the user.

This solution also highlighted one of the most important aspects of the API/client based architecture used in this project. To ensure independence and generalizability, the API must be able to provide descriptive responses to any client. Responses can then be used by the client to understand and react to errors or information locally. The ability to manipulate information locally ensures that development of the back end and front end will stay relatively independent and naturally allow for expansion.

5.2 Functionality Implementation

While the development of the front and back ends were designed as separate entities, their development inevitably occurred concurrently. This guaranteed a coherent system design and ensured the development of specific functionality and not generic, directionless code. Throughout this section, the core supporting elements and interesting aspects of the functionality described in the Requirement Capture section will be discussed. Additionally, the principles and patterns highlighted in the analysis and design stage will be linked to the produced implementation.

5.2.1 Core Functionality

As detailed in the background chapter, no existing VLE implementation was used for the platform. While this decision resulted in more work, it allowed for greater flexibility in the specification of functionality and in the division of responsibility between the front and back ends. Development was assisted by the use of the Spring framework, but required careful thought to ensure it was not overcomplicated by additional framework code.

The extensive range of libraries offered with the Spring framework did assist in development, but the selection and use of libraries was at times difficult. Evaluating the trade-offs between library functionality and any restrictions to development was a necessity and dictated library choice. For example, the Spring-Data-REST library offers a fully-fledged REST API with minimal configuration and set up. However, the generated endpoints are difficult to configure and data must be handled with Hibernate object mapping, something that was not appropriate for the complexity of data managed.

While the libraries add a range of functionality, the common advantage of all libraries is a reduction in boilerplate code. Every library is built to reduce the amount of code written to facilitate actions, and instead allows developers to focus on the application specific functionality offered. This ideology is fundamental to the Spring framework and supported the developer's decision to design the data and external service as configurable interfaces.

Within the application, information is retrieved from storage via a series of services and data-access-objects (DAOs). Both of these components are composed of an interface, specifying the available functionality, and a concrete implementation. Basic functionality was offered in a systematic way that made methods to retrieve, update and delete content available.

This style of architecture, with a layer of abstraction between desired behaviour and implementation allows any data source to be connected in a plug-and-play style. Whether information is stored locally, queried from a persistent database or retrieved from an external service, the specified behaviour and responses are consistently defined.

In addition to implementation flexibility, another core feature of development was the security of the application. The Spring framework provides a simple way to restrict access using a filter based architecture. Each web request that reaches the server is passed through a series of filters that can either propagate, reject or modify the request. Within the API, these filters are used to specify the general permissions surrounding endpoints and to authenticate users.

At a high level, the filtering indicates that all endpoints, unless otherwise specified, require authentication. Authenticated details can then be used to determine if a user is authorized to access an endpoint on a case-by-case basis. To achieve this filtered authentication in a scalable manner, a stateless authentication and authorization was used.

A token-based security workflow was chosen to provide a safe and scalable implementation. With token-based security, user credentials are replaced with a dynamically generated token after initial authentication. Users then provide this authenticated token with each API transaction to identify themselves, reducing the requirement for user credentials to be stored and improving system safety.

Tokens can also provide a method for stateless authentication. Using the newly available JSON Web Tokens (JWTs), token validation can be done with information encoded in the token alone. This removes the requirement for persistent storage of token state and allows for isolated token validation.

JSON Web Tokens (JWT) are a 'method for representing claims securely between two parties' (Anon., 2016). These tokens can encode an arbitrary range of information in a JSON format and secure the claimed content by signing the token with a cryptographic hash of the token contents. When a token is received, it is parsed and verified by the receiving server, without any additional information, to determine the integrity of the contents. Once the content is verified, the encoded information can be used for any task, including authentication, authorization and identification.

To implement this JWT solution, a custom header was added to API transactions and interpreted by a security filter prepended to the Spring security workflow. The filter dynamically verifies the contents of the token, and if valid, decorates the request with the current user's details and authorizations. By allowing token validity to be determined with the token alone, the API becomes stateless and naturally scalable.

While the JWT solution is inherently secure due to the complex encryption used, a token expiry time has been added, mitigating the risk of rogue token use. Additionally, token refresh and cancellation processes could be introduced to add more control in the case of a stolen or compromised token.

5.2.2 Specific Functionality

Implementation of the specific functionality described in the Requirement Capture chapter is of interest because of the design patterns and engineering principles followed. These principles and patterns, described in the Design & Analysis chapter, are discussed here and highlighted in the functionality that was developed.

A prime example of the use of these patterns is the design and implementation of the feedback/mark split. This functionality, which allows markers and course owners to return feedback comments and marks at different times, required relatively simple API development but more complex UI development to display information accurately and intuitively.

The reduced API implementation complexity was due to the annotations and flexibility offered by the Spring framework. Minor changes were required to add visibility markers to the feedback object and annotations to ensure this was rendered correctly. On the other hand, the user interface required more work.

Due to the relative complexity of the submission workflow, there was a requirement to display the status of the submission clearly. This was initially done in a simple, textual manner, but adding more stages to the workflow and a corresponding amount of information made this representation untenable. With users required to fully read and interpret the status messages, there was the possibility for confusion and a complete lack of contextualization as described in the Design chapter.

Several changes were subsequently made to the user interface and API. These changes resulted in a clearer representation of information and a better handling of the added complexity from the feedback/marker split. The first change split the responsibility for information visibility between the API and the user interface. API responses were dynamically defined with only relevant information exposed at each stage of the workflow. This meant that the visibility of UI elements was determined in a binary manner, based on inclusion in the API response.

With such a simple method to determine information visibility, the space on the page could be reused very efficiently. This reuse was achieved by dynamically adapting the content cards with the validity and state of the submission, highlighting changes in an obvious manner. Submission validity, representing whether the submission was for credit, was indicated with coloured highlights of the submission title and different iconography. Additionally, feedback for the submission was displayed as it became available by replacing placeholders and changing icons, which both served to highlight the change of state.

In a more general sense, the bulk management controls offered by the platform build on several of the design patterns and engineering principles discussed. An example of this is the use of an external Angular library that offers functionality for sorting, filtering and manipulating data in a bulk manner for large data sets. Not only does this provide a consistent experience, but these tables are also complemented with a title bar that is used for table context. When selecting data, valid actions and a general description of selected data replace the title and clearly guide the user’s behaviour.

Although all components described in the requirement capture chapter were incorporated in the design of the platform, not all were fully implemented. Two areas not implemented are group administration and marker swapping. They were not fully implemented because the complexity added would not have been handled effectively. This trade-off between functionality and complexity was deemed acceptable because it could be achieved in an isolated manner and would not affect core functionality.

An implementation of the marker swapping API code exists, but the UI functionality has not been built. Marker assignment and the feedback process for submissions have both been abstracted into their own workflow, separating the feedback complexity from submission complexity.

Although this separation simplifies the API code and general submission management workflow, it increases the complexity of the UI. User interface complexity is handled quite well within the designed platform, but required extensive work to accomplish this. Adding an extra stage to the workflow and increasing the information and state displayed would not have been handled well in the time available.

The group administration functionality was removed from the platform because of the significant API complexity it added. To avoid designing multiple cases, the submission workflow was designed with every submission tied to a group. With a single user then sub-classed as a single-member group, this followed the 'default activity' design pattern previously discussed. Taking advantage of this single workflow, dynamic inclusion of information, based on group status, helped guide the group administration UI workflow in a similar manner to the feedback/mark split.

Architecturally, this meant that a single workflow could be followed and the API took most of the responsibility for group management. However, this also meant that the API was very complicated and development slowed significantly. Although more of the responsibility could have been shifted to the UI, the fundamental API complexity still existed and the group management functionality was delayed for later development.

6 TESTING AND RESULTS

Testing of the platform occurred on multiple levels throughout development. This was done to ensure code quality and to evaluate the effectiveness of design choices. Both of these factors were chosen to validate developer decisions and to avoid developer bias and familiarity from hiding poor design decisions. To comprehensively test the platform, unit testing, user acceptance testing and situational testing all occurred. Details of each type of testing and a summary of the results will be provided, with detailed results available in the Appendix.

6.1 CORE FUNCTIONALITY TESTING

Effective testing of the core functionality within the application, at a source code level, was completed with unit testing. This style of testing, which isolates specific functionality and validates behaviour, is very effective in large projects. It forces developers to define specific and isolated tests, which are evaluated in an isolated manner. With this isolation, issues can both be identified and tested at a much finer level.

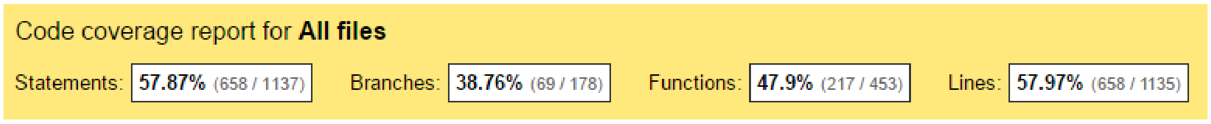
Due to the relative importance and implementation style, the majority of the functionality testing was focused on the user interface. Both information manipulation and workflow management are coordinated by the front end, meaning that the API source code is relatively simple. This simplicity has led to reduced API testing and an emphasis on unit tests that verify the behaviour of the user interface.

