FACULTY OF SCIENCE, ENGINEERING AND COMPUTING

School of *Computer Science & Mathematics*

BSc DEGREE
IN
Computer Science

PROJECT REPORT

Name: Lucas Lawes

ID Number: K1736700

Project Title: Recap

Project Type: Build

Date: 05/05/2020

Supervisor: Paul Neve

Kingston University London

Declaration

I have read and understood the University regulations on plagiarism and I understand the meaning of the word *plagiarism*. I declare that this report is entirely my own work. Any other sources are duly acknowledged and referenced according to the requirements of the School of Computer Science and Mathematics. All verbatim citations are indicated by double quotation marks ("..."). Neither in part nor in its entirety have I made use of another student's work and pretended that it is my own. I have not asked anybody to contribute to this project in the form of code, text or drawings. I did not allow and will not allow anyone to copy my work with the intention of presenting it as their own work.

Date 5th May 2020

Signature Lucas Hawes.

Table of Contents

Abstract 1
Introduction 2
- Aims and Objectives 2
- Scope 3
- Deliverables 3

Literature Review 4
- Student Demand 4
- Adoption Issues 4
- Experience of other HEIs 6
- Existing Offerings 6
- Capability Gap 7

Methodology and Analysis 9
- Work Method 9
- Stakeholders 10
- User Interviews 11
- Positioning 12
- Requirements 13
- Non-Functional Requirements 15
- Prototyping 16
- Project Progress 16

Design 18
- Vision 18
- Target Audience 18
- Design Goals 18
- User Experience 19
- Style Guide 22
- Page Flows 23
- Wireframes 25
  - Portal Home 25
  - Module View 25
  - Lecture Recording 26
- High-Level Design 26
- Database Schema 29
  - Initial Design 29
  - Final Design 30
- Sequence Diagrams 31

Get Portal Context....31
End Lecture Recording....32
Implementation....33
Preliminary Work....33
Inception....35
Elaboration....36
Construction....38
Sprint 1....38
Sprint 2....42
Sprint 3....46
Sprint 4....51
Sprint 5....54
Transition....56
Testing and Evaluation....58
Test Results....58
Evaluation Plan....60
Critical Review....61
Success Criteria....61
Personal Reflection....61
Future Work....62
Aspirations....62
Demonstration....64
Conclusion....65
References....66
Appendix....70
Appendix A – User Personas....70
Director of Learning....70
Lecturer....71
Academic Researcher....72
Engaged Student....73
International Student....74
Motivated Student....75
Appendix B – Impact Map....76
Appendix C – Hierarchical Task Analysis....78
Publish a Lecture....78
Review a Lecture....79
Appendix D – Technology Selection....80

Appendix E – Component Details 82
Recap Client 82
Recap API 82
Mongoose 82
MongoDB Atlas 83
ElasticSearch (and Kibana) 83
Redis 83
Asset Manager 83
Cloudinary 83
Microsoft Azure API 83
Appendix F – Testing Evidence 84
Unit Testing 84
Debugging 84
Back End Testing 86
Integration Testing 88
Front End Testing 89
Browser Compatibility Testing 90
Performance Testing 91
Stress Testing 92
Security Testing 93
Remote User Testing 94
Accessibility Testing 95
Appendix G – Usability Test Pack 97
Student Task Instruction Sheet 97
Lecturer Task Instruction Sheet 98
Observation Sheet 99
Post-Test Questionnaire 100

Abstract

This report describes the work undertaken to build a lecture capture solution for Kingston University (KU) to improve student performance. The literature review shows existing implementations of lecture capture at UK Higher Education Institutions (HEIs) polarise views and are a source of dissatisfaction. Recent surveys found that lecture capture is expensive to implement (Newland, 2017; UCISA, 2019), and the Literature Review concluded there is a capability gap which presents an opportunity to provide an open-source alternative tailored to KU requirements. The project followed a hybrid work methodology combining the Rational Unified Process (RUP) and relevant Agile principles. They provided a framework for realising innovation in iterative steps and close collaboration with the KU client team. Careful attention to the user experience ensured the final product went some way to improving satisfaction and productivity over existing offerings. A High-Level Design provided a robust model for building the solution from the latest digital, web and cloud technologies. It took a fraction of the time and effort to produce a product equivalent to that offered by the market leader, which shows the emerging technology selected has lowered the barrier for entry into the lecture capture market. While the lockdown saw usability testing curtailed, a small set of remote user tests of students provided positive results and feedback. Future work opportunities include enhancing edge cases, integration with a Learning Management System (LMS) and support for live lecture streaming. The project shows it is possible to go some way towards addressing the current satisfaction and adoption issues in a short space of time. The solution is the first-ever lecture capture solution written entirely in JavaScript. This ubiquitous language offers new pedagogical benefits when applied thoughtfully to educational technology.

Introduction

This report describes the work undertaken to provide Kingston University (KU) with a tailor-made lecture capture solution. The project aimed to broaden the use of lecture capture at KU and increase student performance while avoiding adoption issues posed by existing software. Lecture capture is about using digital technology to record a teaching session and disseminate it for later consumption (Gravett, et al., 2020). With over 75% of HEIs in the United Kingdom (UK) using lecture capture in some way (Newland, 2017), it has become a standard component of digital learning. The idea behind lecture capture is that it takes away some of the anxiety associated with the learning process for students (Yorke, 2016; Luke, 2018). Research shows that when students use it to supplement face to face teaching, their performance improves (Nordmann, et al., 2019). The issues with adopting lecture capture include the need for a significant strategic investment, the lack of simple and easy to use tools, and, reservations around the negative impact on student engagement and learning outcomes (Newland, 2017). This report explores the possibility of building Recap as a tactical solution which addresses these issues, is tailored to the requirements of KU and is easy to use.

Aims and Objectives

Global research into the impact of lecture capture shows it to be a divisive technology (Nordmann, et al., 2019), and research at KU in the Summer of 2019 found that any solution should not have a disruptive influence on the relationship between lecturer and student. While student demand for lecture capture was evident in nearly every student consulted, the issues of adoption were more complicated from the perspective of lecturers. The overarching aim of the project was to improve student productivity and performance at KU by providing lecture capture as a supplementary study aid. However, research has shown that lecture capture is not a one-size-fits-all panacea for driving up student engagement and attainment since not all forms of teaching session benefit from it (Evergeti & Garside, 2020). Therefore, a successful approach would also ensure that lecturers decide when they see the benefit of using lecture capture to fulfil teaching or learning needs.

The purpose of the project was to use the time available effectively and build a tailored solution that demonstrates to KU the core functionality of a tactical solution that could then be used by a broader audience. The objectives of the project summarised the steps required to realise a credible set of deliverables and were as follows: -

- Carry out rigorous requirements gathering by interviewing a representative set of KU staff and students.
- Establish a client team to sponsor and shape the project.
- Review existing lecture capture capabilities and ways of working.
- Conduct extensive analysis of the KU requirements.
- Carefully design a fit for purpose user experience, which improves user satisfaction and productivity over existing offerings.
- Research suitable technologies for fulfilling the requirements.
- Produce a medium-fidelity prototype and gather feedback from the client team.

- Design and build a viable solution based on KU requirements and work closely with the client team to improve it.

- Conduct a usability test with Lecturers and Students and elicit feedback for refining the solution.

Scope

Compromises between time, cost and resource constraints shaped the project scope and what was delivered. The time and budget available were enough to demonstrate a viable product without scaling to the KU student population and providing mobile device support. Cloud services offered functionality instead of custom code to ensure on-time delivery. Being an individual project prevented the participation of other people and saw restrictions imposed on access to resources such as the KU LMS. The thrust of the project was to work within these constraints and deliver all the vital software functions to enable a working product. The non-core functionality left out of scope offered enhancements for inclusion in future work.

Deliverables

The following project artefacts were agreed with the client team:

- Project Proposal
- Project Plan
- Client Requirements
- High-Level Design
- Early Working Prototype
- Medium-Fidelity Prototype
- Usability Test
- Working Solution Demonstration
- Final Report

Literature Review

The widespread adoption of lecture capture in both the UK and abroad (Evergeti & Garside, 2020) means there is a wealth of published literature evaluating the experience from other HEIs and this is important to review. The most critical finding is the availability of lecture capture improves access to the teaching session (Gravett, et al., 2020) and is most beneficial for improving student performance when used as a supplement to face to face learning rather than a substitute (Nordmann, et al., 2019). However, lecture capture capabilities are in their infancy, and there is still some way to go to address the concerns and issues raised over the last decade.

Student Demand

HEIs are under pressure to offer lecture capture as a way of recruiting and retaining students (Holmes, 2019), and improve inclusivity for all students. The Project Proposal noted that the current generation of University-bound students, "Generation Z", have a fundamentally different relationship with information than previous generations (Clark, 2019) and see online material as essential study aids. A student survey (N = 118) by Cardiff University showed that 99% of students found lecture capture helpful to some degree (Luke, 2018). The current generation of students is adept at online research and aware of how to self-educate (Clark, 2019). Interviews with students at KU reveal that they expect to consume HEI course content in the same way as provided by YouTube and Massive Open Online Courses (MOOCs). KU has a high number of students who commute, and for whom English is a second language (Linsey, 2019). Lecture capture improves inclusivity for these students (Leadbeater, et al., 2013) as well as students with a disability such as dyslexia (Newland, 2017). A high proportion of the KU student population will, therefore, directly benefit from using lecture capture as a study aid.

Adoption Issues

While the most significant drivers for lecture capture are student expectations and support from senior management (see Figure 1), it tends to polarise academic views (Nordmann, et al., 2019) and along with lack of funding, acts as the biggest barrier to adoption (see Figure 2). Current research tends to concentrate on the positive or negative impact of lecture capture (Evergeti & Garside, 2020) and draws from over a decade of experience. A literature review of 75 studies into the impact of lecture capture found the overall positive findings negate residual scepticism (Panopto, 2020). The key findings were that lecture capture does not negatively impact attendance markedly, and, student access to lecture recordings correlates with higher grades.

Still, the research literature has reported some inconsistent results (Evergeti & Garside, 2020) and there is a need to balance the enthusiasm for the technology by students with the tentative adoption of it by academics (Joseph-Richard, et al., 2018). For example, it is easy to find research which indicates lecture capture has had an overall positive impact on student grades over the last decade (Hove & Corcoran, 2008 to Dommett, et al., 2019) and no compelling evidence of an adverse effect (Nordmann, et al., 2019). At the same time, some studies have highlighted a plethora of adverse effects (Leadbeater, et al., 2013), such as stifling the spontaneity of lecturers (Joseph-Richard, et al., 2018), decreasing student performance due to a lack of self-discipline (Euzent, et al., 2011) and impacting student attendance (Morris, et al., 2019).

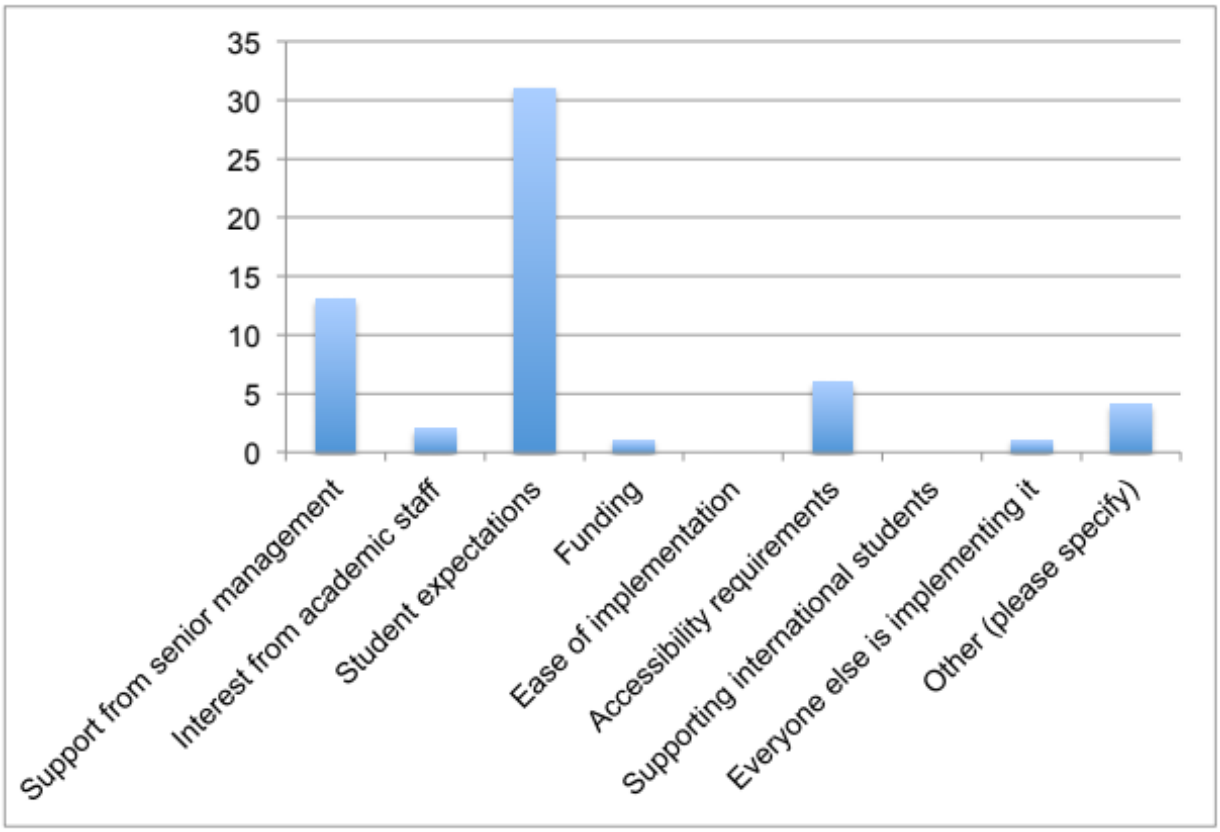


Figure 1 What HEIs perceive as the biggest driver for Lecture Capture (Newland, 2017)

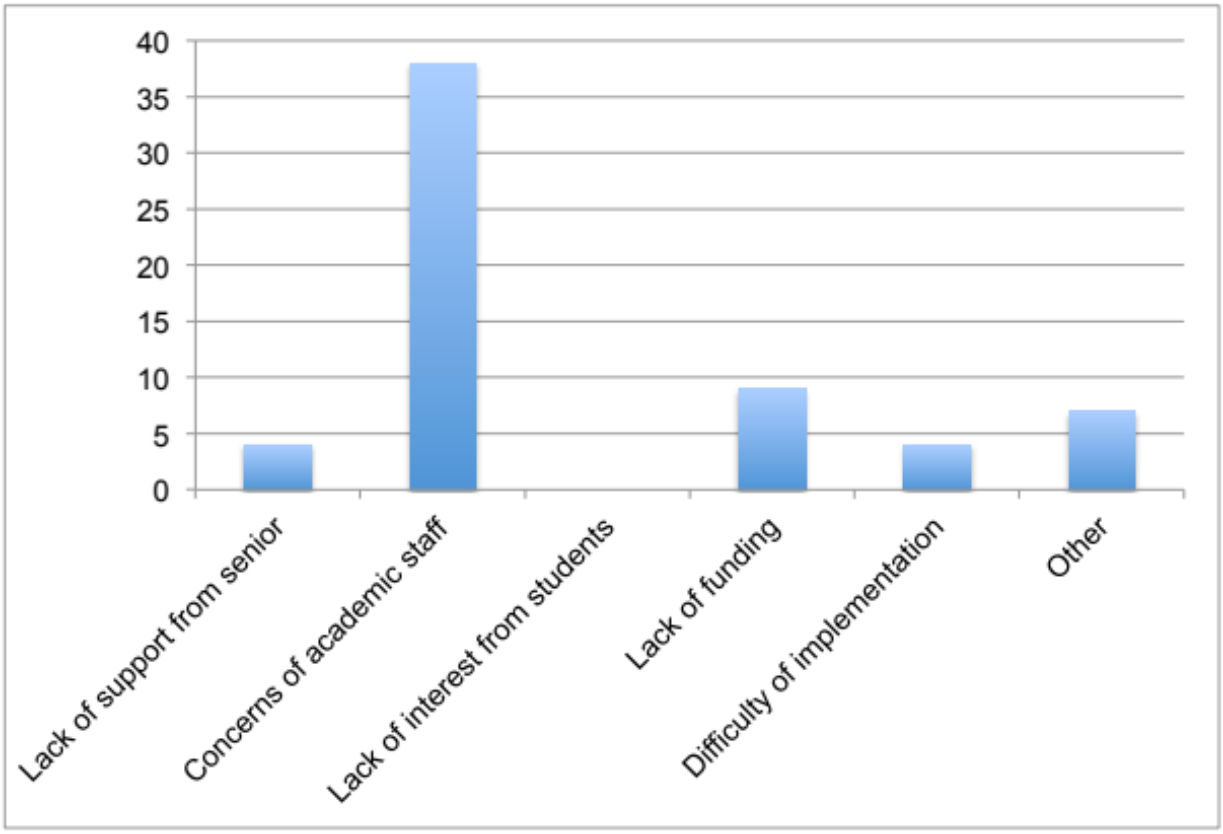


Figure 2 What HEIs perceive as the biggest barrier for Lecture Capture (Newland, 2017)

Much of the literature includes empirical research of teaching and learning in the positivist tradition (Luke, 2020). Recent literature suggests such an approach oversimplifies the multifaceted workings of lecture capture (Morris, et al., 2019). What remains a constant thread throughout the research is that students like to use lecture recordings for learning, assessment preparation and examination revision irrespective of age, gender, enrolment mode or attendance pattern (Dommett, et al., 2019), and, that students benefit from lecture recordings (Caglayan & Ustunluoglu, 2020). While adoption of lecture capture is not straightforward and more research is required into its impact on teaching and learning (Caglayan & Ustunluoglu, 2020), recent research shows that now is a good time for KU to explore the possibility of broadening its use.

Experience of other HEIs

Despite the research findings, academic views remain mixed on the effectiveness of lecture capture to complement educational delivery and active learning (Newland, 2017). What is recognised from the experience of other HEIs, is a single one-size-fits-all, top-down approach by HEIs is inappropriate. It is essential that academics understand the best curriculum design and pedagogical strategies for meeting diverse learning needs and where lecture capture can assist (Morris, et al., 2019). The Heads of eLearning Forum (HeLF) focus on technology-enhanced learning, and their survey findings on the status of implementation at UK universities in 2017 mirrors the conflict of attitudes found in the research:

"Currently, over a third of universities have fully implemented and supported lecture capture, and some have it fully integrated within their VLE and timetabling. However, many others are still in the early days of implementation. One university has decided against using lecture capture for pedagogical reasons" (Newland, 2017)

HEIs are cautious in adopting lecture capture campus-wide with concerns around the risks of infringing statutory rights and mindful of staff concerns (Ibrahim, et al., 2020). When adopting lecture capture, HEIs have a policy choice between an "opt-in" approach which requires staff to instigate a lecture recording before it takes place, and, "opt-out" which assumes the automatic recording of lectures unless staff choose otherwise. Over 50% of universities use the opt-in policy for lecture capture, and only 24% have an opt-out policy (Newland, 2017). Whichever policy is adopted, it is essential to have the consent of lecturers and students, and rather than a blanket approach, acknowledge the variability in lecture capture suitability for different teaching activities (Nordmann & McGeorge, 2018).

The most recent research calls for a more nuanced discussion around how best to use the technology in a way that is pedagogically effective for students and reasonable for staff (Evans & Luke, 2020). When evaluating student satisfaction with Virtual Learning Environments (VLE), lecture capture is the most liked feature across all grade levels (Hamutoglu, et al., 2019). However, there are concerns that students may not have developed the relevant skills (Evans & Luke, 2020) for utilising lecture recordings effectively. Publishing guidance for students is recommended, with particular emphasis on supplementary use and distributed practice; highlighting lecture capture as a constant study resource (Nordmann & McGeorge, 2018). Guidance for Lecturers should focus on education rather than technology (Nordmann & McGeorge, 2018) and address areas of concern, particularly the continuing importance of the live lecture experience over the lecture recording.

Existing Offerings

Lecture capture is gradually becoming a pillar of the digitalisation of the HEI sector on the

assumption that it provides a minimum service provision (Ibrahim, et al., 2020). Nearly 75% of universities in the UK who have implemented lecture capture to some degree use Panopto (Newland, 2017). Panopto see lecture capture transitioning from an optional to a standard component of teaching and learning delivery primarily driven by student demand (Panopto, 2020). Globally, universities have purchased vendor solutions for centralising and managing video content, recording lectures, "flipping" the classroom and integrating video with Learning Management Systems such as Moodle and Canvas. Proprietary software vendors such as Echo360 and Panopto tout video libraries as the future of knowledge management and information retrieval and have seen unprecedented demand for this technology in recent years (Echo 360, 2016). Universities have implemented institution-wide lecture capture systems, using bespoke, open-source and commercial solutions (Morris, et al., 2019).

At KU there have been some innovative pilots by the Faculty of Science, Engineering and Computing involving the use of standalone tools such as Camtasia, KU Media Library, BigBlueButton, ScreenPresso and YouTube Live Stream and publishing lecture recordings to Box. According to Gartner (2018), the critical capabilities of lecture capture are analytics and the capture, creation, modification, and delivery of video. In the more extensive Enterprise Video Content Management space, Panopto and Kaltura are leaders (Gartner, 2018). In the Open Source space, solutions such as OpenCast (https://opencast.org/) provide university-wide lecture capture at scale but lack capabilities for sharing video content with students and requires investment in lecture hall video hardware. Microsoft Stream is a new offering and KU are using the tool for a pilot in 2019/20 to assess the principles of lecture capture and the benefits to students (Linsey, 2019).

Capability Gap

Proprietary software vendors such as Panopto require a strategic investment by HEIs and are not easy to implement successfully. One of the main concerns with lecture capture is that although supplementary use enables students to refresh their knowledge, it is still a passive mode of learning. To address this issue, some vendors include support for flipped classroom and active learning use cases which shrink the value of lecture capture itself in comparison with some other blended learning use cases. However, these solutions offer more than students on traditional lecture-based courses at KU are asking for and make implementation and usability more complicated.

In the open-source space, there are no solutions available which provide the capability to record and deliver lecture recordings. More business-specific proprietary capabilities such as Microsoft Stream are too generic to fulfil a campus-wide educational use case and are currently not easy to customise. This lack of alternatives means if KU were to implement a lecture capture capability, its only sensible choice would be to make a significant strategic investment in an established proprietary software solution such a Panopto or Echo360. These solutions have not addressed the current issues with adoption and satisfaction, and this has an impact on user acceptance and productivity.

Technology has evolved rapidly since the development of existing vendor offerings with the rapidly emerging capabilities of digital, cloud and the web. These capabilities have seen recent releases which significantly reduce the effort required to produce a lecture capture solution to a standard equivalent to existing vendor software. JavaScript has become ubiquitous with building modern web applications quickly, and cloud offerings reduce the ease and cost of adoption. Furthermore, advances over the last decade in the understanding of how to design high-quality user experiences provide new opportunities to address satisfaction issues and

provide productivity gains over existing lecture capture solutions from both an educational and technological perspective.

To conclude this review, emerging technology and better design capabilities create a new opportunity to address dissatisfaction with existing lecture capture solutions. Providing a new solution as an open-source offering would reduce the issues with lack of funding while also providing a cloud offering would enable campus-wide adoption on an as-needed basis. Proprietary software solutions still offer advantages with additional functionality and support. However, the advances in technology should ensure a lecture capture solution is possible to realise in a shorter length of time than these existing solutions initially took to build. The scope of the work effort for the final year project will be about enabling the core concepts of recording and delivering lecture recordings in an easy to use and productive manner. Once this is proven, future work can go into enhancing the offering and making it ready for a campus-wide implementation.

Methodology and Analysis

There was no previous work to refer to for producing the lecture capture solution using new technology and so the selected project methodology needed to facilitate innovation and experimental design. Furthermore, the methodology also needed to enable a rigorous analysis effort in regular and close collaboration with the client team to tailor the solution to KU requirements.

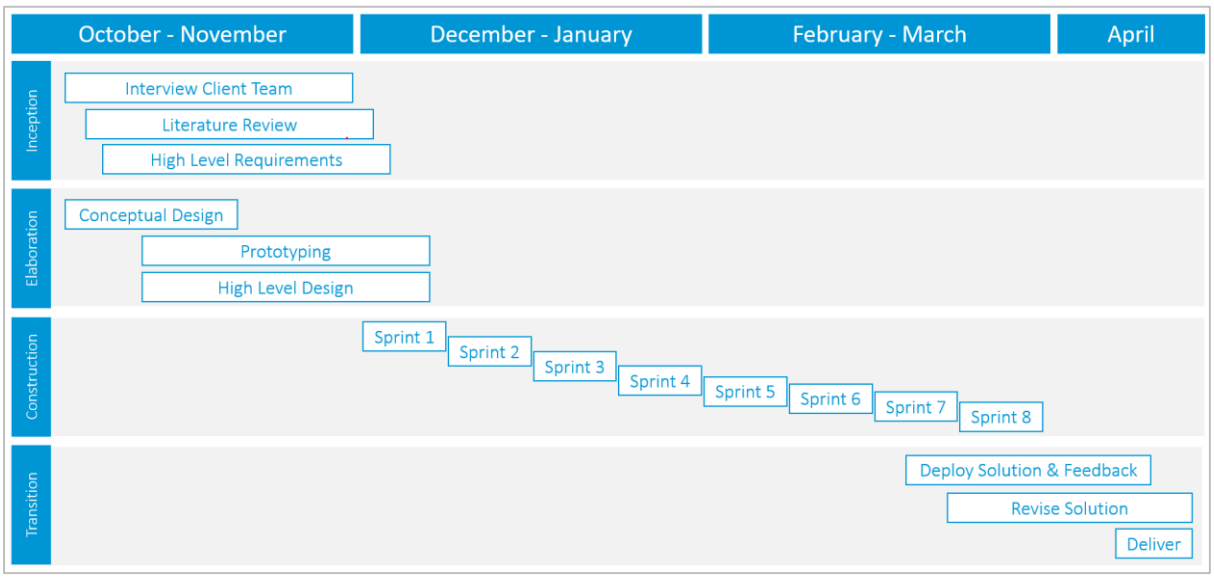
Work Method

The discrete stages of Waterfall loosely follow the outline of the project report and at first glance lent themselves as the natural process steps to follow for the project. However, the empirical nature of innovative solution design meant the project followed a hybrid approach. The Rational Unified Process (RUP) was adapted to use relevant Agile principles and some practices of Scrum. RUP provides an iterative process framework which lessens risk and focuses on developing software (Anwar, 2014). However, client involvement is an influential factor in project success (Tam, et al., 2020) and Agile leads to better communication with the client (Kruse, 2020) and frequent feedback (Petersen & Wohlin, 2009). On the other hand, Agile lacks focus on architectural design (Petersen & Wohlin, 2009) and a set of RUP style High-Level Design artefacts were maintained. While Scrum practices are worth adhering to on a large project, their practice on a final year project is unrealistic since, for instance, it is not possible to work in daily collaboration with the client. Therefore, daily stand-ups and retrospectives are not possible, but it is possible to rely on the concepts of having a Product Backlog, Epics and Sprints with inspection by the client. Like the Critical Path Method, RUP helps with the scheduling and prioritisation of activities, and from a technical perspective, was used to de-risk the project and deliver the critical success factors upfront. The Dynamic System Development Method (DSDM) relies on several proven practices which were used, such as MoSCoW prioritisation. Aspects of the Agile principles were considered essential, such as welcoming change throughout development and delivering working software often so that feedback could be given regularly.

RUP proved a solid foundation for delivering the final product on time while also facilitating a pragmatic approach to overcoming technical challenges early on. Adapting RUP to be "agile", by adhering to relevant principles ensured the quality of the final product. The project followed some aspects of Test Driven Development, such as writing tests first and code refactoring, but practices like Continuous Integration were not adopted. RUP divided the process into four phases, as shown below, and provided a lifecycle which reflected the learning and delivery curves.

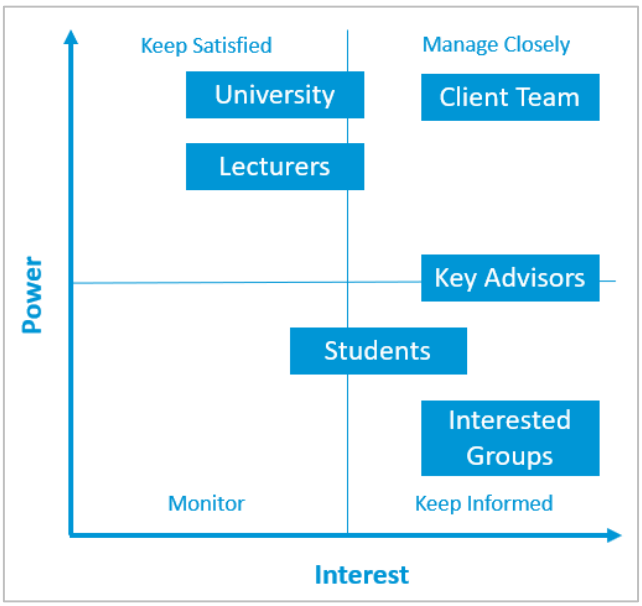
Phase	Work Undertaken	Period
Inception	Problem and scope definition, literature review, early solution prototyping, idea circulation, user interviews, client team formation	October – November
Elaboration	Problem domain analysis, high-level design, requirements gathering, analysis and signoff	November – December
Construction	Development sprints and client review meetings	December – February
Transition	Usability testing, refinement sprints and deployment	March – April

Sprints divided the development effort into two-week timeslots, with a weekly client meeting (except during the December holiday). Complete and accurate communication with the client occurred as frequently as possible. The length of sprints included a degree of flexibility to accommodate other university work, and although 8 sprints in total were planned, only 5 sprints were necessary. The initial high-level project plan, shown below, blended the delivery of RUP based design artefacts with Scrum style sprints and was followed throughout the process.



Stakeholders

Close collaboration with interested parties was essential to deliver a solution tailored to the requirements of KU and its staff and students. The identification of different groups of stakeholders who had an influence, power, or interest in the success of the project helped shape the communication and engagement approach. A Power/Interest Grid, shown below, was produced to prioritise each group and come up with an engagement strategy.

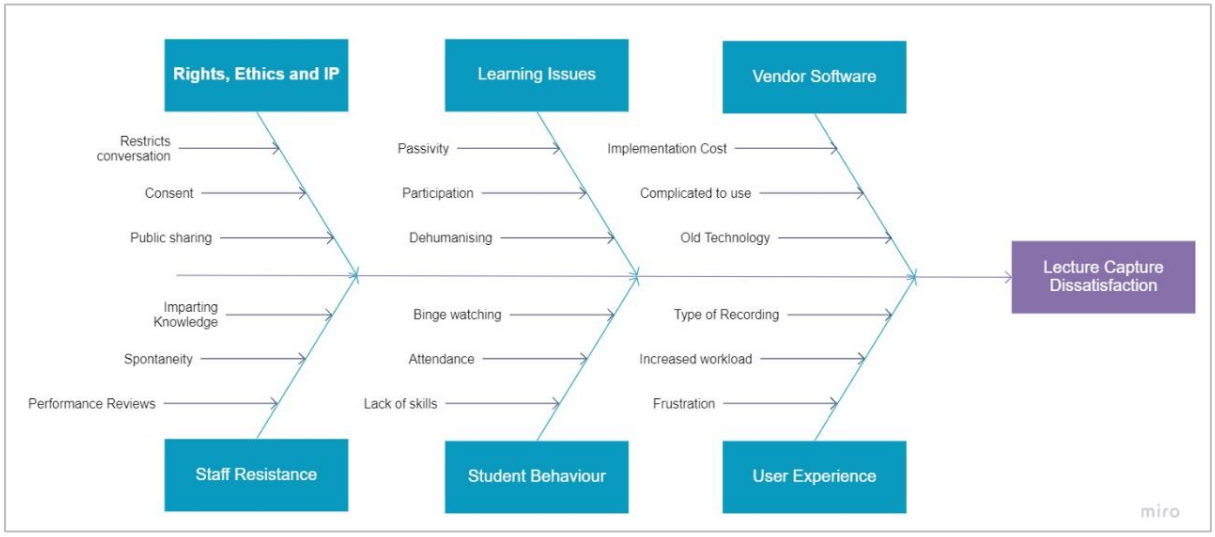


The Client Team were the champions of the projects, instrumental in guiding its success and were fully engaged. The Key Advisors to the project were interested in providing input into the

project, whether that was giving encouragement, insights, tips, or suggestions. However, they were not part of the Client Team and only needed to be kept informed. Interested Groups that are people who wanted to see the project succeed but had little or no influence over it were adequately informed. Students, in general, had high hopes for the project and were interested in its success. They kept engaged and their suggestions and ideas fed back into the design to ensure they liked the final product. The University and Lecturers had a considerable influence over the use of the lecture capture solution, and they stayed in the loop and asked about progress.

User Interviews

The circulation of the initial project idea with KU lecturers and students began at the end of the 2018/19 academic year without first reading the literature about the experiences of other universities. The results of the initial conversations about lecture capture were, therefore, a surprise. On the one hand, students were extremely positive and enthusiastic about the possibility of having access to lecture recordings. While on the other hand, lecturer demand for lecture capture was non-existent, and the project idea in some circles flatly rejected. The response, to begin with, was to shelve the project idea until some background reading highlighted that the polarisation of views was typical across HEIs. It identified an opportunity to address the causes of dissatisfaction, shown below.



A series of interviews with the university, lecturers, and students at the start of the 2019/20 academic year uncovered something many users were experiencing with lecture capture elsewhere – that there is an enormous gap between good idea and reality. The recent literature found the last decade of lecture capture adoption driven by a desire to modernise HEIs (Ibrahim, et al., 2020) rather than an academic enquiry into how to exploit the emerging technology for improving educational outcomes. However, this approach garnered a wealth of experience, and, new recommendations are now beginning to emerge on how to exploit lecture capture best.

Spending time with potential users and stakeholders opened possibilities for addressing the issues with existing lecture capture and generated recommendations for designing an overall solution. These are recorded in the table overleaf.

Cause	Issues/Concern	Recommendation
Rights, Ethics and IP	Restricts conversation	Lecturers are encouraged to design sessions which engage Students and to avoid/pause recordings where necessary
	Consent	An opt-in policy is the most popular approach with Lecturers. Students should be able to ask questions without being recorded.
	Public sharing	Fair usage policies should restrict the sharing of lecture recordings.
Staff Resistance	Imparting knowledge	Face to face teaching sessions should take priority over lecture recording
	Spontaneity	Provide pedagogical support
	Performance Review	Present a policy with clear statements
Learning Issues	Passivity	Promote supplementary use to review the material
	Participation	Avoid/Pause recording where participation is more important than recording
	Dehumanising	Provide pedagogical support
Student Behaviour	Binge Watching	Promote deep learning over time by using recordings as a continuous study aid
	Attendance	Guide for Students to attend lectures and evidence to Lecturers attendance is showing there is generally no impact on it.
	Lack of Skills	Promote best practices for learning
Vendor Software	Cost of Implementation	*Design and build a solution using the latest open-source software*
	Complicated to use	
	Old Technology	
User Experience	Type of Recording	Provide voice-over-presentation recording for teaching sessions, room recordings for talks and speeches.
	Increased workload	*Conduct usability testing to prove the user experience improvements*
	Frustration	

Positioning

The series of interviews held during the inception phase of the project elicited the initial requirements for the project. SCOPE Analysis, shown overleaf, assisted in forming a clear understanding of the situational context in which the project existed. High demand from students for lecture capture was juxtaposed with dissatisfaction with existing offerings and ways of working. By taking advantage of the core competencies and overcoming obstacles, an opportunity emerged to provide an open-source base solution and a cloud-hosted service offering. In the future, there is likely to be increasing demand for lecture capture, improvements in implementation success and maturity in the market proposition. All in all, the findings showed there is a potential gap in the market for a new solution which addresses the concerns

of HEIs and provides an open-source alternative to proprietary software.

SCOPE Situational Analysis

Situation	Core Competencies	Obstacles
High demand from Students Dissatisfaction with existing offerings	Use of free, proven, open-source software for base solution Use of the latest state of the art software and techniques Engineering Excellence	Resistance to adoption Lack of funding

Prospects	Expectations
Provide an Open-Source base solution any HEI can download Provide a cloud hosted service offering to HEIs on a subscription basis	Increasing demand for Lecture Capture Market will mature Improved implementation success stories

Requirements

The candidate requirements were documented as User Stories and discussed with the Client Team who gave feedback for refining them and approved the final version. Epics categorised the User Stories, and the client team prioritised them using the MoSCoW method. There were 4 Epics in all, and these were: -

- **Capture** – to enable Lecturers to capture lecture recordings so they may be shared with and consumed by students.
- **Manage** – to enable administrators to manage lecture recordings so they can be optimised, stored, and accessed by lecturers and students accordingly.
- **Report** – to enable learning analytics to be reported to interested parties.
- **Consume** – to enable students to be able to discover and view lecture recordings.

The User Stories agreed with the client team are documented in the table below.

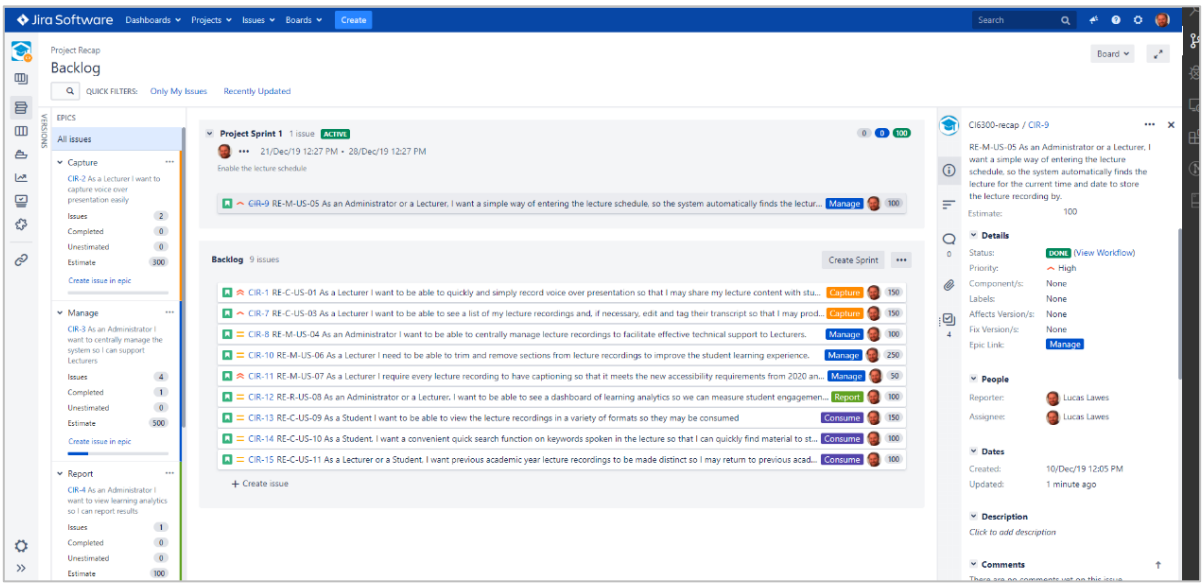
Epic	User Story		Priority
Capture	RE-C-US-01	As a Lecturer, I want to be able to quickly and simply record voice-over presentation so that I may share my lecture content with students for viewing outside the classroom setting.	Must-Have
	RE-C-US-03	As a Lecturer I want to be able to see a list of my lecture recordings and, if necessary, edit and tag their transcript so that I may produce content which is easy for students to retrieve.	Should-Have
Manage	RE-M-US-04	As an Administrator, I want to be able to centrally manage lecture recordings to	Could-Have

Epic	User Story		Priority
		facilitate effective technical support to Lecturers.	
	RE-M-US-05	As an Administrator or a Lecturer, I want a simple way of entering the lecture schedule, so the system automatically finds the lecture for the current time and date to store the lecture recording by.	Should-Have
	RE-M-US-06	As a Lecturer, I need to be able to trim and remove sections from lecture recordings to improve the student learning experience.	Could-Have
	RE-M-US-07	As a Lecturer, I require every lecture recording to have captioning so that it meets the new accessibility requirements from 2020 and so that the material is accessible to all students	Must-Have
Report	RE-R-US-08	As an Administrator or a Lecturer, I want to be able to see a dashboard of learning analytics showing the number of lecture recordings and the viewing statistics for whole lectures and their constituent parts so that we can measure student engagement and progression.	Could-Have
Consume	RE-C-US-09	As a Student, I want to be able to view the lecture recordings in a variety of formats so they may be consumed at University, on the go, from home, from inside any virtual learning environment and on any device, so I may catch up on topics I have missed, study complex topics and revise for my exams.	Must-Have
	RE-C-US-10	As a Student, I want a convenient quick search function on keywords spoken in the lecture so that I can quickly find material to study.	Should-Have
	RE-C-US-11	As a Lecturer or a Student, I want previous academic year lecture recordings to be made distinct from current academic year lecture recordings so I may return to previous academic years, but I am presented with the current academic year by default.	Could-Have

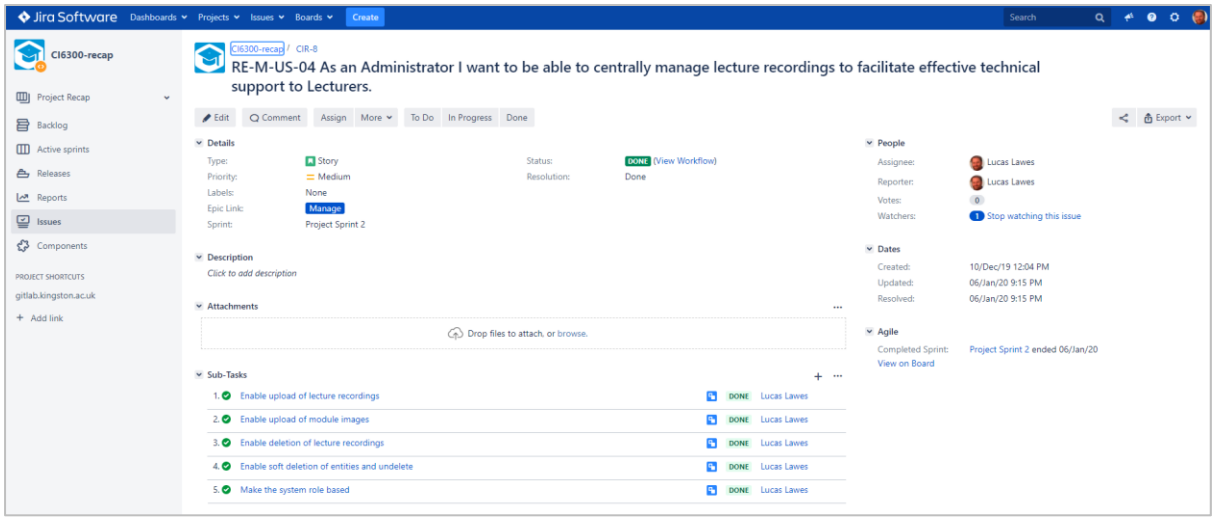
The User Stories were written to be short and descriptive but also left open to interpretation. As development progressed, the client team discussed them and gave feedback for refining the solution. In the early stages, medium-fidelity prototypes supported these discussions, and as soon as the solution emerged, working demonstrations of the product enabled feedback. This process helped improve the quality and suitability of the final product.

Producing a set of High-Level requirements proved to be very useful and an efficient way of capturing requirements over using more prescriptive RUP style Use Cases. It led to a design effort that focused on maximising the benefit that features offered to users rather than detailing

functional requirements that would later change and must be updated. Each User Story was estimated using Story Points and assigned its Priority, which helped inform the critical path. A Project Backlog input in Jira included the Epics and User Stories, shown below.



Significant User Stories were broken down into sub-tasks, as shown below for RE-M-US-04.



Non-Functional Requirements

The project constraints essentially meant the product was a "Proof of Concept" showing the potential for using it in a production environment. Several non-functional requirements specified how the product should behave within these constraints.