Both the Spring and AngularJS frameworks provide unit testing in comprehensive bundled testing libraries. These libraries are accompanied by significant documentation and many examples that help guide the definition and development of tests. In addition, the design of the frameworks, and integration of testing libraries, ensures that testing is a seamless process.

By encouraging a decoupled design with the configuration based 'plumbing' of interfaces, objects can be dynamically 'mocked' and return tester-specified responses, ensuring that tests run in true isolation. Additionally, the bundled libraries provide extra functionality to fake network calls, create temporary databases and validate a large variety of conditions surrounding functionality.

With these conveniences, testing was easily defined alongside development, based on desired behaviour. Testing was written to validate current development and safeguard future changes. Importantly, these tests can be easily expanded to verify bug fixes and guard against their future revival.

Due to the above testing methodologies, the code coverage of the front-end tests has grown steadily. This coverage, measured using a tool called Istanbul, evaluates how much of the code is executed and tested by the defined tests. The below summary (Figure 2), generated by Istanbul indicates the amount of code covered by unit testing.

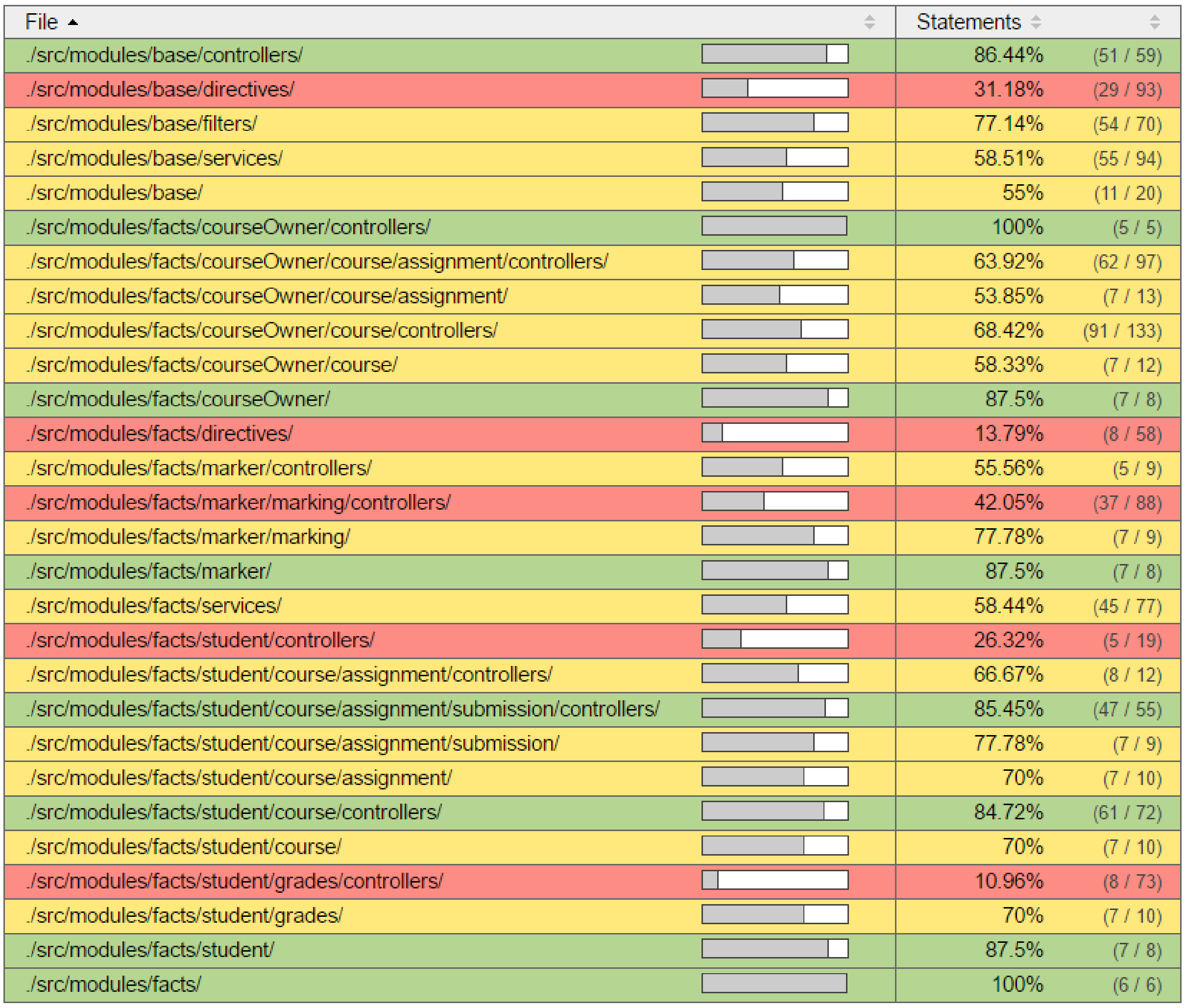


Code coverage report for **All files**

Statements: **57.87%** (658 / 1137) Branches: **38.76%** (69 / 178) Functions: **47.9%** (217 / 453) Lines: **57.97%** (658 / 1135)

Figure 2 Details of Code Coverage Statistics Provided by Istanbul

An interesting aspect of this code coverage is the discrepancy between the types of statistics. As can be seen, almost 60% of the code statements are covered, while only 50% of functions within the code are. This is due to the relative complexity of functionality, and therefore JavaScript functions, within this implementation. To avoid any confusion between statistics, the percentage of statements covered has been chosen to represent code coverage in a more detailed report (Figure 3).



File	Statements	
./src/modules/base/controllers/	86.44%	(51 / 59)
./src/modules/base/directives/	31.18%	(29 / 93)
./src/modules/base/filters/	77.14%	(54 / 70)
./src/modules/base/services/	58.51%	(55 / 94)
./src/modules/base/	55%	(11 / 20)
./src/modules/facts/courseOwner/controllers/	100%	(5 / 5)
./src/modules/facts/courseOwner/course/assignment/controllers/	63.92%	(62 / 97)
./src/modules/facts/courseOwner/course/assignment/	53.85%	(7 / 13)
./src/modules/facts/courseOwner/course/controllers/	68.42%	(91 / 133)
./src/modules/facts/courseOwner/course/	58.33%	(7 / 12)
./src/modules/facts/courseOwner/	87.5%	(7 / 8)
./src/modules/facts/directives/	13.79%	(8 / 58)
./src/modules/facts/marker/controllers/	55.56%	(5 / 9)
./src/modules/facts/marker/marking/controllers/	42.05%	(37 / 88)
./src/modules/facts/marker/marking/	77.78%	(7 / 9)
./src/modules/facts/marker/	87.5%	(7 / 8)
./src/modules/facts/services/	58.44%	(45 / 77)
./src/modules/facts/student/controllers/	26.32%	(5 / 19)
./src/modules/facts/student/course/assignment/controllers/	66.67%	(8 / 12)
./src/modules/facts/student/course/assignment/submission/controllers/	85.45%	(47 / 55)
./src/modules/facts/student/course/assignment/submission/	77.78%	(7 / 9)
./src/modules/facts/student/course/assignment/	70%	(7 / 10)
./src/modules/facts/student/course/controllers/	84.72%	(61 / 72)
./src/modules/facts/student/course/	70%	(7 / 10)
./src/modules/facts/student/grades/controllers/	10.96%	(8 / 73)
./src/modules/facts/student/grades/	70%	(7 / 10)
./src/modules/facts/student/	87.5%	(7 / 8)
./src/modules/facts/	100%	(6 / 6)

Figure 3 Specific Unit Testing Code Coverage Statistics

This report, displaying the percentage of statements covered by testing for different functionalities, is evidence of the quality of testing performed. Examination of the report reveals that core functionality code is well covered. The code for the controllers of complex views, centralized services and filters are systematically well tested. This is reflected both in the percentage of code tested and the number of lines tested.

The relatively low code coverage of directives in the report should also be discussed. Directives are fundamentally difficult to unit test in Angular as they are largely visual and are built in a non-isolated manner, rendering them difficult to unit test. As such, a decision was made to reduce technical testing of directives, but supplement this with testing of their behaviour during use.

6.2 USER ACCEPTANCE TESTING

A major aspect of the application is the user interface and general usability of the system. A round of User Acceptance Testing (UAT) was organized and the results analysed to evaluate the user interface. This UAT split the evaluation of the interface between graphical design and functionality offered. Splitting the testing ensured that two major aspects could be evaluated: the consistency of graphical components and user controls and whether functionality was intuitive.

6.2.1 TESTING FORMAT

To organize the testing, a Google Form was created, with a series of simple questions used to evaluate the design of the user interface and assess ease of adoption. While anonymous, the survey did request demographic information to illustrate the variety of users who tested the application. Throughout the survey, questions were phrased to avoid leading the tester and comment boxes were provided for further expansion. The full list of questions in the survey is detailed in the Appendix.

Survey questions addressed the two major aspects mentioned above. For design consistency, users were asked two questions. One, if graphical design is important to trust in websites; and two, if the design felt consistent, and if not, to comment on the areas that were inconsistent. In a similar fashion, users were asked if the functionality was intuitive to use and, again, asked to expand if they felt it was not so. In addition to the design and functionality specific questions, users were also asked for general commentary on the design and platform itself.

During testing, the application was made available on a public server, and several user accounts, with varying privileges, were given to the testers. To allow all functionality to be explored, a variety of content was pre-created on the application. Beyond the login details, no further instruction or guidance was given. This set up was designed to ensure an accurate understanding of user adoption and the usability of the system.

6.2.2 *Results & Analysis*

The user acceptance testing was run for a five-day period and advertised to a select group of students. These students were selected from several engineering departments that have regular VLE interaction and experience with a variety of VLEs. Twenty-three responses were received during the testing period and are available in full in the Appendix. Consisting of 73.9% of the total survey responses, the majority of evaluation was done on the student interface. This was to be expected, due to the group chosen to do user acceptance testing.

Overall, the user acceptance testing was positive, with more than 90% of responses indicating that the design was consistent and functionality intuitive. This is important as it verifies the work and importance the developer assigned to the consistency and usability of the user-interface.

Of the negative responses, there were indications that this was due to a lack of content, leading to empty spaces and subsequent confusion over where content was accessed. This is an important point and can be addressed with the use of more explicit content containers and a greater awareness of the meaning of blank spaces.

Another overwhelming response was the importance of graphical design in the trust of a website. With only one negative response, it can be seen that graphical design is important for fostering trust in a website. This validates the developer's choice of a well-known and trustworthy visual framework.

Finally, the general commentary of the project was both positive and constructive. Users liked the "smoothness of the operation of the site" and "consistency and simplicity of the design". However, the comments also requested more functionality and gave constructive criticism of specific design features.

Specifically, users felt that certain visual elements were slightly misleading. For example, allowing list elements to be highlighted, but only providing clickable functionality in some cases, led to confusion.

While the comments were helpful and led to the identification of the above issues, user acceptance testing occurred too late in development. This means that the results of the testing have not influenced the user interface yet. However, the identification and understanding of the issues discussed above make them priorities for future improvement.

6.3 Situational Testing

While other testing evaluates the suitability of the platform functionality, situational testing evaluates the suitability of the implementation with respect to the environment it will operate in.

This style of testing stresses the platform with the extremes of the technical environment and establishes usage boundaries and areas for improvement. More specifically, the testing carried out in this project targets the security of and user load expected by the platform.

6.3.1 *Security*

As discussed in the design chapter, the security of the application is extremely important, and several security measures have been implemented. These measures are there both to prevent unauthorized access and to protect information integrity. Unfortunately, security is a difficult aspect of any application and fundamentally biased against the developer. While every effort may be made to secure an application, a malicious user must only find one avenue of access, and can cause irreparable damage.

In an attempt to mitigate this, several vulnerabilities were tested against an isolated, locally hosted version of the platform. These tests covered the three most common styles of attack: injection, broken authentication and session management and cross-site scripting (OWASP, 2013).

Vulnerability to the most common forms of SQL injection was determined to be non-existent. This was tested by sending specially crafted strings to the API, in the hope that extra code might be executed or evaluated. Fundamentally, the design of the system and use of the Spring framework, ensure that many, if not all, forms of SQL injection are mitigated at an architectural level. As the framework sanitizes API input and interfaces render SQL queries as prepared statements, the ability for arbitrary SQL queries to be executed is minimal.

Another major threat mitigated by the use of a framework is cross site scripting (XSS). Attempts at executing XSS snippets within comments and other user-editable areas throughout the application were ineffective. Cross site scripting attacks target the user interface and are easily handled by the Angular framework. With dynamic text rendered after JavaScript has parsed the page, the ability to include functionality in malicious messages or comments is removed. Additionally, the tight coupling of user accounts to content ensures that any such attempts will be easily identifiable and users dealt with accordingly.

The final aspect of security testing was the management of authentication and session management. No tests were carried out throughout development as this style of testing is beyond the scope of the developer. However, the application security, delivered with industry standard JSON Web Tokens (JWT) technology, is extremely robust. While tokens can be stolen, the use of SSL to encrypt client-server communication means that this must be done on the client machine, and as such is out of the developer’s control. A workflow, which would trade statelessness for security, could also be established to allow tokens to be arbitrarily revoked, rather than upon expiration of their validity period.

6.3.2 *User Load*

The availability of the platform is an important aspect of this time-critical workflow. With peaks of activity around deadlines, and the importance of stability during these times, the system was tested to verify the architecture for varying usage. This testing was done on the API with a modelled production environment and simulated user interaction hosted on cloud services.

To model the production environment, approximations of the user load and server specification were made. As the application would be used in the Electrical & Electronic Engineering department, there would be approximately 1000 users spread across the three user types. Additionally, the application would have to be highly available, with the ability to handle bursts of activity around deadlines.

These requirements led to the selection of a high powered, general-purpose web server. Amazon Web Services (AWS) offers this in their M4 range, that provided a "good balance of compute, memory and network resources" (Amazon Web Services, 2016), which are well suited for an application used within the department. With eight virtual CPU cores and 32 GB of memory, the server is powerful enough to deliver a mid-sized application alone.

Load testing of the server was done with a separate server provisioned with increased computation power, selected to provide capacity for the highest number of concurrent connections. The server was hosted within the same network zone on the Amazon Web Services infrastructure to ensure low network latency and to eliminate bandwidth costs. Low network latency ensured that the response times of the application server were indicative of the required processing and not network speed.

On the load-producing server, the simulated user connections were powered by an open source project called 'wrk' (Glozer, 2016). This program provides HTTP benchmarking by generating requests as fast as possible across a user-specified number of threads and connections. Moreover, connections can be customized with LuaJIT (a scripting language) scripts and generate unique benchmark situations.

Taking advantage of this customizability, scripts were developed to test the two major situations experienced by the server: GET-ing course content and POST-ing information to the server. The testing scripts are included in the Appendix for reference. While posting information to the server could include file content, this was not benchmarked for two reasons. Firstly, the variable and uncharacteristic nature of the testing could lead to technical issues, such as network or hard drive failure, which would not be faced by a production application. Secondly, the cost of provisioning storage and memory for this type of testing are beyond the scope of a benchmark for a proof of concept application.

To evaluate the boundaries of the selected situations, testing was done with varying user loads and limited to sub-second responses. If a response took longer than 1000ms, the testing would be short-circuited and restarted with a reduced number of concurrent connections. Luckily, this was never the case in the scenarios tested.

Starting at 200 concurrent connections, the number of simulated users was gradually increased to 1,000, the estimated total number of users, by 200 connection increments. Tests were run across eight threads to take advantage of the multi-threading nature of 'wrk' and were run at approximately 1,500 requests / second, adequately pushing the application boundaries. The request rate was purposely kept static to evaluate the effect of concurrency, rather than rate, on the application. Finally, each test lasted 30 seconds to establish a baseline of behaviour and ensure that observed behaviour was correlated to the delivered load and not network anomalies.

Graphs of the CPU usage during the GET test and POST test are displayed below in Figure 4 and Figure 5 respectively. Detailed results of CPU and memory usage during the tests are available in the Appendix.