- **Scalability** – the production system must eventually scale to serve the student population of KU, approximately 20,000 students. So, the High-Level Design must factor this in and provide components which can be scaled to meet demand later.

- **Capacity** – the system must be able to record, store and retrieve lecture recordings of 2 hours in length, which is the average length of a lecture at KU.

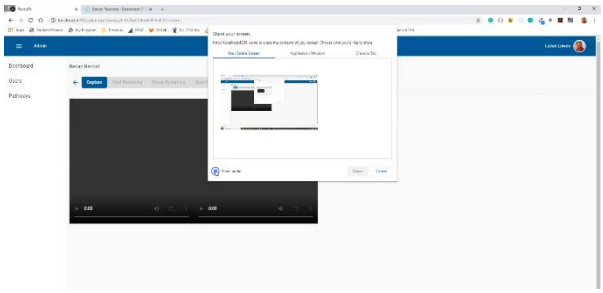
- **Availability** – the system must be available on the internet 24/7 to make lecture recordings accessible to students whenever they need them.

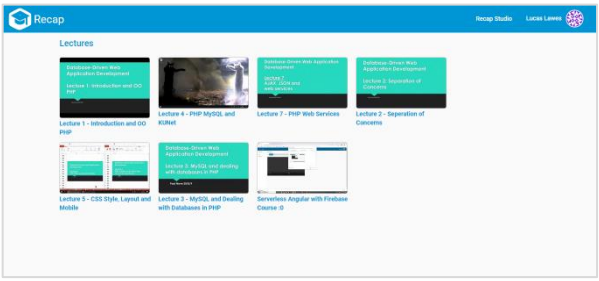
- **Security** – the system must enable Lecturers and Students to authenticate using their KU credentials.

- **Maintainability** – the system must be easy enough for academic staff to maintain without the support of KU IT.

Prototyping

The Project Proposal identified significant project risks and mitigation. Technical risks were also mitigated during the Inception and Elaboration phases of the project. These included ensuring the solution would be able to record, store and share lectures using new techniques. An initial prototype tested the ability to capture the screen and audio necessary for recording a lecture, and the early prototype demo focused on testing the ability to store and share lectures.



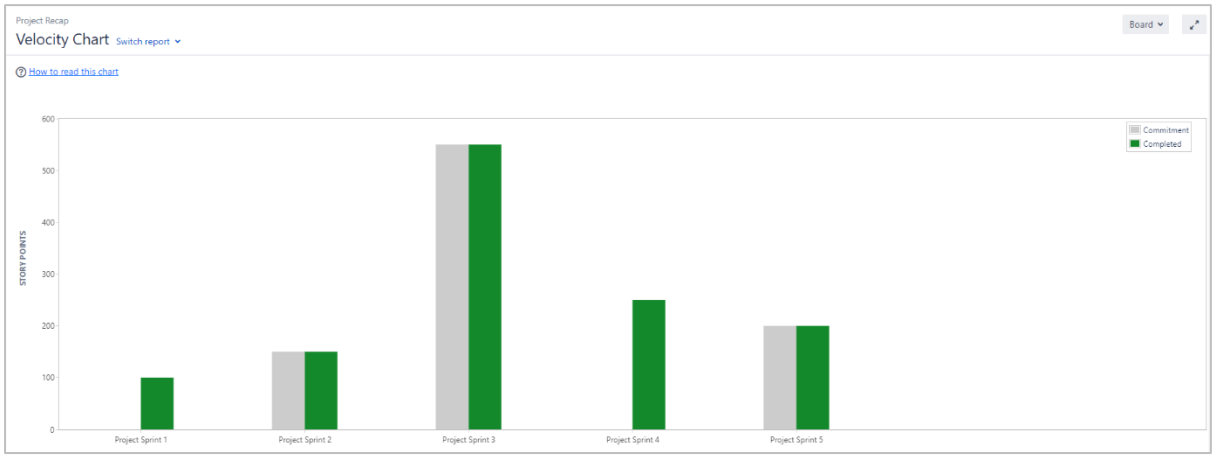


Project Progress

The construction phase commenced in December 2019 and included 5 sprints. The amount of Story Points in a sprint varied as did the length of time taken to complete a sprint.

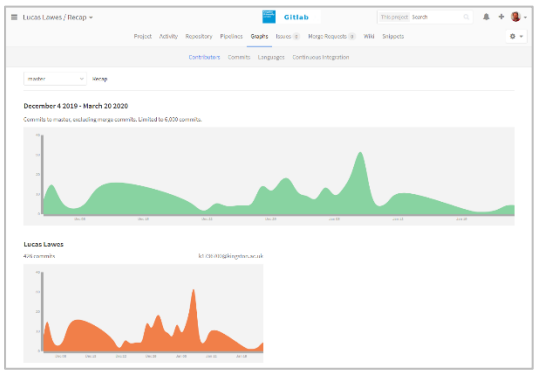
Sprint	Goal	Dates
1	Enable the lecture schedule	21/12/2019 – 30/12/2019
2	Enable lecture recordings and captions to be linked to the lecture schedule	30/12/2019 – 06/01/2020
3	Enable lecturers to make, manage and view lecture recordings	06/01/2020 – 12/01/2020
4	Enable students to be able to view lecture recording easily	12/01/2020 – 28/02/2020
5	To finalise and steady to build	28/02/2020 – 05/03/2020

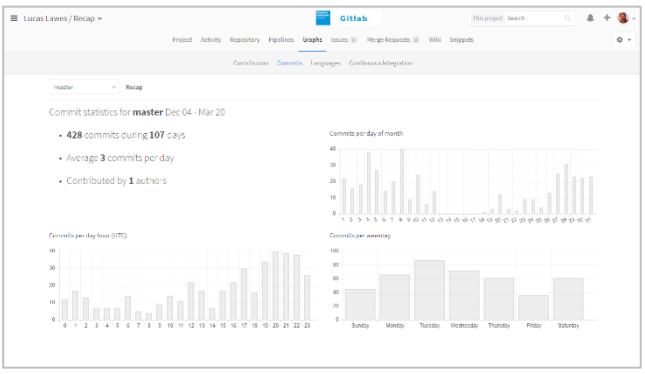
In Scrum methodology, a Sprint is timeboxed. Nevertheless, making the length of time taken to complete a sprint variable was a perfectly workable approach to take given that there were periods during the project where more work could be undertaken than at other times, as shown in the Velocity Chart below.



https://www.kuprojects.org/secure/RapidBoard.jspa?projectKey=CIR&rapidView=141&view=reporting&chart=velocityChart

Gitlab also handled source code management, and the commit history graphs show the variability in work effort throughout the construction phase between December 2019 and March 2020. The total work effort to deliver the four Epic requirements involved 428 commits over four months.





https://gitlab.kingston.ac.uk/K1736700/Recap/graphs/master

Design

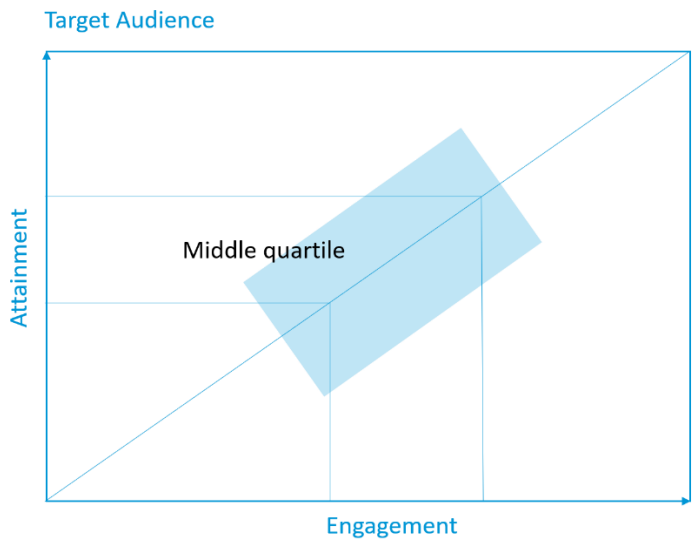
The product name "Recap" came about as the term reinforces the crucial notion of using lecture capture as a supplementary study aid. The design scope focused on improving the user experience and increasing the productivity of core functionality over existing offerings. The adoption of user-centred design enabled user requirements to be more clearly understood, which is vital since it leads to improved usability and user satisfaction (Salah, et al., 2014).

Vision

In the future, the idea for lecture capture is to become an indispensable tool for supporting the student learning process, in the same way that Learning Management Systems have proven to be (Neve, 2016). KU currently has several ad hoc lecture capture initiatives and what came out of the research and analysis was a future vision of a unified, campus-wide capability which helps inspire students to reach their full potential.

Target Audience

To define the target audience for Recap, a member of the client team drew an analogy between student attainment and engagement. The notion was that high performing students would not have much to gain from lecture capture, while low engaging students would either not use it or achieve no significant improvement from using it. The target audience were those in the middle who were motivated, engaged and wanted to attain a higher attainment outcome. While there was no specific research to support the notion, it made much sense and mirrored the experience of mentoring undergraduates.



Design Goals

The design goals addressed the challenges experienced by other HEIs wanting to use lecture capture to improve student outcomes. They were to:

- **Increase lecture capture usage** – research showed that lecture capture offers many benefits when done well, and the ultimate design goal was to enable the use of lecture capture to become widespread at KU.

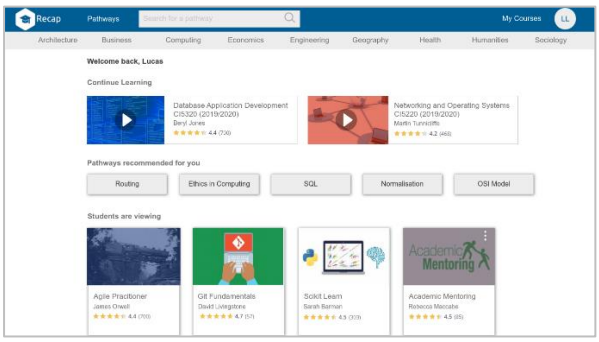
- **Improve user satisfaction** – research showed significant dissatisfaction with existing tools, and taking advantage of the experiences of other HEIs was a design imperative.

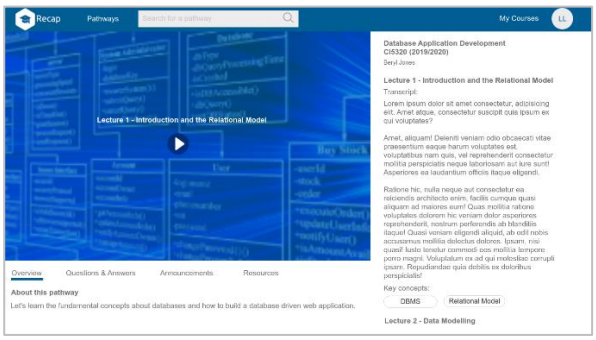
- **Encourage continuous learning** – research also showed that lecture capture works best as a study aid for students when used a little and often.

- **Enable deep learning** – one of the recommendations from the analysis was to enable students to be able to see an overview of lecture recordings and review specific content, e.g. to enable them to go over a complex concept from multiple perspectives.

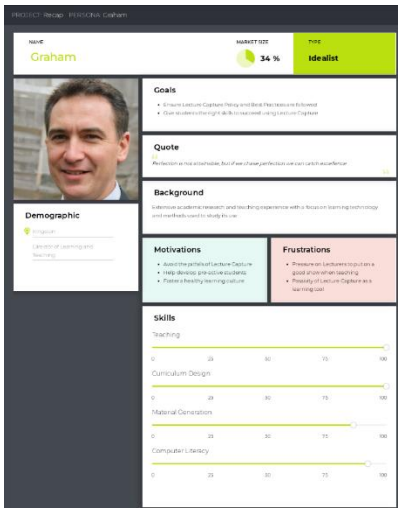
User Experience

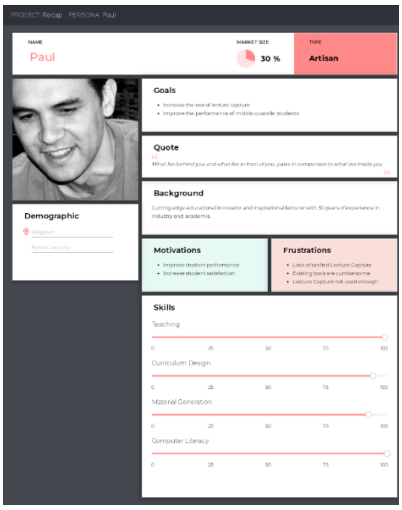
During the Inception and Elaboration phases, medium-fidelity wireframes conveyed the initial idea for the solution and offered a means of eliciting requirements from user interviews.

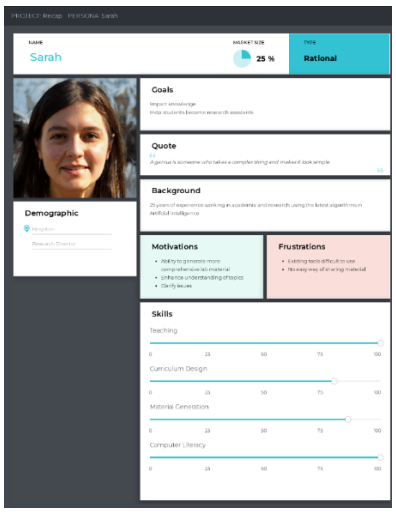


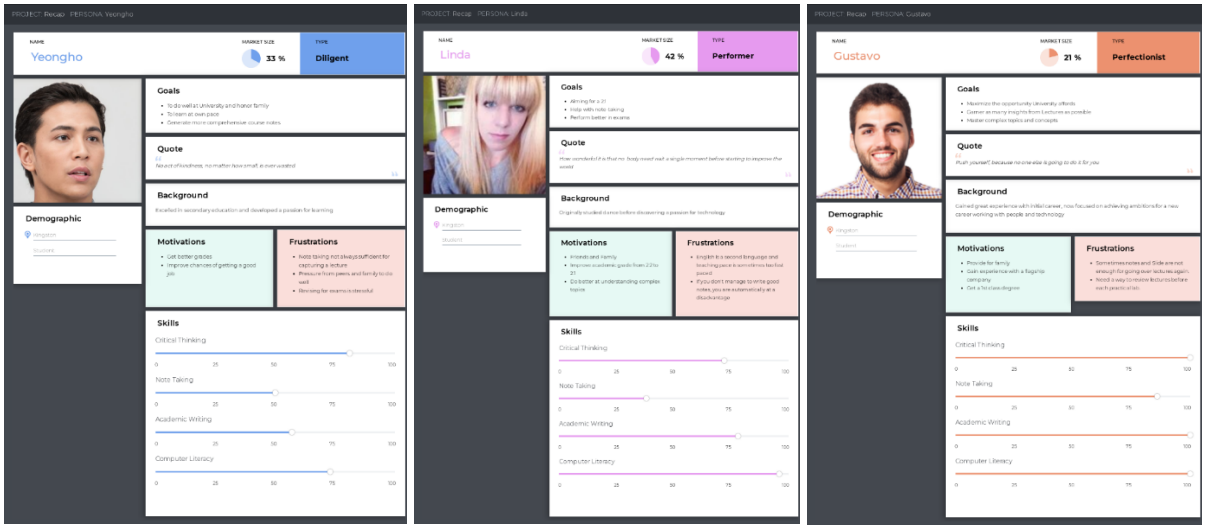


The knowledge collected from user interviews was input into a human-centric design effort which began by creating User Personas which provided both the critical perspectives for conceptualising the target audience and motivation for building the product (see Appendix A – User Personas).







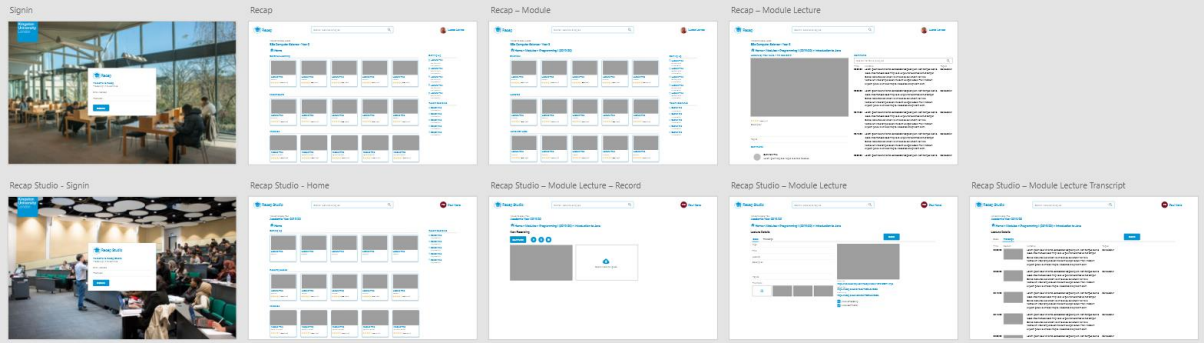


The User Personas helped define the solution. By using six User Personas, the design created harmony between different perspectives to constitute the best conditions for enhancing students learning outcomes. The User Personas reflected six personality types across the roles of staff and student.

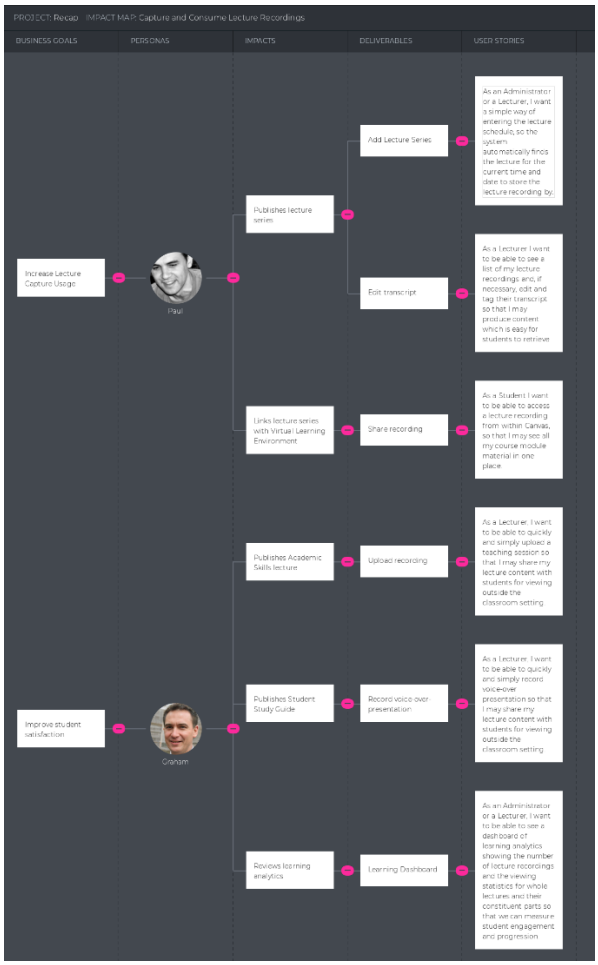
Personality Type	Role	Needs
Idealist	Director of Learning	To ensure lecture capture policies and best practices and equip students with the right skills to succeed.
Artisan	Lecturer	To create relevant and satisfying lecture and supplementary material to support the learning of students, particularly those in the middle quantile of academic performance.
Rational	Academic Researcher	To impart specialist knowledge to students in interesting and engaging ways.
Diligent	Engaged Student	To benefit from self-paced learning and make more comprehensive lecture notes.
Performer	International Student	To improve academic attainment by producing detailed lecture notes and using these to perform better at examinations.
Perfectionist	Motivated Student	To gain a deeper understanding of complex concepts to maximise learning outcomes.

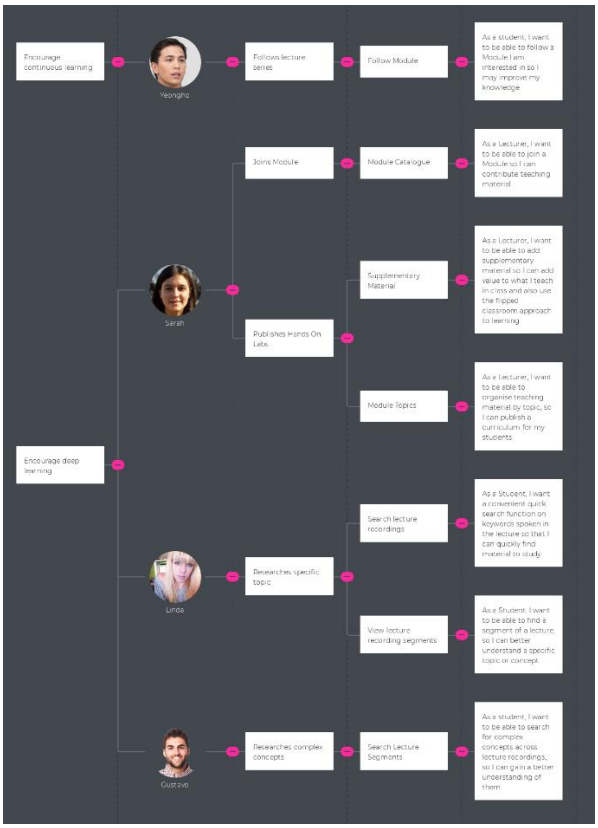
Wanting to learn is central to learning (Ellington, et al., 1993) and particular attention went into ensuring the solution motivated students with the opportunity for self-directed learning. Low-fidelity wireframes of a portal (shown overleaf) fed a design discussion with the client team about how Students would access lecture material, module curriculum and be able to navigate lecture narrative. The design discussion also considered how Lecturers would make lecture recordings available and support in-depth learning across lectures, modules, and academic years. The use of familiar terminology such as “module”, “lecture” and “topic” used in the KU LMS, bore out an immediately recognisable narrative for elements of the system. In addition to the LMS, the initial wireframes for Recap blended visual cues from other systems familiar to students such as YouTube, Udemy and LinkedIn Learning. The discussions helped

generate and select ideas that would improve the user experience. For example, ideas such as student ratings and comments for lecture recordings were dropped while ideas such as the ability to search and download lecture transcripts were kept.



An Impact Map (see Appendix B – Impact Map) documented the outputs from the design discussion, and this replaced a traditional RUP based Use Case Diagram. This approach had the benefit of producing a much more user-centred design which mapped User Stories to user needs and high-level design goals. Plotting the Impact Map identified gaps in the requirements, and User Stories were broken down to support, for example, modules having topics and supplementary material.



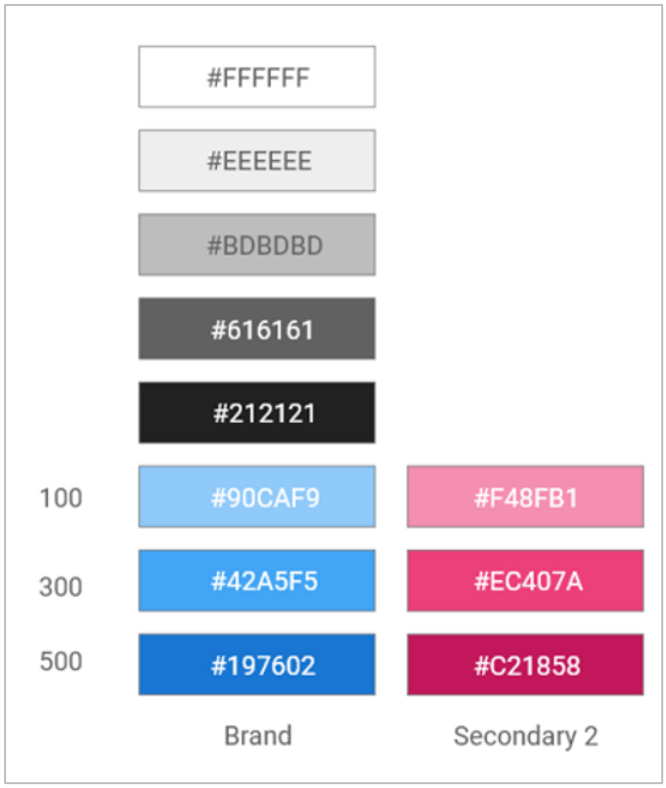


Hierarchical Task Analysis was used to analyse the steps Lecturers and Students would have to perform to undertake tasks (see Appendix C – Hierarchical Task Analysis). Particular attention focused on how task completion would be efficient and productive while minimising errors and frustrations. An example was the ability to record a lecture in an environment where Lecturers would be multi-tasking and would, therefore, need interaction steps to be as short and as straightforward as possible. Analysis effort also went into the consideration of how to hold the attention of students and encourage them to engage with the learning material. The inclusion of a task step for reviewing a transcript enabled students to assimilate an overview quickly and decide whether a lecture recording was relevant to their study needs. Furthermore, a note-taking step provided students with the ability to easily sift through the narrative and identify relevant topics to understand.

Style Guide

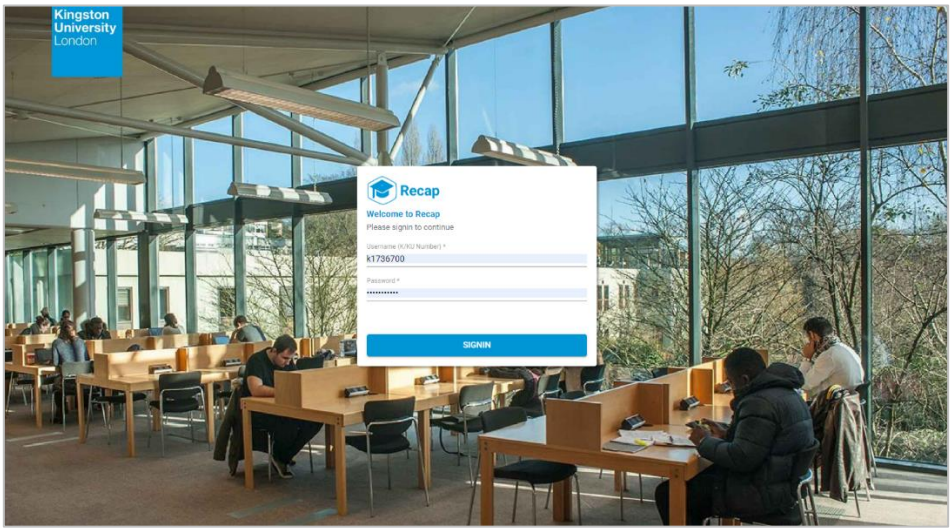
A simple Style Guide outlined a set of standards aimed at providing engaging and persuasive user experiences. The branding took cues from the KU branding to enable it to be instantly recognisable and instil a sense of trust and reflection with its choice of a vibrant light blue (Grzybowski & Kupidura-Majewski, 2019). The logo design for "Recap" conveyed a visual play on words and associated the brand with academic study and attainment.

A small colour palette designed for the user interface provided the basis for indicating how elements relate to other elements and their level of prominence.



The choice of visual imagery aimed to convey to students that they are part of a broader group

looking for a positive learning outcome and reinforce the notion of Recap as being an integral tool for studying.

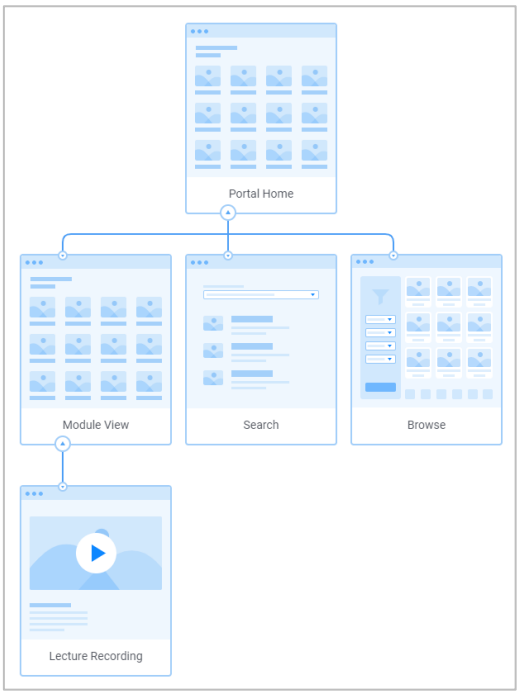


The underlying UI components were provided by Google's Material Design which is a commonly used paradigm familiar to many users. Using Material Design not only sped up the design and development process but also helped make the solution intuitive to use.

Page Flows

The initial design for the solution involved a "Studio" for lecturers and a "Portal" for students to complete their tasks. On reflection, this separation of roles had the disadvantage of potentially causing confusing, and so the design refocused on a single set of screens to support both lecturers and students.

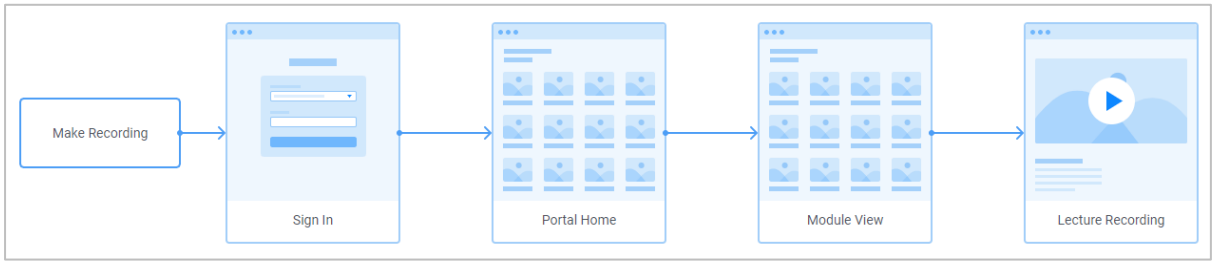
The final design envisaged a single portal with role-based access and a minimal number of pages to empower users to navigate their way around quickly and easily.



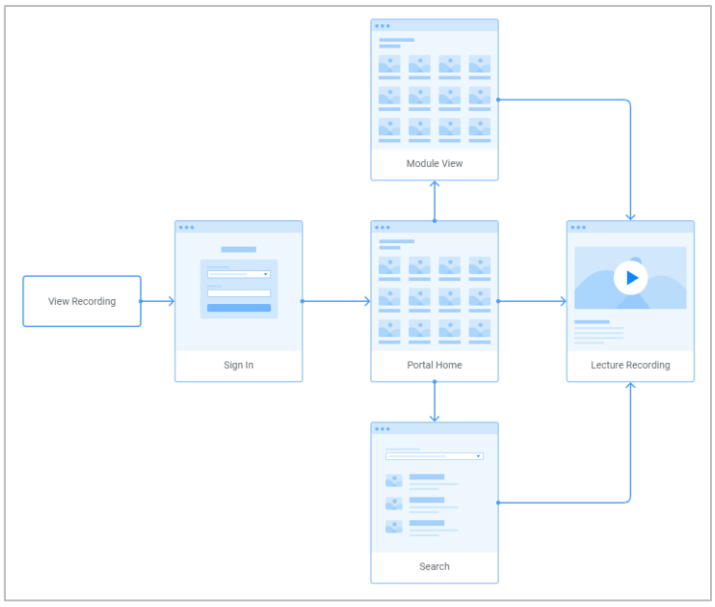
An elementary portal structure broke down into five core pages as follows:

- **Portal Home** – provided quick access to tasks and relevant content, enabling the user to pick up where they left off.
- **Module View** – provided a module and topic container for lecture recordings.
- **Search** – enabled users to discover and retrieve lecture recordings by several facets, including lecturer, topic, and keyword.
- **Browse** – enabled users to find modules to join and follow.
- **Lecture Recording** – enabled users to review and view lecture recordings.

The longest possible page flow for a Lecturer to make a recording involved four pages. As part of the design refinement, Lecturers were able to proactively create a placeholder for a future Lecture Recording and select it from the Portal Home, which made task completion more efficient.



The page flow for a Student wanting to view a recording was designed to be as simple a task to complete as possible. Depending on their usage scenario after signing in, a student can access a Lecture Recording from the Portal Home, or via Search or a Module View.

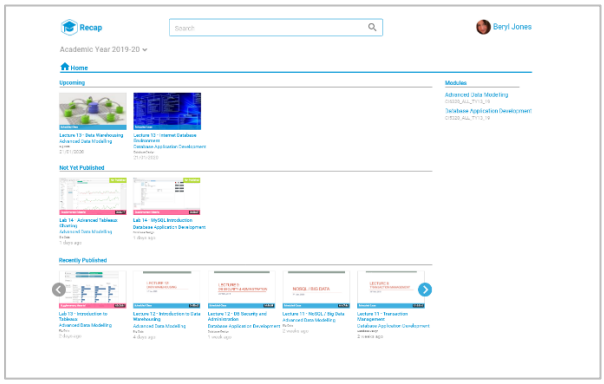


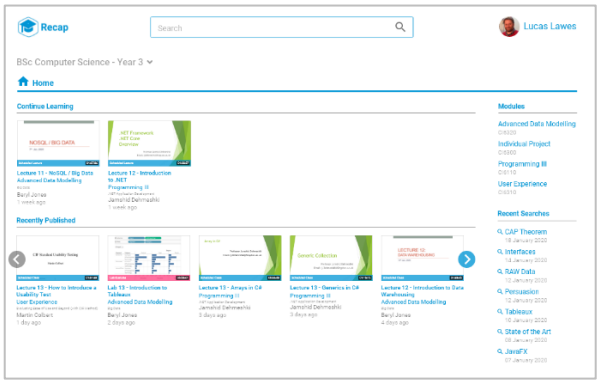
Wireframes

Wireframes of the Recap portal aided requirement discussions with the client team throughout the project. The medium-fidelity wireframes shown below are a subset of the ones reviewed before the beginning of Sprint 3. They show the three main role-based pages for Lecturers and Students respectively without changes applied.

Portal Home

The home page for the portal provided relevant contextual information about lecture recordings and access to modules. For a Lecturer, this provided an efficient way to begin recording an upcoming lecture, review an unpublished lecture and watch recently published lectures. For a Student, the home page provided a way to continue learning by returning to a previous lecture recording and watching recently published ones.

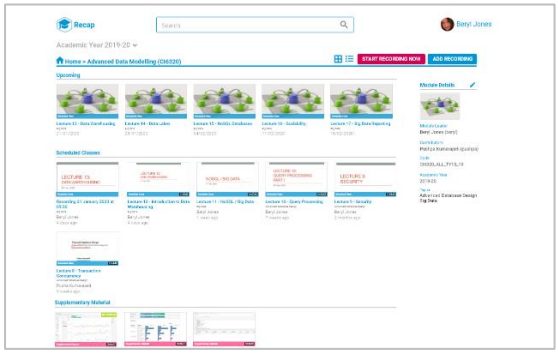


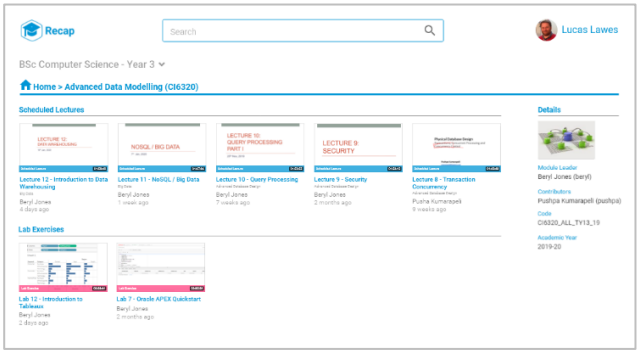


After requirement discussions, changes included the removal of the right navigation pane, the promotion of modules to be displayed as thumbnails above lecture recordings, and the remapping of modules from courses to academic years. These changes simplified the design and improved overall consistency.

Module View

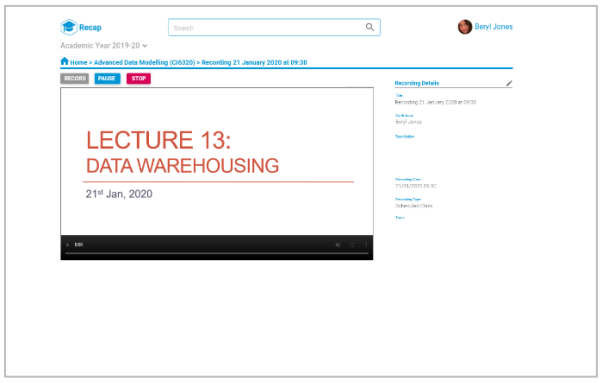
The module page provided a container for upcoming, unpublished, and published lecture recordings for Lecturers. Students only saw published lecture recordings. Lecturers were given a single button to “Start Recording Now” to shortcut the task, and lecture recordings were given a type to distinguish between Scheduled Classes and Supplementary Material. Changes were made to layout lecture recordings by topic rather than a type, and the right navigation pane was removed. Lecture recordings were also given a “traffic light” colour status bar to distinguish between placeholders for new recordings (red), lecture recordings being processed (amber) and lecture recordings ready for review (green). All published lecture recordings were shown with a blue status bar.

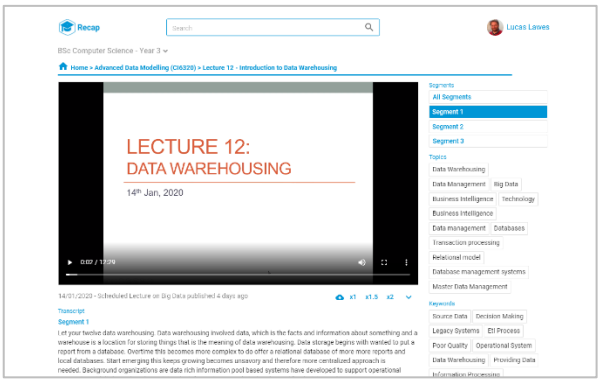




Lecture Recording

The design of the lecture recording view for Lecturers took a substantial amount of effort to get right. For lecturers, the final design saw the lecture recorder have four input buttons (Capture Screen, Record, Pause and Stop) instead of three as this improved task completion with fewer errors. Visual feedback was added to inform the Lecturer of what was happening once the recording had been stopped. The right details pane was removed, and the transcript could be edited in place, enabling greater efficiency for this time-consuming task. For students, the ability to search the transcript was added to enable the task of understanding segments of it to be more productive.

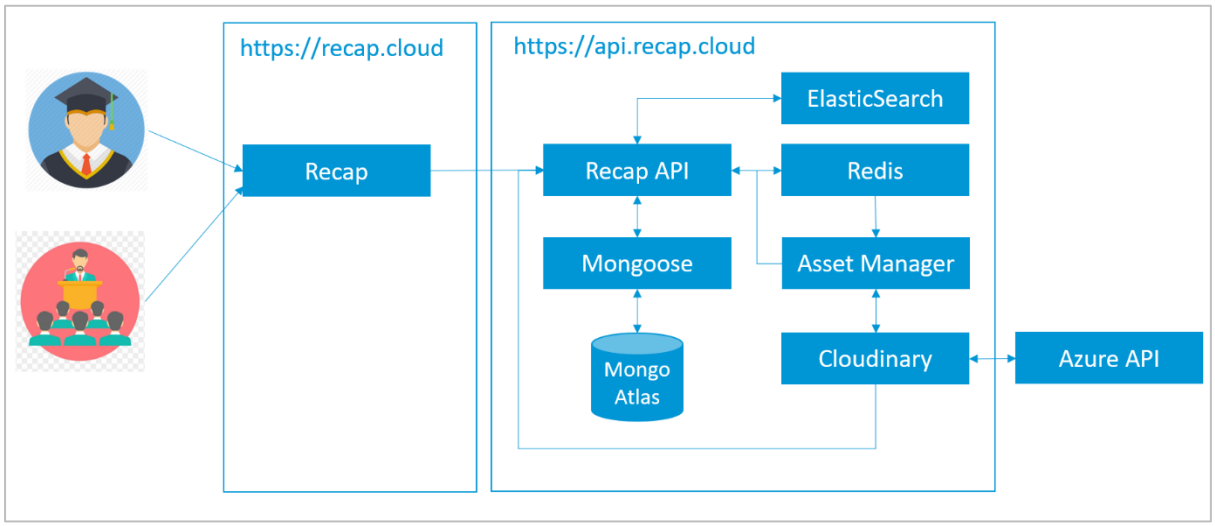




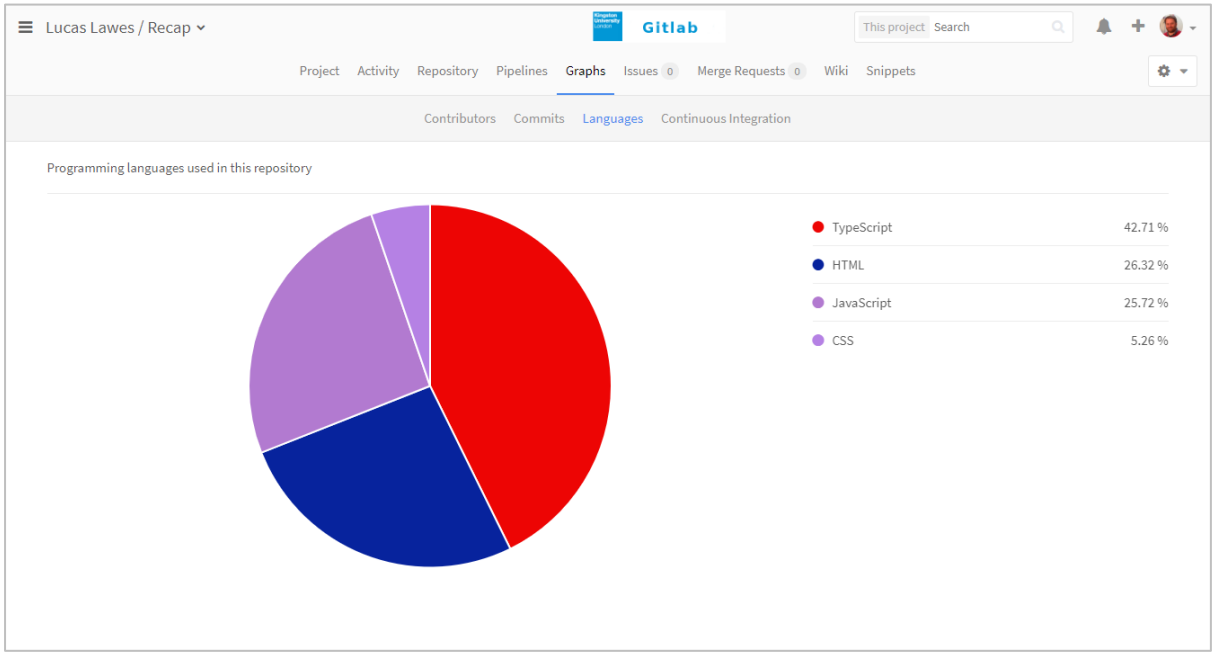
Further design iterations were necessary before the final design for the solution became apparent. All in all, the final design saw the user interface stripped back to be minimal yet informative on the basis this made the portal more intuitive and productive to use.

High-Level Design

The High-Level Design defined an architectural overview of the solution and guidance for the implementation of the different components. The Project Proposal included a conceptual model and a list of candidate technologies. It took considerable effort to evaluate these and find appropriate front and back end technologies that met the requirements. More details of the selection rationale are included in the appendix (see Appendix D - Technology Selection).



At the start of writing the High-Level Design, a decision was made to use JavaScript as it has become synonymous with building modern web applications as once predicted (Atwood, 2007), and appeared to offer a highly productive workflow. Looking at the programming languages committed into the Gitlab code repository at the end of the project shows the entire solution was written using JavaScript, HTML and CSS. Surprisingly, twice as much JavaScript was required to write the frontend than the backend. However, 80% of the complexity was contained in the backend, which included the Recap API and Asset Manager.