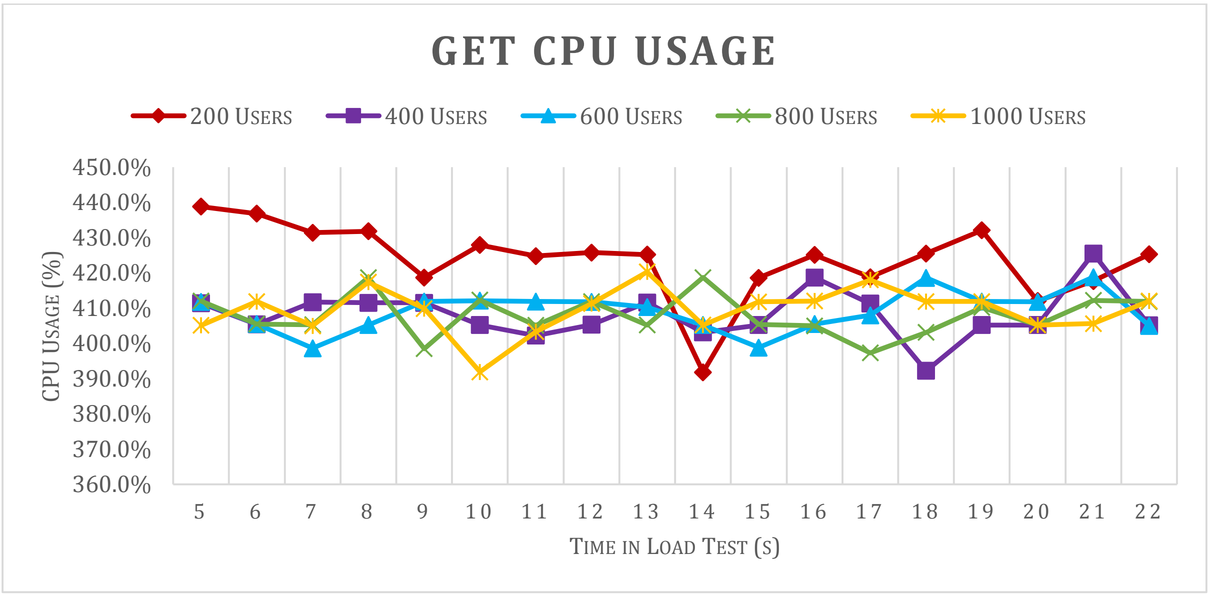


Figure 4 GET Load Test Usage Graph

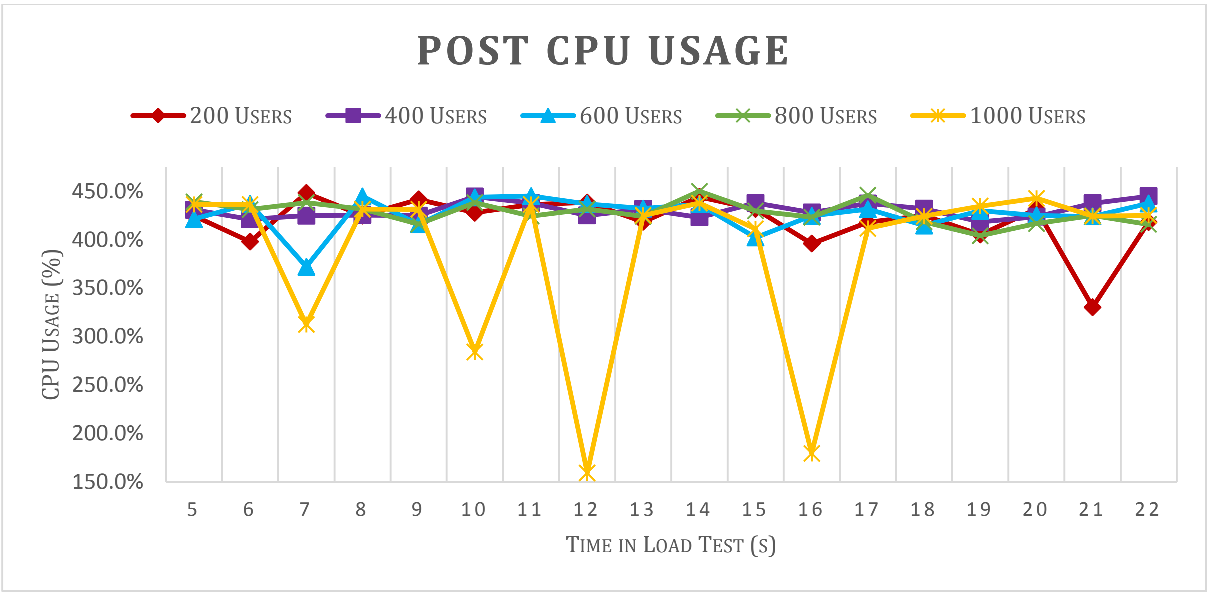


Figure 5 POST Load Test Results Graph

In general, the tests were successful and indicated the stability and suitability of the application for the expected user base. The results of the GET simulation, shown in Figure 4, highlight the stability of the system under all projected user loads.

Results from the testing also indicate that the server selection was accurate for medium-high usage, with all situations bursting to just over 400% processor usage (4 / 8 cores used), leaving adequate headroom. While benchmarking is not indicative of usual behaviour, it provides an idea of the boundaries that the system will operate within and the necessary equipment to increase these boundaries.

One interesting aspect of the testing, seen in Figure 5, are the results of the POST test. CPU usage during the 1000 concurrent user benchmark deviates from the standard of 400% CPU usage to 150% usage on several occasions. These dips in usage can be attributed to the application dropping connections upon database write failures. Fortunately, this behaviour is only seen at extremely high loads (thousands of concurrent requests per second over a sustained period), which could not be produced by the expected user base.

6.4 Summary of Results

Testing of the platform was done to assess the quality of engineering performed during development. This was determined in two ways, firstly, the quality of the technical implementation was verified; and secondly, the engineering principles used to develop the platform were validated.

Technical quality was verified with unit-testing and situational testing, both of which were quantifiably successful. The platform was developed with comprehensive unit testing that both ensures the quality of current code and secures it for future development. A summary of this testing showed the importance it was assigned and the extent to which it was performed. In addition, situational testing highlighted the quality of the code and verified the suitability of the platform for its intended use. Results, contained in the Appendix, indicate the thoroughness of testing performed and the stability of the platform in extreme situations.

While both of the above testing results verify the quality of the technical implementation well, they do not validate the engineering principles in the same way. These conceptual principles cannot be evaluated directly and require more complex testing to assess. This evaluation was achieved through a round of UAT that can be described as successful. While aspects of the UAT are open to interpretation, responses categorically defined the design as intuitive and consistent. This was positive in itself, but more importantly highlighted the accomplishment and success of the developer's implementation of the underlying design principles.

It can be seen from the above results that the engineering of the platform has been well performed. With the verification of technical quality, it can be shown that the platform is built in a robust and safe manner with enough power to operate smoothly. Furthermore, the validation provided by the UAT responses shows that the fundamental principles have been observed and implemented throughout the platform.

7 DOCUMENTATION & GUIDES

Guidance for the installation, configuration and use of the application has been developed to facilitate future use. This chapter highlights the high-level documentation choices and references the location of specific, detailed guides.

7.1 USER GUIDE

There is no user guide available for this project as it would be in direct contradiction of the requirement for an intuitive user interface. While aspects of the user interface may require introduction, which will be done in the 'Basic Functionality Guide', the completion of standard workflows should be obvious and guided naturally by the interface design.

While the user acceptance testing did not explicitly test the completion of standard workflows, the results indicate that users are comfortable with the design and feel it is consistent and functionally intuitive. Additionally, the design decisions and choice of visual framework ensure a level of clarity and consistency that lead user interaction without explicit instruction.

Despite the decisions not to create a user guide, certain visual elements within the application will be described in a simple 'Basic Functionality Guide'. This guide, available in the GUI repository, will ensure that users who have never experienced the Material Design framework are able to understand the general visual language. With an understanding of this, new users should be able to intuitively and naturally navigate the application and execute actions.

7.2 INSTALLATION & CONFIGURATION

Instructions to configure a server and install the API and user interface are all available in the Git repositories used to host the code. Deployment of the user interface relies on the existence of an API, so the instructions are dependent on running the API instruction set first. This style of instruction, embedded in the Git repository, is useful as it can be updated regularly and in close conjunction with the development of code.

Both projects are built with task running tools that are used to build from the source code directly. Once built, the instruction manuals guide the installation of the API and user interface for access on a secured web server. While several security precautions are taken by the application itself and briefly covered in the installation documentation, installation specific security configuration should be executed before public deployment.

7.3 DEPLOYMENT PLAN

The final aspect of guidance available with the project is a deployment plan for the application to be used within the Electrical & Electronic Engineering department. There are two main concerns addressed by this plan: the redundancy of assessment management and adoptability of the user interface. Redundancy refers to the ability for the department to deal with a failure of the application, while adoptability refers to the ease with which users transition. Deployment of the application is designed to occur over an academic year; however, for flexibility, it has been designed in three stages, with specific activities tied to each stage, not time period. Sufficient time should be given between the deployment stages to allow for any fixes and upgrades identified in the previous stage.

Stage one of the deployment will be a limited beta that is designed to happen over a single term (approximately three months). This beta period will contain only a few, smaller courses, with course owners and markers who volunteer their involvement. In addition to this, the courses will be coursework heavy, but with more flexible deadlines. With this flexibility, any changes to the system or failures can be easily mitigated and the use of a backup VLE will not be necessary.

While this stage of development occurs, the volunteers will be asked to complete more comprehensive user acceptance testing (UAT). This round of UAT will foster a high level of communication between the developers and users and ensure that bug fixes happen quickly.

Once the initial limited beta has been completed, the second stage of deployment will include expansion to the remainder of the department. This stage is also designed to occur over a single term, and will expose the system to significant real-life usage and validate the previous user load testing. Statistics on the live usage and server health during this deployment stage will be captured for later analysis.

To provide redundancy, the existing VLE, BlackBoard, will be used in conjunction and will mirror the assigned content and requirement for user submissions. Although user comments and bug reports will still be accepted, it is assumed that the amount of user-developer interaction will reduce. Additionally, dual use during the adoption stage will allow users a gradual transition and ensure smoother adoption.

The final stage of deployment is the removal of BlackBoard as a redundancy, and complete adoption of the new application. Between the end of the second stage and beginning of the third stage, analysis of the usage statistics and server health will pinpoint any areas for improvement. This analysis should be used to verify the infrastructure choices and ensure stability in future usage.

8 EVALUATION

Development of a proof of concept application is an interesting process because the priorities and goals are difficult to define. While specific requirements and functionality can be captured in interviews and created in code, the fundamental goal of development is less tangible. For this project, the underlying goals of development were to demonstrate the viability of and improvements offered by a custom solution. Evaluation of whether these goals have been achieved can be split into two aspects: conceptual and technical, which are discussed below.

8.1 CONCEPTUAL EVALUATION

Although the technical implementation of the project is important, the difficult engineering and problem solving occurred during the analysis and design stage. Gaining an understanding of the problems faced and the ways to manage these was crucial to development. Not only did this fundamentally guide development, it also ensured that functionality was built in a manner that addressed the identified issues.

Developing an understanding of the complexity within the user interface and workflows was difficult, but a critical stage of the project. Once developed, this understanding was used to define the design patterns and engineering principles. These fundamental implementation concepts, discussed in the Design & Analysis chapter, were then used to guide development throughout the project.

The definition of these core implementation guidelines was crucial to the success of the project. Identifying the underlying theme of convenience, that drives the user experience led to a principled user interface and workflow design. Instead of designing functionality, and testing for exhibited behaviour, the desired design principles were codified in the style of development and resulted in a principled, consistent and user-friendly application.

8.2 TECHNICAL EVALUATION

The technical aspect of this project can be evaluated with respect to the technical quality, and inclusion, of functionality. In general, a trade-off between available functionality and technical quality was made; this resulted in a slightly less feature filled application, but a correspondingly stronger implementation.

Functionality that was not implemented, as discussed in the Implementation chapter, was excluded in an isolated manner. It was external to the core implementation and is not crucial to the demonstration of platform viability. Additionally, time-restricted development of this functionality would have detracted from other UI and workflow improvements, reducing the impact of the project.

For the implemented functionality, development was supported by principled engineering and design decisions and by the use of industry practices. This level of engineering was achieved in three mains ways: explicit definition of design patterns and engineering principles, selection and adoption of mature and robust frameworks and testing to verify the integrity and to validate the use of developed functionality.

Throughout the analysis and design stage, the underlying theme, convenience, that tied together the problems with existing VLE implementations was identified. This theme was then codified in the described design patterns and engineering principles and these were used to guide development of desired functionality.

Development of the platform, following these principles, was legitimized and simplified with the use of industry frameworks. With the selection of two powerful frameworks, a firm foundation for development was created. This foundation was, and will be, supported by a large base of industry and community knowledge, ensuring the continued relevance of developed code.

Finally, the quality of both the developed code and functionality was verified in several rounds of testing. Unit testing during development was used to verify the quality of code and to validate the functionality offered. Similarly, situational testing ensured the overall quality of the system and validated the suitability of the application for the desired implementation environment. Finally, UAT confirmed the general usability of the system and validated the developer's requirement for usability and ease of use.

9 CONCLUSION

FACTS is an application that demonstrates the viability and suitability of a custom VLE implementation. The implementation, designed with input from members of the Electrical & Electronic Engineering Department at Imperial College London, addresses the interface difficulties and complexities offered by existing solutions.

Not all functionality offered by existing VLEs is available in this implementation; however, the delivered functionality has been designed in a principled manner and is consistently implemented. As the fundamental goal of the project is dependent on an interface that addresses existing difficulties, a decision was made to remove functionality in favour of an improved user interface.

Improvements to the user interface were primarily achieved with the definition of design patterns and engineering principles. Once defined, these design guidelines were used to produce a platform based on strong design decisions and developed in a systematic manner.

Evaluation and verification of the work was completed through several tests. These tests, designed to evaluate code quality, platform stability and user experience, combined to provide a positive holistic view of the platform and validated the developer's design choices.

Built in a principled manner and with the use of the practical and theoretical skills available to an Electrical & Electronic Engineer, this is a well-designed application and a strong indication of the developer's potential.

9.1 FURTHER WORK

It is unlikely that this implementation will be deployed in a production environment, but will instead be a source of inspiration and design for an in-house system. As such, this section will focus on the additional functionalities that could be added, once the system is production ready, and on the creation of more comprehensive system documentation.

Two concepts were briefly covered in the requirements capture, but should be highlighted here. These are: a configurable bulk management interface powered by CSV files and a user-customizable report-generating engine. Both these added functionalities would continue the theme of convenience, and if delivered well, would enable course owners, markers and administrators to have more control over the workflow.

With CSV files, a user would be able to manipulate information in their own environment. This in turn could lead to a faster marker turn around and more advanced and innovative ways of determining or moderating marks.

Similarly, the creation of a user-customizable report-generating engine would give users the freedom to analyse and interpret information on their own. Results of these reports could be coupled with the marker assignment stage and could guide fairer marker assignment. Additionally, the reports could be fed into other user-customizable modules. These could be used to automatically moderate marks or, using machine learning, identify struggling students at an earlier stage.

In addition to further functionality development, future work could be done to formalize the design patterns and engineering principles as a document. Storing these resources in a central manner would enable future development to reference them more easily. Similarly, simple examples of the patterns and principles would ensure developers understand the reasoning and style of development that underpins them.

10 BIBLIOGRAPHY

Amazon Web Services, 2016. *EC2 Instance Types.* [Online]
Available at: https://aws.amazon.com/ec2/instance-types/

Anon., 2016. *JSON Web Tokens,* Seattle: s.n.

Business Wire, 2014. *Geenio Launches Full Cycle Learning System for the Enterprise, Raises $2M Seed Round.* [Online]
Available at: http://www.businesswire.com/news/home/20141219005952/en/Geenio-Launches%C2%A0Full-Cycle-Learning-System%C2%A0for-Enterprise%C2%A0Raises-2M

Clarke, T. J. W., 2015. *FACTS - "Preventing Broken Windows".* London: EEE, Imperial College London.

Empson, R., 2014. *Education Giant Blackboard Buys MyEdu To Help Refresh Its Brand And Reanimate Its User Experience.* [Online]
Available at: http://techcrunch.com/2014/01/16/education-giant-blackboard-buys-myedu-to-help-refresh-its-brand-and-reanimate-its-user-experience/

Geenio, n.d. *Geenio,* Los Angeles: s.n.

Glozer, W., 2016. *wrk.* [Online]
Available at: https://github.com/wg/wrk

Google, 2016. *Material Design.* [Online]
Available at: https://www.google.com/design/spec/material-design/introduction.html

Harris, P., Sillence, E., Briggs, P. & Fishwick, L., 2004. Trust and Mistrust of Online Health Services. *CHI,* pp. 663-670.

Kornukopia, n.d. *Kornukopia,* Saint Petersburg: s.n.

Lambda Solutions, n.d. *What is Moodle?.* [Online].

Lichtenstein, S. & Williamson, K., 2006. Understanding Consumer Adoption of Internet Banking: An Interpretive Study in the Australian Banking Context. *Journal of Electronic Commerce Research,* pp. 50-66.

Moodle Org, 2002. *Moodle,* s.l.: s.n.

OWASP, 2013. *Top 10 2013 Web Threats.* [Online]
Available at: https://www.owasp.org/index.php/Top_10_2013-Top_10

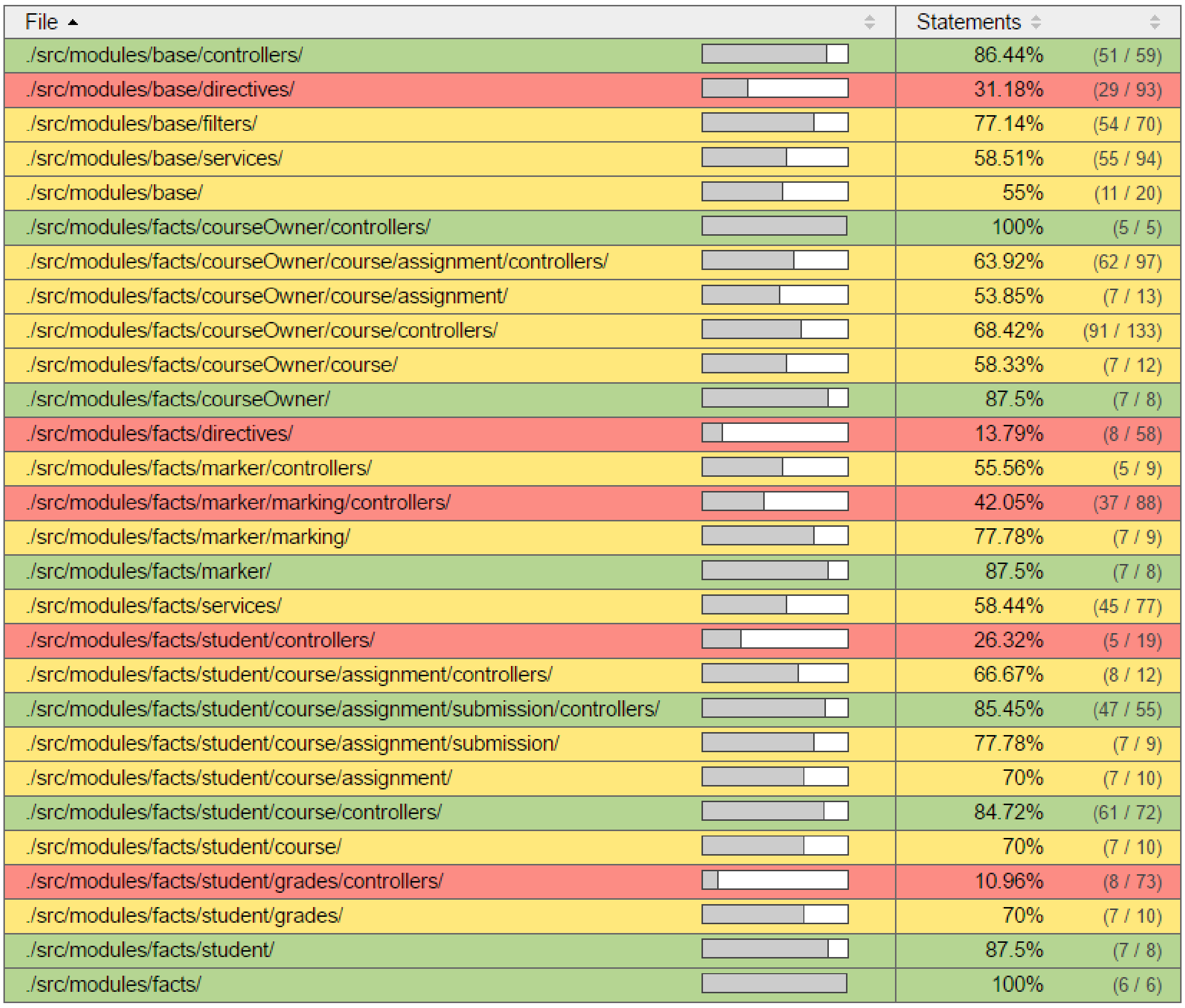
Schneider, S., 2016. *Single-Page vs. Multi-page UI Design: Pros & Cons.* [Online]
Available at: https://studio.uxpin.com/blog/single-page-vs-multi-page-ui-design-pros-cons/