The Recap API acted as a hub which interconnected the other backend components via microservices. The High-Level Design defined the components shown in the table below (for more details, see Appendix E – Component Details):

Component	Role	Notes
Recap (client)	Provides role based portal for lecturers and students	Angular Single Page Application (JavaScript, HTML, CSS)
Recap API	Provides a RESTful API for the Recap client to access backend components	NodeJS/Express
Mongoose	Provides an Object Data Modeller and wrapper for accessing Mongo Atlas	NodeJS package
Mongo Atlas	Provides a NoSQL document store for user, module and lecture information.	Cloud hosted
ElasticSearch	Provides an intelligent search engine for published lecture information	Also includes Kibana used for learning analytics

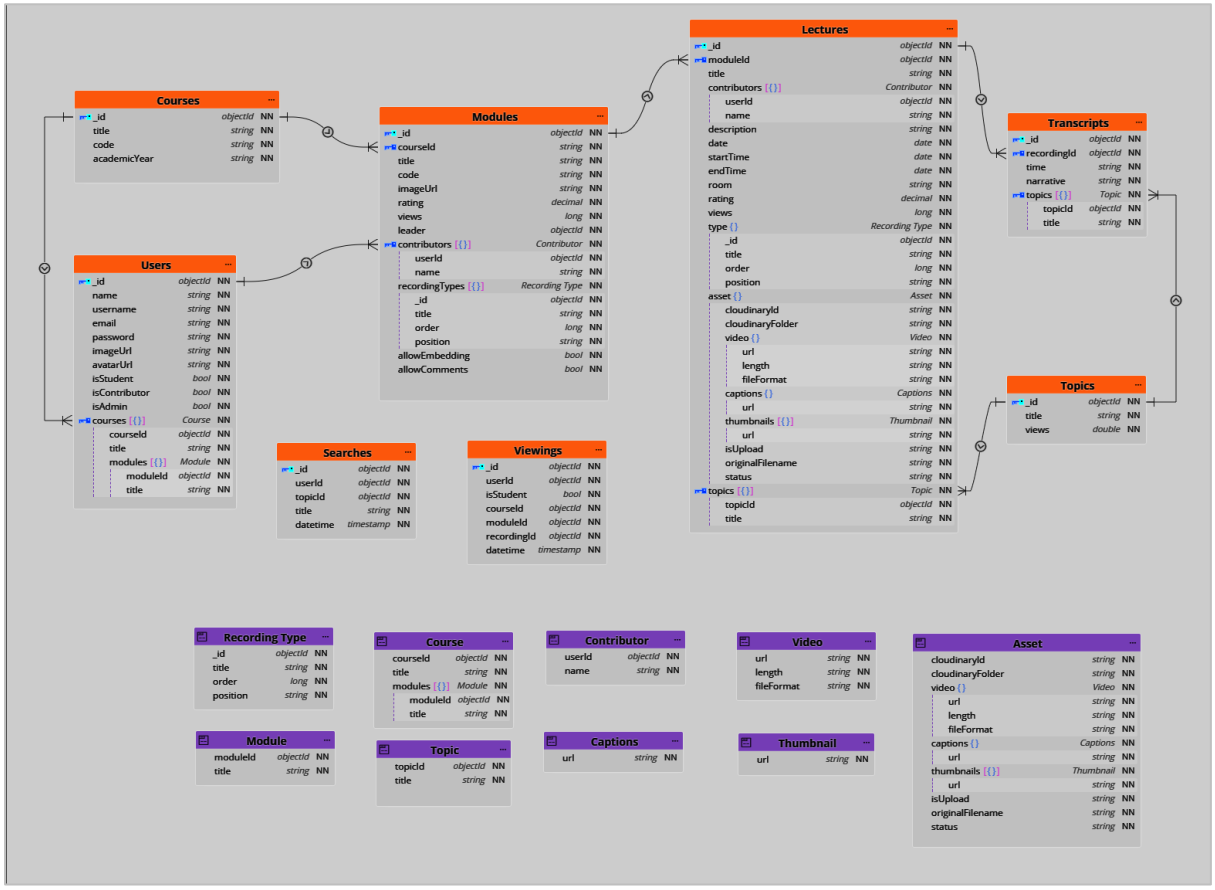
Component	Role	Notes
Redis	Provides an in-memory cache and message queue for improving the performance of the Recap API	Queue provided by NodeJS/Bull
Asset Manager	Provides a process for uploading, processing and deleting images and videos stored in Cloudinary.	NodeJS/Cloudinary.v2
Cloudinary	Provides a digital asset library and transformation API	Cloud hosted
Azure API	Provides an Artificial Intelligence API for system generating lecture transcriptions	Cloud hosted

Database Schema

The scope of the MongoDB database design included storing user and lecture information with a link to the video library provided by Cloudinary cloud services.

Initial Design

An initial NoSQL schema provided conceptual guidance for beginning the implementation of the MongoDB database. Collections contained documents which could contain sub-documents, and object references provided relationships between document types.



The initial design envisaged the following collections:

- **Users** – contained a single document type storing different attributes depending on the user role mappings. Boolean flags identified the user role mappings. The document type for students contained module mappings.

- **Courses** – contained a single document type storing course information.

- **Modules** – provided a parent container for lecture recordings.

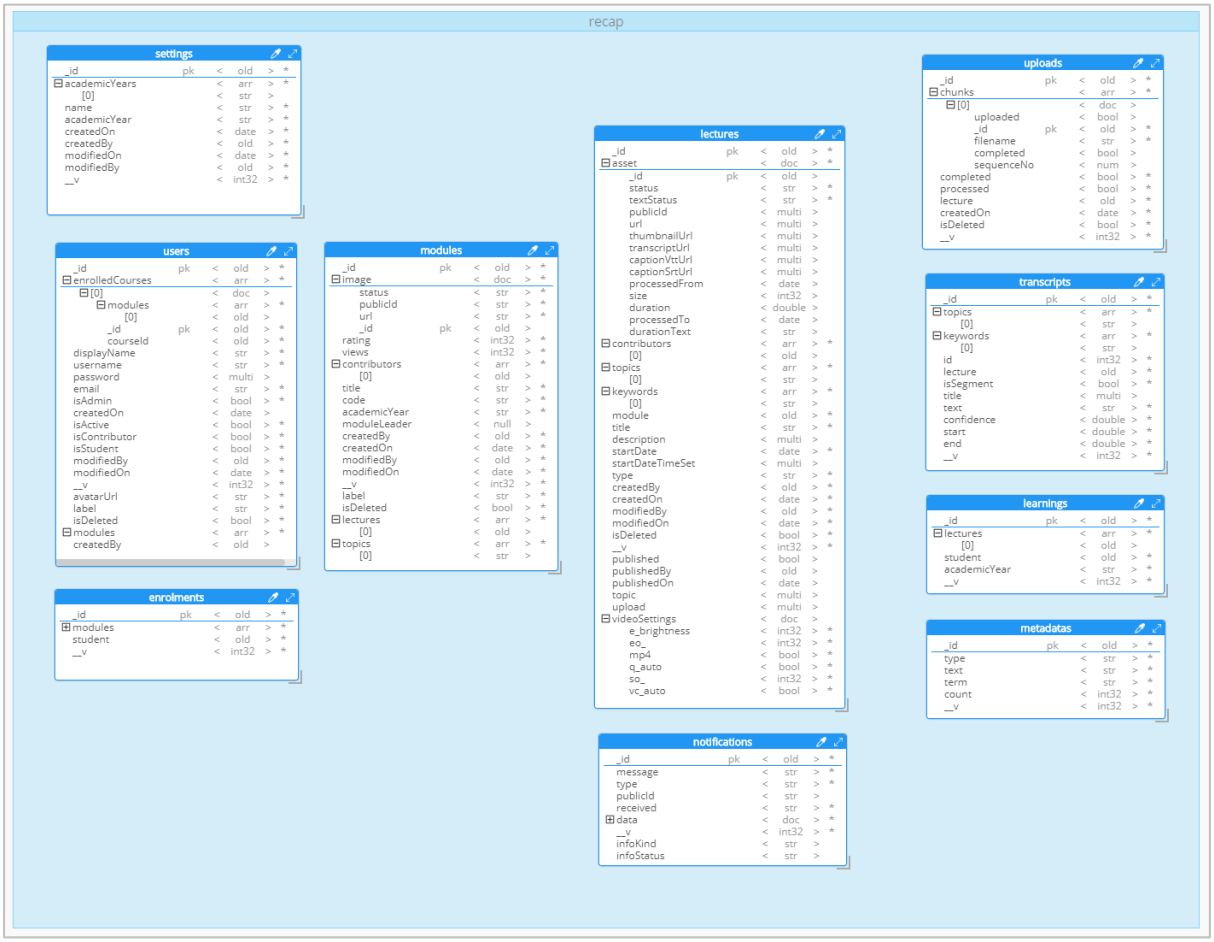
- **Lectures** – contained a complex document storing information about the lecture recordings, mappings for the video stored in Cloudinary and metadata.

- **Transcripts** – stored multiple documents containing the transcript for each lecture recording.

- **Topics** – stored documents containing metadata for each lecture recording.
- **Searches** – contained a single document type storing user search history.
- **Viewings** – contained a single document type storing user viewing history.

Final Design

Changing requirements and the experimental nature of the development process meant the MongoDB database schema evolved significantly throughout the Construction phase of the project. The final database schema shown below is a view extracted from the production database. Notable changes included the Settings collection superseding the Courses collection and the Enrolments collection replacing the student module mappings. The Lectures document type included an improved Cloudinary mapping and a Video Settings sub-document to support editing of the lecture recording videos. The Transcripts collection included the ability to segment the lecture recording and the Metadatas collection stored both keyword and topic metadata for each lecture recording.

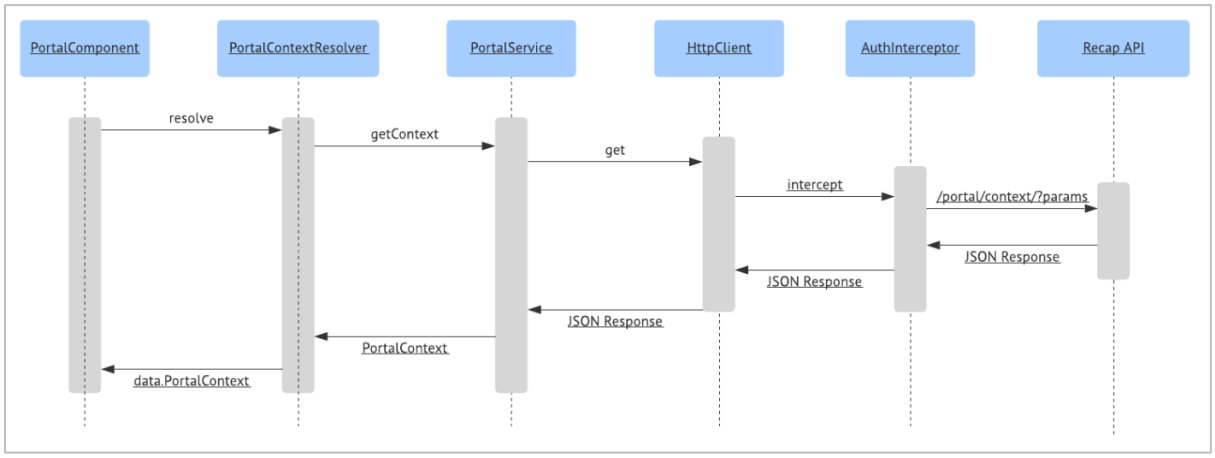


Sequence Diagrams

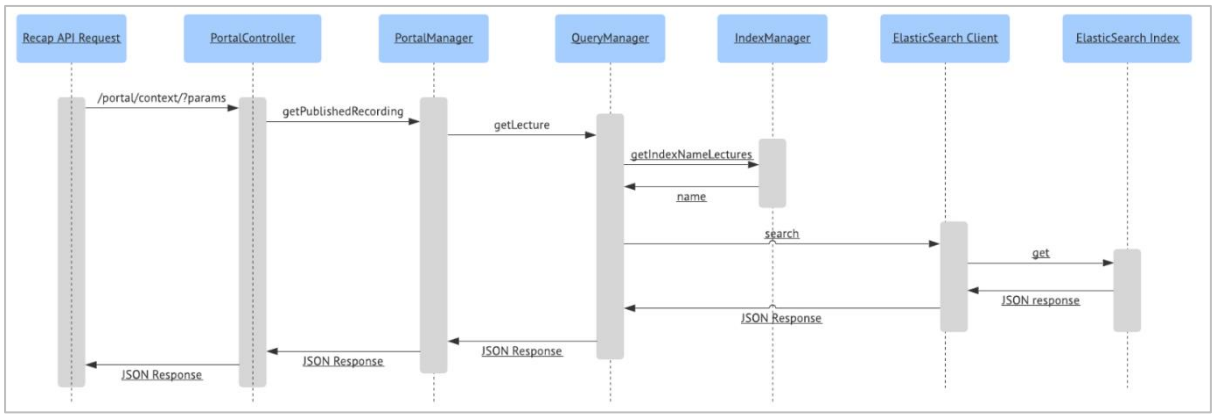
A series of sequence diagrams provided a guide for building modular code and defining how messages pass between components. The design for two usage scenarios has been included to illustrate the design patterns used with fundamental interactions for both the frontend and backend.

Get Portal Context

The Recap client rendered contextual page views for each user depending on their security and module mappings.



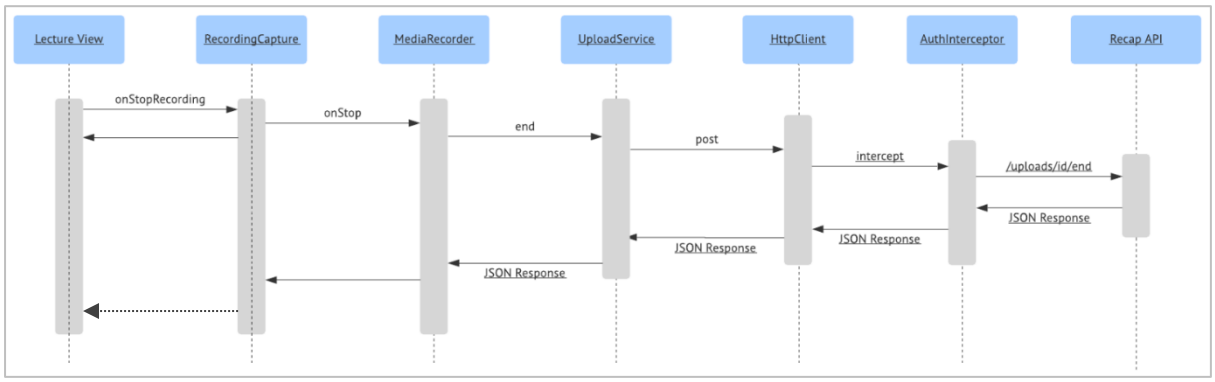
When a user accessed a portal page, the PortalComponent resolved the context for the view before it was loaded to provide a fast and seamless user experience. The PortalContextResolver used the routing information to pass context parameters to the PortalService, which requested the context data from the Recap API via the HttpClient. Before the request was sent, the AuthInterceptor added a security token to enable the user to be authenticated.



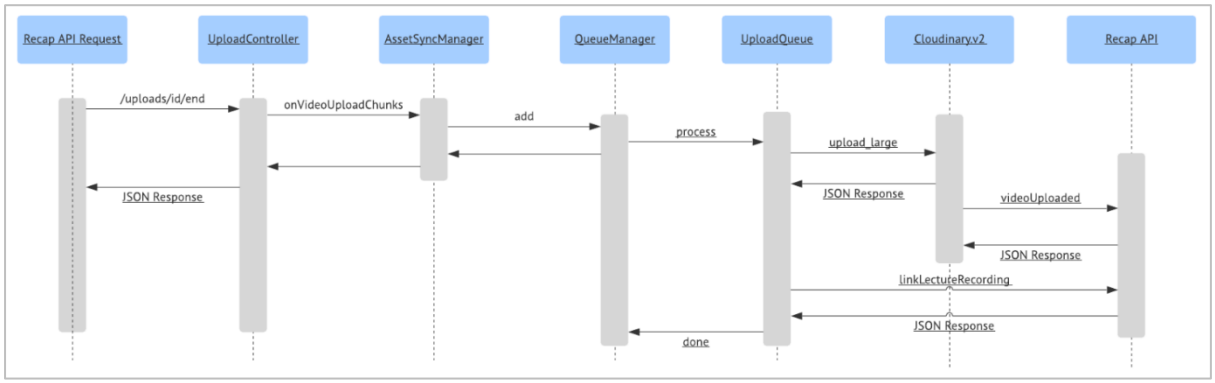
On the backend, the PortalController received the request from the Recap client after being authenticated by middleware components (not shown). The responsibility for processing the request was passed from the Controller to the Manager component; in this example, the PortalManager. The PortalManager handled multiple data requests in parallel (not shown) to reduce the response times. To get lecture recordings published in ElasticSearch, the PortalManager would call the QueryManager, and there were other Manager components to access data from MongoDB and Redis. This approach kept the codebase clean and made it easy to write, read and maintain.

End Lecture Recording

The Lecture View in the Recap client enables Lecturers to make lecture recordings by using the JavaScript Screen Capture and Media Recorder APIs.



When a Lecturer clicks Stop on the Lecture View component to finish a lecture recording, it sets off an asynchronous onStop event handler on the Media Recorder to finalise the upload (not shown), and once completed, end the "transaction". The transaction is finished by the UploadService which sends a post request via the HttpClient. The response contains a message stating whether the process succeeded or failed, and the Recording Capture component updates the Lecture View appropriately.



The request from the Recap client is processed on the backend by the UploadController. This Controller checks the upload is successful before calling the AssetSyncManager which adds an upload job to the message queue using QueueManager. In the Asset Manager, the job is processed using the UploadQueue event handler. The Cloudinary API is called to upload the lecture recording video file from the file system to the video library in the cloud. A successful upload returned a "public id" which is linked to the lecture recording stored in MongoDB by calling back to the Recap API. Cloudinary receives the video file and starts eager transformations for converting the video file to MP4 and getting the transcript from Microsoft Azure. The successful completion of these transformations triggers a stream of notifications by Cloudinary back to the Recap API, which makes the necessary updates to the lecture recording data stored in MongoDB.

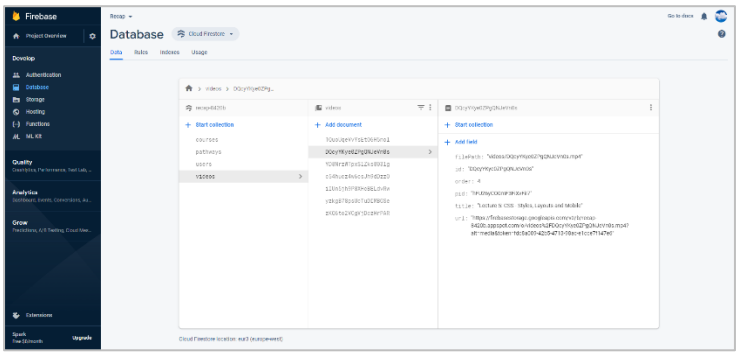
Implementation

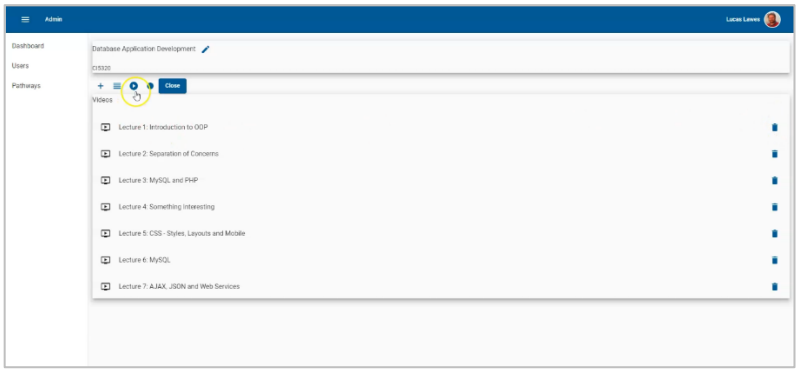
Preliminary work undertaken in the summer of 2019 explored the basis for an innovative solution to lecture capture. This work continued through the Inception and Elaboration phases before implementation commenced in the Construction phase of the project. Refinement work and bug fixes finalised the solution during the Transition phase.

Preliminary Work

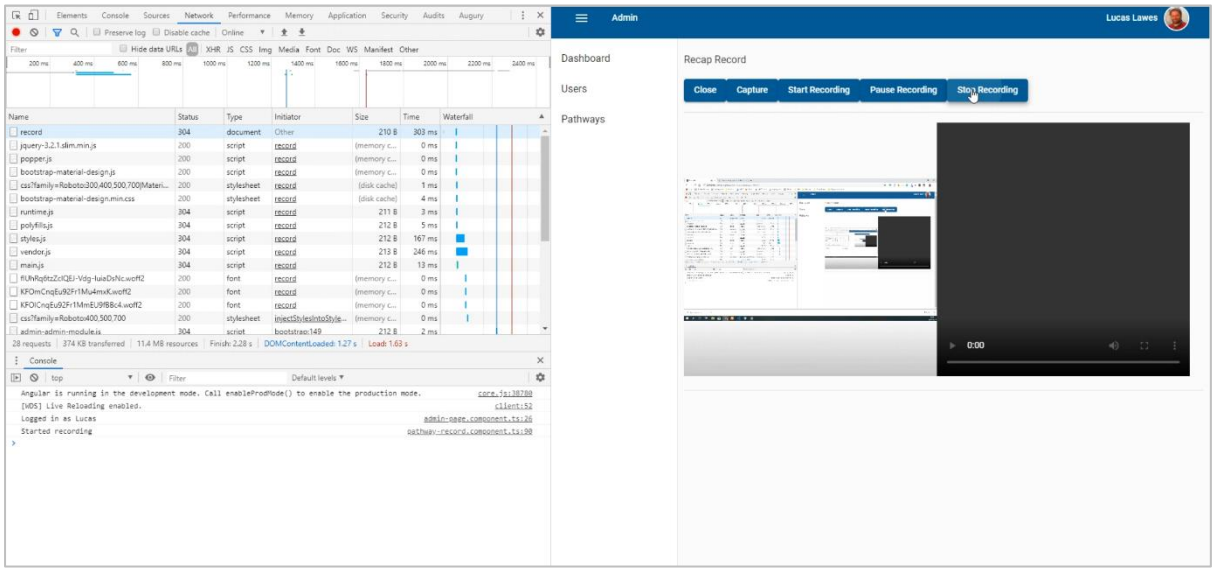
The initial idea for Recap came from the realisation the open-source software Opencast lacks a frontend for students to use to access lecture recordings. A close inspection of the source code revealed the codebase for video capture to be comprehensive, vast, and complex. It appeared unclear how to provide a frontend for Opencast despite the contributors expressing an interest in building one. At the same time, Google's latest UI toolkit called Flutter was emerging as a cross-platform development tool which provided the ability to build apps for mobile, web and desktop from a single codebase. The first idea was to explore how the two could be combined to produce a compelling and more widely used open-source alternative to proprietary lecture capture software. However, while researching how to integrate with the Opencast API, it became clear that getting a frontend to communicate with the Opencast stack would be technically challenging and problematic in that it would require a team effort to make the application more open and interoperable.

In 2019, the JavaScript Screen Capture API was emerging as a supported capability on modern web browsers, and this addition meant a desktop application is no longer necessary to make lecture recordings. The discovery of this new API gave way to realising the second idea of building an entire lecture capture solution in JavaScript. A prototype developed using a serverless computing architecture demonstrated a relatively small JavaScript codebase could reproduce the essential functionality found in a desktop-based screen recording tool. The prototype used Firebase for cloud database and storage, and Angular for a web frontend.

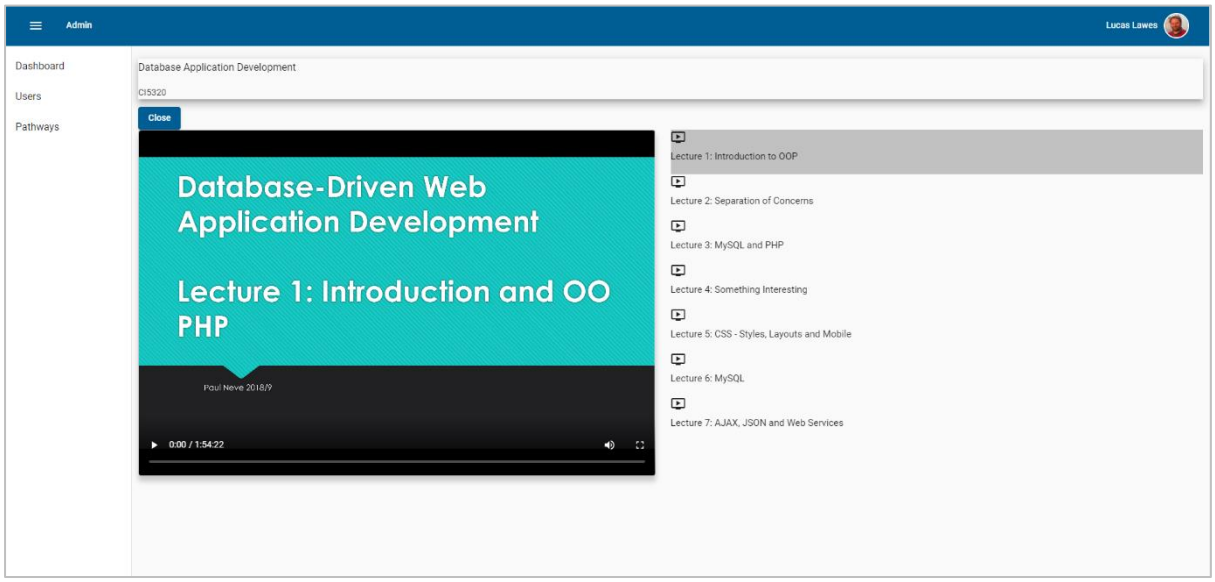




The development effort focused on recording a screen, saving the recording to the cloud, and retrieving the media for playback in a video player only using HTML, CSS, and JavaScript.



The initial results were promising, and what is more, making lecture recordings in a web browser seemed to provide a better user experience over desktop tools. The prototype included the ability to upload lecture recording files meaning desktop tools could still make lecture recordings for edge use cases.



The prototype only took five days to produce, and the necessary code for capturing the screen and saving it to the cloud took up some forty lines. While it took some significant mental effort to come up with the code in the first place, the overall development effort required to reproduce a screen recording tool equivocal to existing proprietary software written in more powerful languages was negligible.

The preliminary work promised to lead to a more open and accessible lecture capture solution than existing offerings. Even if a much more fully functional version of the software were paid for, perhaps as a cloud offering, it was clear an open-source base offering could outweigh the

benefits of purchasing proprietary software. The original 40 lines of code which took the idea for Recap and made it a reality are shown below.

```
captureScreen() {        You, 7 months ago • Initial admin commit
  this.invokeGetDisplayMedia((stream: any) => {
    this.videoStream = stream;
    if (this.videoRecorder.duration === Infinity) {
      this.videoRecorder.currentTime = 1e101;
    }
    this.mediaRecorder = new MediaRecorder(stream, { mimeType: 'video/webm; codecs=vp8, opus' });
    this.mediaRecorder.ondataavailable = (e) => {
      console.log('ondataavailable', e);
      this.chunks.push(e.data);
    };
    this.mediaRecorder.onstop = (e) => {
      console.log('onstop', e);
      this.videoStream.getTracks().forEach(track => {
        console.log(track);
        track.stop();
      });
      const blob = new Blob(this.chunks, { type : 'video/webm' });
      const downloadLink = document.createElement('a');
      const url = URL.createObjectURL(blob);
      downloadLink.href = url;
      downloadLink.download = 'test.webm';
      document.body.appendChild(downloadLink);
      downloadLink.click();
      setTimeout(() => {
        document.body.removeChild(downloadLink);
        window.URL.revokeObjectURL(url);
      }, 100);
      const task: AngularFireUploadTask = this.storage.upload('videos/test.webm', blob);
      task.snapshotChanges().pipe(finalize(() => {
        const fileRef: AngularFireStorageReference = this.storage.ref('videos/test.webm');
        fileRef.getDownloadURL().subscribe(fileUrl => {
          this.videoURL = fileUrl;
        });
      })).subscribe();
    };
  }, (error: any) => {
    console.log('Got an error!', error);
  });
}
```

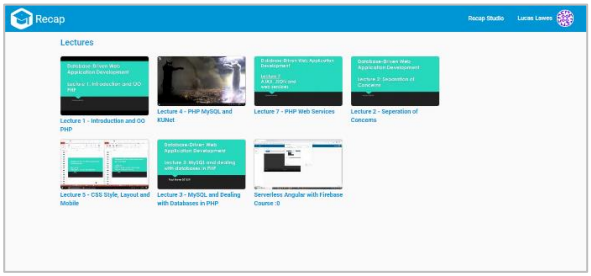
Inception

With preliminary work completed successfully, the Inception phase focused on understanding KU requirements and wondering what technology to use for a solution. While the idea was to use JavaScript for the whole solution, it was unclear whether this was possible. A wide range of backend technologies offered convincing alternatives and Spring Boot, ASP.NET Core and Go were given careful consideration over NodeJS; however, it is hard to avoid the gravitational pull of JavaScript (Gawron, 2018).

Research into each component of the High-Level Design revealed NodeJS as an integration option every time. The backend solution is essentially a hub for a set of interconnected microservices which necessitates flexibility to accommodate changing requirements over time. While NodeJS is lightweight, scalable and cross-platform (Gawron, 2018) its ability to integrate with cloud services and other open-source software made it difficult not to select. The

Early Prototype Demo provided the first opportunity to use NodeJS and the development workflow, compared to more procedural languages, seemed more productive.

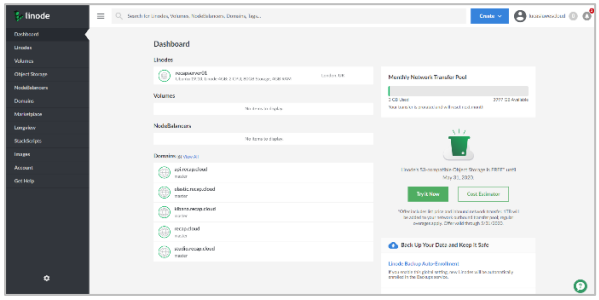
The prototype produced for the Early Prototype Demo included a significant amount of backend development work getting Recap API messages to flow between a client, MongoDB and Cloudinary. The frontend demonstrated the ability for a lecturer to upload a lecture recording and publish it to a portal for students to consume.

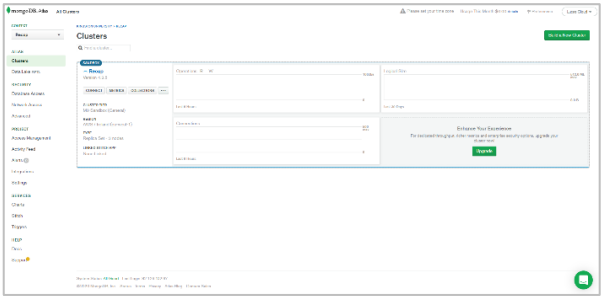


Elaboration

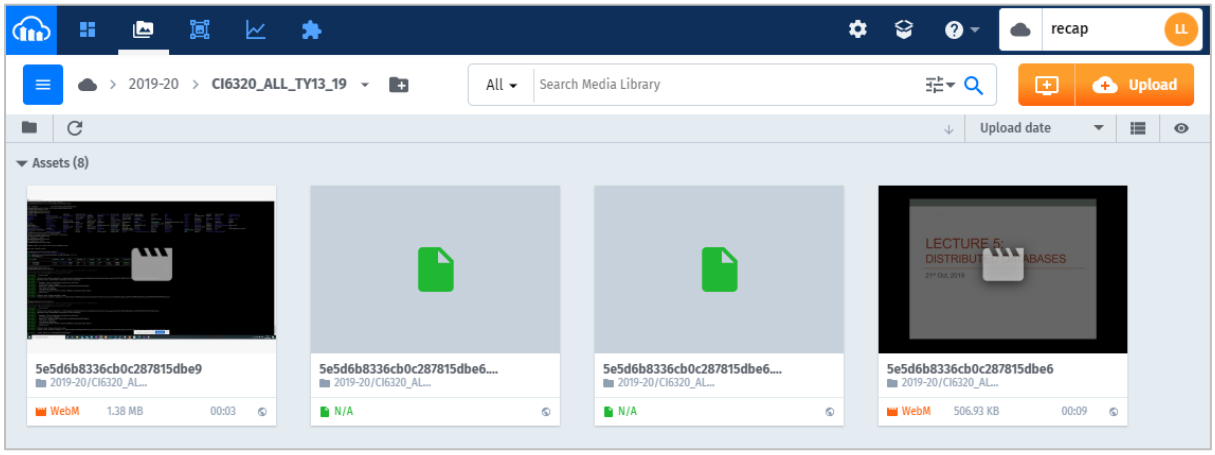
The focus of the Elaboration phase shifted from prototyping any further to preparing the development and production environments and, ramping up the technical skills necessary to implement the solution.

A server hosted on Linode was configured with Ubuntu, and a database cluster was created on the MongoDB Atlas cloud platform.



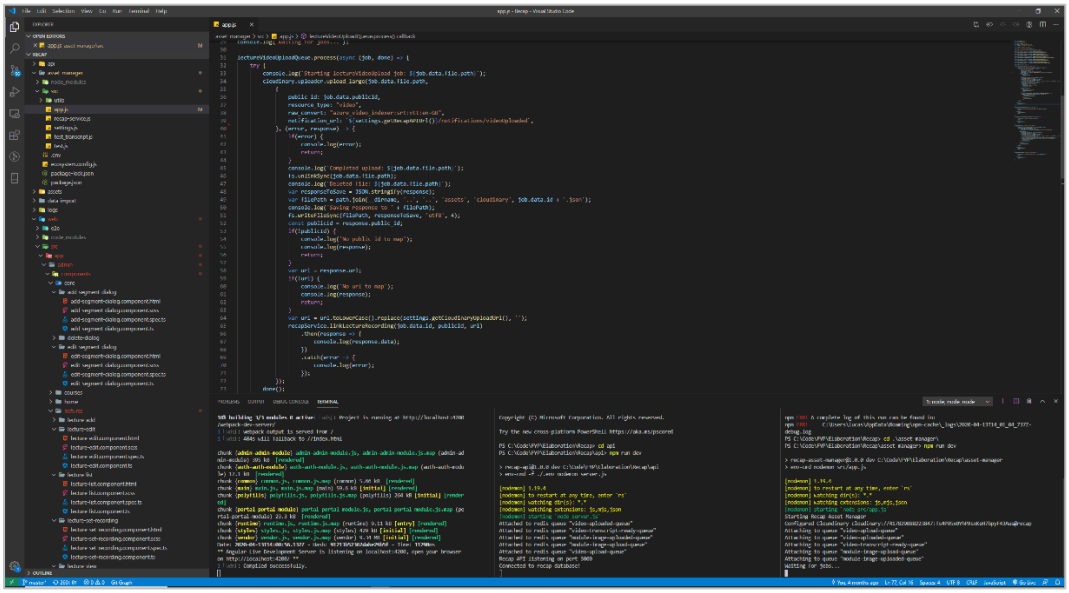


A Media Library was created in Cloudinary, and the API was carefully researched.

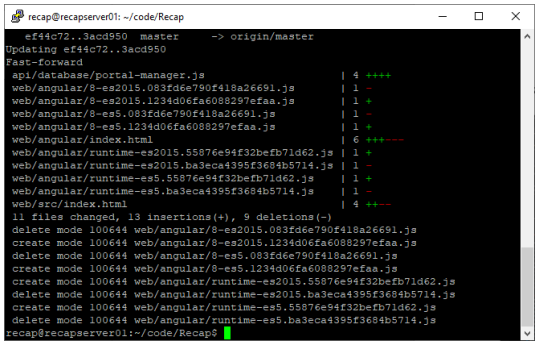


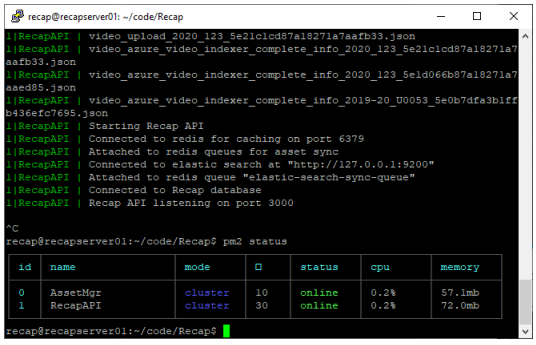
Test scripts were written against Cloudinary to learn how the platform handled video assets,

and one of these is shown below.

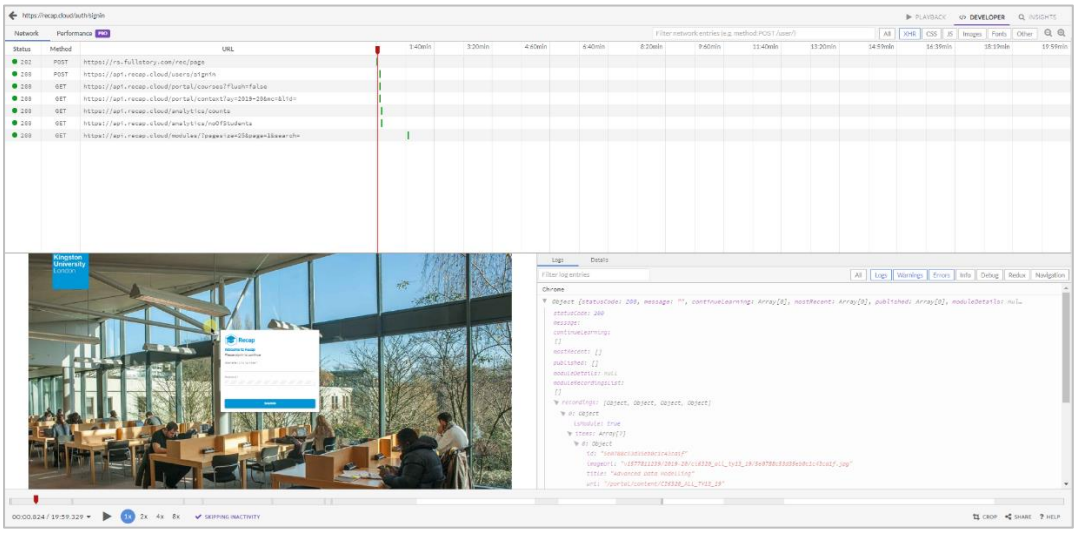


The solution was deployed to Linode using Git and the NodeJS processes hosted using PM2.





User sessions on the production environment were captured in LogRocket so they could be played back. This monitoring instrument improved the debugging process by helping to catch bugs and issues and later aided remote usability testing.



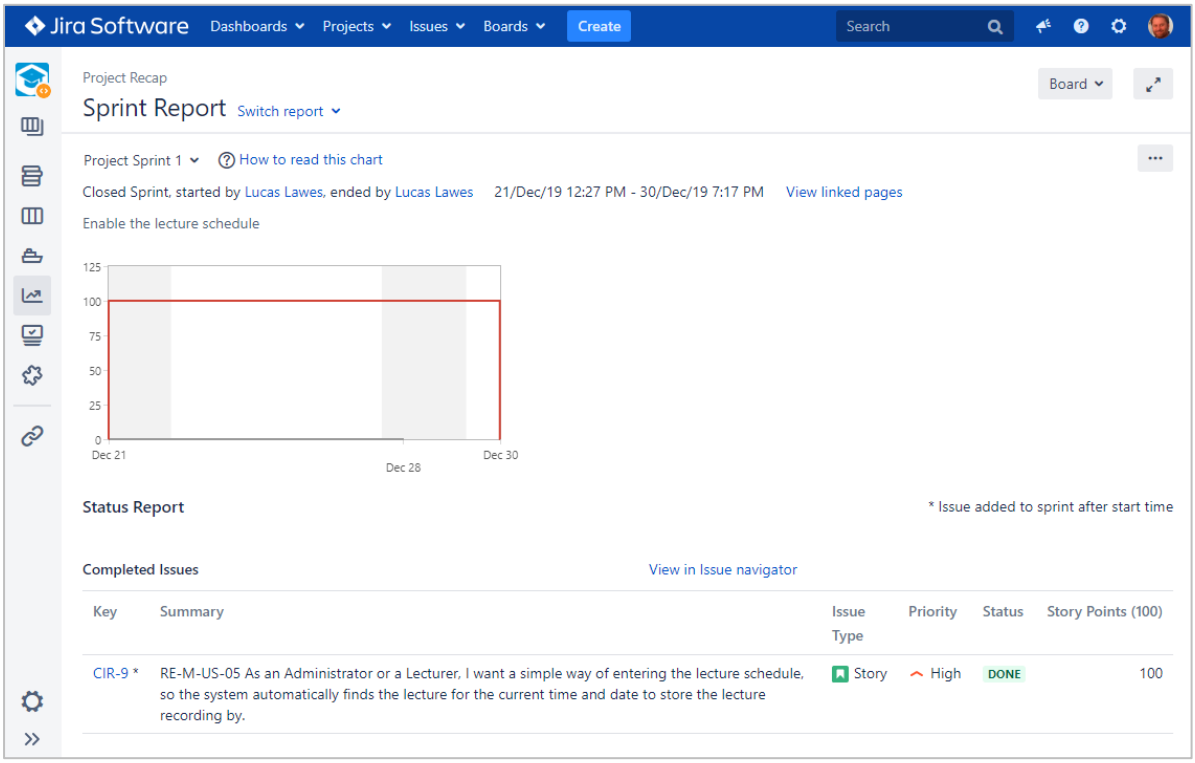
Construction

Once the requirements were clearly understood, and the High-Level Design tried and tested, the Construction phase commenced. One of the advantages of using RUP as a process framework is that lots of critical work can be carried out before the main implementation effort begins (Anwar, 2014), de-risking the project and saving time further along the process.

The Construction phase divided into five "sprints", which are probably better thought of as *six* distinct, sequential steps in the development process undertaken. While each step built incrementally on the previous step, progress was not linear. Sprint 1 produced a significant amount of administration functionality, which is not used in the final product as requirements changed later. However, lots of valuable experience and groundwork was gained from putting the effort in which benefited the project later. Sprint 3 took on more work than ideal but produced much useful functionality in a shorter amount of time than expected. Sprint 4, arguably two sprints, saw a total rework of functionality to better support the requirements and improve the user experience and essentially included a repeat step. The overall effort put into the Construction phases produced a solution quite different from the one initially envisaged before eliciting requirements and showed a collaborative approach had paid off.

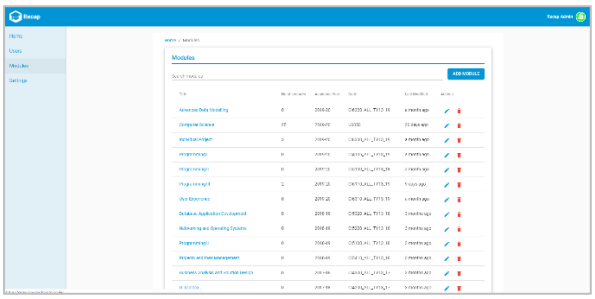
Sprint 1

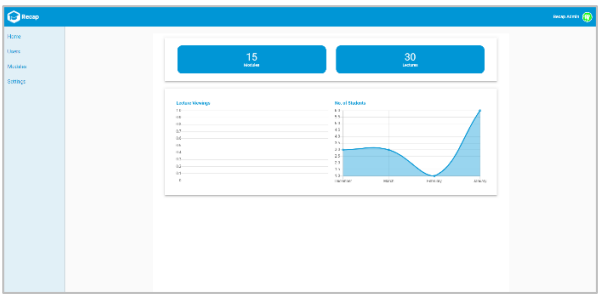
The goal of the first Sprint was to enable the lecture schedule, and one User Story (RE-M-US-05) estimated to be 100 story points was assigned.



The initial approach for delivering the User Story was to produce an Administration microsite for managing Courses, Modules and Lectures. Later, in the project, the concept of "crowdsourcing" was introduced to simplify the implementation and enable an "opt-in" approach. Crowdsourcing saw Courses dropped, and Lecturers tasked with managing Modules and Lectures in the portal.

The experience of building an Administration microsite helped improve technical skills and knowledge before developing the more complex Recap client. This portal required advanced concepts and patterns to be implemented, and some of these were used for the first time while building the Administration microsite; for example, Dependency Injection and Observables. The Administration microsite comprised of 29 UI view components for performing standard CRUD operations and mapping Users, Courses, Modules and Lecture Recordings. Development also began on a dashboard to provide a summary view of lectures stored in MongoDB, but this was deprecated and replaced by Kibana, which is a more powerful data visualisation tool.





While the functionality displayed to the user changed after Sprint 1, an underlying framework was established for communicating between the Recap client and the Recap API. JSON Web tokens issued by the Recap API authentication service allowed the Recap client to be authorised to perform other requests. In Angular, an HttpInceptor added the token to the Authorization Header of every request, while in NodeJS, a routing middleware function verified the token and identified the authenticated user before the router processed each request.

```
web > src > app > services > auth-interceptor.ts > ...
You, 4 months ago | 1 author (You)
import {
  HttpInterceptor,
  HttpRequest,
  HttpHandler
} from '@angular/common/http';
import { Injectable } from '@angular/core';

import { AuthService } from './auth.service';

You, 4 months ago | 1 author (You)
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  constructor(private authService: AuthService) {}

  intercept(req: HttpRequest<any>, next: HttpHandler) {
    const token = this.authService.getToken();
    const authRequest = req.clone({
      headers: req.headers.set('Authorization', 'Bearer ' + token)
    });
    return next.handle(authRequest);
  }
}
```

```
api > middleware > check-auth.js > ...
You, 3 months ago | 1 author (You)
const jwt = require('jsonwebtoken');

const settings = require('../settings');

module.exports = (req, res, next) => {
  try {
    const token = req.headers.authorization.split(' ')[1];
    const decodedToken = jwt.verify(token, settings.getJWTSecret());
    req.userData = {
      userId: decodedToken.userId
    };
    next();
  }
  catch (error) {
    res.status(401).json({ message: 'You are not authenticated!' });
  }
};
```

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

router.post('', checkAuth, LectureController.create);

router.post('/quickStart', checkAuth, LectureController.createQuickStart);
```

The NodeJS function used for starting the Recap API permitted Cross-Origin Resource Sharing (CORS) to support requests coming from the Recap client. The start function (shown overleaf) also established backend service connections with the Recap API and registered the routes exposed over HTTPS with compression enabled.

```
api > app.js > start
32  const LogRocket = require('logrocket');
33  LogRocket.init('ltbx7y/recap');
34
35  const start = async () => {
36
37    console.log(`Starting ${settings.getAppName()}`);      You, 3 months ago • Add asset sync manager
38
39    console.log(await CacheManager.connect());
40
41    console.log(await AssetSyncManager.connect());
42
43    console.log(await SearchManager.connect());
44
45    console.log(await SearchSyncManager.connect());
46
47    await mongoose.connect(settings.getDatabaseUrl(), {
48          useNewUrlParser: true,
49          useCreateIndex: true,
50          useUnifiedTopology: true
51        });
52
53    console.log('Connected to Recap database');
54
55    app.use(compression());
56    app.use(bodyParser.json({limit: '5000mb'}));
57    app.use(bodyParser.urlencoded({ limit: '5000mb', extended: false }));
58
59    app.use('/images', express.static(path.join(__dirname, 'images')));
60
61    app.use((req, res, next) => {
62      res.setHeader('Access-Control-Allow-Origin', '*');
63      res.setHeader(
64        'Access-Control-Allow-Headers',
65        'Origin, X-Requested-With, Content-Type, Accept, Authorization'
66      );
67      res.setHeader(
68        'Access-Control-Allow-Methods',
69        'GET, POST, PATCH, PUT, DELETE, OPTIONS'
70      );
71      next();
72    });
73
74    app.use('/test', testRoutes);
75    app.use('/settings', settingsRoutes);
76    app.use('/notifications', notificationRoutes);
77    app.use('/users', userRoutes);
78    app.use('/courses', courseRoutes);
79    app.use('/modules', moduleRoutes);
80    app.use('/lectures', lectureRoutes);
81    app.use('/uploads', uploadRoutes);
82    app.use('/transcript', transcriptRoutes);
83    app.use('/metadata', metadataRoutes);
84    app.use('/analytics', analyticsRoutes);
85    app.use('/portal', portalRoutes);
86    app.use('/catalogue', catalogueRoutes);
87  }
```

A Unit Testing framework, called Jasmine, was going to be used to help find bugs, ensure quality, and facilitate refactoring. However, there was very little logic to test, as most component APIs already had unit tests, and most code was communicating with another component. So instead, integration test scripts were written for NodeJS. Example test scripts (see overleaf) show NodeJS connecting with ElasticSearch, and, processing a captured copy of Cloudinary notification messages and storing them into MongoDB.

```
api > test_content.js > ...
You, 3 months ago | 1 author (You)
1  const mongoose = require('mongoose');        You, 3 months ago • Implement onLectureChange, Delete and onReindex
2
3  const searchManager = require('./database/search/search-manager');
4  const ContentManager = require('./database/search/content-manager');
5
6  const settings = require('./settings');
7
8  searchManager
9    .connect()
10     .then((response) => {
11       console.log(response);
12     })
13     .catch(error => {
14       console.log(error);
15     });
16
17 mongoose
18     .connect(
19       settings.getDatabaseUrl(),
20       {
21         useNewUrlParser: true,
22         useCreateIndex: true,
23         useUnifiedTopology: true
24       }
25     )
26     .then(() => {
27       console.log('Connected to recap database');
28     })
29     .catch((error) => {
30       console.log('Connection to recap database failed! ' + error);
31     });
32
33 const run = async () => {
34     await ContentManager.onReindex({});
35     await ContentManager.onLectureChange({
36       lectureId: '5e1706904ce19b059ad50896'
37     });
38     await ContentManager.onLectureDelete({
39       lectureId: '5e1706904ce19b059ad50896'
40     });
41 };
42
43 run()
44     .then()
45     .catch(error => console.log(error));
```

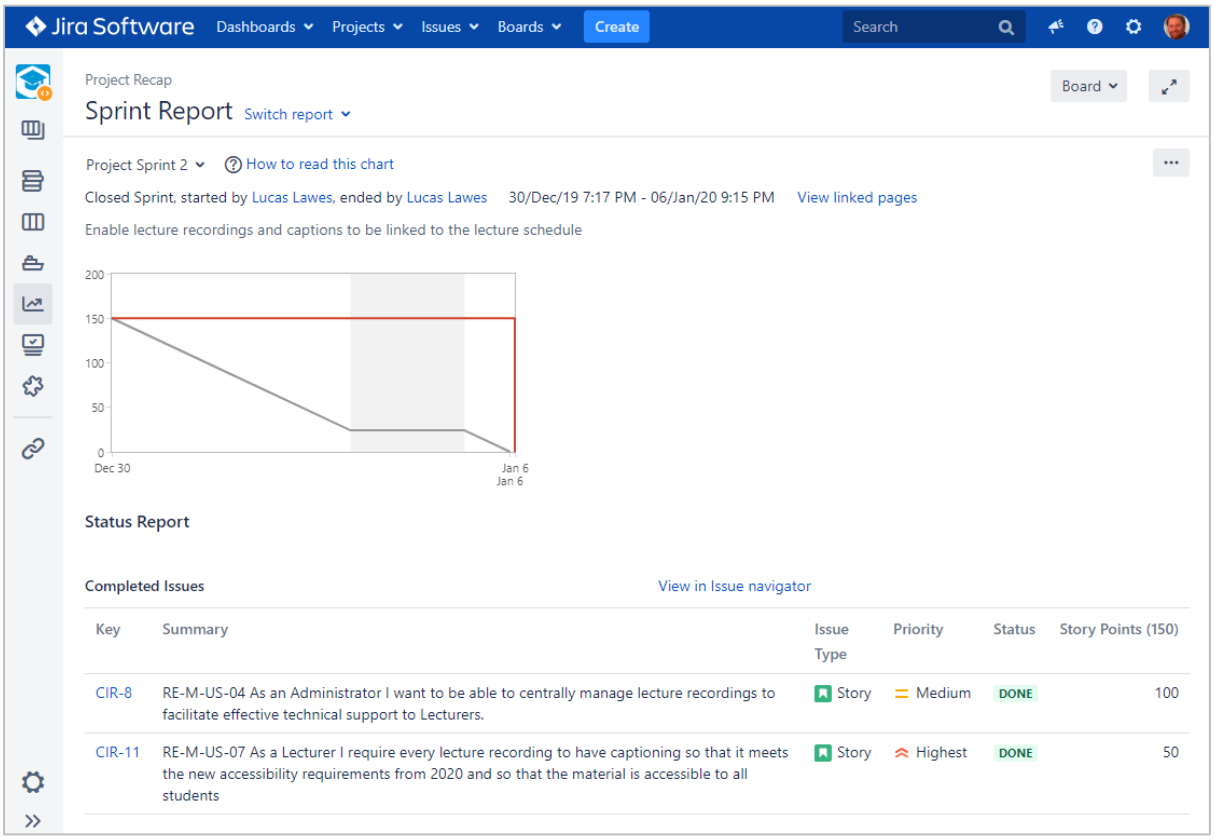
```
api > test_transcript.js > ...
You, 3 months ago | 1 author (You)
1  const fs = require('fs');        You, 3 months ago • Add new transcript sync algorithm
2  const path = require('path');
3  const settings = require('./settings');
4  const mongoose = require('mongoose');
5
6  const TranscriptSync = require('./middleware/transcript-sync');
7
8  mongoose
9    .connect(
10     settings.getDatabaseUrl(),
11     {
12       useNewUrlParser: true,
13       useCreateIndex: true,
14       useUnifiedTopology: true
15     }
16   )
17   .then(() => {
18     console.log('Connected to recap database!');
19   })
20   .catch((error) => {
21     console.log('Connection to recap database failed! ' + error);
22   });
23
24 const filename = 'video_azure_video_indexer_complete_info_2019-20_CI6110_ALL_TY13_19_5e0e9b492b570771e506cff8.json';
25
26 const getNotificationFromFile = async (filename) => {
27   const filePath = path.join(__dirname, '..', 'assets', 'cloudinary', 'notifications', filename);
28   const file = fs.readFileSync(filePath);
29   const notification = JSON.parse(file);
30   return notification.responseData;
31 };
32
33 const run = async (notification) => {
34   return await TranscriptSync(notification);
35 };
36
37 getNotificationFromFile(filename)
38   .then(notification => {
39       run(notification)
40         .then(response => {
41           console.log(response);
42         })
43         .catch(error => {
44           console.log(error)
45         });
46   })
47   .catch(error => {
48     console.log(error);
49   });
```

Unit test fixtures were left as placeholders in Angular so that the Recap client could be easily upgraded to be production quality code over the Summer of 2020.

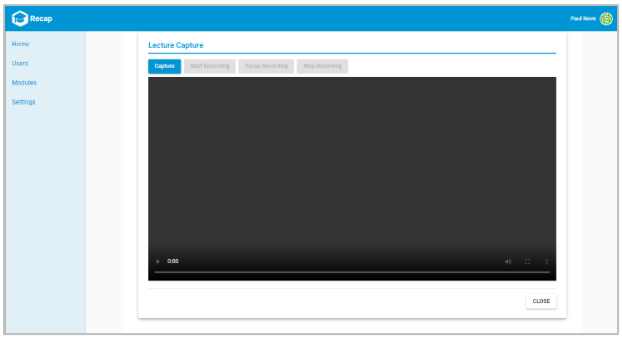
```
web > src > app > admin > components > lectures > lecture-view > lecture-view.component.spec.ts > ...
You, 4 months ago | 1 author (You)
1  import { async, ComponentFixture, TestBed } from '@angular/core/testing';
2
3  import { LectureViewComponent } from './lecture-view.component';
4
5  describe('LectureViewComponent', () => {
6    let component: LectureViewComponent;
7    let fixture: ComponentFixture<LectureViewComponent>;
8
9    beforeEach(async(() => {
10     TestBed.configureTestingModule({
11       declarations: [ LectureViewComponent ]
12     })
13     .compileComponents();
14   }));
15
16   beforeEach(() => {
17     fixture = TestBed.createComponent(LectureViewComponent);
18     component = fixture.componentInstance;
19     fixture.detectChanges();
20   });
21
22   it('should create', () => {
23     expect(component).toBeTruthy();
24   });
25 });
26
```

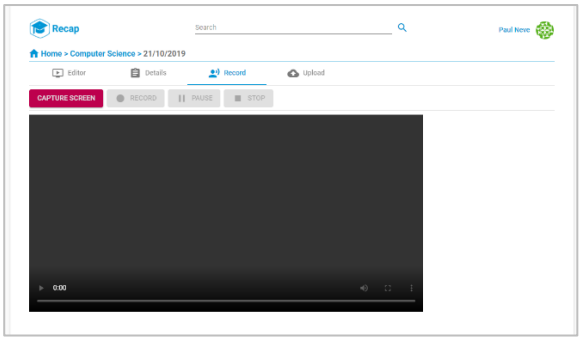
Sprint 2

The sprint goal of Sprint 2 was to enable lecture recordings and captions to be linked to the lecture schedule and comprised of two User Stories (RE-M-US-04/07). While Sprint 1 had focused on setting up the MongoDB database schema to support lecture recordings, the focus of Sprint 2 moved onto their generation using the Recap client and their storage in Cloudinary.



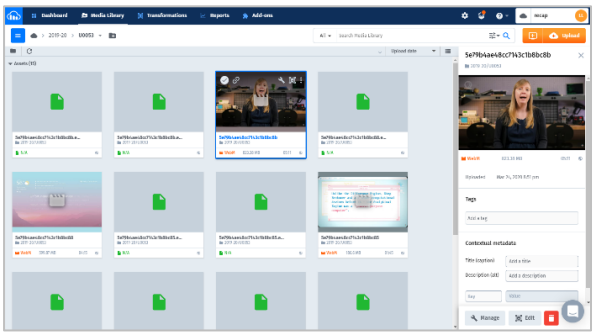
The Sprint delivered the necessary functionality for making and storing a lecture recording, and improvements to reliability and user experience were added in subsequent Sprints when the feature was ported from across the Administration microsite to the Portal.

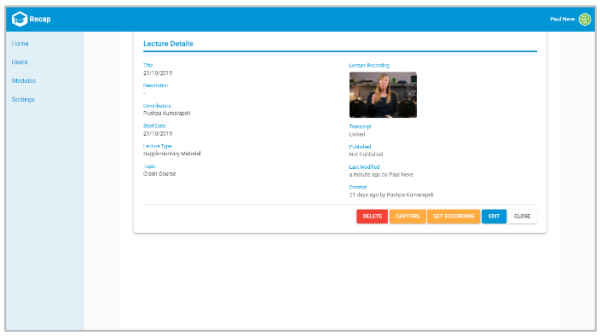




The High-Level Design opted to use the Recap API to upload the lecture recordings to the Linode Server before uploading the raw video file to the Media Library in Cloudinary. The rationale behind this was uploading the screen capture in chunks avoided forcing Lecturers to wait a long time after stopping a lecture recording, and these chunks needed to be reassembled on a server before being uploaded to Cloudinary. This two-step process caused a delay between when the lecture recording was stopped and when the raw video file was ready to be delivered

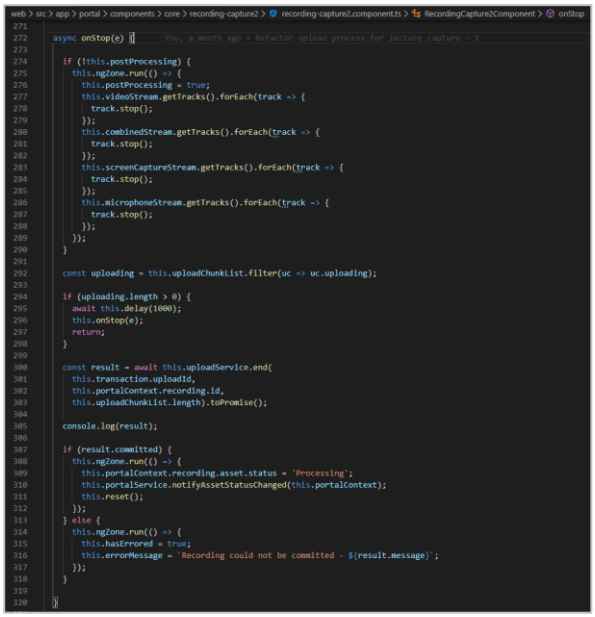
to the browser. A status associated with a lecture recording compensated for the delay. It enabled the user interface to inform the user when the video and transcript were processed, and thumbnails gave reassurance the footage was stored safely.





There were reliability and transcoding issues with the two-step upload process. The code was refactored several times to mitigate the issues, but in subsequent Sprints, problems persisted. The original code used promises for processing asynchronous requests and investigation discovered these did not always complete on the Recap client side. The code was refactored to use async-wait syntax, and this simplified the code and made it less convoluted.

```
web > src > app > portal > components > core > recording-capture > recording-capture.component.ts > RecordingCaptureComponent > pollToStop
280
281   pollToStop() {        You, 2 months ago • Add in outline for recording capture and editor
282     const delay = time => new Promise(resolve => setTimeout(resolve, time));
283     delay(1000).then(() => {
284       if (this.chunkQueue.findIndex(cq => !cq.completed) > -1) {
285         this.pollToStop();
286         return;
287       }
288       const unsuccessfulChunks = this.chunkQueue.filter(cq => !cq.success);
289       if (unsuccessfulChunks.length === 0) {
290         this.lectureService.uploadCaptureChunkProcess(this.portalContext.recording.id, this.chunkQueue.map(cq => cq.filename))
291           .subscribe((response) => {
292             this.portalContext.recording.asset.status = 'Processing';
293             this.portalService.notifyAssetStatusChanged(this.portalContext);
294             this.reset();
295           }, (error) => {
296             console.log(error);
297           });
298       } else {
299         const unsuccessfulChunksToRetry = unsuccessfulChunks.filter(c => !c.retried);
300         if (unsuccessfulChunksToRetry.length > 0) {
301           this.retryUnsuccessfulChunks(unsuccessfulChunksToRetry);
302           this.pollToStop();
303           return;
304         }
305         this.reset();
306         this.hasErrored = true;
307         this.errorMessage = 'Could not upload all parts of the recording';
308         this.postProcessing = false;
309       }
310     });
311   }
312
313   retryUnsuccessfulChunks(unsuccessfulChunksToRetry) {
314     console.log('Retrying ' + unsuccessfulChunksToRetry.length + ' chunk(s)');
315     unsuccessfulChunksToRetry.forEach((chunkQueueItem => {
316       chunkQueueItem.completed = false;
317       chunkQueueItem.retried = true;
318       this.lectureService.uploadCaptureChunk(this.portalContext.recording.id, chunkQueueItem.filename, chunkQueueItem.data)
319       .subscribe((event: HttpEvent<any>) => {
320         if (event.type === HttpEventType.UploadProgress) {
321         } else if (event.type === HttpEventType.Response) {
322           chunkQueueItem.completed = true;
323           chunkQueueItem.success = true;
324           chunkQueueItem.data = null;
325           this.chunkQueueSuccess++;
326         }
327       }, error => {
328         console.log(error);
329         chunkQueueItem.completed = true;
330         chunkQueueItem.success = false;
331         chunkQueueItem.data = null;
332         chunkQueueItem.error = error;
333       });
334     }));
335   }
```



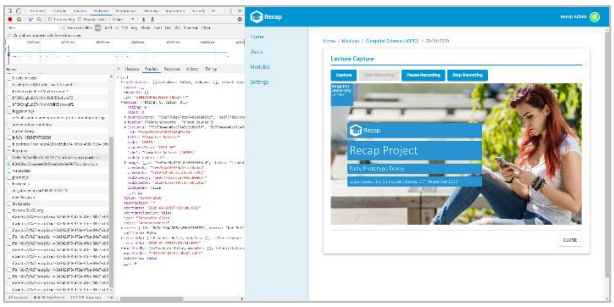
```
web > src > app > portal > components > core > recording-capture2 > recording-capture2.component.ts > RecordingCapture2Component > onStop
271
272   async onStop(e) {        You, a month ago • Refactor upload process for lecture capture - 1
273
274     if (!this.postProcessing) {
275       this.ngZone.run(() => {
276         this.postProcessing = true;
277         this.videoStream.getTracks().forEach(track => {
278           track.stop();
279         });
280         this.combinedStream.getTracks().forEach(track => {
281           track.stop();
282         });
283         this.screenCaptureStream.getTracks().forEach(track => {
284           track.stop();
285         });
286         this.microphoneStream.getTracks().forEach(track => {
287           track.stop();
288         });
289       });
290     }
291
292     const uploading = this.uploadChunkList.filter(uc => uc.uploading);
293
294     if (uploading.length > 0) {
295       await this.delay(1000);
296       this.onStop(e);
297       return;
298     }
299
300     const result = await this.uploadService.end(
301       this.transaction.uploadId,
302       this.portalContext.recording.id,
303       this.uploadChunkList.length).toPromise();
304
305     console.log(result);
306
307     if (result.committed) {
308       this.ngZone.run(() => {
309         this.portalContext.recording.asset.status = 'Processing';
310         this.portalService.notifyAssetStatusChanged(this.portalContext);
311         this.reset();
312       });
313     } else {
314       this.ngZone.run(() => {
315         this.hasErrored = true;
316         this.errorMessage = `Recording could not be committed - ${result.message}`;
317       });
318     }
319
320   }
```

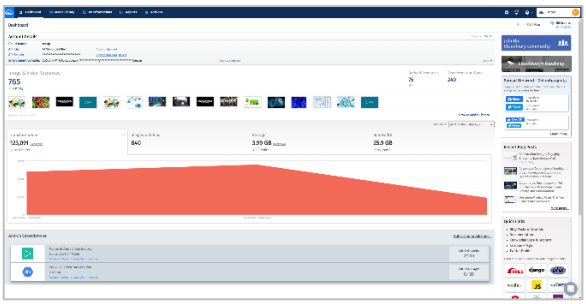
What was implemented, though, was a new UploadService to provide a more reliable messaging system between the Recap client and Recap API. To begin a new upload, the Recap client would request a transaction id. As the lecture recording was captured, each chunk was uploaded with the transaction id and a sequence no. After the lecture recording was stopped, the Recap client resent the transaction id with a sequence length, and the Recap API checked the sequence of chunks against the file system to validate whether all the chunks had been received. Some of the code implemented in the Recap API is shown overleaf.

```
api > controllers > upload-controller.js > begin > exports.begin

exports.begin = async (req, res, next) => {
  try
  {
    const lectureId = req.body.lectureId;
    const user = req.user;

    const newUploadId = await UploadManager.getNewUploadId(lectureId, user);

    res.status(200).send({
      lock: (newUploadId !== null),
      uploadId: newUploadId
    });

  } catch(error) {
    console.log(error);
    res.status(500).send({
      error: error.message
    });
  }
  finally {
    next();
  }
};

exports.uploadChunk = async (req, res, next) => {
  try
  {
    const uploadId = req.params.id;
    const sequenceNo = parseInt(req.query.sn);
    const chunkFilename = req.uploadChunk.filename;
    const file = req.file;
    let completed = (file && file !== null);

    if (completed) {
      const chunksPath = path.join(__dirname, '..', '..', 'assets', 'chunks');
      const chunkFilePath = path.join(chunksPath, chunkFilename);
      const chunkFileExists = fs.existsSync(chunkFilePath);
      completed = chunkFileExists;
      await UploadManager.setUploadChunkCompleted(uploadId, sequenceNo, completed);
    }

    console.log(`Chunk upload ${uploadId}-${sequenceNo} completed - ${completed}`);

    res.status(200).json({
      completed
    });

  } catch(error) {
    console.log(error);
    res.status(500).send({
      error: error.message
    });
  }
  finally {
    next();
  }
};
```

```
api > controllers > upload-controller.js > end > exports.end

exports.end = async (req, res, next) => {
  try        You, a month ago • Refactor upload process for lecture capture - 1
  {
    const uploadId = req.params.id;
    const lectureId = req.body.lectureId;
    const sequenceLength = req.body.sequenceLength;
    const upload = await UploadManager.getUpload(uploadId);

    const chunksPath = path.join(__dirname, '..', '..', 'assets', 'chunks');

    let sequencesReady = 0;
    for(let sequenceNo = 0; sequenceNo < sequenceLength; sequenceNo++) {
      const chunk = upload.chunks.find(c => c.sequenceNo === sequenceNo);
      if (chunk && chunk.completed) {
        const chunkFilePath = path.join(chunksPath, chunk.filename);
        const chunkFileExists = fs.existsSync(chunkFilePath);
        if (chunkFileExists) {
          sequencesReady++;
        }
      }
    }

    if (sequencesReady !== sequenceLength) {
      res.status(200).json({
        committed: false,
        message: `Sequence length ${sequenceLength}, Sequences Ready ${sequencesReady}`
      });
      return;
    }

    const lecture = await Lecture.findById(lectureId)
      .where('isDeleted')
      .equals(false);

    if(!lecture) {
      res.status(200).json({
        committed: false,
        message: 'Associated lecture not found!'
      });
      return;
    }
```

Integration and End-to-End testing instigated using test script or in the browser, with the benefit of Google Chrome Developer Tools for debugging purposes, proved sufficient for developing the solution to Proof of Concept standards. A considerable amount of testing was conducted, and the Cloudinary dashboard shows 840 images and videos were uploaded using 25.9GB of bandwidth over three months of testing.





The second User Story in the Sprint required video captioning to be enabled, which is a legal requirement. The transcript, sent in a response from Microsoft Azure to the Recap API by Cloudinary, provided the sentences needed to add a text track to the video player used by students to watch lecture recordings.

```
setVideoPlayerCaptions() {
  const textTrack = this.videoPlayer.addTextTrack('captions', 'English', 'en');
  this.portalContext.transcript.forEach(sentence => {
    textTrack.addCue(new VTTCue(sentence.start, sentence.end, sentence.text));
  });
}       You, 2 months ago • Add in outline for recording capture and editor
```

The response data sent by Cloudinary contained a UK English version of the transcript as requested by the Asset Manager when uploading the raw video file. The *info_data* attribute of the JSON document provided the transcript, along with keywords and topics. The transcript is

broken down into sentences with a confidence rating given by the algorithm indicating how accurate it thinks the audio to text conversion was.


```
{
  "responseData": {
    "info_kind": "azure_video_indexer",
    "info_status": "complete",
    "public_id": "2019-20/U0053/5e0b7dfa3b1ffb436efc7695",
    "uploaded_at": "2020-01-01T20:17:27Z",
    "version": 1577909847,
    "url": "http://res.cloudinary.com/recap/video/upload/v1577909847/2019-20/U0053/5e0b7dfa3b1ffb436efc7695.mp4",
    "secure_url": "https://res.cloudinary.com/recap/video/upload/v1577909847/2019-20/U0053/5e0b7dfa3b1ffb436efc7695.mp4",
    "etag": "e2f84f2b341e70228c4ff5bffb11374af",
    "notification_type": "info",
    "info_data": {
      "en-GB": {
        "version": "1.0.0.0",
        "duration": "0:03:01.023",
        "sourceLanguage": "en-US",
        "sourceLanguages": [
          "en-US"
        ],
        "language": "en-GB",
        "languages": [
          "en-GB"
        ],
        "transcript": [ ... ], // 48 items
        "keywords": [ ... ], // 30 items
        "topics": [ ... ], // 18 items
        "namedPeople": [ ... ], // 1 item
        "sentiments": [ ... ], // 2 items
        "emotions": [ ... ], // 1 item
        "blocks": [ ... ], // 6 items
        "speakers": [ ... ], // 5 items
        "textualContentModeration": { ... }, // 4 items
        "statistics": { ... } // 5 items
      },
      "en-US": { ... } // 16 items
    }
  }
}
```



```
{
  "responseData": {
    "info_kind": "azure_video_indexer",
    "info_status": "complete",
    "public_id": "2019-20/U0053/5e0b7dfa3b1ffb436efc7695",
    "uploaded_at": "2020-01-01T20:17:27Z",
    "version": 1577909847,
    "url": "http://res.cloudinary.com/recap/video/upload/v1577909847/2019-20/U0053/5e0b7dfa3b1ffb436efc7695.mp4",
    "secure_url": "https://res.cloudinary.com/recap/video/upload/v1577909847/2019-20/U0053/5e0b7dfa3b1ffb436efc7695.mp4",
    "etag": "e2f84f2b341e70228c4ff5bffb11374af",
    "notification_type": "info",
    "info_data": {
      "en-GB": {
        "version": "1.0.0.0",
        "duration": "0:03:01.023",
        "sourceLanguage": "en-US",
        "sourceLanguages": [
          "en-US"
        ],
        "language": "en-GB",
        "languages": [
          "en-GB"
        ],
        "transcript": [
          {
            "id": 1,
            "text": "Hello welcome my name is Lucas and I'm part of Project Group A 16 and I'd like to give you a brief overview of some of the prototyping work.",
            "confidence": 0.9139,
            "speakerId": 1,
            "language": "en-GB",
            "instances": [
              {
                "adjustedStart": "0:00:00",
                "adjustedEnd": "0:00:06.78",
                "start": "0:00:00",
                "end": "0:00:06.78"
              }
            ]
          },
          {
            "id": 2,
            "text": "I've done to date and steps are taken towards that for the project familiar friends with my team.",
            "confidence": 0.9129,
            "speakerId": 1,
            "language": "en-GB",
            "instances": [
              {
                "adjustedStart": "0:00:06.78",
                "adjustedEnd": "0:00:14.68",
                "start": "0:00:06.78",
                "end": "0:00:14.68"
              }
            ]
          },
```


The Notification Controller in the Recap API provided endpoints for processing Cloudinary notifications, and the Asset Manager set the corresponding endpoint. After a raw video file was uploaded, multiple notifications were sent to the Recap API and processed by the postVideoUploaded function. When the notification type was marked as “info” this function forwarded the notification for the Transcript Manager to handle.

```
api > controllers > notification-controller.js > postVideoUploaded > exports.postVideoUploaded
exports.postVideoUploaded = async (req, res, next) => {        You, 3 months ago • Add queues to support uploaded and transcript
  const notification = req.body;
  const notificationType = notification.notification_type;
  const publicId = notification.public_id;
  const infoKind = notification.info_kind;
  const infoStatus = notification.info_status;
  const Message = {
    message: 'Video notification received',
    notification_type: notificationType,
    public_id: publicId,
    info_kind: infoKind,
    info_status: infoStatus,
    received: Date(Date.now()).toString(),
    error: null
  };

  console.log('Received video upload notification from cloudinary');
  console.log(message);

  try
  {
    const notificationData = new Notification({
      message: 'Received video upload notification from cloudinary',
      type: notificationType,
      publicId: publicId,
      infoKind: infoKind,
      infoStatus: infoStatus,
      received: message.received,
      data: notification
    });
    await notificationData.save();
    if (notificationType) {
      const lecture = await lecture.findOne({'asset.publicId': publicId})
      if (!lecture || lecture.isDeleted) {
        console.log('Could not find associated lecture or lecture deleted.')
      }
      if (notificationType === 'upload') {
        const job = await AssetSyncManager.onVideoUploaded(lecture._id, publicId);
      } else if (notificationType === 'info') {
        const transcriptSync = await TranscriptManager.sync(notification);
        console.log(transcriptSync);
        if(transcriptSync.synced) {
          const job = await AssetSyncManager.onVideoTranscriptReady(lecture._id, publicId, infoKind, infoStatus);
        }
      }
    }
    var response = JSON.stringify(req.body);
    var filename = req.body.public_id ? req.body.public_id.replace(/\//g,'_') : 'test';
    if (notificationType) {
      filename = notificationType + '_' + filename;
    }
    if (infoKind) {
      filename = infoKind + '_' + infoStatus + '_' + filename;
    }
    var filePath = path.join(__dirname, '..', '..', 'assets', 'cloudinary', 'notifications', 'video_' + filename + '.json');
    console.log('Saving video upload response to ' + filePath);
    fs.writeFileSync(filePath, response, 'utf8', 4);

    res.status(200).json(message);
  }
  catch(error) {
    message.message = 'Could not process notification';
    message.error = error;
    console.log(message);
    res.status(500).json(message);
  }
};
```

```
api > database > transcript-manager.js > sync > exports.sync
        const infoKind = notification.info_kind;
        if (!infoKind || infoKind !== 'azure_video_indexer') {
            response.message = 'Unsupported notification info kind ' + infoKind;
            return response;
        }

        const infoStatus = notification.info_status;
        if (!infoStatus) {
            response.message = 'Unsupported notification format - info status not provided';
            return response;
        }

        if (infoStatus !== 'complete') {
            lecture.asset.textStatus = infoStatus;
            await lecture.save();
            response.message = 'Unsupported notification info status ' + infoStatus;
            return response;
        }

        const infoDataEnglish = notification.info_data['en-GB'] ? notification.info_data['en-GB'] : null;
        if (infoDataEnglish === null) {
            response.message = 'Language en-GB not found';
            return response;
        }

        console.log('en-GB transcript found');

        if (!infoDataEnglish.transcript) {
          infoDataEnglish.transcript = [];        You, 3 months ago • Fix null array bug in sync script
        }

        const transcript = infoDataEnglish.transcript
            .map(sentence => new Transcript({
                id: sentence.id,
                lecture: lecture._id,
                isSegment: false,
                title: null,
                topics: [],
                keywords: [],
                text: sentence.text,
                confidence: sentence.confidence,
                start: getTimeInSeconds(sentence.instances[0].start),
                end: getTimeInSeconds(sentence.instances[0].end)
            }));

        console.log(transcript.length + ' sentences to be processed');

        if (!infoDataEnglish.topics) {
          infoDataEnglish.topics = [];
        }

        const topics = infoDataEnglish.topics
            .filter(topic => topic.confidence > 0.6)
            .map(topic => {
                return {
                term: ' ',
                text: topic.name,
                instances: topic.instances.map(instance => {
                    return {
                    start: getTimeInSeconds(instance.start),
                    end: getTimeInSeconds(instance.end)
                    }
                })
                }
            });
```

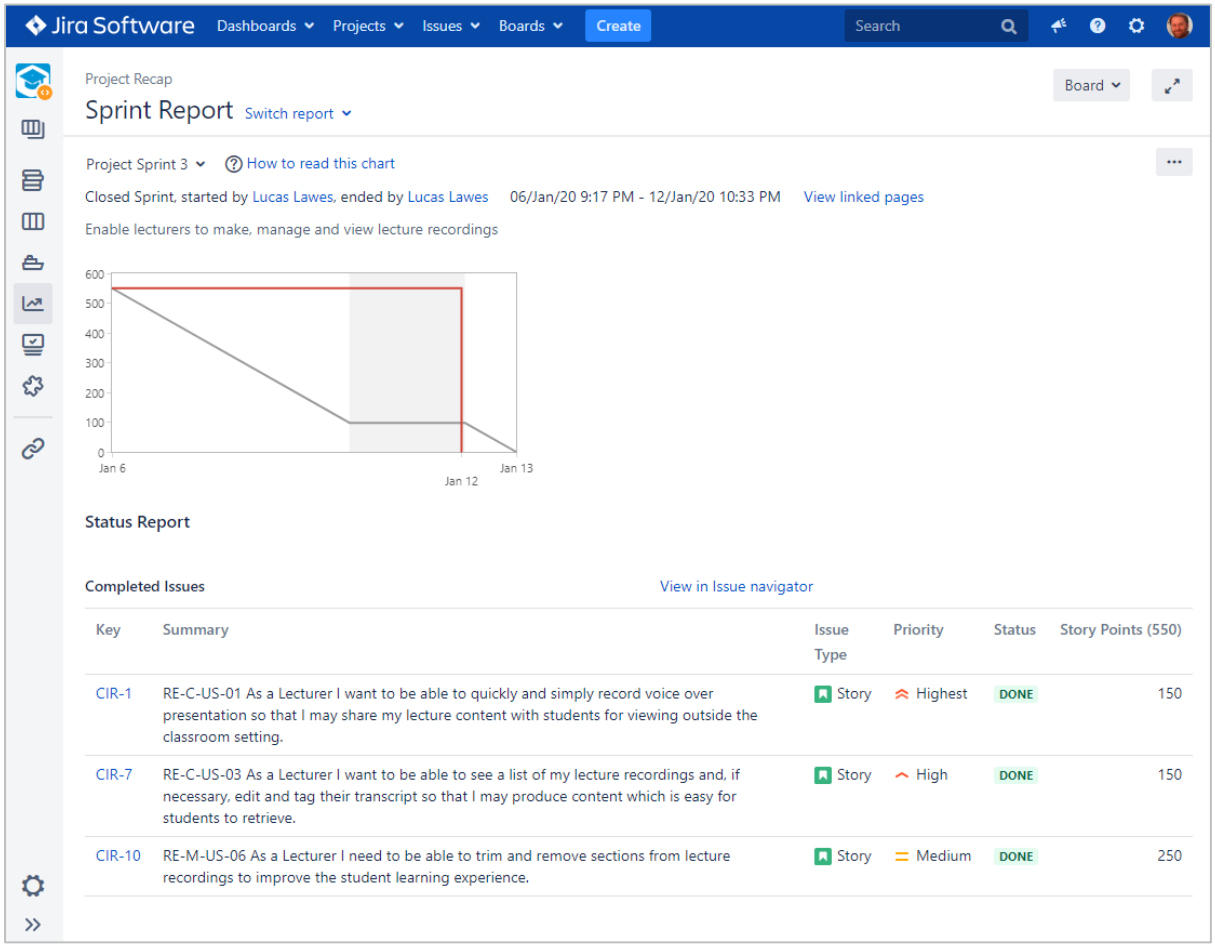
Technically the audio-to-text conversion and transcript synchronisation worked well although there were a few bugs to fix during testing. However, the accuracy of the transcript text from a translation perspective was variable and, testing identified the Microsoft Azure Video Index API had trouble understanding all voice types, mainly when the audio recording was of inferior quality. There was a temptation to switch the transcript duties to the Google AI Video Transcription service. However, ultimately audio-to-text conversion is still in its infancy, and

so the implementation was left unchanged for the final year project.

The end of Sprint 2 marked a milestone in the implementation with the completion of the critical path. Integration between the key backend components and an Administration microsite which supported lecture capture in a web browser laid the foundations for a viable product. Most importantly, the solution now had data and an established link between lecture information and recordings. The challenge now was how best to exploit this data to realise a lecture capture solution for KU.

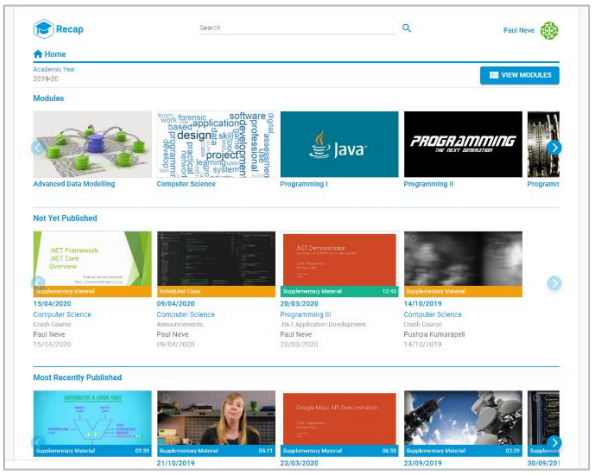
Sprint 3

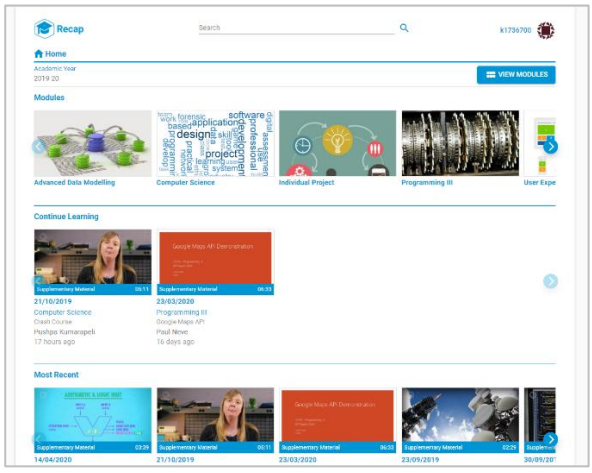
A demonstration was given to the client team, and feedback input into Sprint 3 (see Design Wireframes for changes). The Sprint goal was to make a start on the Consume Epic and enable Lecturers to make, manage and view lecture recordings. The main feedback was that Lecturers would not want to use the Administration microsite. So, some of the development work involved porting across some of the functionality into the Portal. The three User Stories totalled 550 story points. Still, some of the work had already been completed, and, the ability to edit lecture recordings (RE-M-US-06) would be delivered using the Cloudinary video transformation API; drastically reducing development effort.



The Recap client portal became a key differentiator over competitor products. By designing a role-based portal, lecturers and students had a single point of access for recording and consuming lectures.

The lecturer's view of the home page is shown on the left, and the student's view is shown on the right.





An Angular application is modular and supports lazy loading, and this enabled the administration microsite and portal to be contained in the same application. The modular approach allowed components to be shared, and lazy loading meant only necessary components downloaded, ensuring the application loaded quickly despite hosting two complete applications.

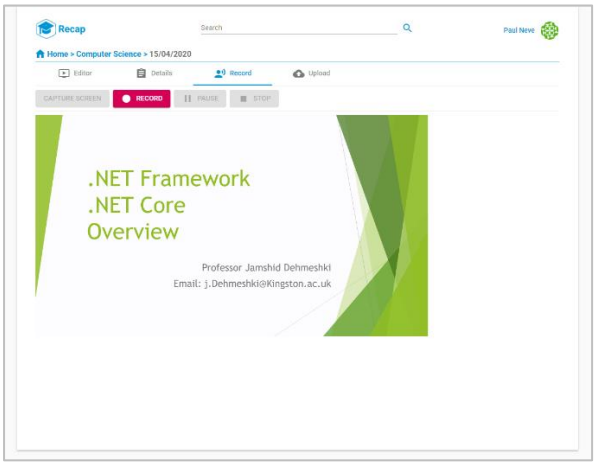
Lazy loading was configured in the routing definition and resolvers, such as the PortalContextResolver, enabled data to be loaded before the view components were initialised, making the user experience clean and fluid.