Torvalds, L., 2016. *The mind behind Linux* [Interview] (April 2016).

11 APPENDIX

11.1 Unit Testing Code Coverage Statistics

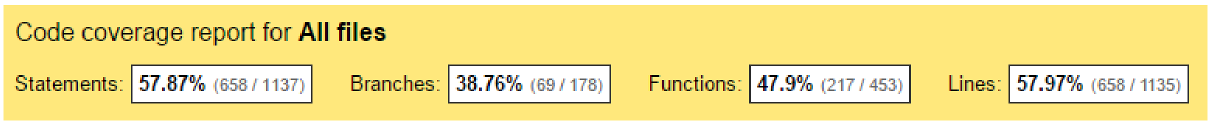
The code coverage statistics for the unit testing of the GUI are displayed in the table below.



File ▲	Statements	
./src/modules/base/controllers/	86.44%	(51 / 59)
./src/modules/base/directives/	31.18%	(29 / 93)
./src/modules/base/filters/	77.14%	(54 / 70)
./src/modules/base/services/	58.51%	(55 / 94)
./src/modules/base/	55%	(11 / 20)
./src/modules/facts/courseOwner/controllers/	100%	(5 / 5)
./src/modules/facts/courseOwner/course/assignment/controllers/	63.92%	(62 / 97)
./src/modules/facts/courseOwner/course/assignment/	53.85%	(7 / 13)
./src/modules/facts/courseOwner/course/controllers/	68.42%	(91 / 133)
./src/modules/facts/courseOwner/course/	58.33%	(7 / 12)
./src/modules/facts/courseOwner/	87.5%	(7 / 8)
./src/modules/facts/directives/	13.79%	(8 / 58)
./src/modules/facts/marker/controllers/	55.56%	(5 / 9)
./src/modules/facts/marker/marking/controllers/	42.05%	(37 / 88)
./src/modules/facts/marker/marking/	77.78%	(7 / 9)
./src/modules/facts/marker/	87.5%	(7 / 8)
./src/modules/facts/services/	58.44%	(45 / 77)
./src/modules/facts/student/controllers/	26.32%	(5 / 19)
./src/modules/facts/student/course/assignment/controllers/	66.67%	(8 / 12)
./src/modules/facts/student/course/assignment/submission/controllers/	85.45%	(47 / 55)
./src/modules/facts/student/course/assignment/submission/	77.78%	(7 / 9)
./src/modules/facts/student/course/assignment/	70%	(7 / 10)
./src/modules/facts/student/course/controllers/	84.72%	(61 / 72)
./src/modules/facts/student/course/	70%	(7 / 10)
./src/modules/facts/student/grades/controllers/	10.96%	(8 / 73)
./src/modules/facts/student/grades/	70%	(7 / 10)
./src/modules/facts/student/	87.5%	(7 / 8)
./src/modules/facts/	100%	(6 / 6)

Figure 1 Specific Unit Testing Code Coverage Statistics

The colour of rows corresponds to the level of coverage offered with red being the lowest and green the most.



Code coverage report for **All files**

Statements: **57.87%** (658 / 1137) Branches: **38.76%** (69 / 178) Functions: **47.9%** (217 / 453) Lines: **57.97%** (658 / 1135)

Figure 2 General Code Coverage Statistics

11.2 User Acceptance Testing Results

11.2.1 Survey Format

The full survey format is displayed below, with detailed responses below that. Questions are displayed as the full question text, the options, if present, in square brackets. Optionality is encoded as a bolding of required questions.

1. **What degree do you do?**
2. **What year are you in? [First, Second, Third, Fourth, Post Graduate]**
3. **What interface are you testing?**
4. Do you have experience with other VLEs? [BlackBoard, Moodle, Geenio, Kornukopia, CaTE]
5. **Is graphical design important to your trust in a website? [Yes, No, N/A]**
6. **Did the design have a consistent feel? [Yes, No]**
    a. **What was inconsistent about the design? Why? (Asked if Q6 was No)**
7. **Did you feel the functionality was intuitive to use? [Yes, No]**
    a. **What functionality did you feel was confusing? Why? (Asked if Q7 was No)**
8. Was there any functionality you felt was missing?
9. Any other comments?

11.2.2 Responses

The responses to the survey are available below, with the questions covered and the information displayed in a tabular or graphical format.

What year are you in?

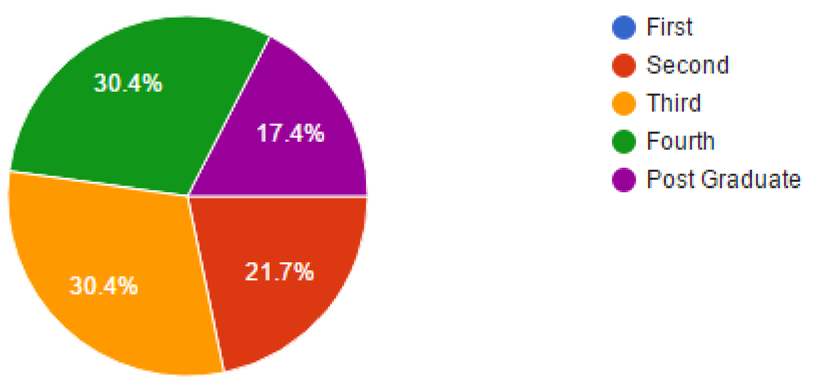


Figure 3 User Acceptance Testing Year Of Study Results

What degree do you do? What interface are you testing?

Degree	Total Responses	Student	Marker	Course Owner
Electrical Engineering	6	5	0	1
General Engineering	3	1	1	1
Politics	3	1	1	1
Aeronautical Engineering	1	1	0	0
Mechanical Engineering	2	1	1	0
Bio Medical Engineering	1	1	0	0
Bio Medical Sciences	1	1	0	0
Chemistry	1	1	0	0
Physics	1	1	0	0
Medicine	1	1	0	0
Biology	1	1	0	0
Medicine	1	1	0	0
Geophysics	1	1	0	0
Total	23	17	3	3

Table 1 User Acceptance Testing Interface Response Results

Do you have any experience with other VLEs?

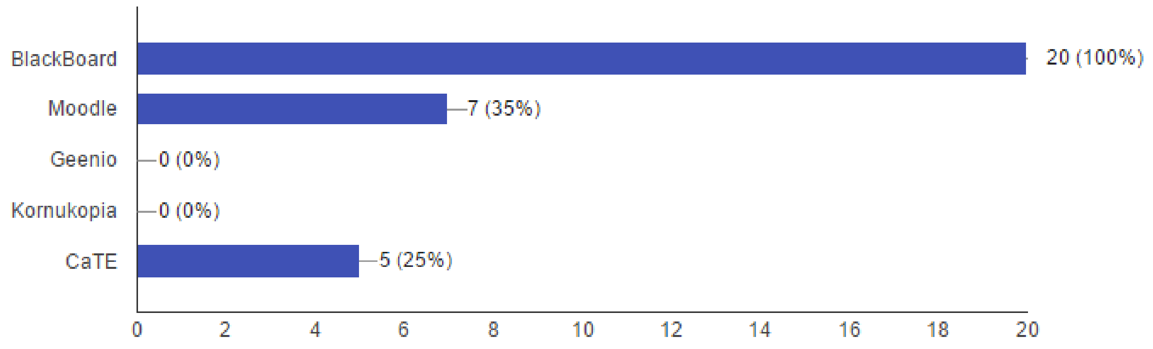


Figure 4 User Acceptance Testing VLE Experience Results (Optional question, 20 out of 23 responded)

Is graphical design important to your trust in a website?