```
web > src > app > portal > portal-routing.module.ts > routes > children
You, 2 months ago | 1 author (You)
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './components/home/home.component';
import { PortalComponent } from './components/portal/portal.component';
import { SearchComponent } from './components/search/search.component';
import { PortalContextResolver } from '../services/portal-context-resolver';
import { ModuleViewComponent } from './components/module-view/module-view.component';
import { LectureViewComponent } from './components/lecture-view/lecture-view.component';
import { ModuleListComponent } from './components/catalogue/module-list/module-list.component';

const routes: Routes = [
    {
        path: '',
        component: PortalComponent,
        children: [
            {
                path: 'home',
                component: HomeComponent,      You, 3 months ago • Wire up portal context
                resolve: {
                    portalContext: PortalContextResolver
                }
            },
            {
                path: 'search',
                component: SearchComponent
            },
            {
                path: 'browse',
                component: ModuleListComponent
            },
            {
                path: 'content/:moduleCode',
                component: ModuleViewComponent,
                resolve: {
                    portalContext: PortalContextResolver
                }
            },
            {
                path: 'content/:moduleCode/:lectureId',
                component: LectureViewComponent,
                resolve: {
                    portalContext: PortalContextResolver
                }
            },
            {
                path: '**',
                redirectTo: 'home'
            }
        ]
    }
];

export const PortalRoutingModule = RouterModule.forChild(routes);
```

```
web > src > app > portal > components > home > home.component.ts > ...
You, 3 months ago | 1 author (You)
import { Component, OnInit, OnDestroy } from '@angular/core';      You, 3 months ago • Get flickity working
import { Router, ActivatedRoute, ParamMap, NavigationEnd } from '@angular/router';
import { AuthService } from 'src/app/services/auth.service';
import { PortalContext } from 'src/app/models/portal-context';
import { CurrentUser } from 'src/app/models/current-user';
import { PortalService } from 'src/app/services/portal-service';
import { PortalRequest } from 'src/app/models/portal-request';
You, 3 months ago | 1 author (You)
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss']
})
export class HomeComponent implements OnInit, OnDestroy {

  currentUser = new CurrentUser();
  portalRequest = new PortalRequest();
  portalContext = new PortalContext();

  navigationSubscription: any;

  constructor(
    private authService: AuthService,
    private portalService: PortalService,
    private route: ActivatedRoute,
    private router: Router
  ) {
    this.navigationSubscription = this.router.events.subscribe((e: any) => {
      if (e instanceof NavigationEnd) {
        this.portalRequest = this.portalService.getRequestFromUrl();
      }
    });
    this.route.paramMap.subscribe((paramMap: ParamMap) => {
      this.portalContext = route.snapshot.data.portalContext;
    });
  }

  ngOnInit() {
    this.currentUser = this.authService.getCurrentUser();
    this.portalRequest = this.portalService.getRequestFromUrl();
  }

  ngOnDestroy() {
    if (this.navigationSubscription) {
      this.navigationSubscription.unsubscribe();
    }
  }
}
```

Using resolvers also allowed related components to share the same data. The portal page loaded once and as the context changed, the PortalContextResolver supplied the data needed to drive the child components. The child components used 25 components to tailor the views for the Lecturers and Students.

Providing a UI for recording a lecture in the browser gave lecturers control over what to record and when. No additional downloads or installations were necessary.

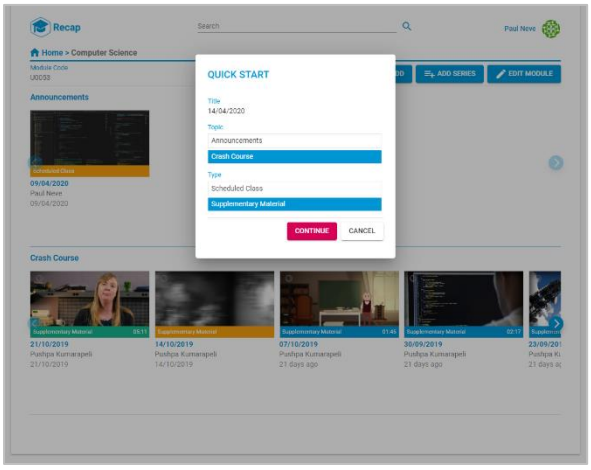


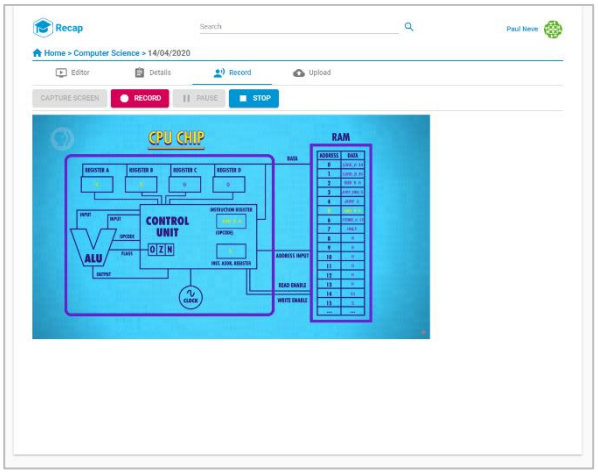
The lecture capture component captured both the screen and the microphone using the Media API. The *onScreenCapture* function in the *RecordingCaptureComponent* was responsible for using the Media API. Testing uncovered issues with capturing the screen and audio tracks using the Display Media API, which prompts the user to select a screen source and the audio. Capturing audio using this API was reliant on users remembering to select audio as well, and when audio was selected, the soundtrack quality was sometimes poor. The User Media and Audio Stream APIs enabled the microphone to be selected, and after extensive research, a pattern was discovered for combining the video and audio tracks. These combined tracks could then be passed to the Media Recorder API which provided output in chunks of the combined recording. Event handlers were attached to the Media Recorder to pass the lecture recording to the Upload Service which sent it to the cloud for processing and storage.

```
web > src > app > portal > components > core > recording-capture2 > recording-capture2.component.ts > RecordingCapture2Component > onCaptureScreen
153   async onCaptureScreen() {      You, a month ago • Refactor upload process for lecture capture - 1
154     try {
155
156       const displayMediaConstraints = {
157           video: true,
158           audio: false
159       };
160
161       const navigatorWrapper = (navigator as unknown) as any;
162       this.screenCaptureStream = null;
163
164       try {
165         this.screenCaptureStream = await navigatorWrapper.mediaDevices.getDisplayMedia(displayMediaConstraints);
166         this.screenCaptureStream.getVideoTracks().forEach(videoTrack => {
167           videoTrack.onended = (e) => {
168             this.onStopSharing();
169           };
170         });
171       } catch (error) {
172         if (error.message === 'Permission denied' || (error.name && error.name === 'NotAllowedError')) {
173           return;
174         }
175         this.errorMessage = 'Browser does not support screen capture';
176         this.hasErrored = true;
177         return;
178       }
179
180       this.microphoneStream = null;
181
182       try {
183         this.microphoneStream = await navigator.mediaDevices.getUserMedia({ video: false, audio: true });
184       } catch (error) {
185         if (error.message === 'Permission denied' || (error.name && error.name === 'NotAllowedError')) {
186           return;
187         }
188         this.errorMessage = 'No audio microphone found';
189         this.hasErrored = true;
190         return;
191       }
192
193       // tslint:disable-next-line: no-string-literal
194       const audioContext = new (window['AudioContext'] || window['webkitAudioContext'])();
195       const audioDestination = audioContext.createMediaStreamDestination();
196       const audioSource = audioContext.createMediaStreamSource(this.microphoneStream);
197       const voiceGain = audioContext.createGain();
198
199       voiceGain.gain.value = 0.7;
200       audioSource.connect(voiceGain).connect(audioDestination);
201
202       const audioTracks = audioDestination.stream.getAudioTracks();
203
204       const combinedTracks = [
205         ...this.screenCaptureStream.getVideoTracks(),
206         ...audioTracks
207       ];
208
209       this.combinedStream = new MediaStream(combinedTracks);
210       this.videoStream = this.screenCaptureStream;
211       this.mediaRecorder = new MediaRecorder(this.combinedStream, { mimeType: 'video/webm; codecs=vp9,opus' });
212       this.mediaRecorder.ondataavailable = (e) => this.onDataAvailable(e);
213       this.mediaRecorder.onstop = (e) => this.onStop(e);
214     } catch (error) {
215       this.errorMessage = 'Could not capture screen and microphone. ' + error.message ? ' - ' + error.message : '';
216       this.hasErrored = true;
217       console.error('Error: ' + error);
218     }
219   }
```

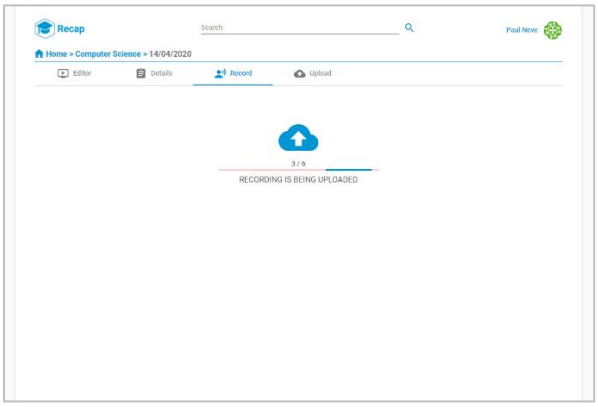
```
web > src > app > services > upload-service.ts > UploadService > constructor
      You, a month ago | 1 author (You)
1     import { Injectable } from '@angular/core';
2     import { HttpClient, HttpErrorResponse, HttpHeaders } from '@angular/common/http';
3     import { Observable, throwError } from 'rxjs';
4     import { map, catchError } from 'rxjs/operators';
5
6     import { environment } from '../../environments/environment';
7     import { sequence } from '@angular/animations';
8
9     const BACKEND_URL = environment.apiUrl + '/uploads/';
10
      You, a month ago | 1 author (You)
11    @Injectable({ providedIn: 'root' })
12    export class UploadService {
13
14      constructor(private http: HttpClient) {      You, a month ago • Refactor upload process for lecture capture - 1
15      }
16
17      begin(lectureId: string): Observable<any> {
18        return this.http.post<{ lock: boolean, uploadId: string }>(
19            BACKEND_URL + 'begin', {
20              lectureId
21            });
22      }
23
24      uploadChunk(uploadId: string, sequenceNo: number, data: Blob): Observable<any> {
25        const params = '?sn=${sequenceNo}';
26        const formData = new FormData();
27        formData.append('chunk', data);
28        return this.http.post<{ completed: boolean }>(
29            BACKEND_URL + uploadId + '/uploadChunk' + params,
30            formData);
31      }
32
33      reset(lectureId: string): Observable<any> {
34        return this.http.post<{ status: string }>(
35            BACKEND_URL + 'reset', {
36              lectureId
37            });
38      }
39
40      end(uploadId: string, lectureId: string, sequenceLength: number): Observable<any> {
41        return this.http.post<{ committed: boolean, message: string }>(
42            BACKEND_URL + uploadId + '/end', {
43              lectureId,
44              sequenceLength
45            });
46      }
47    }
48
```

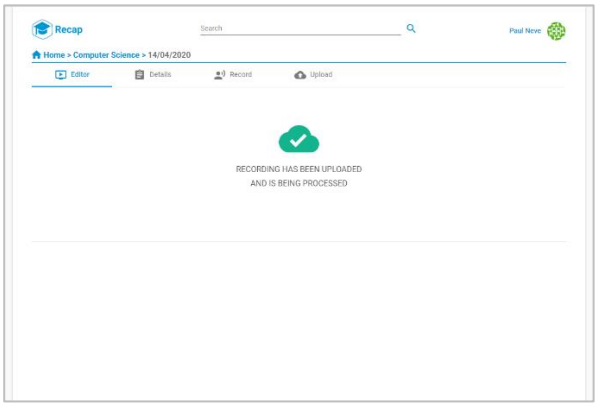
Particular attention was paid to the user experience for quickly creating new lecture recordings on the fly. A “quick start” workflow was added in Sprint 4 and is shown below in three stages. Lecturers were expected to select a Module such as “Computer Science” from the home page, from where they could start a “quick start”.



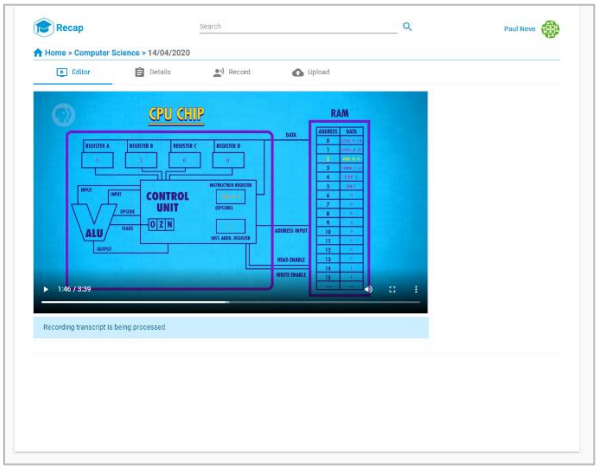


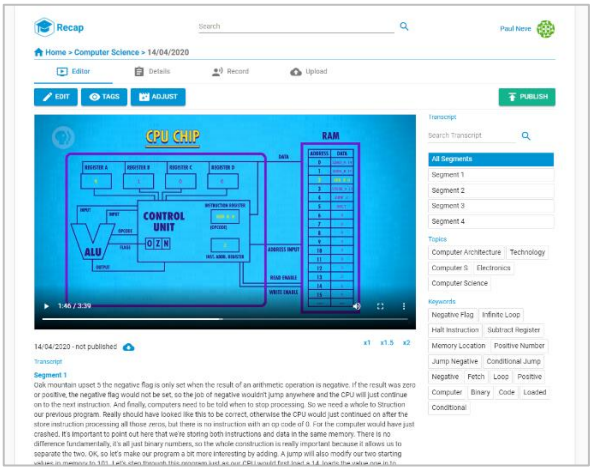
When a Lecturer stopped a lecture recording, the screen displayed a progress bar while the remaining chunk uploads completed.



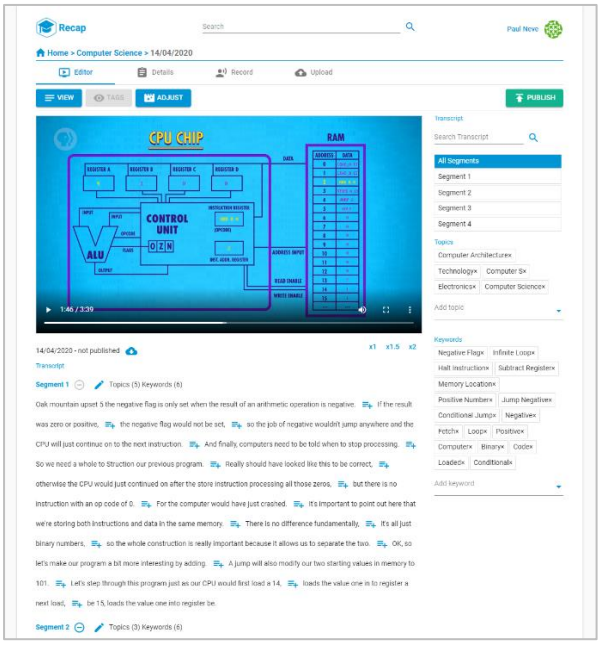


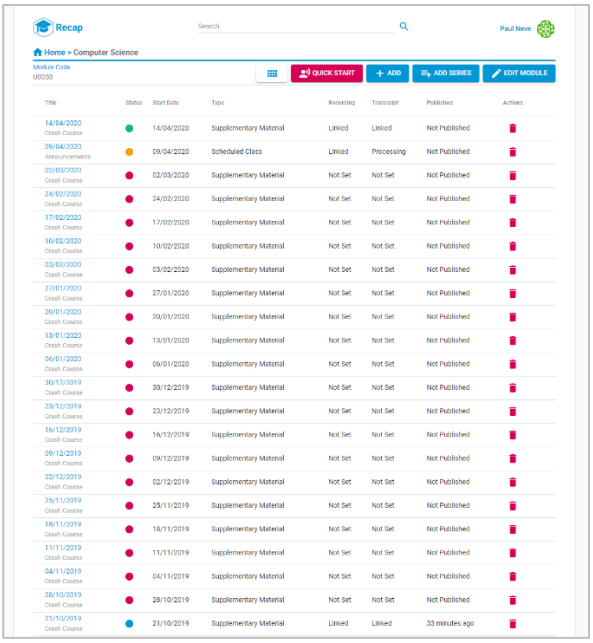
Once the upload completed, the Lecturer was free to close the browser or wait for the raw video file to be processed.



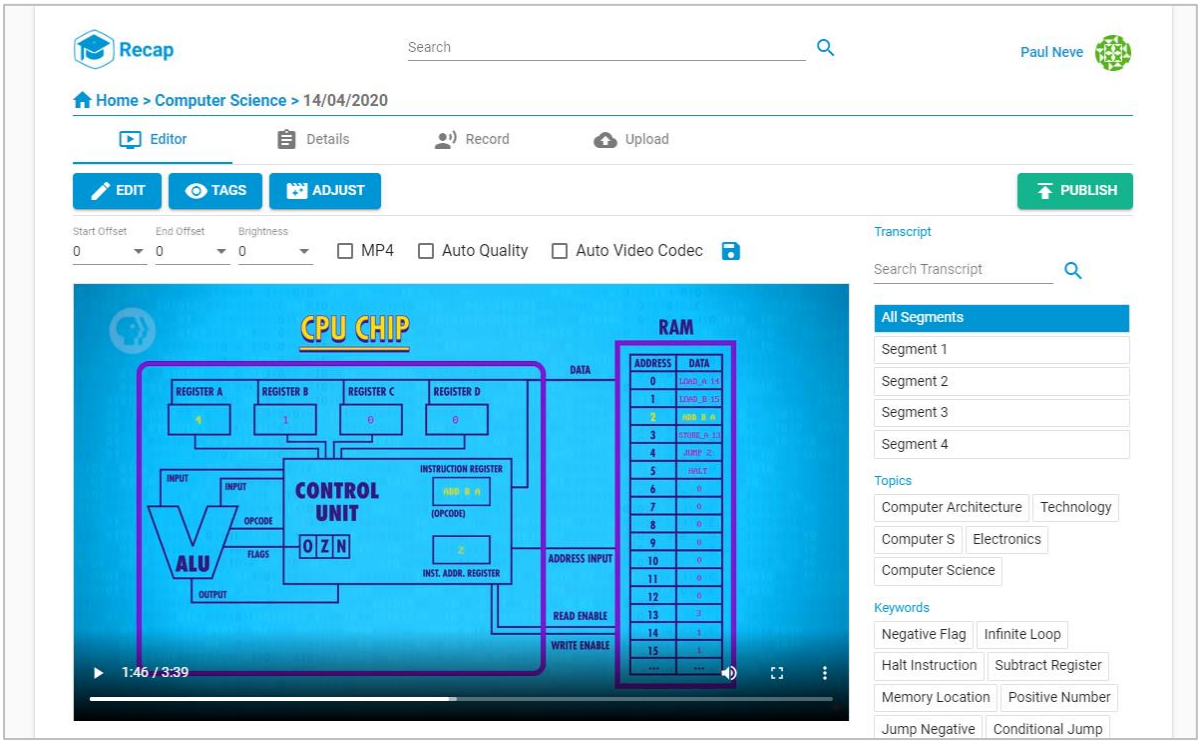


Not many lecturers would want to regularly edit the lecture recording transcript as it is a labour-intensive task. However, simple to use functionality was added so this could still be done. Lectures were able to change the text, add and remove segments, and edit the topics and keywords without refreshing the page. A list view was provided to enable Lecturers to manage lecture recordings.



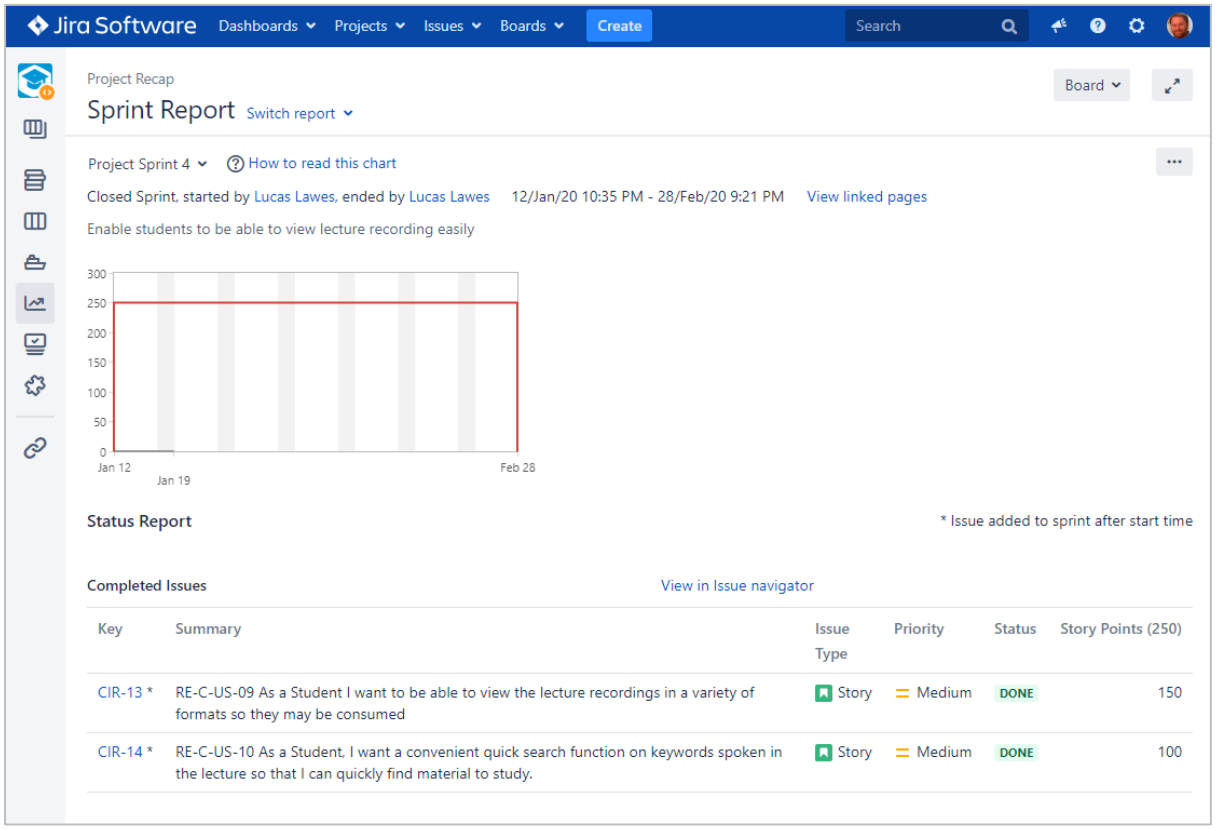


The finally User Story in Sprint 3 was to enable Lecturers to make changes to the lecture recording video. The Cloudinary video transformation API made it possible to provide the means to make adjustments on the fly via the browser. The API is URL parameter-driven, and video settings for trimming, adjusting the brightness and correcting the playback could be made on the fly by Lecturers.

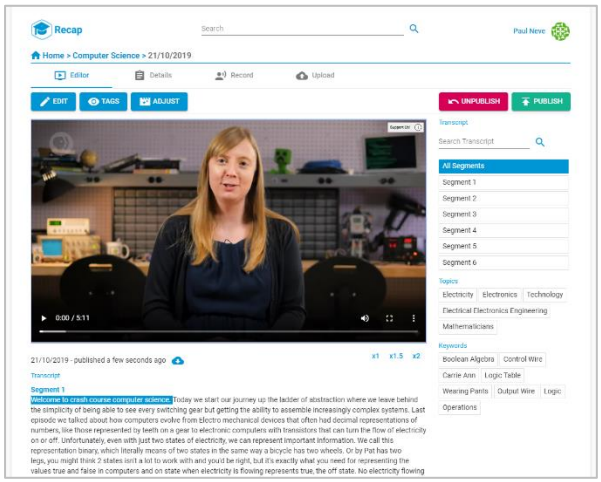


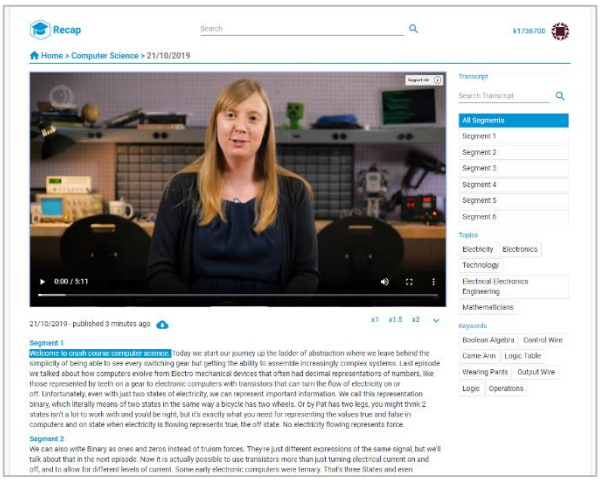
Sprint 4

The goal of Sprint 4 was to enable students to view lecture recordings easily. However, it also involved an immense amount of rework for improving the user experience of the portal for lecturers as well.



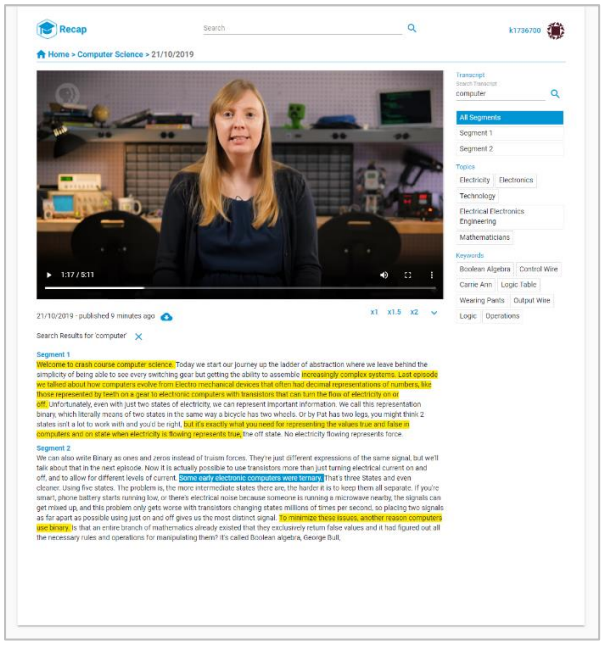
Separate view components were developed to render the lecture recording views for lecturers and students (shown below) to make the codebase easy to write and maintain as they offered different levels of functionality and drew data from separate data sources.

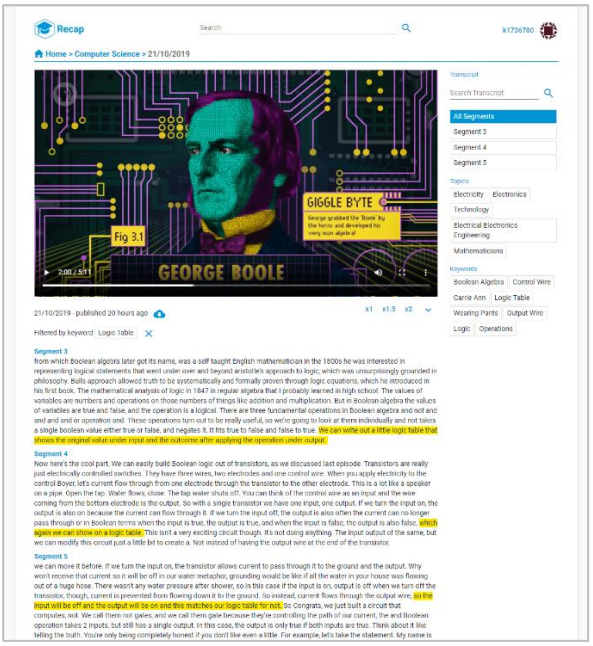




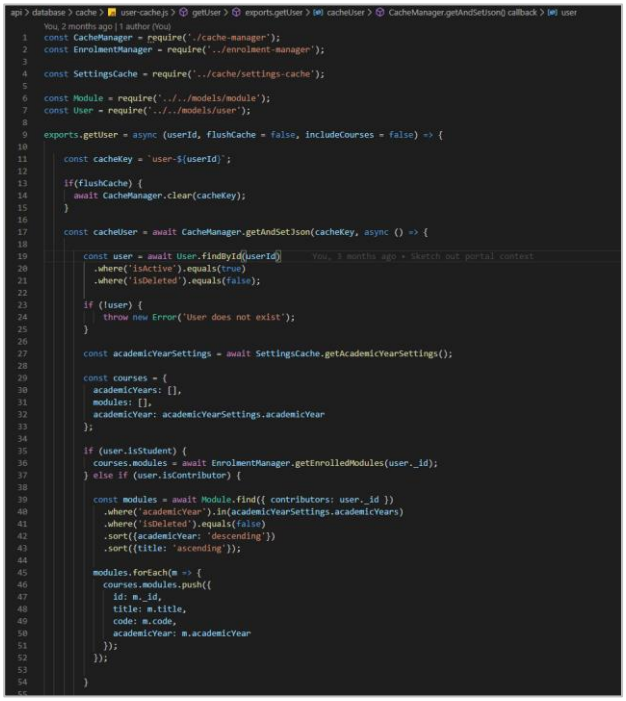
The Media Recorder API output the lecture recording track as “webm” and while Cloudinary supported the format, the playback experience was not satisfactory. When the lecture recording was transcoded to MP4, the playback experience was vastly superior, and so this file format was used to deliver video.

Students and lecturers were given the ability to drill down into the transcript by using segments, topics, and keywords, as well as text search. Selecting the transcript text moved the position of the video track to the same spot enabling users to drill into segments of the lecture recording as well.

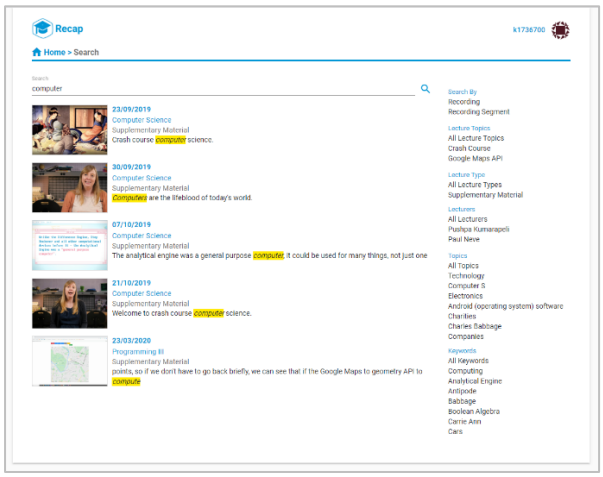


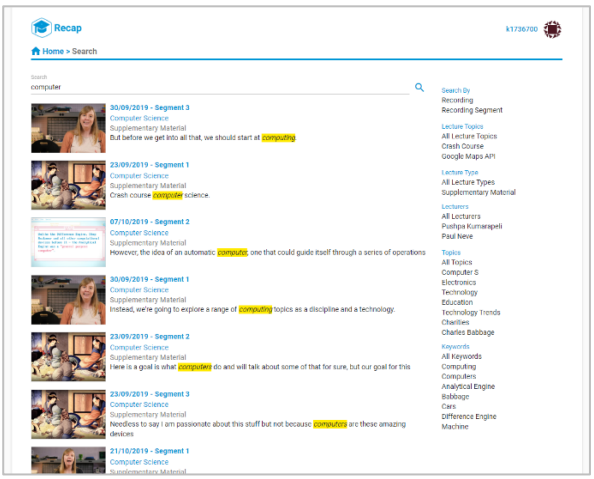


Part of the refactoring effort was to accommodate using Redis as a caching engine. A simple Cache Manager was developed and used to cache settings and other commonly used data objects, such as user information. This in-memory cache reduced Recap API response times.

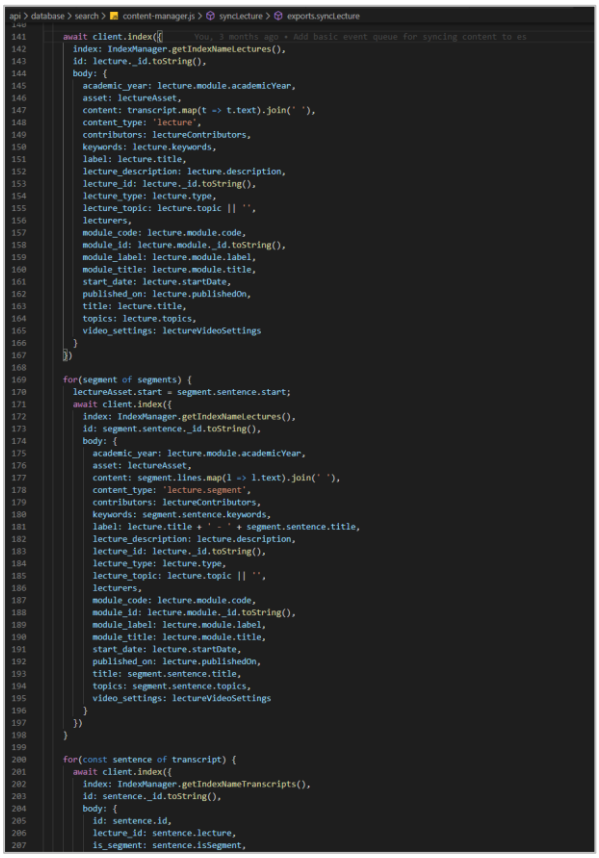


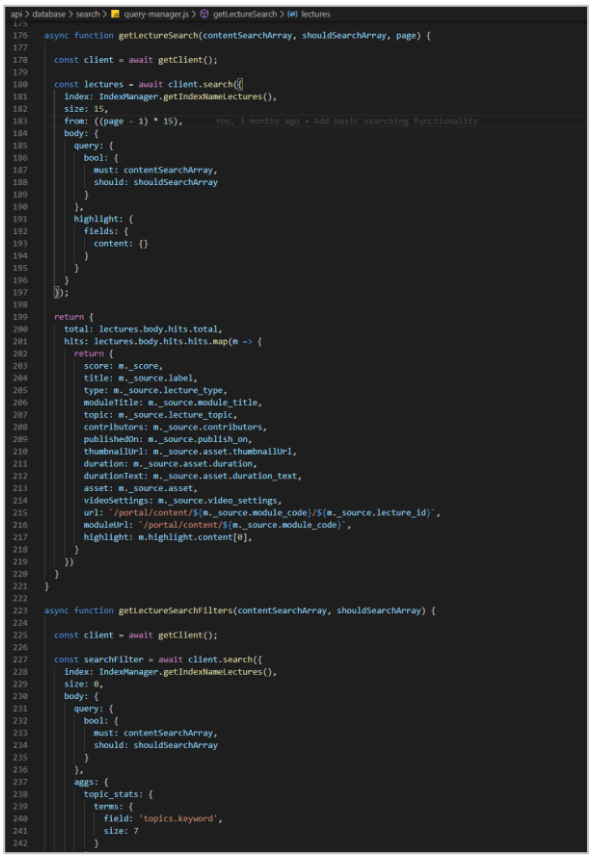
ElasticSearch provided intelligent keyword search across lecture recordings and segments. It also provided search filters for refining searches, such as to search by Recording or Recording Segment.





When a Lecturer published a lecture recording, it triggered an event in the Recap API to index it in ElasticSearch. The lecture recording could then be retrieved by running a lecture search.

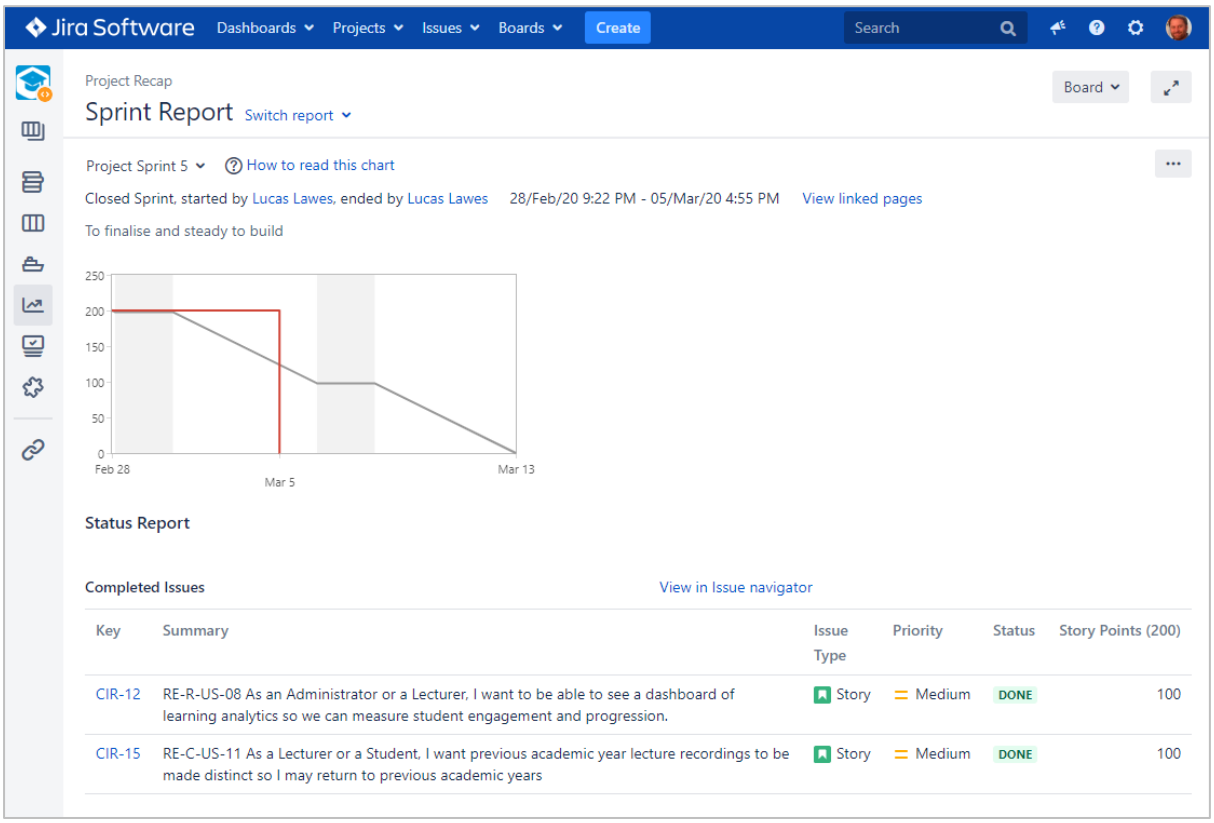




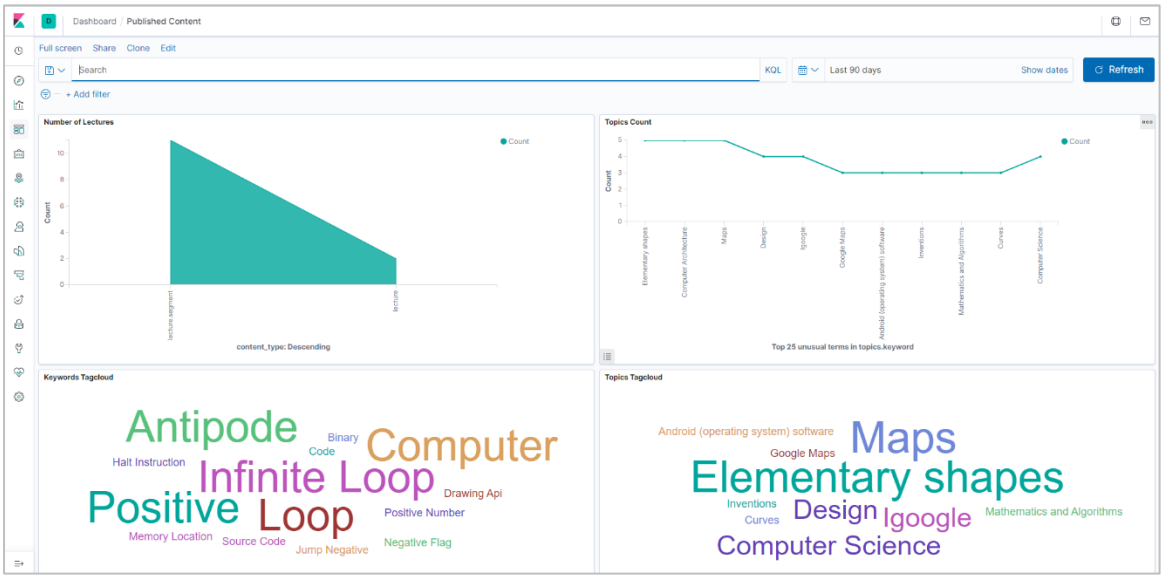
Feedback from the client team during Sprint 4 was incredibly helpful, and the solution began to take shape. Courses were dropped from the database schema and replaced with Academic Years. Changing a root document during the later stages of the development process created a fundamental change to the UI and codebases. Refactoring in the requirement changes vastly improved the solution but made Sprint 4 the equivalent of two sprints.

Sprint 5

While the sprint goal of Sprint 5 was to finalise and steady the build, it also involved setting up a Learning Analytics dashboard and enabling historical lecture recordings to be accessed.



With lecture recordings synchronising to ElasticSearch, Kibana was installed on the Linode Server, and a dashboard created. It displayed the number of lectures, segments, topics, and keywords in the index. The data visualisations configured manually and provided interesting insights.

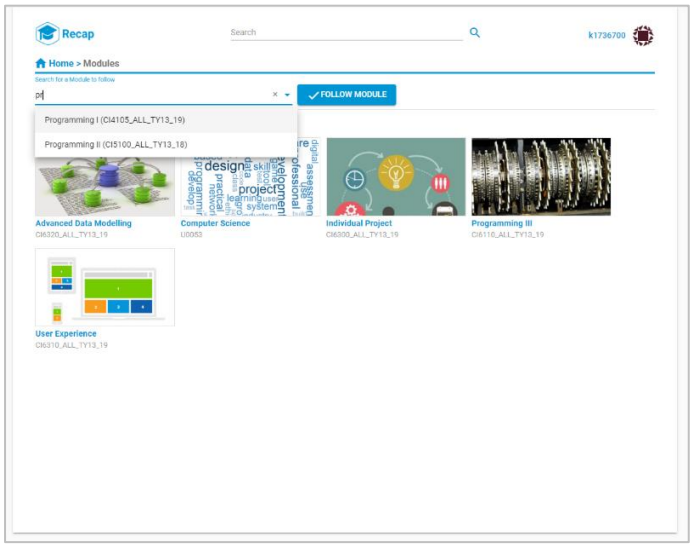


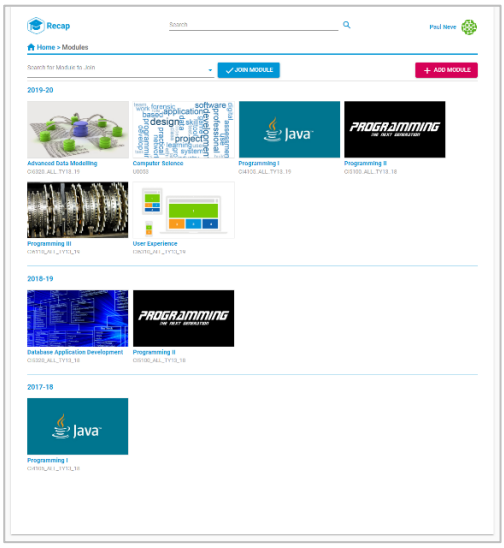
During Sprint 5, the Search Sync Manager and Index Manager were refactored to reflect Course schema changes from Sprint 4, and improve the quality of the code; for example, to increase the performance of deleting documents from the index which had been too slow up until then.

```
api > database > search > search-sync-manager.js > connect > exports.connect
You, 3 months ago | 1 author (You)
const Queue = require('bull');
const ContentManager = require('./content-manager');

const SYNC_QUEUE_NAME = 'elastic-search-sync-queue';
const SYNC_QUEUE_JOB_ON_REINDEX = 'onReindex';
const SYNC_QUEUE_JOB_ON_MODULE_CHANGE = 'onModuleChange';
const SYNC_QUEUE_JOB_ON_MODULE_VIEW = 'onModuleView';
const SYNC_QUEUE_JOB_ON_MODULE_DELETE = 'onModuleDelete';
const SYNC_QUEUE_JOB_ON_LECTURE_CHANGE = 'onLectureChange';
const SYNC_QUEUE_JOB_ON_LECTURE_VIEW = 'onLectureView';
const SYNC_QUEUE_JOB_ON_LECTURE_RECORDING_VIEW = 'onLectureRecordingView';
const SYNC_QUEUE_JOB_ON_LECTURE_DELETE = 'onLectureDelete';

exports.connect = async() => {      You, 3 months ago • Add basic event queue for syncing content to es
    try {

        const qm = await this.getQueueManager();

        if (qm.isConnected) {
            return 'Already connected!';
        }

        qm.syncQueue = new Queue(SYNC_QUEUE_NAME);
        qm.syncQueue.process(this.syncQueueJobHandler);

        qm.isConnected = true;

        return `Attached to redis queue "${SYNC_QUEUE_NAME}"`;
    } catch (error) {
        throw new Error('Could not connect to redis queue');
    }
}

exports.onReindex = async (moduleId) => {
    return await this.addSyncQueueJob(SYNC_QUEUE_JOB_ON_REINDEX, { moduleId })
}

exports.onModuleChange = async (moduleId) => {
    return await this.addSyncQueueJob(SYNC_QUEUE_JOB_ON_MODULE_CHANGE, { moduleId })
}

exports.onModuleView = async (moduleId) => {
    return await this.addSyncQueueJob(SYNC_QUEUE_JOB_ON_MODULE_VIEW, { moduleId })
}

exports.onModuleDelete = async (moduleId) => {
    return await this.addSyncQueueJob(SYNC_QUEUE_JOB_ON_MODULE_DELETE, { moduleId })
}

exports.onLectureChange = async (lectureId) => {
    return await this.addSyncQueueJob(SYNC_QUEUE_JOB_ON_LECTURE_CHANGE, { lectureId })
}

exports.onLectureView = async (moduleId) => {
    return await this.addSyncQueueJob(SYNC_QUEUE_JOB_ON_LECTURE_VIEW, { moduleId })
}

exports.onLectureRecordingView = async (moduleId) => {
    return await this.addSyncQueueJob(SYNC_QUEUE_JOB_ON_LECTURE_RECORDING_VIEW, { moduleId })
}

exports.onLectureDelete = async (lectureId) => {
    return await this.addSyncQueueJob(SYNC_QUEUE_JOB_ON_LECTURE_DELETE, { lectureId })
}
```

```
api > database > search > index-manager.js > createLecturesIndex > exports.createLecturesIndex
You, 3 months ago | 1 author (You)
const searchManager = require('./search-manager');

const INDEX_NAME_LECTURES = 'lectures';
const INDEX_NAME_TRANSCRIPTS = 'transcripts';

exports.getIndexNameLectures = () => {
    return INDEX_NAME_LECTURES;
};

exports.getIndexNameTranscripts = () => {
    return INDEX_NAME_TRANSCRIPTS;
};

exports.createLecturesIndex = async () => {

    const client = await getClient();

    const exists = await client.indices.exists({ index: INDEX_NAME_LECTURES});

    if(exists.body) {
        console.log(`Deleting index - ${INDEX_NAME_LECTURES}`);
        await client.indices.delete({
            index: INDEX_NAME_LECTURES
        });
    }

    console.log(`Creating index - ${INDEX_NAME_LECTURES}`);
    await client.indices.create({
      index: INDEX_NAME_LECTURES
    });

    console.log(`Mapping index - ${INDEX_NAME_LECTURES}`);
    await client.indices.putMapping({
        index: INDEX_NAME_LECTURES,      You, 3 months ago • Implement onLectureChange, Delete and onReindex
        body: {
            properties: {
                content_type: {
                    type: 'keyword',
                },
                module_id: {
                    type: 'keyword',
                },
                module_title: {
                    type: 'text',
                    fields: {
                        keyword: {
                            type: 'keyword'
                        }
                    }
                },
                module_code: {
                    type: 'text',
                },
                module_label: {
                    type: 'text',
                },
                lecture_id: {
                    type: 'keyword',
                },
                lecturers: {
                    type: 'keyword',
                },
                lecture_description: {
                    type: 'text',
                    analyzer: 'english'
                },
                lecture_type: {
```

A module catalogue was added to the portal to enable lecturers to “Join” a module and students to “Follow” a module. The module catalogue also allowed users to browse historical lecture recordings.





The Module List component was developed to provide the Module Catalogue views using separation of concerns and dependency injection – just as all the other components in Angular had been. For completeness, the top parts of the Controller and View code are shown overleaf

as code examples.

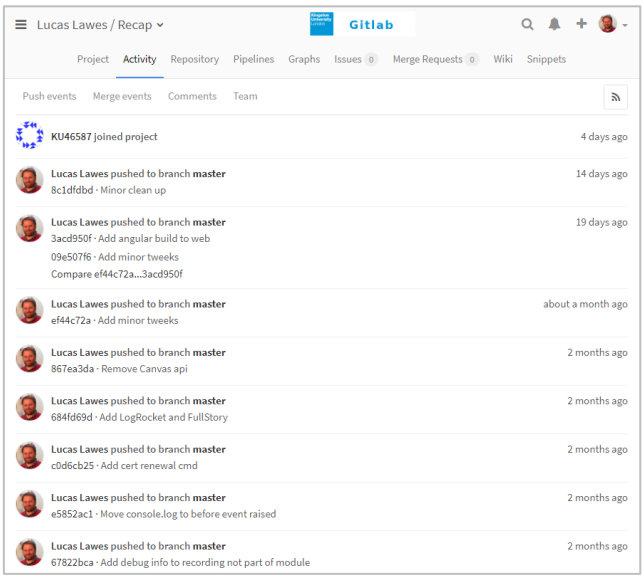
```
web > src > app > portal > components > catalogue > module-list > module-list.component.ts > ModuleListComponent > modulesInput$
You, 2 months ago | 1 author (You)
1  import { Component, OnInit, OnDestroy } from '@angular/core';
2  import { CurrentUser } from 'src/app/models/current-user';
3  import { AuthService } from 'src/app/services/auth.service';
4  import { ActivatedRoute, Router } from '@angular/router';
5  import { CatalogueService } from 'src/app/services/catalogue-service';
6  import { AssetService } from 'src/app/services/asset-service';
7  import { MatDialogConfig, MatDialog } from '@angular/material';
8  import { PortalService } from 'src/app/services/portal-service';
9  import { NewModuleDialogComponent } from '../../core/new-module-dialog/new-module-dialog.component';
10 import { Observable, Subject, concat, of } from 'rxjs';
11 import { switchMap, catchError, distinctUntilChanged, tap } from 'rxjs/operators';
12
   You, 2 months ago | 1 author (You)
13 @Component({
14   selector: 'app-module-list',
15   templateUrl: './module-list.component.html',
16   styleUrls: ['./module-list.component.scss']
17 })
18 export class ModuleListComponent implements OnInit {
19
20   imageRootUrl = '';
21   currentUser = new CurrentUser();
22
23   modulesList: any;
24
25   modules$: Observable<any[]>;
26   modulesLoading = false;
27   modulesInput$ = new Subject<string>();        You, 2 months ago • Add catalogue
28   selectedModule: any;
29
30   constructor(
31     private assetService: AssetService,
32     private authService: AuthService,
33     private portalService: PortalService,
34     private catalogueService: CatalogueService,
35     private route: ActivatedRoute,
36     private router: Router,
37     private dialog: MatDialog
38   ) {
39   }
40
41   ngOnInit() {
42     this.imageRootUrl = this.assetService.getImageRootUrl();
43     this.currentUser = this.authService.getCurrentUser();
44     this.catalogueService.getModulesList().subscribe(modulesList => {
45       this.modulesList = modulesList;
46     }, error => {
47       console.log(error);
48     });
49     this.modules$ = concat(
50       of([]),
51       this.modulesInput$.pipe(
52           distinctUntilChanged(),
53           tap(() => this.modulesLoading = true),
54           switchMap(term => this.catalogueService.getModulesLookup$(term).pipe(
55               catchError(() => of([])),
56               tap(() => this.modulesLoading = false)
57               ))
58       )
59     );
60   }
61
62   async onNewModule() {
63
64     const dialogConfig = new MatDialogConfig();
65     dialogConfig.disableClose = false;
66     dialogConfig.autoFocus = true;
```

```
web > src > app > portal > components > catalogue > module-list > module-list.component.html > div.breadcrumb-container > div.breadcrumb-content
You, a month ago | 1 author (You)
1  <div class="breadcrumb-container">
2      <div class="breadcrumb-content d-inline-flex flex-row justify-content-start flex-wrap">
3          <div class="breadcrumb-study-item">
4              <h3><a routerLink="/portal/home"><img class="pb-1" src="assets/home-icon.png">Home > </a></h3>
5          </div>
6          <div class="breadcrumb-study-item selected">
7              <h3>Modules</h3>                You, 2 months ago • Add catalogue
8          </div>
9      </div>
10 </div>
11 <ng-container *ngIf="currentUser.isContributor">
12   <div class="d-flex flex-row justify-content-between align-items-end btn-bar-top">
13     <div class="d-flex flex-row justify-content-start align-items-end">
14       <ng-select class="contributors moduleLookup"
15         autocomplete="no"
16         [(ngModel)]="selectedModule"
17         [items]="modules$ | async"
18         bindLabel="label"
19         dropdownPosition="auto"
20         [multiple]="false"
21         [loading]="modulesLoading"
22         [typeahead]="modulesInput$"
23         typeToSearchText="Type module code or title"
24         placeholder="Search for Module to Join"
25         notFoundText="No modules found - Add Module?"
26         >
27       </ng-select>
28       <button mat-raised-button color="primary" type="button" (click)="onAddModule()" aria-label="Add Module" title="
29     </div>
30     <div class="content-toolbar d-flex flex-row justify-content-end align-items-end">
31       <div>
32         <button mat-raised-button color="primary" type="button" class="button-warn" (click)="onNewModule()" aria-labe
33       </div>
34     </div>
35   </div>
36   <div class="content">
37     <ng-container *ngFor="let item of modulesList">
38       <div class="content-item pb-3 mb-3">
39           <div class="content-item-title">
40               <h3>{{ item.academicYear }}</h3>
41           </div>
42           <div class="recording-items d-flex justify-content-start flex-wrap">
43               <div class="recording-item" *ngFor="let module of item.modules">
44                   <div class="recording-item-thumbnail d-flex flex-column justify-content-end"
45                   routerLink="{{module.url}}"
46                   [ngStyle]="{'background-image': 'url(' + imageRootUrl + module.imageUrl + ')'}">
47                   </div>
48                   <div class="recording-item-details">
49                       <div class="recording-item-title"><a routerLink="{{module.url}}">{{module.title}}</a></div>
50                       <div class="recording-item-topic">{{module.code}}</div>
51                   </div>
52               </div>
53           </div>
54       </div>
55     </ng-container>
56   </div>
57 </ng-container>
58 <ng-container *ngIf="currentUser.isStudent">
59     <div class="d-flex flex-row justify-content-between align-items-end btn-bar-top">
60       <div class="d-flex flex-row justify-content-start align-items-end">
61         <ng-select class="contributors moduleLookup"
62           autocomplete="no"
63           [(ngModel)]="selectedModule"
64           [items]="modules$ | async"
65           bindLabel="label"
66           dropdownPosition="auto"
67           [multiple]="false"
```

The development of the solution could have continued passed Sprint 5. Still, a body of work sufficient for a final year project had been delivered, and the concept of writing an entire lecture capture solution in JavaScript had been proven technically to an acceptable standard.

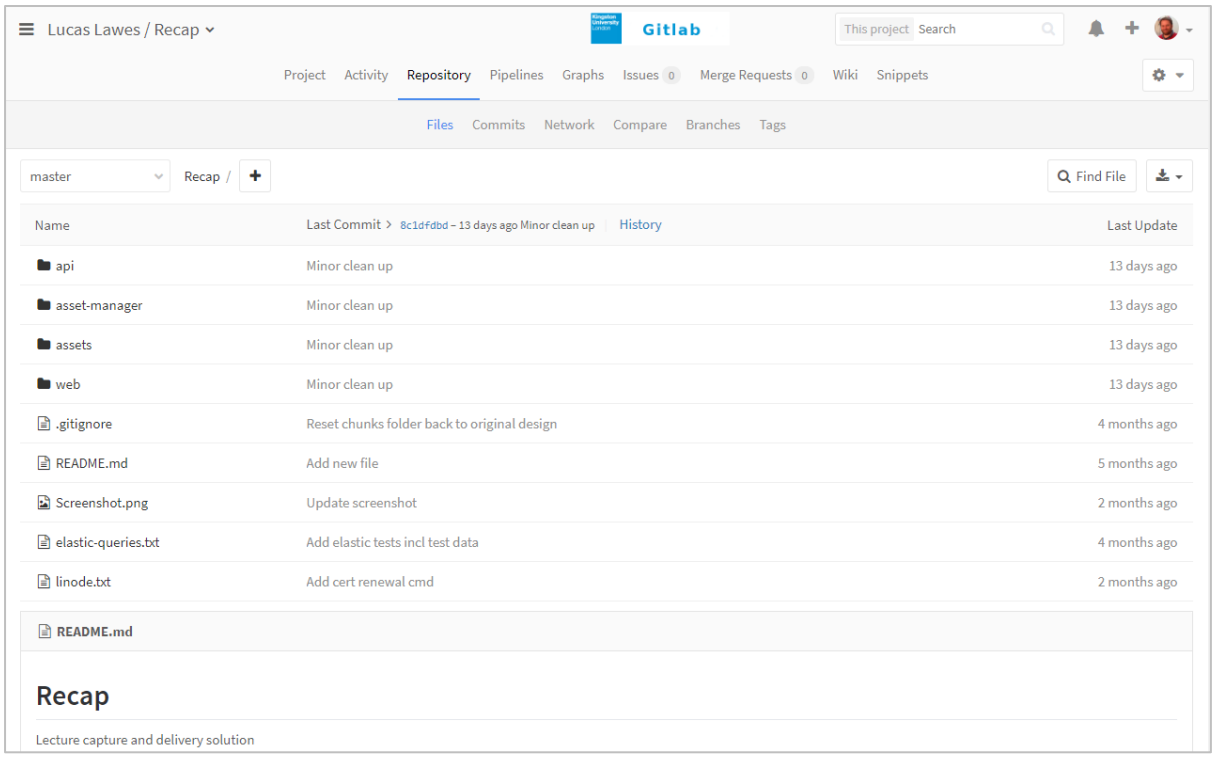
Transition

By March 2020, development had finished, and a stable release of Recap was in production. There were a few further commits for minor bug fixes and code clean up, as shown by the Gitlab activity.



The source code repository for the Recap API is hosted in Gitlab. The root folder contains the following source code solution folders:

- **api** – NodeJS/Express JavaScript application for the Recap API.
- **asset-manager** – NodeJS JavaScript application for the Asset Manager.
- **assets** – temporary folder storing raw files and JSON relating to the lecture recordings.
- **web** – Angular Typescript/HTML/CSS single page application for the Recap client.



The source code repository can be accessed via this link:
https://gitlab.kingston.ac.uk/K1736700/Recap/tree/master

A copy of the source code released to production can be downloaded from box via this link:
https://kingston.box.com/s/a0xff5i77s1se9cv5uq5apnss8rlxix6

Testing and Evaluation

Testing is an essential part of the Agile process (Kruse, 2020) and the ability to plan, design and create efficient tests is critical (Pawlak & Poniszewska-Marańda, 2019). The innovative and experimental nature of the project meant a proactive approach was taken to testing, and every sprint started with a test to evaluate whether each requirement would be met. Testing was a rigorous process for ensuring quality and confidence in the final product (Kruse, 2020).

Test Results

The following types of testing were integrated into the development process, and for traceability and completeness, supporting evidence for tests carried out is included in Appendix F – Testing Evidence.

Testing Type	**Details**	**Results**
Unit	Manual unit test scripts were written to test logic, including back end algorithms for sorting lecture recordings, and segmenting the transcript, and, front end algorithms for setting the portal context.	The algorithms were all relatively straightforward and passed unit and functional testing.
Integration	Test scripts were written to individually test the Recap client against the Recap API, and the Recap API against all the back end components, cloud services and the KU authentication service.	Extensive integration testing was conducted throughout the construction phase. When refactoring affected the integration, scripts were rerun to ensure every test passed.
End to End	Manually executed lecture capture was carried out to test each component of the system. For instance, a screen recording was sent by the Recap client to the API, via Redis to the Asset Manager, on to Cloudinary and Microsoft Azure, back to the Recap API and onto MongoDB. Likewise, when a lecture recording is published by the Recap client to the API, via Redis to Elastic Search and on to Kibana. Furthermore, when a lecture recording is deleted, it is deleted by the Recap API from MongoDB, ElasticSearch and by the Asset Manager from Cloudinary.	Extensive end to end testing was conducted throughout the construction and transition phases. There were conditions under which the tests failed, and the code had to be revised. E2E testing proved that the final product was robust and reliable.

Testing Type	Details	Results
Functional	Test cases for the three of the four Epics (Capture, Manage, Report) were completed manually. Testing for Consume Epic was automated using Selenium in development and Mabl in production.	Task completion gradually increased throughout the construction phase. By transition test cases supporting all user stories passed.
Browser Compatibility	Detailed compatibility testing was undertaken of the portal in all modern desktop web browsers, including Chrome, Safari, Edge, Firefox, and Opera. Internet Explorer is not supported. Mobile browser testing was not included, although the portal runs on Apple iPhone, iPad, and Android (Phone, Tablet and Chromecast). Testing was undertaken at KU using workstations in the Library and Labs.	All modern desktop web browsers completed all functional tests except for Safari, which the Recap client does not currently support for lecture recording. Future releases of Recap are expected to support Safari. Support had been restricted by the development environment being Windows 10 based, and Safari not currently supporting the JavaScript Screen Capture API standard completely.
Performance	The Nginx web server which hosts the portal on “recap.cloud” was performance tested using GT Metrix automation tools.	Results showed that Nginx needs to be configured to support compression, and a future release of Recap needs to reduce the bundle size of the Recap client by upgrading Angular and deprecating the Administration microsite.
Stress	The Recap API and client were stress-tested by using Loadster to run high loads of functional test scripts for viewing a lecture recording against the Recap API and Cloudinary.	The breaking point was not found, and the solution did not crash under peak traffic. There is an outstanding issue with the NodeJS process, which runs the Recap API, stopping when left idle for a few weeks and this will have to be remedied in a future release.
Security	The component security lockdown test was performed on the production	All security test passed to an acceptable standard for a final

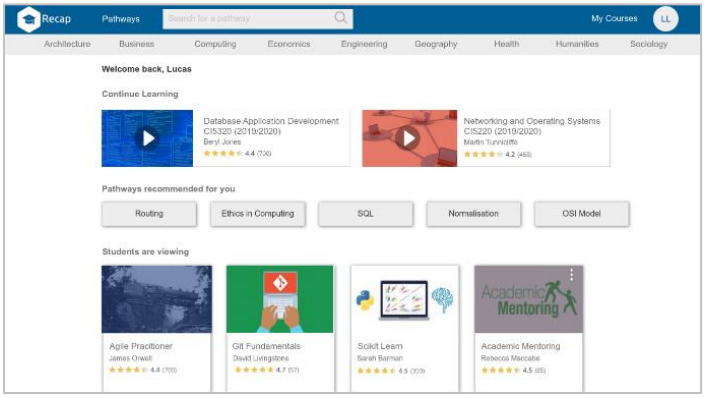
Testing Type	Details	Results
	environment after it was set up and configured. Authentication and authorisation testing against the Recap API and client was undertaken manually. A vulnerability scan of the Recap website and HTTPS SSL certificate.	year project. There is more work to do for a production environment. For practicality and cost reasons, Kibana and Cloudinary do not have security authentication fully configured yet.
Accessibility	The Recap website was accessibility tested by manually using the Windows 10 screen reader, and the Web Accessibility Evaluation Tool (WAVE) was used to evaluate the HTML markup and visual contrast.	Accessibility testing revealed a weakness in the Recap client implementation. A future release needs to improve the structure of the HTML generated by the Angular SPA and provide a high contrast theme. These changes will improve Screen Reader support and better assist the visually impaired.
Remote User	The Usability Test planned for the Transition project phase was curtailed due to the lockdown, but some remote user testing was completed in March and April. Final year Computer Science students were invited to participate in short evaluation sessions and give feedback on Recap as a study aid during construction and transition. Students signed in using their KU credentials and followed the modules they were undertaking. They were then asked to choose a topic to learn, prepare for an assessment and revise for an examination. Sessions were conducted remotely using Discord and TeamViewer.	Participants were able to complete all tasks with minimal assistance and no errors. Feedback was positive with comments like “it is really fast” and “it is amazing, and I wish I had it for my degree”. In general, testing with students confirmed the demand for lecture capture at KU is there. Two participants offered to write emails supporting the use of lecture capture they were so enthusiastic about the capability.

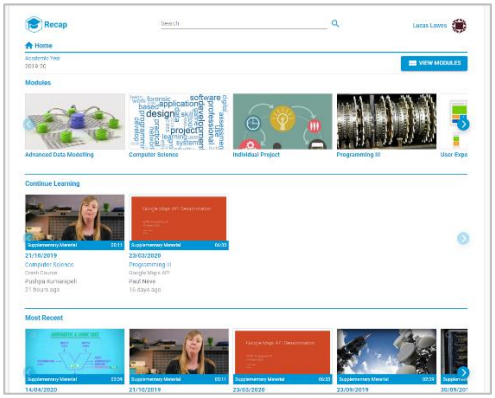
Evaluation Plan

Initial feedback from the client team has been very encouraging. The short term goals for the evaluation of Recap are to measure improvements in ease of use and productivity over existing offerings. However, the first step is to conduct the Usability Test with KU students and staff (see Appendix G – Usability Test Pack). Longer-term, the evaluation of Recap needs to measure the impact of using it as a study aid on student learning and performance.

Critical Review

The solution evolved from an initial concept in the Summer of 2019 to a viable product by the Spring of 2020. The solution had taken input from a wide variety of stakeholders at KU, and the evolution of the design from concept to completion was evident.





Success Criteria

The project set out to explore the possibility of providing KU with a lecture capture solution. While the concept of building a solution entirely in JavaScript was proven in principle, there was no extensive usability testing by Lecturers and Students. There may however be an opportunity to rectify this over the Summer of 2020 and determine whether Recap has the potential to be a better product over proprietary software. It would be helpful to prove the solution went some way to resolving some of the dissatisfaction with existing attempts at lecture capture. Still, unless the solution is evaluated, this cannot be determined.

Feedback from the client team and other interested parties has been positive, and the project at least went some way towards achieving its aim and objectives all bar a usability test. Leaving evaluation until the end of a project always poses a risk of running out of time. There is also a trade-off with evaluating too early whereby the solution might not be ready or of sufficient quality. That said, the risk of not completing the objective of conducting a usability test could have been mitigated more effectively. So, although the outbreak of coronavirus meant a usability test could not be carried out in March as planned, it might have made more sense to deliver a less detailed solution for testing in February. This planning approach would have left more time available to refine the solution and accommodate changing circumstances.

Personal Reflection

The main achievement of this individual project was to discover a passion and pursue it, despite setbacks early on. Good fortune meant the project received the support of an excellent client team and a free license to invest time integrating open-source software of exceptional quality. There was a moment when the project was nearly jettisoned in favour of researching an Artificial Intelligence (AI) algorithm. However, the opportunity to build a software product that utilised AI proved more of a draw. Moreover, the first time an end-to-end test succeeded in capturing a lecture recording and returned its transcript from an AI API was immensely satisfying and showed this had been the right choice.

There were many challenges along the way. Finding the right support for a project was as tricky as finding the right idea. However, things fell into place over time and presented a new challenge; having to do the project. At first, the project seemed the appropriate size for a final

year project to fit in neatly alongside other subjects. Nevertheless, by the end of the project, it seemed like an entire degree. This exponential growth in size was not the result of unreasonable requirements by the client. The reverse was true. The work was so fascinating to undertake it produced a swell in creativity, and the next challenge became finding a balance between exploiting it and containing it. In the end, like writing a report, the streams of written code whittled down to a mere trickle. In terms of project management, the approach to blend RUP with agile principles seemed about right. There was much groundwork to do before the construction began, and having this knowledge before writing any coding was empowering and produced a higher quality solution. Technically the project went smoothly as while the learning curve was steep, the numerous technologies all fell into place and just worked the first time of asking.

Future Work

The most significant limitations of the project were a lack of inclusivity edge cases and integration with an LMS. However, as with other capabilities such as scalability and mobile device support, these can be added in future work. Inclusivity is about supporting as many different usage scenarios as possible, such as commuters, and it was disappointing not to have added this as a design principle. This addition would have factored it into the design process more explicitly and helped shape the product to be a little more user friendly and accessible than it currently is. Integration with an LMS would have established the solution and Instructure offer a cloud service for Canvas, and so this was a new opportunity for future work. There is a NodeJS package for accessing the Canvas API, and this can be used to extract student module mappings automatically.

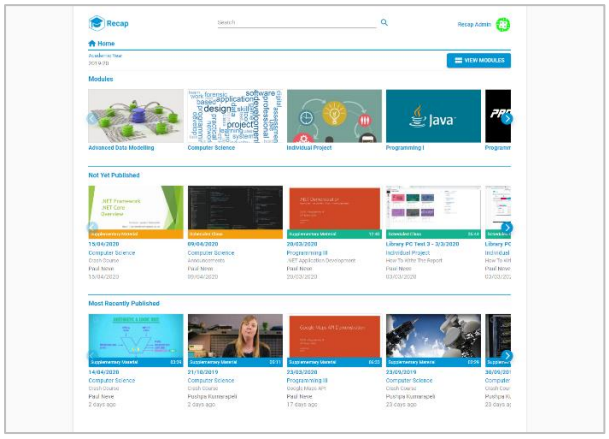
The educational landscape changed during the transition phase of the project, and if the project has a second cycle, the requirements will undoubtedly look very different. The closure of KU campuses in March 2020 saw teaching shift to online ways of working and digital communication tools, such as Zoom and Microsoft Stream, enabled Lecturers and Students to stay connected and engaged. It is probably fair to state that before this pivotal moment, lecture capture was perceived by KU to be a nice-to-have. In the 2020/21 academic year, lecture capture, as well as live lecture streaming, are likely to be must-haves, and this transforms the proposition from eight months ago.

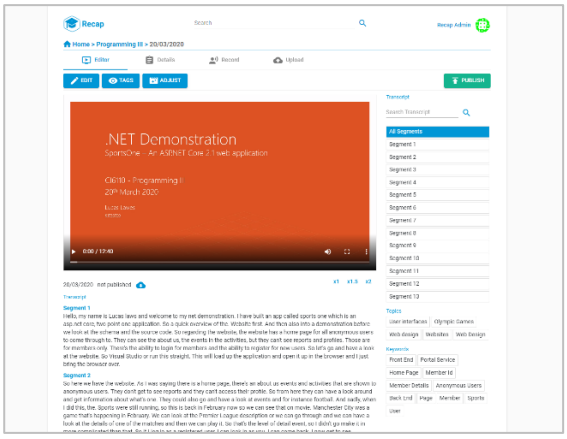
Aspirations

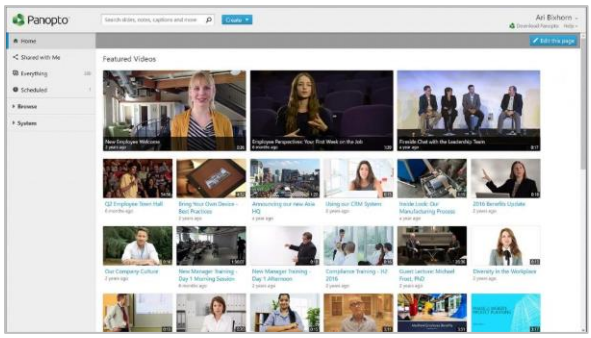
It is therefore also fair to say that the current state of the solution is at a pivotal moment, teetering between being just another good idea and something of meaningful value. In the SCOPE analysis, the prospect for the solution in the future was to provide an open-source base solution as well as a subscription-based cloud-hosted offering. The solution itself was put together using software built by many developers in the open-source community. It would be fitting, therefore, to distribute the underlying source code using the GNU GPLv3 license so that others can do almost anything they wish except distributing closed source versions. The key differentiator of the solution is that it is written entirely in JavaScript, which seems more potent than procedural languages for building modern web applications. The lesson learnt from coding with JavaScript is that it is a compelling language with infinite possibilities.

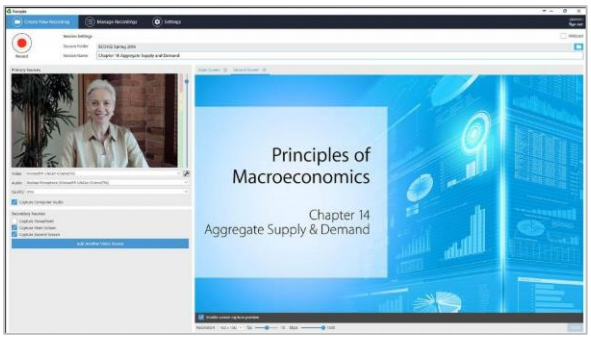
Looking to the future state of the solution, assuming that it is proven to work, there is a clear opportunity to become a new entrant in the lecture capture market by providing a software as a service (SaaS) platform. Being based on open-source software and JavaScript means that only suppliers of cloud services such as the likes of Linode and Cloudinary can drive costs up. Customers would be in a position to force costs down by switching to Recap and over time this

could lead to an emerging generation of Content Capture tools which would provide new pedagogical benefits for students engaging with a variety of lecture materials (Evergeti & Garside, 2020). A quick comparison of Recap and Panopto, the market leader (Gartner, 2018), shows that such an ambition is perhaps not too unrealistic.









SOAR Analysis (shown overleaf) summarises the position of Recap post project. The experience of undertaking the project has provided a greater insight into HEI "pain points" and requirements. Lecture capture should not be about capturing the lecture hall interaction, but about recalling the narrative and content, so students can revisit the gist of what was taught. The project built a new solution from scratch, which leverages open-source software to an equivalent quality of competitor software. Significant dissatisfaction with lecture capture can be resolved, not just by technological means, but by how users experience it. The majority of HEIs have not implemented lecture capture campus-wide (Newland, 2017), and this is understandable. Lecture capture in its on-premise form is expensive to deploy and must be a self-service offering; it cannot be successfully implemented top-down because it does not have a clear benefit for all teaching and learning needs.

The aspirations would be to provide a Recap download which HEIs can use to replace their existing implementations. A cloud-hosted service offering would provide lecture capture capabilities on-demand in much the same way that the uptake of Zoom occurred; by there being a definite individual and group requirement rather than a top-down decision. If Recap became a market leader, it would only be because it provides staff and students with increased satisfaction and improves learning outcomes.

SOAR Analysis

Strengths	Opportunities
Insight into HEI pain points and requirements New solution built from scratch Leverages open-source software	Majority of HEIs have not implemented campus wide Significant dissatisfaction targeted primarily at Panopto, Kaltura and Echo360
Aspirations	**Results**
Provide an Open-Source base solution any HEI can download Provide a cloud hosted service offering to HEIs on a subscription basis	Become the market leader Increased satisfaction Improved student performance

Demonstration

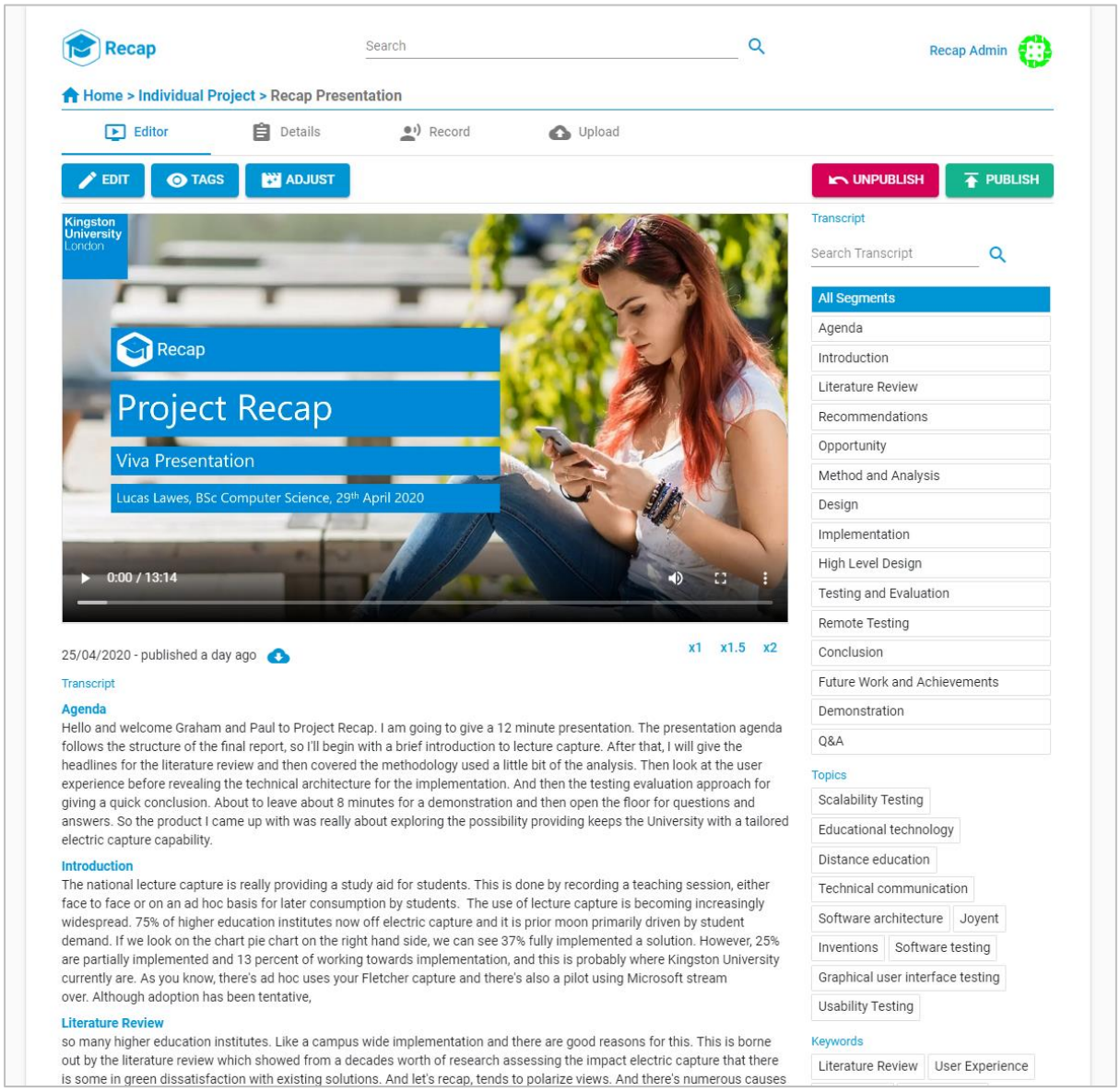
A ten-minute product demonstration video of the Recap lecture capture solution is available to download from the final year project folder and watch.

https://kingston.box.com/s/wsf35ei5g12ueg2z8w7qmuugs201g0qw

The demonstration shows how:

- Lecturers record a teaching session using voice-over presentation.
- Module Leaders prepare a Module for Lecture Recording.
- Students use Recap as a study aid for learning new topics, researching, and preparing for assessment.
- Lecturers publish lecture recordings which have been recorded online and offline.
- Lecturers and Students discover lecture recordings across Modules and Academic Years.
- Lecturers gain an overview and insights into the published content using the Learning Analytics dashboard.

KU students and staff can evaluate Recap by accessing it via https://recap.cloud and signing in with their K/KU credentials.



Conclusion

To conclude this Critical Review, the next step for the project is to take a step back to reflect on what has been a significant effort and then find some way to evaluate the solution. The project has shown that it is possible to go a fair way towards addressing the current satisfaction and adoption issues in a short space of time. This progress provides confidence and motivation for preparing an enhanced solution over the Summer of 2020. The Recap solution is the first-ever lecture capture solution to be written entirely in JavaScript. While the future uptake of Recap is as yet uncertain, what is clear is that JavaScript has matured as a ubiquitous language and offers new pedagogical benefits when applied thoughtfully to educational technology.

References

Abuhussein, A., 2020. *Toward an Effective Requirement Engineering Approach for Cloud Applications.* [Online]
Available at: https://link.springer.com/chapter/10.1007/978-3-030-33624-0_2
[Accessed 28 April 2020].

Anon., 2019. *A Model-Driven Approach to Generate Schemas for Object-Document Mappers.* [Online]
Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8708919
[Accessed 12 December 2019].

Anwar, A., 2014. A Review of RUP (Rational Unified Process). *International Journal of Software Engineering.*

Atwood, J., 2007. *The Principle of Least Power.* [Online]
Available at: https://blog.codinghorror.com/the-principle-of-least-power/
[Accessed 20 March 2020].

Berners-Lee, T., 1998. *Principles of Design.* [Online]
Available at: https://www.w3.org/DesignIssues/Principles.html
[Accessed 12 September 2019].

Caglayan, E. & Ustunluoglu, E., 2020. *A Study Exploring Students' Usage Patterns and Adoption of Lecture Capture.* [Online]
Available at: https://link.springer.com/article/10.1007/s10758-020-09435-9
[Accessed 28 April 2020].

Clark, S., 2019. *Meet Generation Z* [Interview] (6 December 2019).

Cohen, E., 2019. *The Definitive Guide to Project Management Methodologies.* [Online]
Available at: https://www.workamajig.com/blog/project-management-methodologies
[Accessed 6 12 2019].

Dinmore, S., 2019. Beyond Lecture Capture: Creating digital video content for online learning. *Journal of University Teaching and Learning Practice,* 16(1).

Dommett, E., Gardner, B. & van Tilburg, W., 2019. Staff and student views of lecture capture: a qualitative study. *Journal of Educational Technology in Higher Education,* 16(23).

Ebbert, D. & Dutke, S., 2020. Patterns in students' usage of lecture recordings: a cluster analysis of self-report data. *Research in Learning Technology,* Volume 28.

Echo 360, 2016. *Responding to Students' Expectations at King's College London.* [Online]
Available at: https://echo360.com/responding-to-students-expectations-at-kings-college-london/
[Accessed 30 09 2019].

Edwards, M. R. & Clinton, M. E., 2019. A study exploring the impact of lecture capture availability and lecture capture usage on student attendance and attainment. *Higher Education,* Volume 77, pp. 403-421.

Ellington, H., F, P. & P, R., 1993. *Handbook of educational technology.* 3rd ed. London: Kogan Page.

Euzent, P., Martin, T., Moskal, P. & Moskal, P., 2011. *Assessing Student Performance and Perceptions in Lecture Capture vs. Face-to-Face Course Delivery.* [Online]
Available at: https://www.learntechlib.org/p/111523/
[Accessed 15 January 2020].

Evans, G. & Luke, K., 2020. *Lecture capture and peer working: exploring study practices through staff-student partnerships.* [Online]
Available at: http://orca.cf.ac.uk/129824/1/2314-Article%20Text-14086-1-10-20200219.pdf
[Accessed 1 April 2020].

Evergeti, V. & Garside, H., 2020. Captured Content and Lecture Recordings: Perceptions and Experiences of Students and Lecturers. In: K. Gravett, I. Kinchin & N. Yadovchuk, eds. *Enhancing Student-Centred Teaching in Higher Education.* Cham: Springer, pp. 115-132.

Gartner, 2018. *Magic Quadrant for Enterprise Video Content Management.* [Online] Available at: https://www.gartner.com/en/documents/3894379 [Accessed 09 December 2019].

Gawron, K., 2018. *6 Main Reasons Why Node.js Has Become a Standard for Enterprise-Level Organizations.* [Online] Available at: https://www.monterail.com/blog/nodejs-development-enterprises [Accessed 12 December 2019].

Gravett, K., Yakovchuk, N. & Kinchin, I., 2020. *Enhancing Student-Centred Teaching in Higher Education.* 1st ed. s.l.:Springer.

Grzybowski, A. & Kupidura-Majewski, K., 2019. *What is color and how it is perceived?.* [Online] Available at: https://www.sciencedirect.com/science/article/pii/S0738081X19301191 [Accessed 28 April 2020].

Hamutoglu, N. et al., 2019. *Evaluating students experiences using a virtual learning environment: satisfaction and preferences.* [Online] Available at: https://link.springer.com/article/10.1007/s11423-019-09705-z [Accessed 1 April 2020].

Hetherington, L.-D., Glowatz, M., McDonald, E. & Dempsey, A., 2020. *Business students' experiences of technology tools and applications in higher education.* [Online] Available at: https://onlinelibrary.wiley.com/doi/abs/10.1111/ijtd.12168 [Accessed 17 January 2020].

Holmes, V., 2019. *New project to introduce Lecture Capture.* [Online] Available at: https://www.reading.ac.uk/internal/staffportal/news/articles/spsn-793391.aspx [Accessed 31 March 2020].

Hove, C. & Corcoran, K., 2008. *If You Post It, Will They Come? Lecture Availability in Introductory Psychology.* [Online] Available at: https://www.tandfonline.com/doi/abs/10.1080/00986280802004560 [Accessed 17 February 2020].

Ibrahim, Y., Howarth, A. & Stone, I., 2020. *Lecture Capture Policies: A Survey of British Universities.* [Online] Available at: https://link.springer.com/article/10.1007/s42438-020-00102-x [Accessed 31 March 2020].

Johnston, A., Massa, H. & Burne, T., 2013. Digital lecture recording: A cautionary tale. *Nurse Education in Practice,* Issue 13, pp. 40-47.

Joseph-Richard, P. et al., 2018. *Big brother or harbinger of best practice: Can lecture capture actually improve teaching?.* [Online] Available at: https://onlinelibrary.wiley.com/doi/full/10.1002/berj.3336 [Accessed 15 January 2020].

Khee, C. W. G. & Jamaluddin, A., 2014. Students' perception towards lecture capture based on the Technology Acceptance Model. *Procedia - Social and Behavioral Sciences,* Issue 123, pp. 461-469.

Kruse, P., 2020. Systematic Testing in the Agile Development Process. *ATZ Worldwide,* Volume 122, pp. 42-45.

Leadbeater, W., Shuttleworth, T., Couperthwaite, J. & Nightingale, K., 2013. Evaluating the use and impact of lecture recording in undergraduates: Evidence for distinct

approaches by different groups of students. *Computers & Education,* Issue 61, pp. 185-192.
Linsey, T., 2019. *Lecture Capture Pilots Proposal,* s.l.: Kingston University.
Luke, K., 2018. *Studying learning journeys with lecture capture through Staff-Student partnerships.* [Online]
Available at: http://orca.cf.ac.uk/120956/
[Accessed 31 March 2020].
Luke, K., 2020. *The pause/play button actor-network: lecture capture recordings and (re)configuring multi-spatial learning practices.* [Online]
Available at: https://www.tandfonline.com/doi/full/10.1080/10494820.2019.1706052
[Accessed 15 January 2020].
MacKay, J., 2019. Show and 'tool': How lecture recording transforms staff and student perspectives on lectures in higher education. *Computers & Education,* Issue 140.
Morris, N., Swinnerton, B. & Coop, T., 2019. *Lecture recordings to support learning: A contested space between students and teachers.* [Online]
Available at:
https://www.sciencedirect.com/science/article/abs/pii/S0360131519301575
[Accessed 15 January 2020].
Morris, N., Swinnerton, B. & Coop, T., 2019. Lecture recordings to support learning: A contested space between students and teachers. *Computers & Education,* Issue 140.
Neve, P. H. G. a. L. D., 2016. Keeping them Interested and Keeping them Honest: Using an On-Line Learning and Development Environment called NoobLab to improve outcomes and prevent plagiarism in undergraduate programming pedagogy. In: P. Novais & S. Konomi, eds. *Intelligent Environments 2016: Workshop Proceedings of the 12th International Conference on Intelligent Environments.* London: IOS Press, pp. 207-215.
Newland, B., 2017. *Lecture Capture in UK HE - A HeLF Survey Report,* Brighton: HeLF.
Nordmann, E. et al., 2019. Turn up, tune in, don't drop out: the relationship between lecture attendance, use of lecture recordings and achievement at different levels of study. *Higher Education,* Volume 77, pp. 1065-1084.
Nordmann, E. & McGeorge, P., 2018. *Lecture capture in higher education: time to learn from the learners.* [Online]
Available at: https://psyarxiv.com/ux29v
[Accessed 2 April 2020].
Panopto, 2020. *75 Studies Reveal The Impact Of Lecture Capture.* [Online]
Available at: https://www.panopto.com/blog/75-studies-reveal-the-impact-of-lecture-capture/
[Accessed 23 March 2020].
Pawlak, M. & Poniszewska-Marańda, A., 2019. Software Testing Management Process for Agile Approach Projects. *Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies,* Volume 30, pp. 63-84.
Petersen, K. & Wohlin, C., 2009. *A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case.* [Online]
Available at: https://www.sciencedirect.com/science/article/pii/S0164121209000855
[Accessed 28 April 2020].
Ready Education, 2018. *The Ultimate Guide to Understanding Gen-Z Students.* [Online]
Available at: https://www.readyeducation.com/blog/the-ultimate-guide-to-understanding-gen-z-students
[Accessed 22 September 2019].
Salah, D., Paige, R. & Cairns, P., 2014. A Systematic Literature Review for Agile

Development processes and user-centred design integration. *EASE '14: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, pp. 1-10.

Stackoverflow, 2019. *Developer Survey Results 2019.* [Online] Available at: https://insights.stackoverflow.com/survey/2019#technology [Accessed 28 December 2019].

Tam, C., Joia da Costa Moura, E., Oliveira, T. & Varajao, J., 2020. *The factors influencing the success of on-going agile software development projects.* [Online] Available at: https://www.sciencedirect.com/science/article/pii/S0263786320300089 [Accessed 28 April 2020].

The Educators, 2020. *Barriers to Motivation in an Online Learning Environment.* [Online] Available at: http://www.theeducators.co/2018/11/12/barriers-to-motivation-in-an-online-learning-environment/ [Accessed 10 April 2020].

UCISA, 2019. *2018 Survey of Technology Enhanced Learning for higher education in the UK*, Oxford: UCISA.

Appendix

Appendix A – User Personas

Director of Learning

PROJECT: Recap PERSONA: Graham

NAME

Graham

MARKET SIZE

34 %

TYPE

Idealist

Goals

- Ensure Lecture Capture Policy and Best Practices are followed
- Give students the right skills to succeed using Lecture Capture

Quote

Perfection is not attainable, but if we chase perfection we can catch excellence

Background

Extensive academic research and teaching experience with a focus on learning technology and methods used to study its use

Demographic

Kingston

Director of Learning and Teaching

Motivations

- Avoid the pitfalls of Lecture Capture
- Help develop pro-active students
- Foster a healthy learning culture

Frustrations

- Pressure on Lecturers to put on a good show when teaching
- Passivity of Lecture Capture as a learning tool

Skills

Teaching

0 25 50 75 100

Curriculum Design

0 25 50 75 100

Material Generation

0 25 50 75 100

Computer Literacy

0 25 50 75 100

Lecturer

PROJECT: Recap PERSONA: Paul

NAME

Paul

MARKET SIZE

30 %

TYPE

Artisan

Demographic

Kingston

Senior Lecturer

Goals

- Increase the use of lecture capture
- Improve the performance of middle quantile students

Quote

What lies behind you and what lies in front of you, pales in comparison to what lies inside you

Background

Cutting edge educational innovator and inspirational lecturer with 30 years of experience in industry and academia.

Motivations

- Improve student performance
- Increase student satisfaction

Frustrations

- Lack of unified Lecture Capture
- Existing tools are cumbersome
- Lecture Capture not used enough

Skills

Teaching

0 25 50 75 100

Curriculum Design

0 25 50 75 100

Material Generation

0 25 50 75 100

Computer Literacy

0 25 50 75 100

Academic Researcher

PROJECT: Recap PERSONA: Sarah

NAME

Sarah

MARKET SIZE

25 %

TYPE

Rational

Goals

Impart knowledge
Help students become research assistants

Quote

A genius is someone who takes a complex thing and makes it look simple

Background

25 years of experience working in academia and research using the latest algorithms in Artificial Intelligence

Demographic

Kingston

Research Director

Motivations

- Ability to generate more comprehensive lab material
- Enhance understanding of topics
- Clarify issues

Frustrations

- Existing tools difficult to use
- No easy way of sharing material

Skills

Teaching

0 25 50 75 100

Curriculum Design

0 25 50 75 100

Material Generation

0 25 50 75 100

Computer Literacy

0 25 50 75 100

Engaged Student

PROJECT: Recap PERSONA: Yeongho

NAME

Yeongho

MARKET SIZE

33 %

TYPE

Diligent

Goals

- To do well at University and honor family
- To learn at own pace
- Generate more comprehensive course notes

Quote

No act of kindness, no matter how small, is ever wasted

Background

Excelled in secondary education and developed a passion for learning

Demographic

Kingston

Student

Motivations

- Get better grades
- Improve chances of getting a good job

Frustrations

- Note taking not always sufficient for capturing a lecture
- Pressure from peers and family to do well
- Revising for exams is stressful

Skills

Critical Thinking

0 25 50 75 100

Note Taking

0 25 50 75 100

Academic Writing

0 25 50 75 100

Computer Literacy

0 25 50 75 100

International Student

PROJECT: Recap PERSONA: Linda

NAME

Linda

MARKET SIZE

42 %

TYPE

Performer

Goals

- Aiming for a 2:1
- Help with note-taking
- Perform better in exams

Quote

How wonderful it is that no body need wait a single moment before starting to improve the world

Demographic

Kingston

Student

Background

Originally studied dance before discovering a passion for technology

Motivations

- Friends and Family
- Improve academic grade from 2:2 to 2:1
- Do better at understanding complex topics

Frustrations

- English is a second language and teaching pace is sometimes too fast paced
- If you don't manage to write good notes, you are automatically at a disadvantage

Skills

Critical Thinking

0 25 50 75 100

Note Taking

0 25 50 75 100

Academic Writing

0 25 50 75 100

Computer Literacy

0 25 50 75 100

Motivated Student

PROJECT: Recap PERSONA: Gustavo

NAME

Gustavo

MARKET SIZE

21 %

TYPE

Perfectionist

Demographic

Kingston

Student

Goals

- Maximize the opportunity University affords
- Garner as many insights from Lectures as possible
- Master complex topics and concepts

Quote

Push yourself, because no one else is going to do it for you

Background

Gained great experience with initial career, now focused on achieving ambitions for a new career working with people and technology

Motivations

- Provide for family
- Gain experience with a flagship company
- Get a 1st class degree

Frustrations

- Sometimes notes and Slide are not enough for going over lectures again.
- Need a way to review lectures before each practical lab.

Skills

Critical Thinking

0 25 50 75 100

Note Taking

0 25 50 75 100

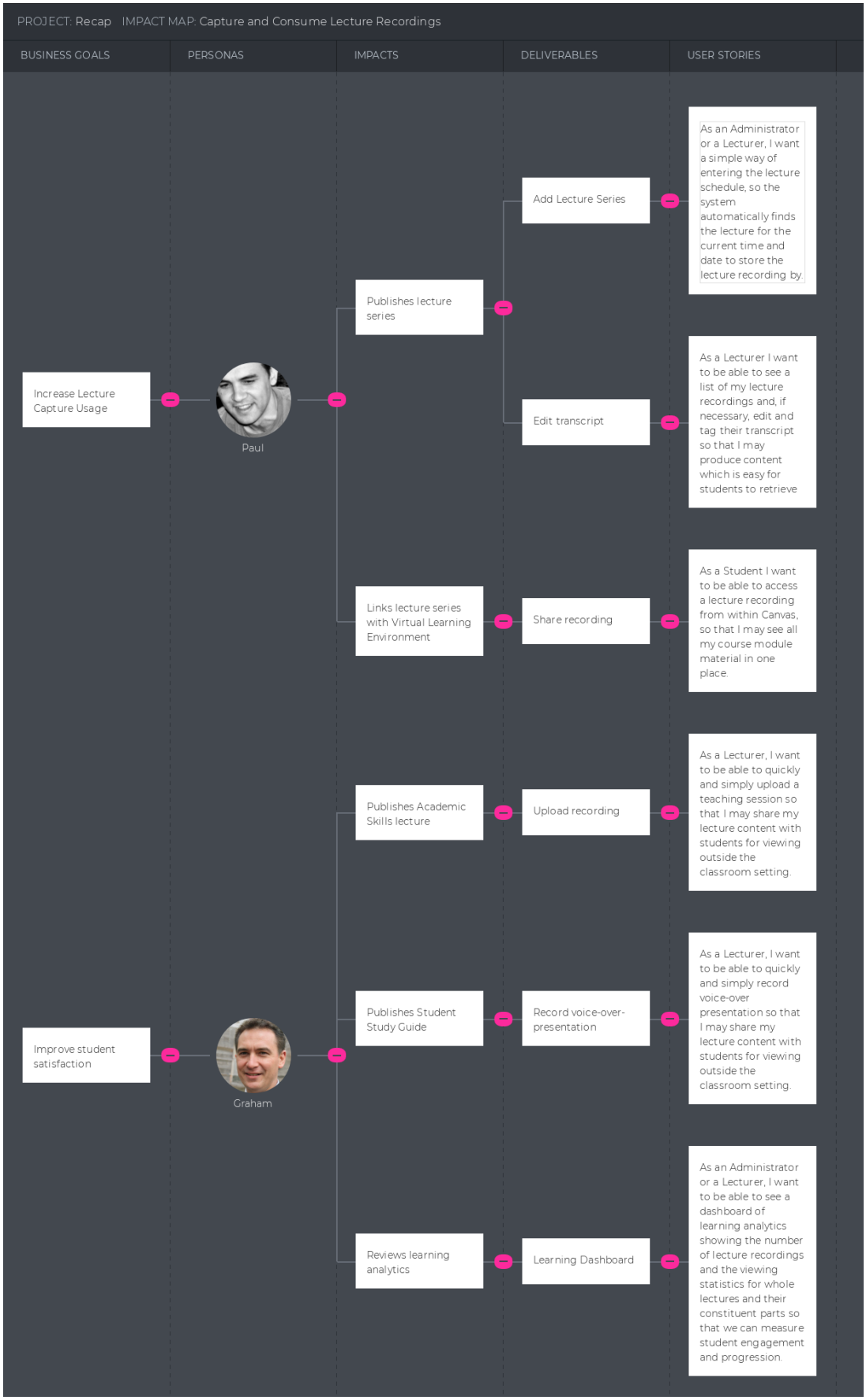
Academic Writing

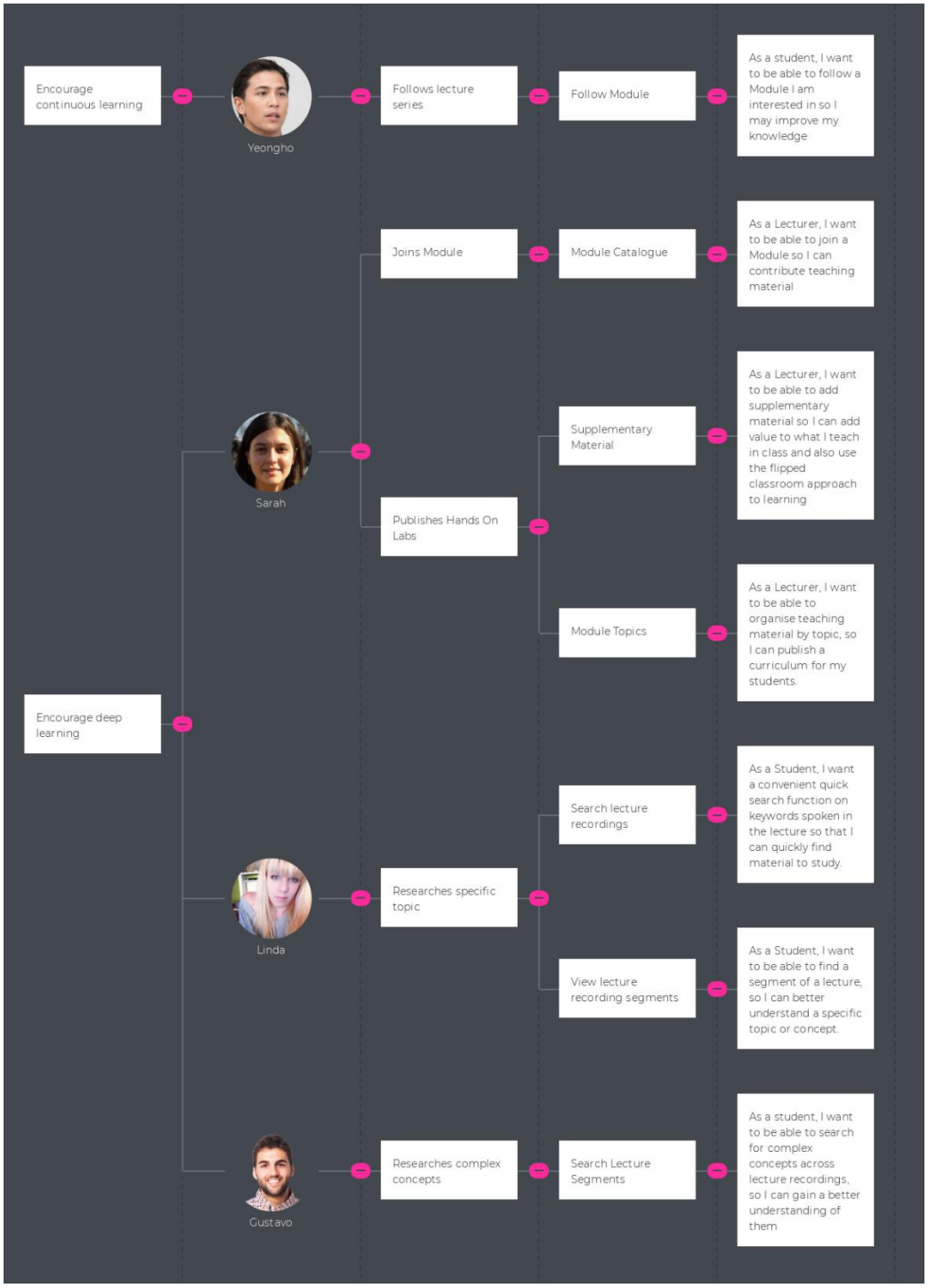
0 25 50 75 100

Computer Literacy

0 25 50 75 100

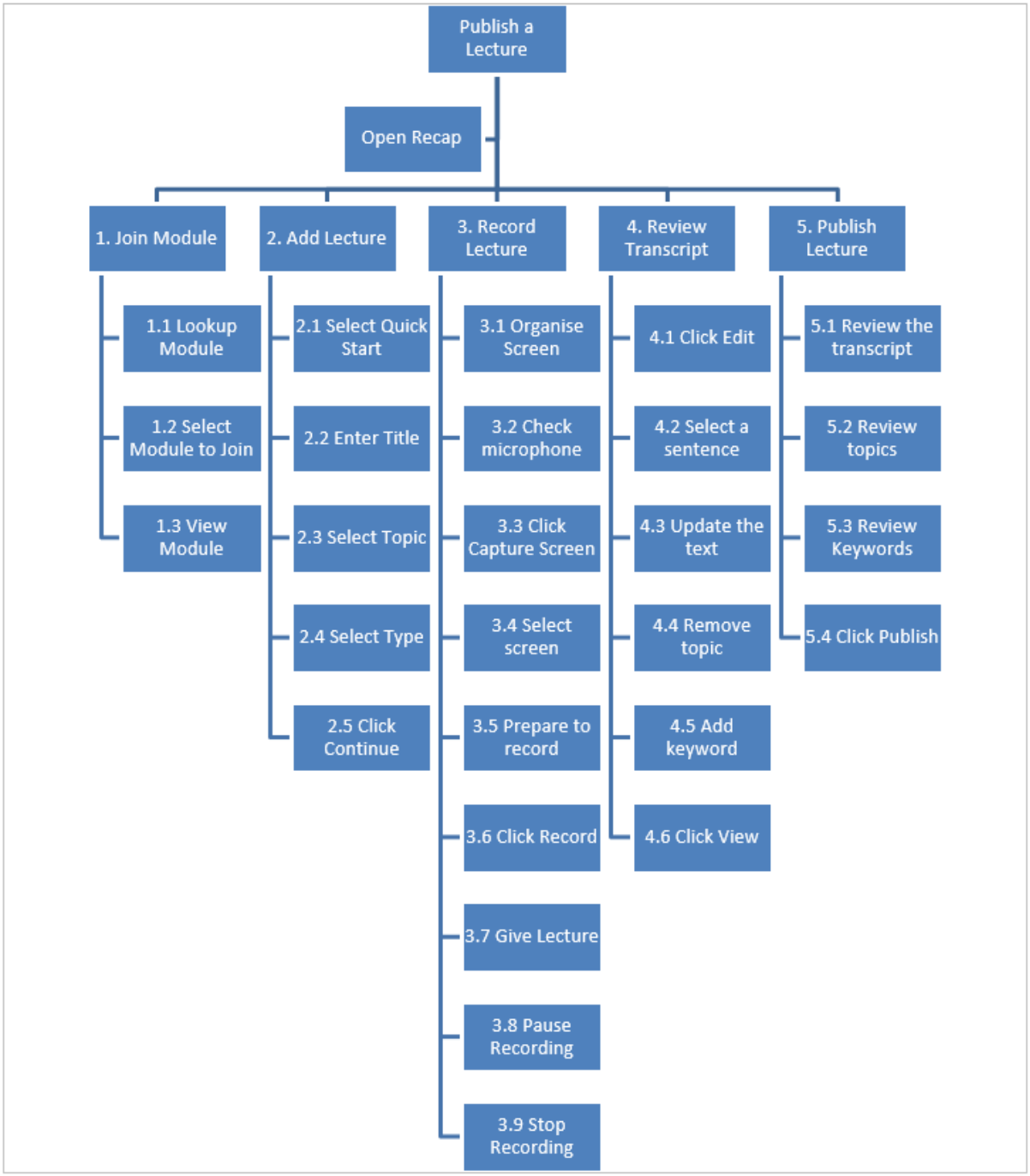
Appendix B – Impact Map



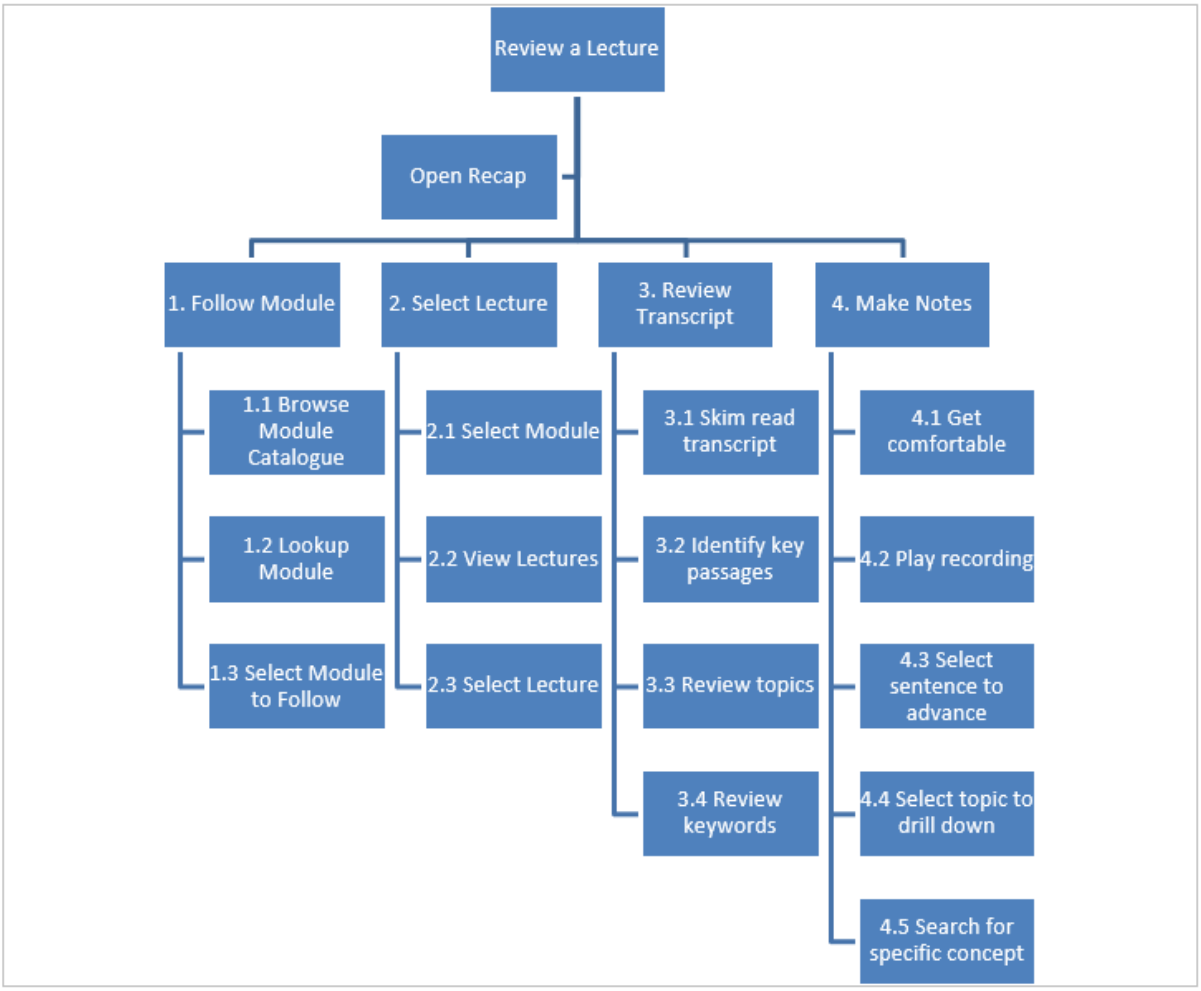

Encourage continuous learning
Yeongho
Follows lecture series
Follow Module
As a student, I want to be able to follow a Module I am interested in so I may improve my knowledge
Joins Module
Module Catalogue
As a Lecturer, I want to be able to join a Module so I can contribute teaching material
Sarah
Publishes Hands On Labs
Supplementary Material
As a Lecturer, I want to be able to add supplementary material so I can add value to what I teach in class and also use the flipped classroom approach to learning
Module Topics
As a Lecturer, I want to be able to organise teaching material by topic, so I can publish a curriculum for my students.
Encourage deep learning
Linda
Researches specific topic
Search lecture recordings
As a Student, I want a convenient quick search function on keywords spoken in the lecture so that I can quickly find material to study.
View lecture recording segments
As a Student, I want to be able to find a segment of a lecture, so I can better understand a specific topic or concept.
Gustavo
Researches complex concepts
Search Lecture Segments
As a student, I want to be able to search for complex concepts across lecture recordings, so I can gain a better understanding of them

Appendix C – Hierarchical Task Analysis

Publish a Lecture

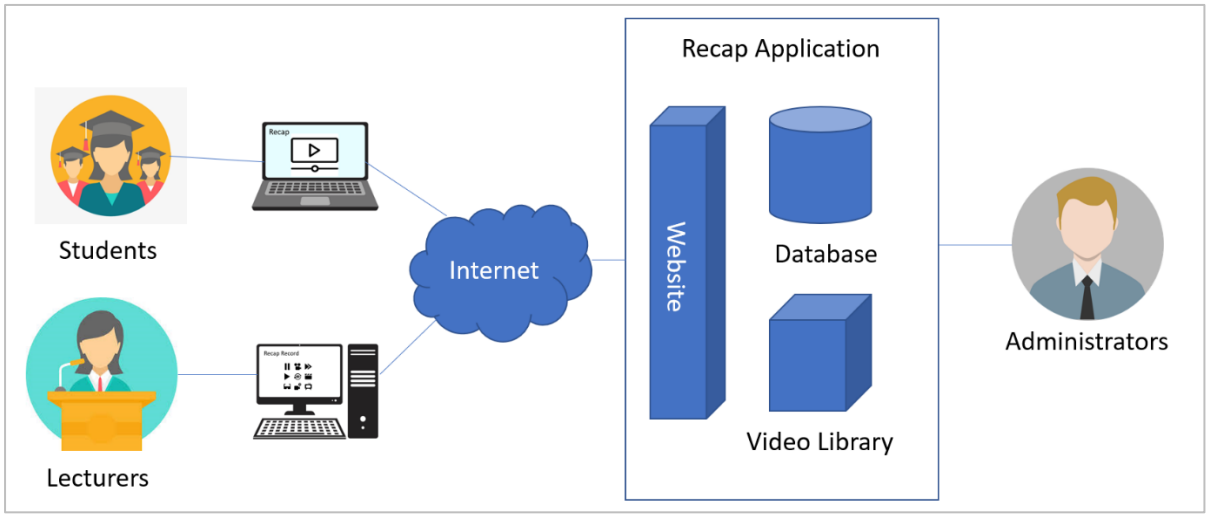


Review a Lecture



Appendix D – Technology Selection

The conceptual model showed the fundamental components of a potential solution and was produced during the Inception phase. One of the advantages of RUP is that it encourages proactive research upfront and allowed research to be undertaken before the requirements were finalised.



The candidate technology evaluated during this early stage included:

Functionality	Technology
Video Library	Local File System, Firebase Storage, YouTube, OpenCast, Cloudinary
Video Database	Firebase with Algolia and Cloud Functions, MongoDB with Atlas, MySQL, OpenCast with SOLR and ElasticSearch
Web Site / Services	Firebase, NodeJS, Spring Boot, ASP.NET Core, PHP
Client	Angular, React, HTML5 MediaRecorder
Hosting	Linode, Heroku, AWS, GCS

With the requirements agreed, the appropriate technology almost selected itself for the project during the Elaboration phase. However, a technology evaluation against criteria was carried out although new technologies, such as Redis Server and the Microsoft Azure Audio-to-Text API, were added during the Construction phase to cover off gaps in the technology evaluation.

All the technology had to be open-source, except for cloud services, and the criteria for technology selection involved:

- **Time to Benefit** – rapid application development was a critical success factor for the project and technologies which offered significant benefits quickly were given a higher priority.

- **Rating** – highly recommended the technology was given priority.

- **Suitability** – how well the technology fits the requirements for the solution.

The candidate technologies evaluated had a score out of 10 awarded for each of the selection criteria, and the following became the technology selected for the solution.

Functionality	Technology Selected	Time to Benefit	Rating	Suitability	Total
Screen Recording	JavaScript Screen Capture API	10	10	10	30
Client	Angular	10	8	8	26
Web Server	Nginx	10	10	10	30
Web Site	NodeJS	10	10	10	30
Video Library	Cloudinary	10	10	7	27
Audio-to-Text API	Microsoft Azure	10	7	7	24
Database	MongoDB	10	8	7	25
Search	ElasticSearch	10	10	10	30
Learning Analytics	Kibana	10	10	10	30
Caching	Redis Server	10	10	10	30
Hosting	Linode	10	10	10	30

The JavaScript Screen Capture API is a brand new web-based technology used in leading-edge software like Cisco WebEx Teams and was the way to go without comparison. For the client, React is a more accessible web framework than Angular. Still, Angular provides a better user experience out of the box when developing a single page application with Material Design. Flutter would be the client of choice for any future mobile app development for the same reason. NodeJS was selected as the backend application framework once MongoDB was chosen as the database.

MySQL was the preferred option for the database, but it is not able to store unstructured content as easily as a NoSQL database. MongoDB has its limitations with no search capability (although Atlas was in beta at the time of development) and difficulty maintaining referential integrity – something important in an application which is role-based and has a hierarchical content structure. A hybrid approach of using MySQL and MongoDB databases was considered, but the development effort would have been too high for a final year project. Other NoSQL databases such a Couchbase and Firebase were considered early on, but MongoDB provided sufficient functionality and storage for no cost. NodeJS and MongoDB integrate well (with the aid of a package called Mongoose), and both can scale massively, and so it seemed a natural choice to select both together. Nginx is a web server which can also be used as a reverse proxy and was preferred over Apache for its ability to host the Angular client and NodeJS process endpoints.

JavaScript Object Notation (JSON) was the data interchange format used throughout the selected technologies. Once NodeJS was chosen as the backend application process engine, it then became a case of selecting other compatible technologies. ElasticSearch, Kibana and Redis Server selected themselves for no other reasons than being an excellent fit and highly regarded in the developer community (Stackoverflow, 2019). ElasticSearch filled in the Atlas gap as the full-text search engine for the video database. Kibana is a sister product to

ElasticSearch and provided Learning Analytics gleaned from the video database metrics. NodeJS does not come with session caching, and Redis Server plugged this gap and could also offer a messaging queue between NodeJS processes.

The original intent for the video library was to use FFmpeg for transcoding, the file system or Amazon S3 for storage, and the Google Audio-to-Text API for transcription. This approach represented a significant amount of work which could otherwise be offloaded to a cloud service. Cloudinary selected itself over other cloud providers such as Mux and Imagekit.io by offering upload, transcoding and transcription on the fly, storage, and delivery for a basic subscription fee. Cloudinary integrated with both Google Cloud Platform (GCP) and Microsoft Azure for video transcription, and the actual transcription responses from these services were compared. After initial testing, GCP appeared to provide slightly more accurate text recognition over the equivalent Microsoft Azure API. However, the format of the response, and, the lack of topics and keywords, meant GCP was harder to code against and provided an arguably weaker user experience. The cost was also a factor taken into consideration, with Microsoft Azure being cheaper than GCP, and so it was selected for the final year project.

The hosting provider Linode was selected for the solution platform because KU was already using it. Ubuntu Server was used as the operating system on a single Linode server. This server provided a sufficient platform on which to deploy most of the selected software. However, to save system resources and costs, MongoDB was implemented in the cloud along with Cloudinary.

Appendix E – Component Details

Recap Client

A Single Page Application (SPA) written in Angular using HTML5, CSS 3 and Typescript (which transcompiles to JavaScript) provided the web client (portal) for Lecturers and Students to access the system. The main advantage of building the client as a SPA was the ability to produce the equivalent of the full-blown desktop application in a web browser. The views rendered for Lecturers were very complicated under the hood and Angular made the development effort required to make these a lot easier.

Recap API

A RESTful API built using NodeJS (and Express) enabled a single application process to serve as the central hub of the application which sent and received messages between the Recap client and backend components. Writing a RESTful API with JSON is so much easier than writing a web service with XML, and all the backend components supported this communication protocol and message format. The Recap API communicated in the same way with Mongo Atlas, Elastic Search, Redis and Cloudinary. Choosing NodeJS was a distinct advantage for its ability to integrate with these other components, and application development with JavaScript is rapid and robust.

Mongoose

Mongoose provided an Object Data Modelling (ODM) wrapper for MongoDB and NodeJS and made storing and retrieving unstructured data extremely easy. While MongoDB is a schema-less NoSQL document database and provides agility and flexibility, Mongoose brought some necessary structure to ensure there were data contracts for API messages. For instance, the system relied on messages from Cloudinary to always be in the same format so they could be processed successful and stored in MongoDB in a standardised structure. That said, one of the

critical success factors for the project was to welcome change, and one of the advantages of using a MongoDB with Mongoose is the data schema was easy to update.

MongoDB Atlas

The NoSQL database stores information about users, lecture recordings and their mappings in the cloud which provided a highly scalable and performant cluster with everything already configured. MongoDB uses collections with data stored in the form of JSON style documents. This document orientated storage pattern made the database an excellent tool for building a modern web application with a large amount of unstructured data.

ElasticSearch (and Kibana)

The ElasticSearch search engine stored a copy of the published lecture recording content to provide views for students. The content could be queried using fuzzy logic to deliver a powerful search capability for students wanting to drill down into lecture recordings and find specific material. Learning Analytics were provided by a Kibana dashboard, which enabled insights to be gained about the lecture recordings, segments and viewing statistics.

Redis

The messaging queue and caching formed a crucial layer in the backend architecture and used Redis, a fast, in-memory key-value database. The queue system used Bull, a NodeJS library, to provide a means to smooth out processing peaks and create a robust communication channel between NodeJS processes and threads. Messages flowed from the Recap API to the Asset Manager and ElasticSearch. It enabled long-running jobs such as uploading lecture recordings to Cloudinary, and, synchronising the ElasticSearch index with MongoDB, could be executed behind the scenes without requiring the user to have to wait for task completion on the same thread. Caching of the user session sped up the frontend user experience.

Asset Manager

The Asset Manager was a NodeJS process tasked with handling the backend processing of lecture recordings. New jobs were initiated by the Recap API placing messages on the queue. The jobs responded to events in the system, such as uploading and deleting lecture recordings. When a user used the screen recording feature to create a new lecture recording, the Recap client uploaded a video stream in chunks. The Asset Manager reassembled the video from the chunks, uploaded it to Cloudinary and called back to the Recap API to update a lecture status stored in MongoDB.

Cloudinary

Cloudinary is a highly scalable cloud-based digital asset library used to prepare, store, and deliver images and videos to the Recap client. Cloudinary exposes a RESTful web API, and this was used by the Asset Manager to upload, process, and delete resources upon request from the Recap API. Request parameters asynchronously converted uploaded videos to MP4 using eager transformations and generated the video transcript using the Microsoft Azure Video Index API. The Recap API provided Cloudinary with a notification endpoint, and when jobs completed on the Cloudinary and Microsoft Azure platforms, messages were received and processed. This architecture enabled the Recap API to update the video playback format to MP4 and store the video transcript.

Microsoft Azure API

The Microsoft Azure Video Index API was accessed via Cloudinary, making development

more straightforward. The Video Index API returns a transcript of a video as a JSON message with the audio-to-text conversion stored in chunks. The message also contained keywords which are commonly used terms and topics which are overarching themes of the narrative. This low level of granularity was used to enable the user to drill down into the segments of the lecture recording to focus on a specific topic or concept.

Appendix F – Testing Evidence

Unit Testing

Perhaps the biggest surprise from developing a service-orientated JavaScript-based solution was the line between unit tests and integration tests being fuzzy. The expectation going into the project was to achieve 70% code coverage with automated unit testing. However, an initial investigation found that unit tests required too much use of mocking and confidence in unit tests was low. Furthermore, component APIs already had unit tests, and these did not need to be repeated. Instead, moving up the testing stack and focusing on integration and end-to-end testing increased confidence, speed of development, and usefulness.

On the frontend, the Angular application was developed in Visual Studio Code using TypeScript and ESLint. Static typing and linting tools such as these gave a remarkable level of confidence in the quality of the written code. Manual unit tests tested logic such as that contained in *getRequestFromUrl* while writing unit tests with mock objects for functions which used services such as the *SettingsService* was deemed un-useful and were integration tested instead.