	Yes	**No**	**N/A**
Responses	22	1	0

Table 2 User Acceptance Testing Importance of Graphical Design for Website Trust Results

Did the design have a consistent feel?

	Yes	**No**
Responses	22	1

Table 3 User Acceptance Testing Design Consistency Results

What was inconsistent about the design? Why?

#	**Response**
1	Inconsistent folder organisation for course content, problem sheets etc

Table 4 User Acceptance Testing Inconsistent Design Comment Results

Did you feel the functionality was intuitive to use?

	Yes	**No**
Responses	21	2

Table 5 User Acceptance Testing Intuitive Functionality Results

What functionality did you feel was confusing? Why?

#	**Response**
1	Not clear where work should be submitted
2	Headings, subheadings etc. You click on a heading which theoretically should lead you to more subheadings and then smaller sub-categories within those, but I found that the flow down from heading to smaller categories was muddled.

Table 6 User Acceptance Testing Confusing Functionality Comment Results

Was there any functionality you felt was missing?

#	Response
1	No
2	No
3	No
4	Didn't see course content (maybe i wasnt looking?) Maybe some panopto links or summit
5	Toggling the announcements for each course / going into the course page.
6	I think the course webpage should include a link to the professor's main course page (where the content is), or have a separate section for course contents. Maybe even a short-cut to panopto so it links directly to the lecture videos.
7	Maybe show me how many more things i have to mark - like a countdown, or a task that the leader has set, e.g. mark 25 items and then percentage complete
8	Settings!
9	NO
10	Not really.
11	"Home / Courses " or "Home / Dashboard" - near the top as a navigational tool - instinctively tried to click them. Felt like it would be intuitive to allow clicking Maybe also some form of "upcoming assignments" if already scheduled a la CATE
12	Discussion/student groups, summary of assignments on dashboard
13	It took a while to get used to, ESESIS is a great example of how to categorise everything neatly and in a logical manner.
14	It'd be nice to have a link directly to published exam results, or when they'll be released
15	On the dashboard, under Announcements, didn't seem to be able to click on the announcements to bring up the full text? Otherwise, it seems to have everything I'd want as a student.
16	The only thing I couldn't work out is how I get assigned things to mark. Presumably that's by the Course Owner? But when I checked out that user (survey response incoming), it wasn't clear how I would assign a marker to a given assignment.
17	To assign a marker to an assignment?

Table 7 User Acceptance Testing Missing Functionality Results

Any other comments?

#	Response
1	I like the smoothness of the operation of the site and the transitions of selection boxes. Very elegant.
2	Suprisingly ok on a mobile (s6 has large screen tho). Dont want to have to click on the sidebar to get into a course - inevitably there is nothing interesting on the 'news' ticker (not on bb anyhow) i just want to jump straight into a particular course
3	Initially, I wasn't sure if I was able to click on announcements in a course (i.e. the one in High Performance Computing for Engineers that says Howdy everyone!). It was only after I tried that I realized it was not a link to a page where I could see the announcements for that course, but I'm unsure how that works for a larger announcement.
4	Ordering of announcements is based on last viewed page, which seems confusing when the order seems different when revisiting the dashboard. It feels like the breadcrumb thing, "Home/Dashboard", should be clickable (which is the case when viewing pages with more specific information, but 'Home' is never clickable, and I think it should be removed). I get why it was included, as it just seems random to have 'dashboard' repeated twice. But other than that, the consistency and simplicity of the design makes it easy for the users to intuitively use. These are the FACTS!!!!
5	Nope, These are the FACTS!!!!
6	The only part of this I found slightly strange is that the notifications tab would open with "No Notifications" and this text filled the notifications tab, but it was still possible to scroll down a bit into empty space.
7	The window resized well to keep the design the same even when the page wasn't being shown full screen.
8	Nice work cian
9	These are the FACTS!!!!
10	Like the modular look - even better if I could drag and drop / customize
11	I like CATE better than blackboard, I'd prefer if it was closer to that.
12	I created an assignment (woohoo!) but when I did, I got stuck with a message that said 'submission upload' for a couple of minutes and didn't go away, so I eventually closed the tab. When I logged back into the course owner thing, though, the assignment seemed to have been created just fine. The issue might have been that the deadline was today, so when I went to check on the assignment it said it was already closed - maybe that confused the computer when trying to work out submissions. Otherwise, all seems good!

Table 8 User Acceptance Testing Other Comments Results

11.3 Situational Testing Results

The tables and graphs below are the detailed results of the load tests, with CPU and memory usage of the server during the simulation. Lighter grey text has been excluded from the graphs to remove test initialization and wrap-up artefacts.

	200 Users		400 Users		600 Users		800 Users		1000 Users	
Time	CPU (%)	MEM (%)	CPU (%)	MEM (%)	CPU (%)	MEM (%)	CPU (%)	MEM (%)	CPU (%)	MEM (%)
1	285.6%	5.2%	0.0%	4.9%	346.8%	4.8%	305.3%	4.6%	383.6%	4.4%
2	493.8%	5.2%	295.3%	4.9%	449.5%	4.8%	431.4%	4.6%	388.5%	4.4%
3	487.4%	5.2%	408.9%	4.9%	430.3%	4.8%	458.1%	4.6%	383.0%	4.4%
4	488.8%	5.2%	437.4%	4.9%	393.2%	4.8%	351.1%	4.6%	318.8%	4.4%
5	424.7%	5.2%	430.8%	4.9%	421.3%	4.8%	439.1%	4.6%	436.4%	4.4%
6	397.9%	5.2%	421.1%	4.9%	437.1%	4.8%	431.2%	4.6%	436.3%	4.4%
7	448.1%	5.2%	424.9%	4.9%	372.1%	4.8%	438.2%	4.6%	312.4%	4.4%
8	424.8%	5.2%	425.4%	4.9%	444.7%	4.8%	431.7%	4.6%	431.5%	4.4%
9	441.6%	5.2%	424.9%	4.9%	415.8%	4.8%	415.9%	4.5%	431.3%	4.4%
10	427.9%	5.2%	444.5%	4.9%	443.7%	4.8%	438.3%	4.5%	284.1%	4.4%
11	436.1%	5.1%	437.7%	4.9%	445.1%	4.8%	424.4%	4.5%	436.3%	4.3%
12	438.4%	5.1%	425.1%	4.9%	437.0%	4.8%	431.6%	4.5%	159.2%	4.4%
13	418.4%	5.1%	431.7%	4.9%	432.2%	4.8%	424.1%	4.5%	425.0%	4.4%
14	444.2%	5.1%	423.1%	4.9%	436.6%	4.8%	450.0%	4.5%	437.8%	4.3%
15	431.6%	5.1%	438.0%	4.8%	402.3%	4.7%	429.6%	4.5%	411.1%	4.3%
16	395.9%	5.1%	428.1%	4.8%	424.5%	4.7%	423.2%	4.5%	179.5%	4.3%
17	418.0%	5.0%	437.5%	4.8%	431.5%	4.7%	445.8%	4.5%	411.6%	4.3%
18	425.0%	5.0%	432.0%	4.8%	414.6%	4.7%	418.5%	4.4%	424.2%	4.3%
19	405.2%	5.0%	418.2%	4.8%	429.8%	4.7%	404.0%	4.4%	434.6%	4.3%
20	431.7%	5.0%	423.7%	4.8%	425.1%	4.7%	416.6%	4.4%	442.7%	4.3%
21	330.4%	5.0%	437.6%	4.8%	424.0%	4.7%	424.9%	4.4%	424.2%	4.3%
22	418.3%	5.0%	444.9%	4.8%	437.5%	4.7%	415.8%	4.4%	425.0%	4.3%
23	428.5%	4.9%	422.4%	4.8%	424.4%	4.6%	430.3%	4.4%	405.3%	4.2%
24	425.4%	4.9%	461.1%	4.8%	428.7%	4.6%	431.7%	4.4%	403.2%	4.2%
25	430.9%	4.9%	429.8%	4.8%	424.5%	4.6%	423.9%	4.4%	425.8%	4.2%
26	433.6%	4.9%	431.6%	4.8%	431.7%	4.6%	365.7%	4.4%	430.0%	4.2%