```
web > src > app > services > portal-service.ts > PortalService > getRequestFromUrl
52
53    getRequestFromUrl() {
54      const currentUser = this.authService.getCurrentUser();
55      let url = document.URL.replace(document.baseURI, '');
56      if (url.includes('?'))
57      {        You, 2 months ago • Fix bug with url ?tab=Record
58        const urlSubParts = url.split('?');
59        url = urlSubParts[0];
60      }
61      const urlParts = url.replace('portal/', '').split('/');
62      const request = new PortalRequest();
63      request.isSearchPage = false;
64      if (urlParts.length === 1 && urlParts[0].startsWith('search')) {
65        request.isSearchPage = true;
66        request.academicYear = currentUser.courses.academicYear;
67      } else if (urlParts.length === 1 && urlParts[0] === 'browse') {
68        request.academicYear = currentUser.courses.academicYear;
69      } else if (urlParts.length === 1 && urlParts[0] === 'home') {
70        request.academicYear = currentUser.courses.academicYear;
71      } else if (urlParts.length === 2 && urlParts[0] === 'content') {
72        request.moduleCode = urlParts[1];
73      } else if (urlParts.length === 3) {
74        request.moduleCode = urlParts[1];
75        request.lectureId = urlParts[2];
76      } else {
77        return request;
78      }
79      this.fillRequest(currentUser, request);
80      return request;
81    }
82
83    private fillRequest(currentUser: CurrentUser, request: PortalRequest): void {
84      if (request.moduleCode) {
85        request.module = currentUser.courses.modules.find(m => m.code === request.moduleCode);
86      } else {
87        request.module = null;
88      }
89      if (request.module) {
90        request.academicYear = request.module.academicYear;
91      } else {
92        request.academicYear = currentUser.courses.academicYear;
93      }
94    }
```

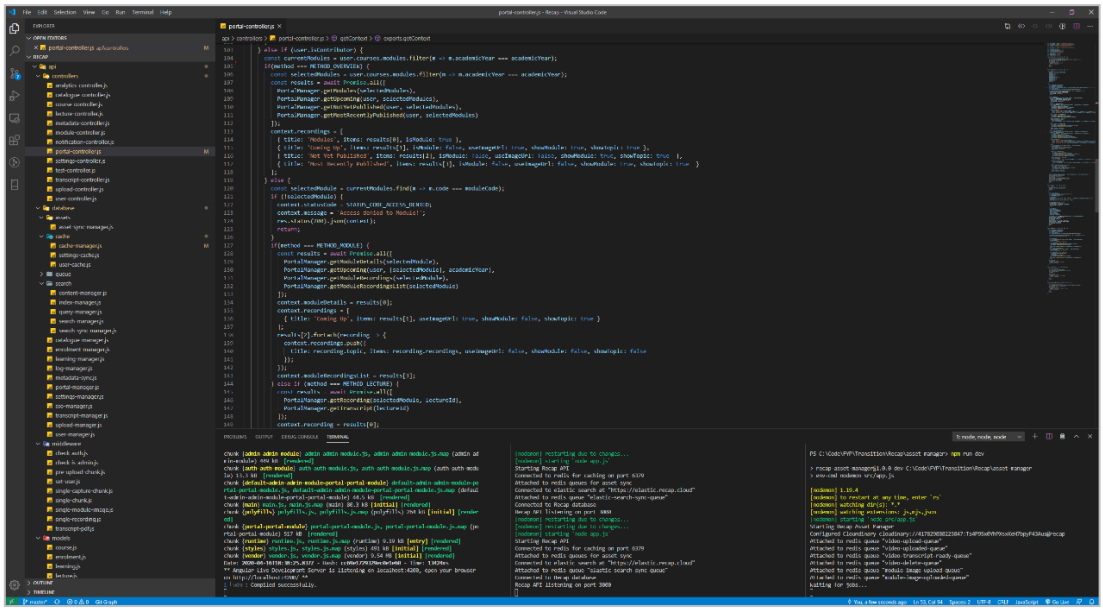
```
web > src > app > services > settings-service.ts > SettingsService > constructor
      You, 3 months ago | 1 author (You)
1     import { Injectable } from '@angular/core';
2     import { HttpClient } from '@angular/common/http';
3
4     import { environment } from '../../environments/environment';
5
6     import { SystemSettings } from '../models/system-settings';
7     import { Observable } from 'rxjs';
8
9     const BACKEND_URL = environment.apiUrl + '/settings/';
10
      You, 3 months ago | 1 author (You)
11    @Injectable({ providedIn: 'root' })
12    export class SettingsService {
13
14      constructor(        You, 3 months ago • Add settings editor
15        private http: HttpClient) {
16      }
17
18      getSystem(): Observable<SystemSettings> {
19        return this.http.get<SystemSettings>(BACKEND_URL + 'system');
20      }
21
22      updateSystem(newSystemSettings: Partial<SystemSettings>) {
23        return this.http.put(BACKEND_URL + 'system', newSystemSettings);
24      }
25
26    }
27
```

The same issue with unit testing was also found to be true of the back end. The same testing framework, Jasmine, was going to be used for NodeJS because it makes writing unit tests easy. In theory, writing a series of unit tests with mock objects enables the developer to use a testing framework and receive confidence when the tests all pass. However, mocking led to false confidence since very little of the actual NodeJS code was being tested. So integration test scripts proved more useful than automated unit tests. Real confidence in the solution came from extensive debugging, integration, and system-level testing.

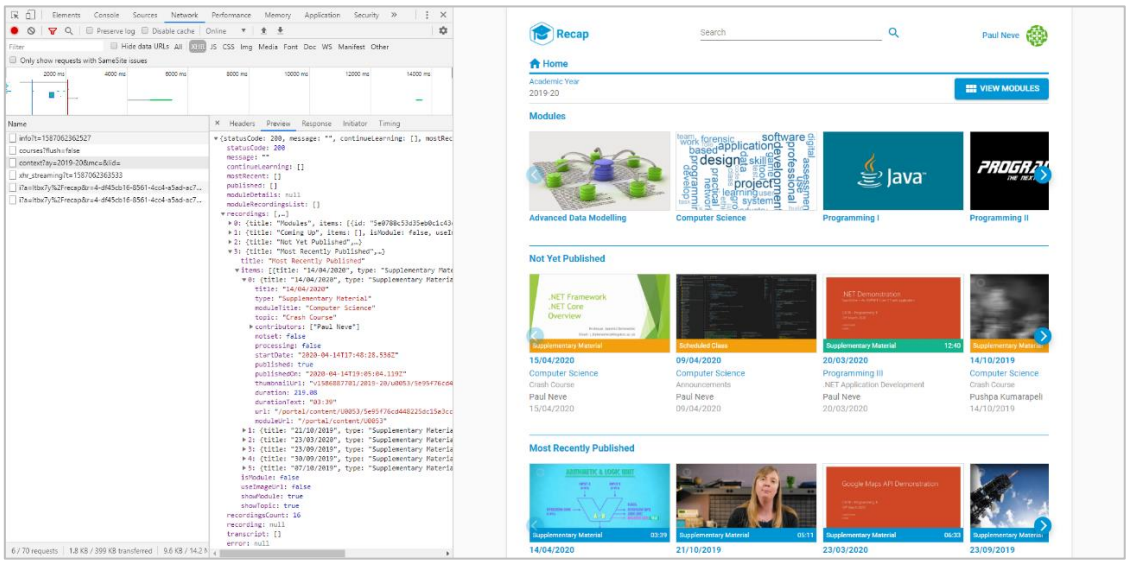
Debugging

From a development perspective, writing the entire solution in JavaScript had a distinct

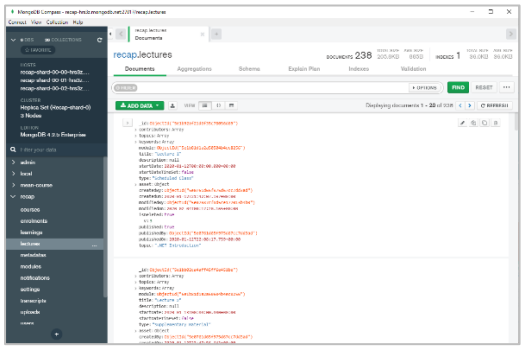
productivity advantage. It was possible to write all the code in a single Visual Studio Code solution folder, debug it using Google Chrome Developer Tools, and run the processes (for the Recap client, API and Asset Manager) via the terminal. Trace and log information was output to the terminals for each process, making the debugging workflow simple and easy to follow.

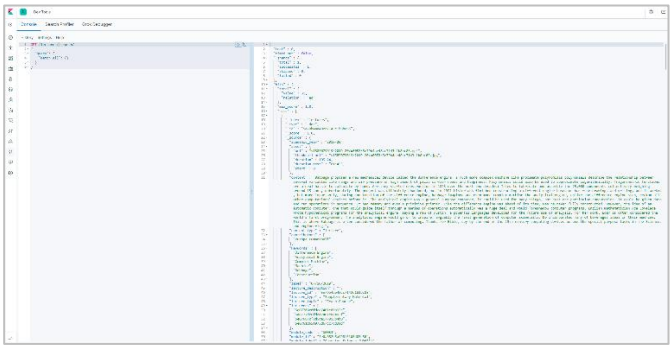


Both Angular and NodeJS supported fast and automatic recompilation, meaning a code change would cause the process to recycle with very little waiting time. The back and front end code could be adjusted, and the result seen in Chrome Developer Tools almost immediately.

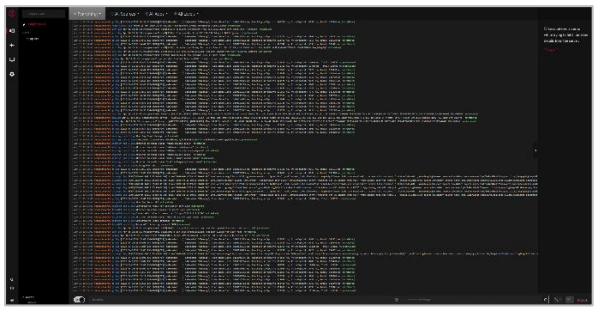


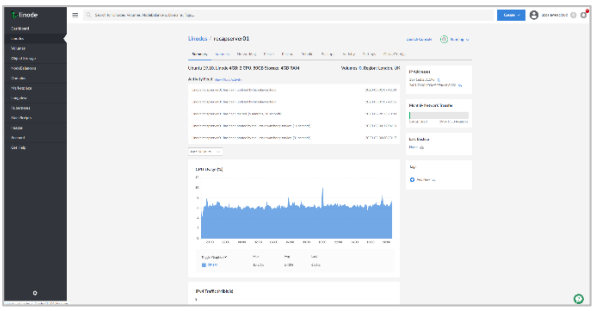
During the initial integration and setup work of the backend, MongoDB Compass and Kibana Dev Tools were used to query the database and search index, respectively.





From time to time, the Linode server logs and process metrics needed to be interrogated. The server and application logs were sent to LogDNA, a cloud-hosted service which provided a real-time view across all the log files (shown below on the left). Linode provided server metrics with notifications, which proved useful in identifying server configuration errors, such as when ElasticSearch kept crashing regularly, and it was discovered the server needed more memory to keep it loaded in memory.





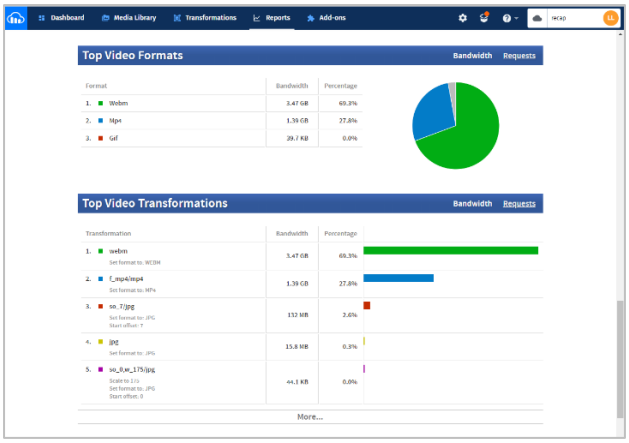
Most solution code passed through into a stable API such as the Angular core framework, Mongoose, Cloudinary, ElasticSearch and Bull (Redis). A considerable amount of effort was involved in testing these back end services and integrating them.

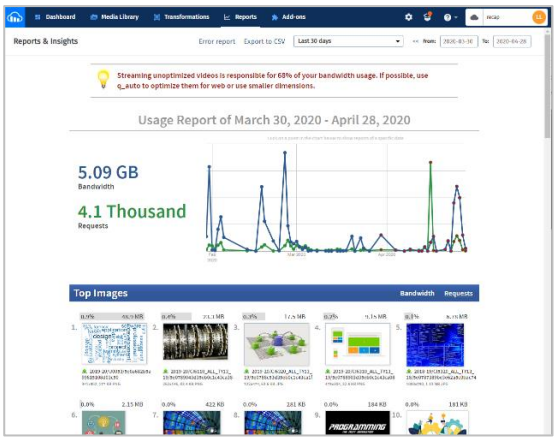
Back End Testing

The plugging in of each service into the Recap API and Asset Manager started by writing small test scripts for each component and function. Tests were coded for things like reassembling lecture recording chunk files into a raw video file, and, uploading it to Cloudinary.

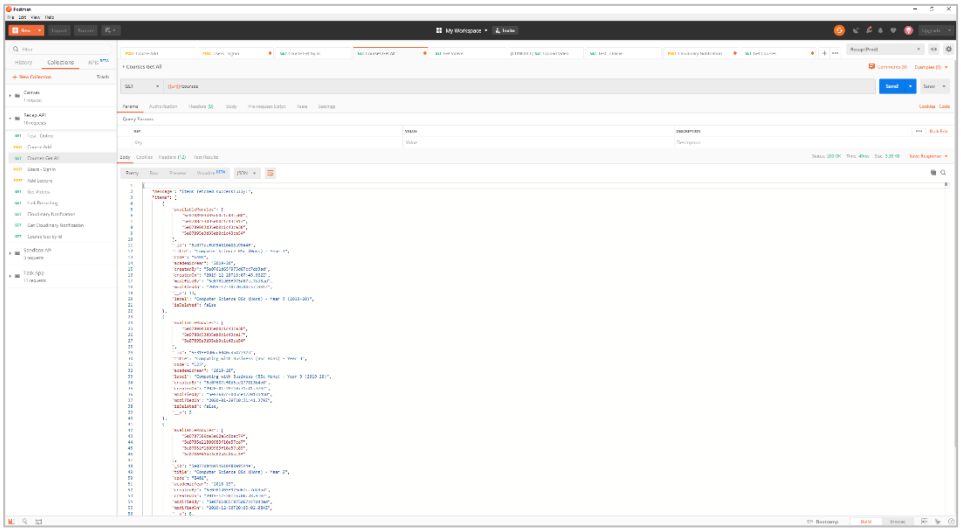
```
asset-manager > src > test_chunks.js > ...
You, 3 months ago | 1 author (You)
const path = require('path');
const fs = require('fs');

const chunksPath = path.join(__dirname, '..', '..', 'assets', 'chunks');
const uploadsPath = path.join(__dirname, '..', '..', 'assets', 'uploads');

const lectureId = '5e1484fc123d332454804119';

console.log(chunksPath);      You, 3 months ago • Add basic support for capture chunks

const fileStream = fs.createWriteStream(uploadsPath + '\\' + lectureId + '.webm', {flags: 'a'});

fs.readdir(chunksPath, (e, chunkFiles) => {
    chunkFiles.forEach((chuckFilename) => {
        if(chuckFilename.includes(lectureId)) {
            const data = fs.readFileSync(chunksPath + '\\' + chuckFilename);
            const dataBuffer = Buffer.from(data, 'base64');
            fileStream.write(dataBuffer);
            fs.unlinkSync(chunksPath + '\\' + chuckFilename);
        }
    });
});
```

```
asset-manager > src > test_eager.js > eager
You, 2 months ago | 1 author (You)
var fetch = require('node-fetch');
var cloudinary = require('cloudinary').v2;
const settings = require('./settings');

cloudinary.uploader.upload_large('C:\\Temp\\video.webm', {
        resource_type: 'video',
        folder: 'Test',
        eager: [        You, 2 months ago • Set eager with fetch format only
          { fetch_format: "mp4" }
        ],
        eager_async: true,
        use_filename: false,
    }, (error, result) => {
        console.log(error, result.public_id);
    });
```





The Recap API was tested using Postman, which simplified the workflow for developing the RESTful API and testing the instance in the development and production environments.



Using NodeJS, Mongoose and MongoDB together proved an incredibly quick development workflow. Getting Redis, MongoDB and ElasticSearch to talk to one another required more advanced development skills and in-depth integration testing.

Integration Testing

It was at the integration level that testing proved most valuable and confidence in the solution was dependent on getting this right. Extensive testing took hours of effort to determine whether the way the microservices were wired together worked and would scale. Test scripts were written to connect the services and see whether the expected results were returned to the terminal and logs. Examples included seeing whether lecture information stored in MongoDB could be synchronised with ElasticSearch via a Redis Queue, and, examining whether the same lecture information stored in ElasticSearch could be returned from the index.