Table 9 POST Load Test Usage Results

	200 Users		400 Users		600 Users		800 Users		1000 Users	
Time	CPU (%)	MEM (%)	CPU (%)	MEM (%)	CPU (%)	MEM (%)	CPU (%)	MEM (%)	CPU (%)	MEM (%)
1	478.0%	7.4%	424.6%	5.9%	352.0%	5.7%	94.7%	5.5%	26.6%	5.4%
2	425.4%	7.3%	418.6%	6.0%	405.4%	5.7%	411.8%	5.5%	245.0%	5.4%
3	418.7%	7.3%	412.0%	6.0%	411.6%	5.7%	405.4%	5.5%	411.9%	5.4%
4	425.2%	7.3%	424.9%	5.9%	405.4%	5.7%	411.9%	5.5%	405.1%	5.4%
5	438.8%	7.2%	411.4%	5.9%	411.8%	5.7%	412.0%	5.5%	405.1%	5.4%
6	436.8%	7.2%	405.4%	5.9%	405.5%	5.7%	405.5%	5.5%	411.9%	5.4%
7	431.4%	7.2%	411.7%	5.9%	398.6%	5.7%	405.3%	5.5%	405.0%	5.4%
8	431.8%	7.2%	411.5%	5.9%	405.2%	5.7%	418.7%	5.5%	417.4%	5.4%
9	418.7%	7.2%	411.5%	5.9%	411.9%	5.7%	398.5%	5.5%	409.9%	5.4%
10	427.9%	7.1%	405.2%	5.9%	412.1%	5.7%	412.3%	5.5%	391.9%	5.3%
11	424.8%	7.1%	402.2%	5.9%	411.9%	5.7%	405.1%	5.5%	403.3%	5.3%
12	425.8%	7.1%	405.3%	5.9%	411.8%	5.6%	411.9%	5.5%	411.3%	5.3%
13	425.2%	7.0%	411.6%	5.9%	410.3%	5.6%	405.2%	5.5%	420.3%	5.3%
14	391.8%	7.0%	403.1%	5.9%	405.2%	5.6%	418.7%	5.5%	405.2%	5.3%
15	418.6%	7.0%	405.2%	5.9%	398.8%	5.6%	405.4%	5.5%	411.8%	5.3%
16	425.1%	6.9%	418.7%	5.9%	405.5%	5.6%	405.0%	5.4%	412.0%	5.2%
17	418.8%	6.9%	411.3%	5.9%	408.0%	5.6%	397.3%	5.4%	418.0%	5.2%
18	425.5%	6.8%	392.2%	5.9%	418.6%	5.6%	403.1%	5.4%	411.9%	5.2%
19	432.1%	6.8%	405.2%	5.9%	411.9%	5.6%	410.2%	5.4%	411.9%	5.2%
20	412.0%	6.8%	405.2%	5.9%	411.8%	5.5%	405.2%	5.4%	405.2%	5.2%
21	417.7%	6.7%	425.5%	5.9%	418.8%	5.5%	412.2%	5.4%	405.6%	5.2%
22	425.3%	6.6%	405.1%	5.9%	405.0%	5.5%	411.9%	5.4%	412.0%	5.1%
23	425.1%	6.6%	412.0%	5.8%	398.5%	5.5%	405.3%	5.4%	405.3%	5.1%
24	424.7%	6.5%	398.5%	5.8%	405.3%	5.5%	405.4%	5.4%	411.8%	5.1%
25	418.5%	6.5%	418.5%	5.8%	412.2%	5.5%	392.1%	5.4%	405.4%	5.1%
26	431.5%	6.4%	405.2%	5.8%	398.7%	5.5%	398.8%	5.4%	405.3%	5.1%

Table 10 GET Load Test Usage Results

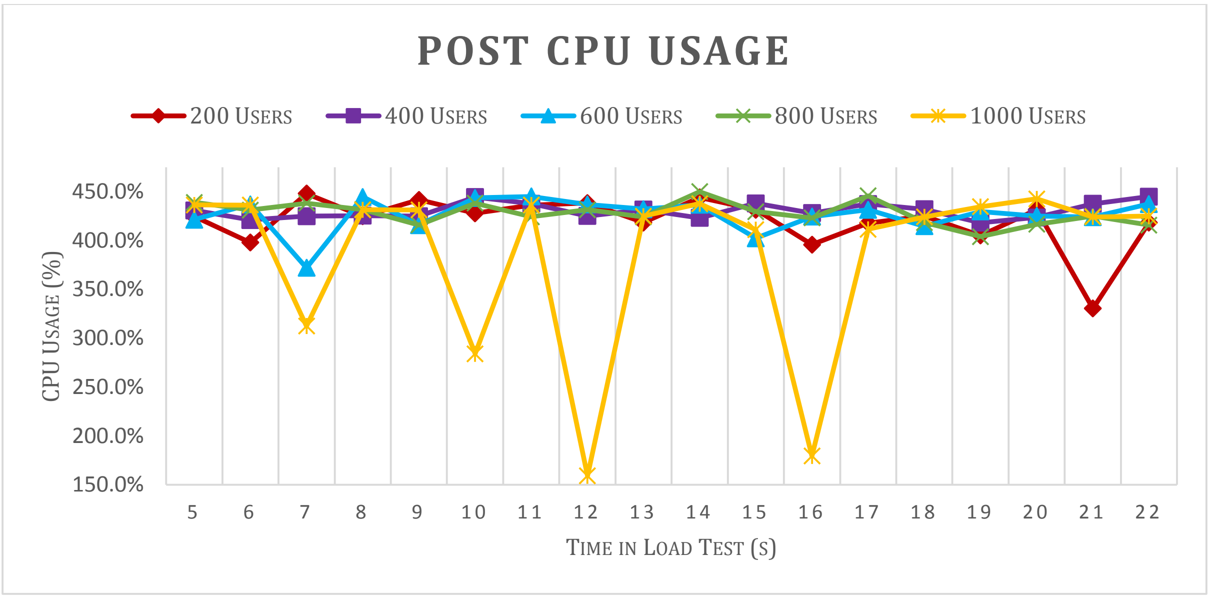


Figure 5 POST Load Test Usage Graph

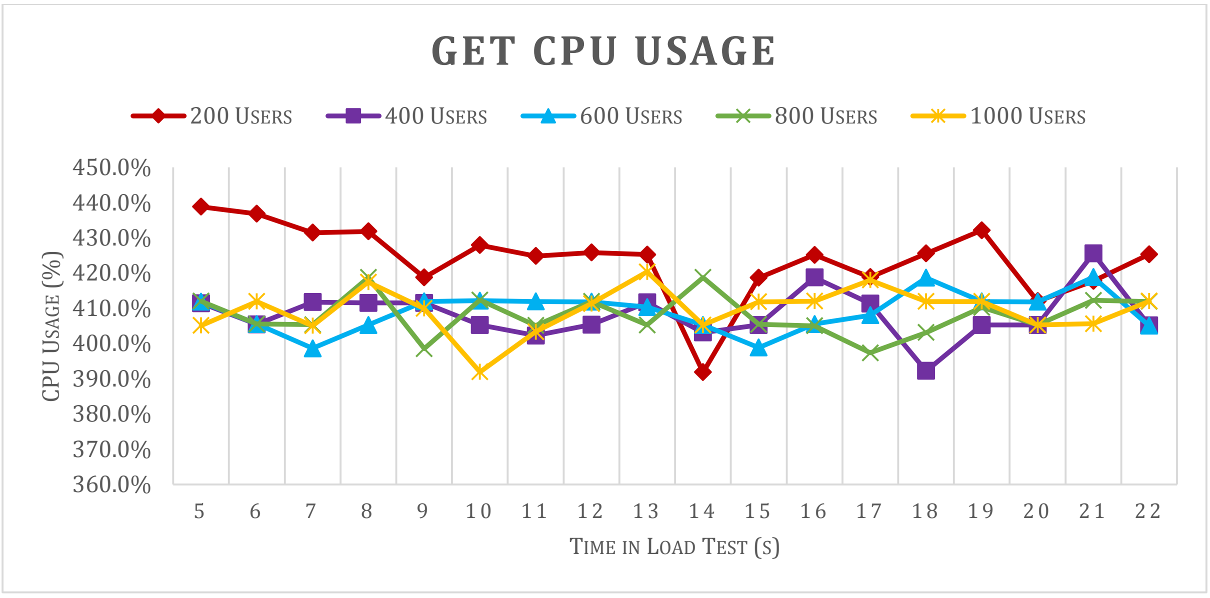


Figure 6 GET Load Test Usage Graph

11.4 Source Code

Source code for the project was managed with Git source-control management software. Code was backed up offsite in private repositories on GitHub. The repositories below contain the source code for the API and user interface respectively.

FACTS API: https://github.com/cianyleow/facts

FACTS GUI: https://github.com/cianyleow/facts-gui

Both repositories can be viewed online at the above addresses or cloned locally with Git tools.

11.4.1 Load Testing Scripts and Commands

The LuaJIT scripts used to execute the load testing are included below. During testing, $TOKEN_VALUE was replaced with a valid pre-generated token to remove the authentication workflow from scripting.

get.lua

```
token = “$TOKEN_VALUE”

wrk.headers[“X-AUTH-TOKEN”] = token
```

Figure 7 LuaJIT Script to Load Test GET Situation

post.lua

```
token = “$TOKEN_VALUE”

wrk.headers[“X-AUTH-TOKEN”] = token

wrk.headers[“Content-Type”] = “application/json;charset=UTF-8”

wrk.method = “POST”

wrk.body = ‘{“title”:”Announcement”,”content”:”An Announcemnt!”}’
```

Figure 8 LuaJIT Script to Load Test POST Situation

These scripts were used by executing the below commands:

GET: `wrk -t 8 -c 200 -d 30s -s get.lua https://DOMAIN/api/courses/1`

POST: `wrk -t 8 -c 200 -d 30s -s post.lua https://DOMAIN/api/courses/1/announcements`