```
api > test_content.js > ...
You, 3 months ago | 1 author (You)
1  const mongoose = require('mongoose');        You, 3 months ago • Implement onLectureChange, Delete and onReindex
2
3  const searchManager = require('./database/search/search-manager');
4  const ContentManager = require('./database/search/content-manager');
5
6  const settings = require('./settings');
7
8  searchManager
9    .connect()
10     .then((response) => {
11       console.log(response);
12     })
13     .catch(error => {
14       console.log(error);
15     });
16
17 mongoose
18     .connect(
19       settings.getDatabaseUrl(),
20       {
21         useNewUrlParser: true,
22         useCreateIndex: true,
23         useUnifiedTopology: true
24       }
25     )
26     .then(() => {
27       console.log('Connected to recap database');
28     })
29     .catch((error) => {
30       console.log('Connection to recap database failed! ' + error);
31     });
32
33 const run = async () => {
34     await ContentManager.onReindex({});
35     await ContentManager.onLectureChange({
36       lectureId: '5e1706904ce19b059ad50896'
37     });
38     await ContentManager.onLectureDelete({
39       lectureId: '5e1706904ce19b059ad50896'
40     });
41 };
42
43 run()
44     .then()
45     .catch(error => console.log(error));
```

```
api > test_elastic.js > ...
You, 2 months ago | 1 author (You)
1  const mongoose = require('mongoose');        You, 3 months ago • Add elastic tests incl test data
2
3  const SearchManager = require('./database/search/search-manager');
4  const IndexManager = require('./database/search/index-manager');
5  const ContentManager = require('./database/search/content-manager');
6  const QueryManager = require('./database/search/query-manager');
7
8  const Lecture = require('./models/lecture');
9
10 const settings = require('./settings');
11
12 mongoose
13   .connect(
14     settings.getDatabaseUrl(),
15     {
16       useNewUrlParser: true,
17       useCreateIndex: true,
18       useUnifiedTopology: true
19     }
20   )
21   .then(() => {
22     console.log('Connected to recap database!');
23   })
24   .catch((error) => {
25     console.log('Connection to recap database failed! ' + error);
26   });
27
28 const resetIndex = async () => {
29   console.log(await SearchManager.connect());
30   await IndexManager.createTranscriptsIndex();
31   await IndexManager.createLecturesIndex();
32   await ContentManager.syncLectures();
33   return 'Done!'
34 };
35
36 const deleteLecture = async (lectureTitle) => {
37   console.log(await SearchManager.connect());
38   const lecture = await Lecture
39                           .findOne()
40                           .where('title').equals(lectureTitle);
41   console.log(lecture._id);
42   await ContentManager.onLectureDelete( {
43     lectureId: lecture._id
44   });
45 };
46
47 const search = async () => {
48   console.log(await SearchManager.connect());
49   const result = await QueryManager.getLectures({
```

Another area of integration testing was writing scripts to see if user information stored in MongoDB could be persisted and returned from the Redis cache. Likewise, integration testing also saw scripts written to see whether the same result could be achieved for data stored in the ElasticSearch index.

```
api > test_pm.js > ...
You, 3 months ago | 1 author (You)
1  const mongoose = require('mongoose');        You, 3 months ago • Refactor student modules
2
3  const CacheManager = require('./database/cache/cache-manager');
4  const UserCache = require('./database/cache/user-cache');
5
6  const User = require('./models/user');
7
8  const settings = require('./settings');
9
10 const run = async(displayName) => {
11
12   console.log(await CacheManager.connect());
13
14   await mongoose.connect(settings.getDatabaseUrl(), {
15           useNewUrlParser: true,
16           useCreateIndex: true,
17           useUnifiedTopology: true
18         });
19
20   console.log('Connected to Recap database');
21
22   const user = await User.findOne()
23     .where('displayName').equals(displayName)
24     .where('isDeleted').equals(false);
25
26   console.log(user.displayName);
27
28   const cachedUser = await UserCache.getUser(user._id, true);
29   console.log(cachedUser);
30 };
31
32 run('Paul Neve').then().catch(error => console.log(error));
```

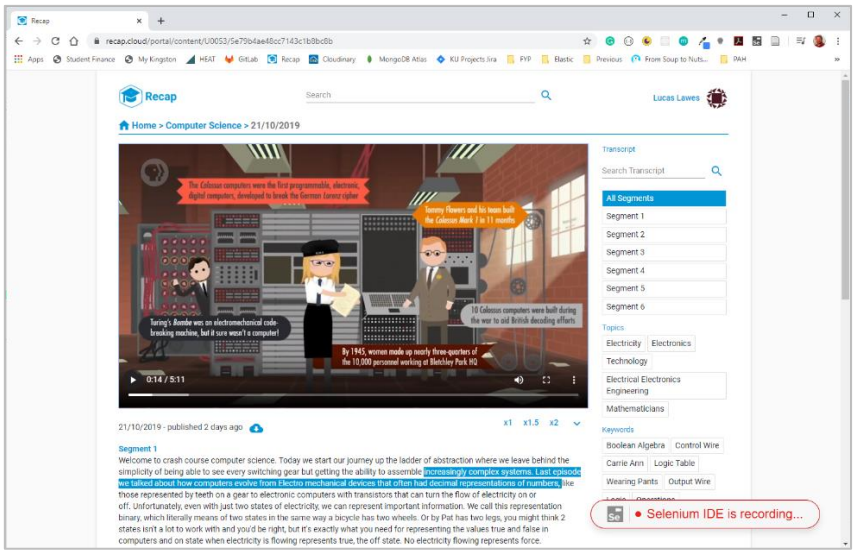
```
api > test_elastic_query.js > ...
You, 3 months ago | 1 author (You)
1  const queryManager = require('./database/search/query-manager');        You, 3 months ago
2  const cacheManager = require('./database/cache/cache-manager');
3
4  const run = async (clearCache) => {
5    if (clearCache) {
6      await cacheManager.clear('query');
7    }
8    let cachedQuery = await cacheManager.getAndSetJson('query', async () => {
9      return await queryManager.getLectureSegments({
10       query: 'sql',
11       lectures: ['5e159fb93feca971142a9e84'],
12       contributor: null,
13       topic: 'Software',
14       keyword: 'Operating System'
15     }, 1);
16   });
17   return cachedQuery.value;
18 };
19
20 const start1 = Date.now();
21 run(true).then(response => {
22     console.log(Date.now() - start1);
23     const start2 = Date.now();
24     run(false).then(response => {
25       console.log(Date.now() - start2);
26     })
27     .catch(error => console.log(error.body ? error.body : error));
28   })
29   .catch(error => console.log(error.body ? error.body : error));
```

Once the integration of the microservices had been successfully tested, end-to-end testing was undertaken using both manual and automated Front End testing.

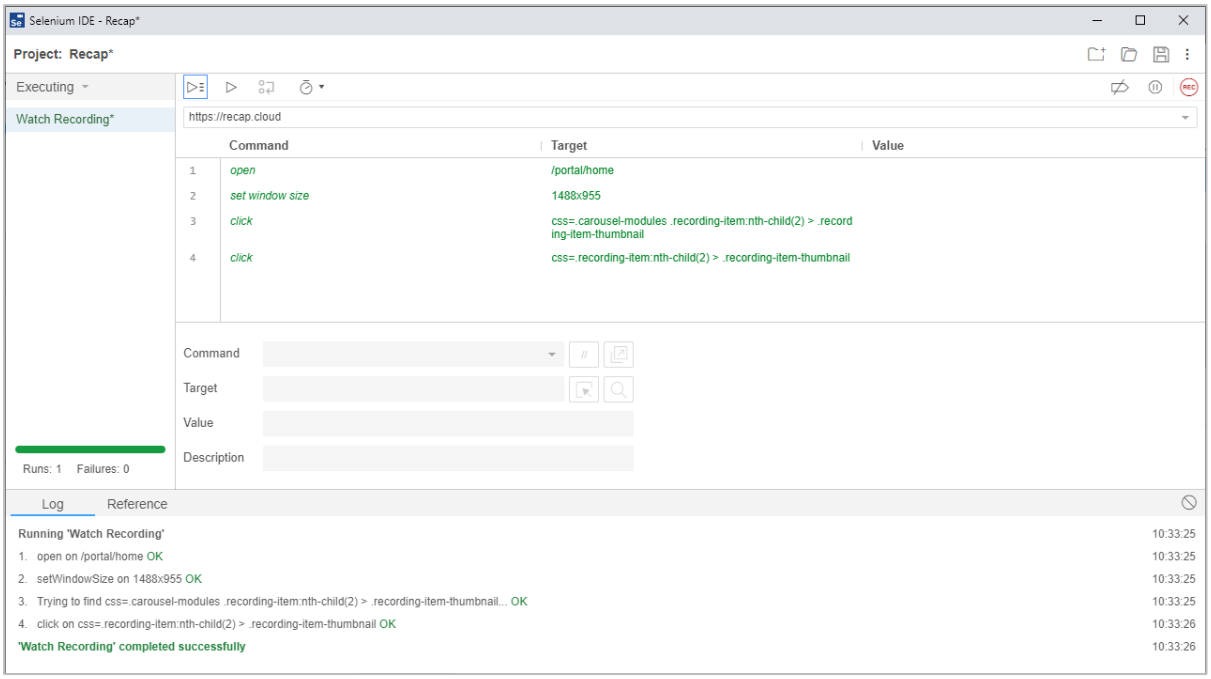
Front End Testing

The most rewarding moment in all the testing happened whenever a Front End test passed successfully. A significant amount of testing was completed manually by interacting with the user interface. Sometimes the testing needed to be automated so that repeatable tests could be performed.

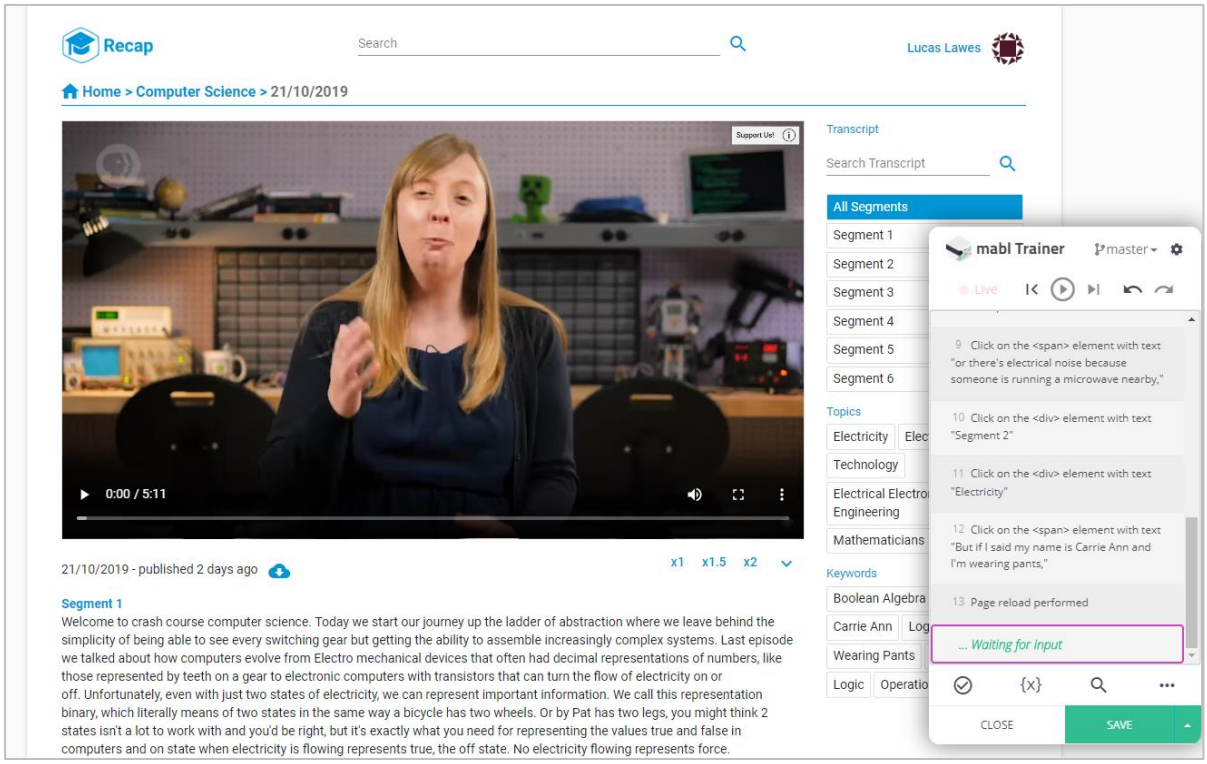
Selenium provided a playback tool for automating functional tests which could trigger end-to-end testing. Steps for page flows such as watching a lecture recording were recorded in the web browser and created a repeatable process for testing the back end services, and components functioned as expected.



The user interface contained many user interactions and being able to record different combinations and replaying them enabled repeatable tests to be conducted when the Recap client was refactored.

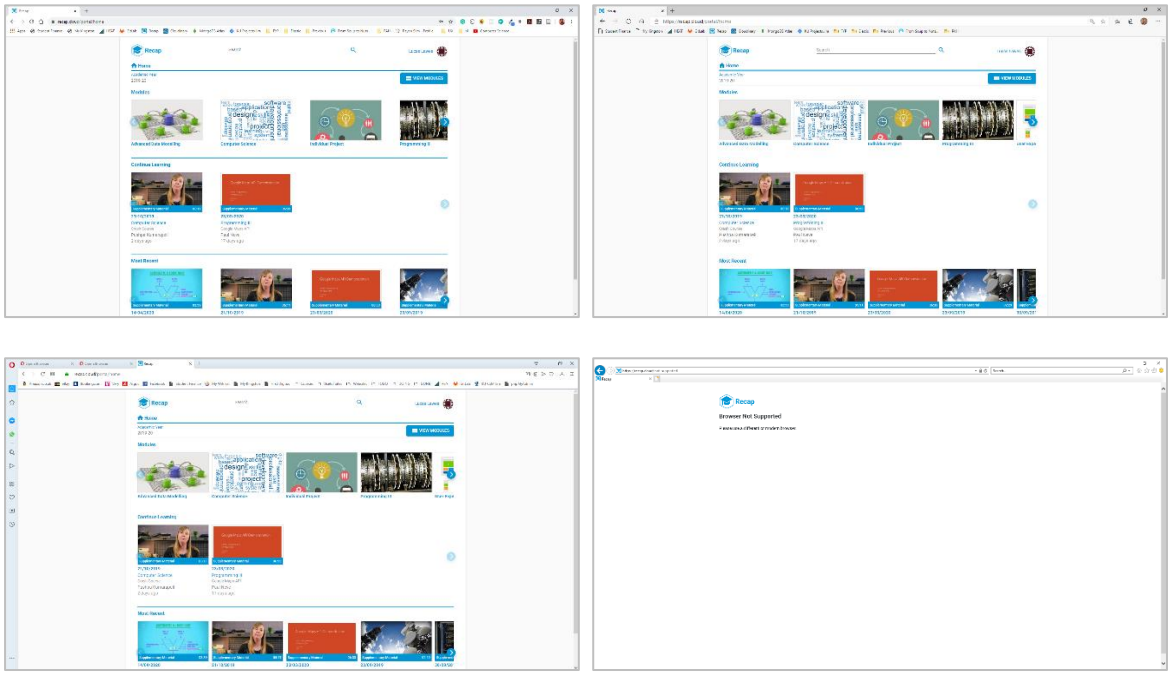


Mabl is a cloud-hosted test automation solution which was also used to record functional tests which triggered the back end. While Selenium was used in the development environment, Mabl was used in the production environment and had the advantage of being able to conduct the same functional test across different web browsers at the same time.

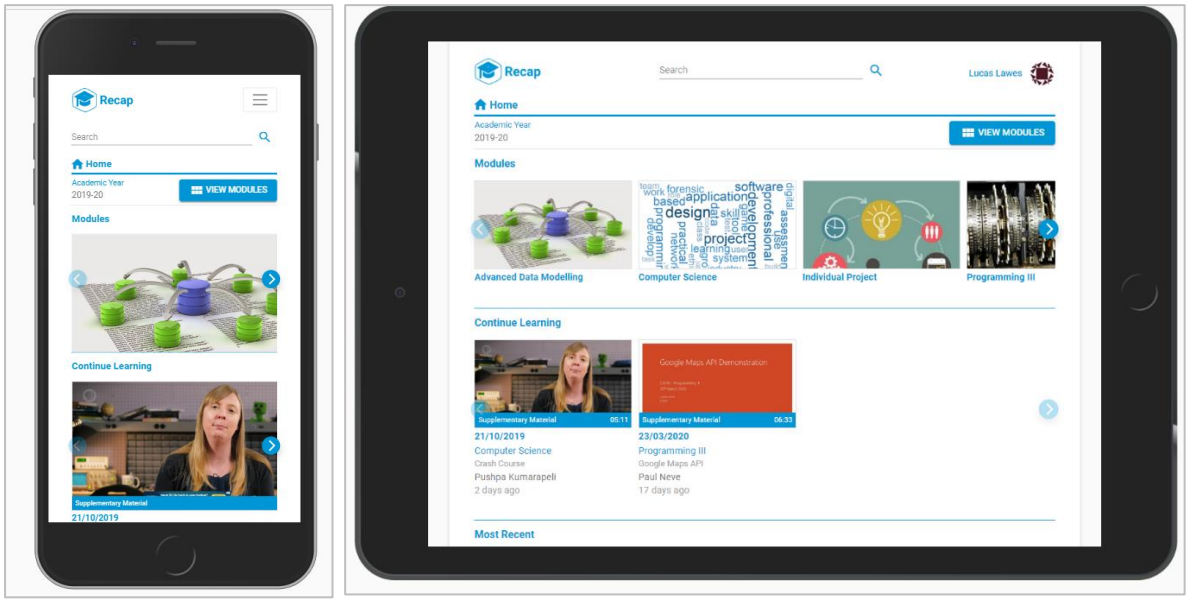


Browser Compatibility Testing

The Recap client was thoroughly tested in all modern browsers, including Chrome, Edge, and Opera. Internet Explorer was not supported, and the user asked to choose a modern browser.



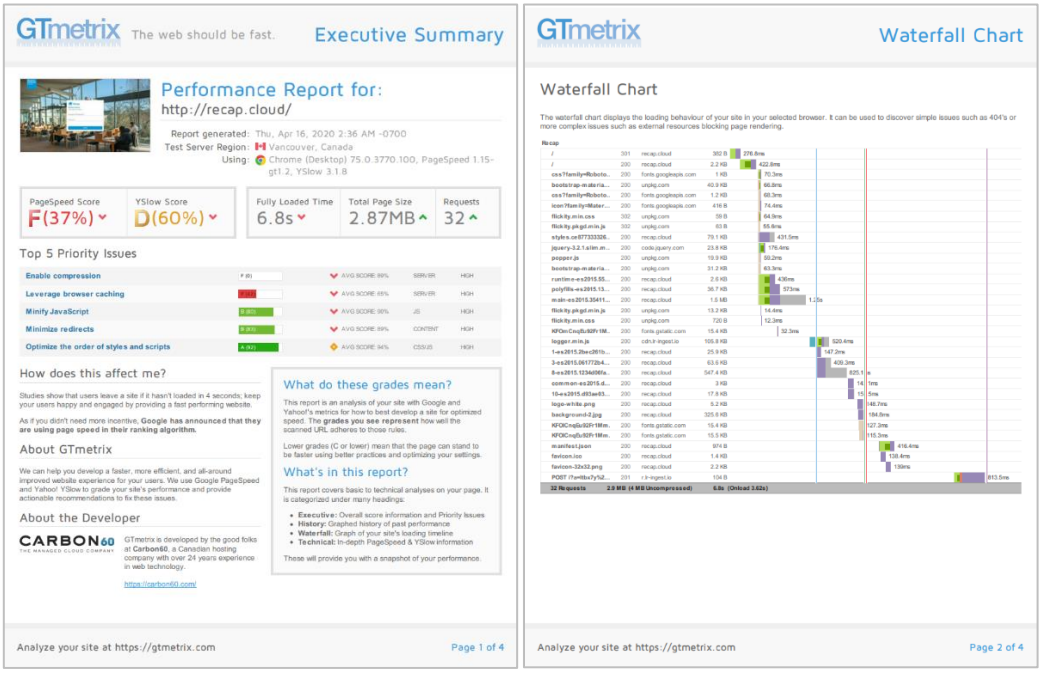
Testing with Safari uncovered issues with completing the lecture capture task. While all other user stories could be completed in Safari on a Mac, iPhone and iPad, the lecture capture task issues were left unresolved despite repeated attempts to remedy them.



Manual browser testing was also conducted at KU in the library and lecture rooms, and students were asked to test the Recap client by logging in with their KU credentials. The Recap client performed well for ad hoc users, and so the focus of testing shifted to performance testing.

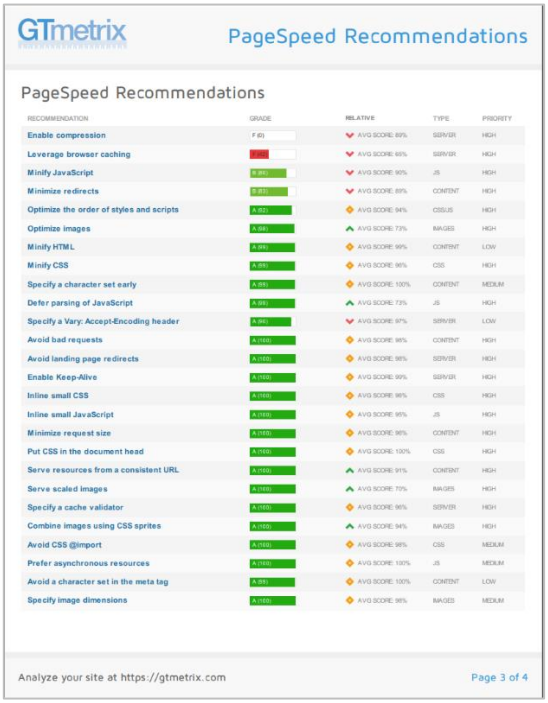
Performance Testing

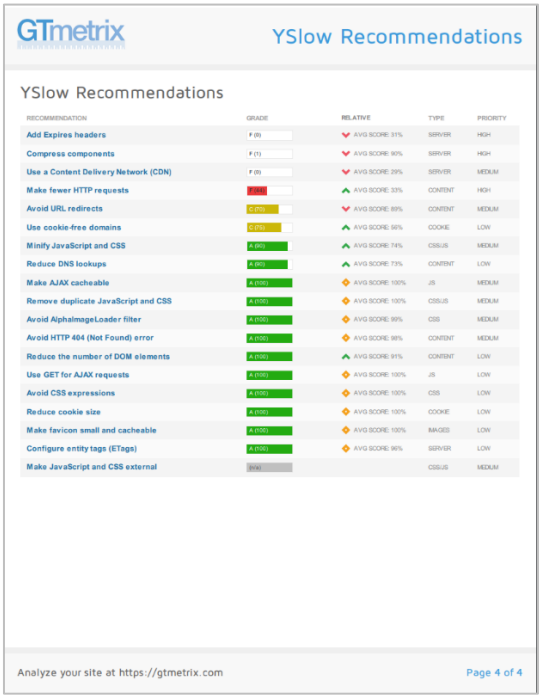
While a single Linode server was configured in the production environment – therefore, the NodeJS processes were not clustered - lightweight performance testing was conducted using GT Metrix automation tools.



While the results summary looked rather alarming at first glance, the automation tool was downloading the entire Angular application, and, triggering requests to LogRocket, which was only there itself for testing purposes.

Page loadings times could be reduced by enabling compression on the Nginx web server which hosted the Angular application. In future work, the Administration microsite could be removed from the Recap client to reduce the size of the SPA download bundle by half.

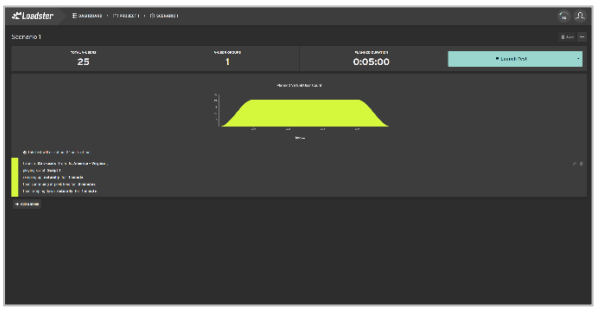


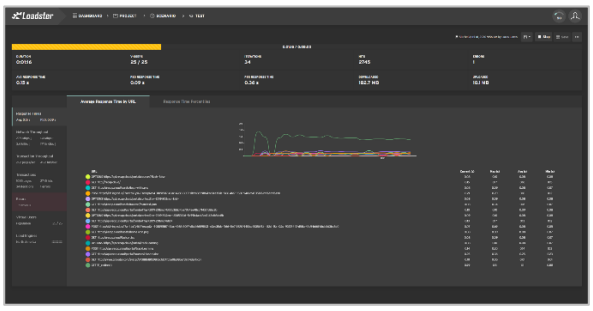


Further performance testing of the RESTful Recap API is required; however, it was tested as part of Stress Testing.

Stress Testing

After the final release of the software was deployed to production, stress testing was conducted to confirm that the production environment could handle Usability Testing. Stress Testing was performed using a cloud-hosted platform called Loadster, which was given the same test scripts as Mabl. When LogRocket was factored out, all the requests, including those to the Recap API, performed well enough to handle more than the load expected in Usability Testing and other ad hoc testing. Testing showed there was no need to alter the configuration and specification of the Linode server.



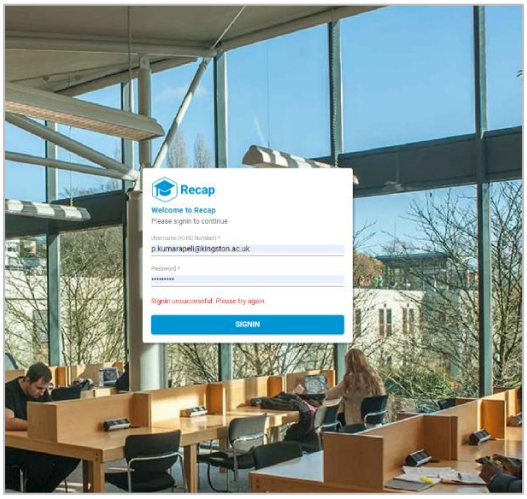


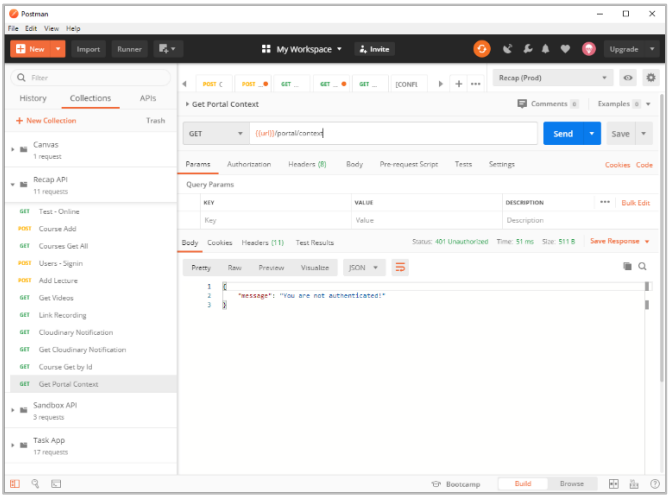
With the production environment prepared for user testing, the Linode server was security

tested.

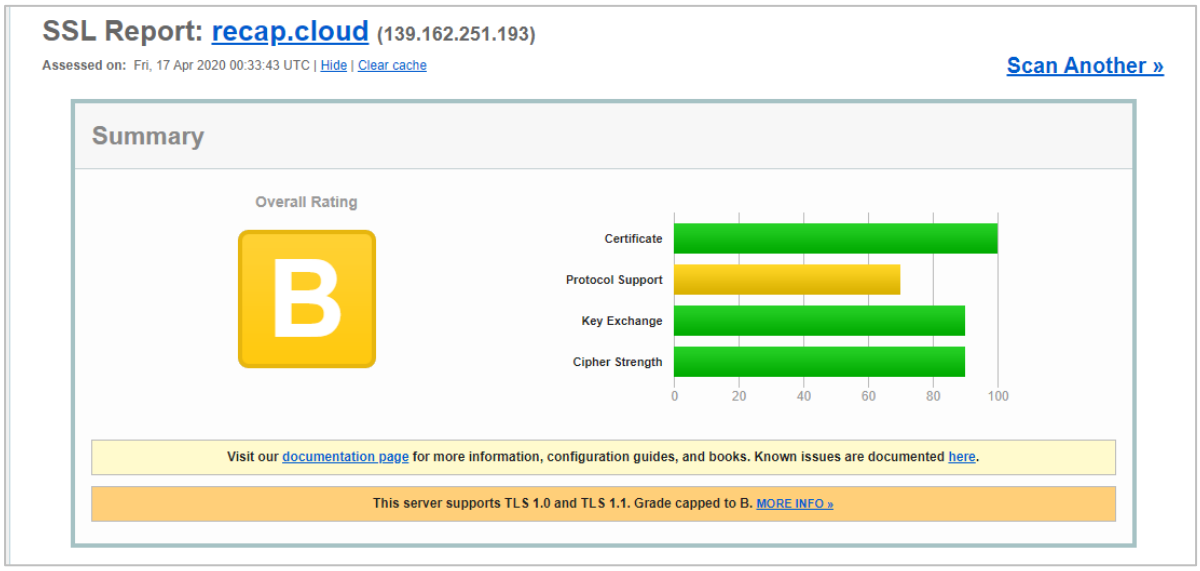
Security Testing

The Recap client and API were designed to be secured by requiring users to authenticate against the KU active directory. If users do not provide the correct credentials, they are not granted access to the Recap portal, and access to the Recap API is not authorised unless a user is authenticated against the Auth service beforehand.

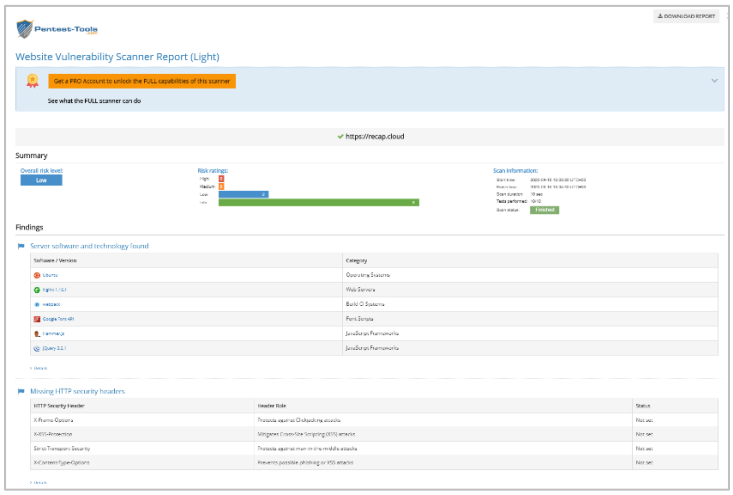


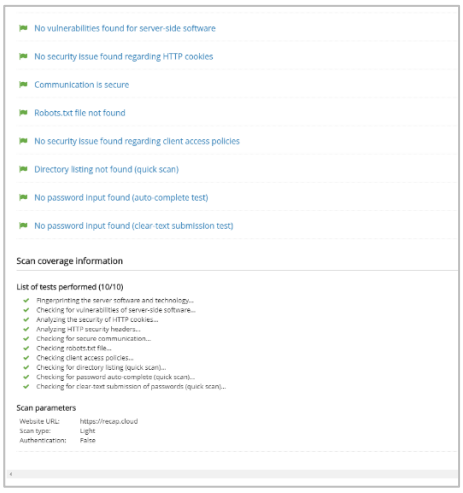


The Nginx web server has a free certbot HTTPS certificate installed which SSL Labs gave a B Grade rating to because it is only configured for TLS 1.x. However, this is sufficient for the project and means all microservice endpoints are running over HTTPS.



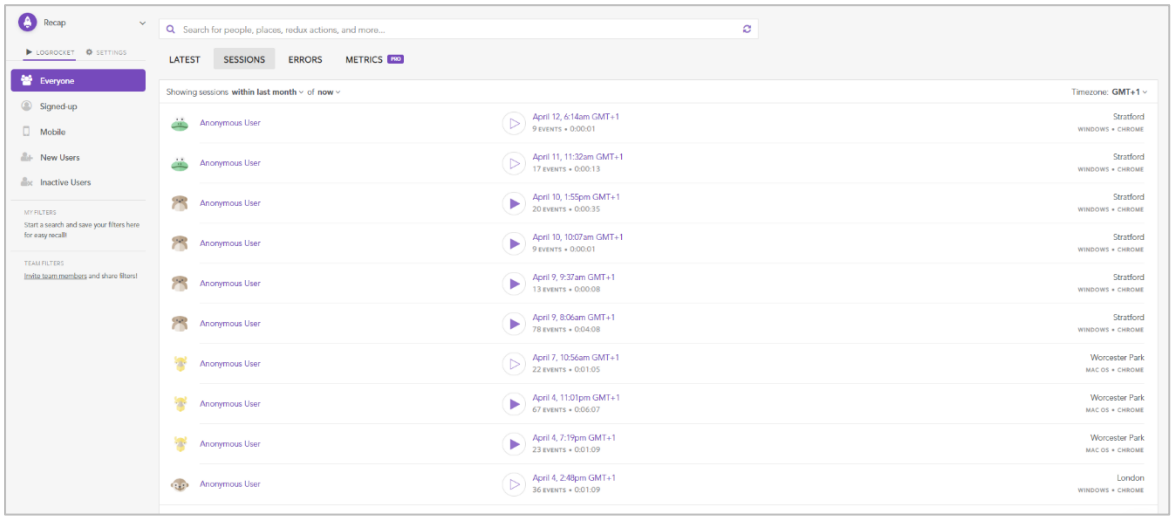
Pentest provided a necessary website vulnerability scanner report for recap.cloud and showed positive results. However, there is a vulnerability with the Recap client missing some HTTP Security Headers, and some further research is required to understand how to configure Nginx and NodeJS to serve these. For NodeJS, a package called Helmet was initially used to set these headers for the Express based RESTful API. However, it caused technical issues with the solution and was turned off to enable the Recap API to work. Resolving these issues and setting the headers is a must to protect against Cross-Site Scripting (XSS) attacks.

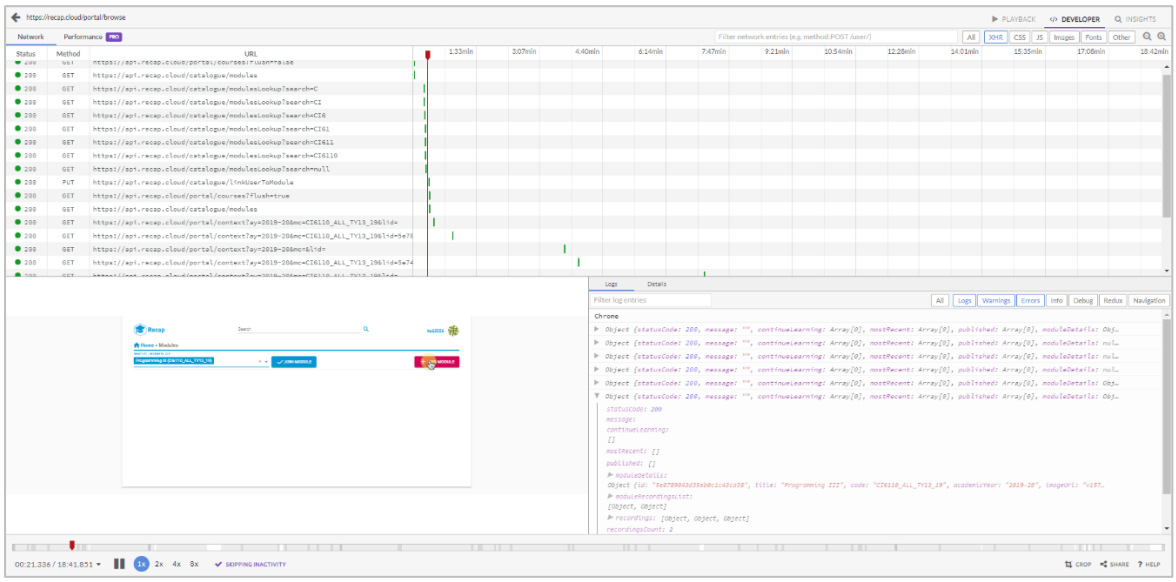




Remote User Testing

The Recap client was configured to enable a degree of remote user test recording by sending user-session information to LogRocket, which helps to understand problems affecting users.



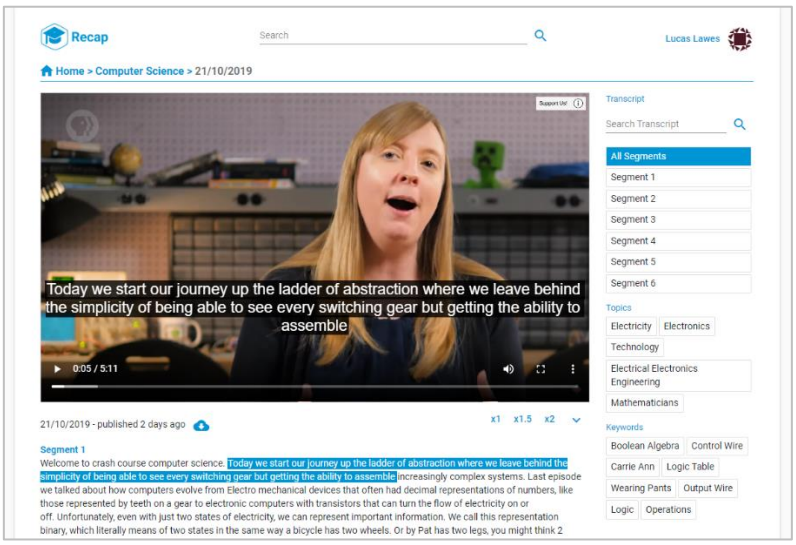


LogRocket allowed remote user testing sessions to be played back. From a usability standpoint, it was possible to understand where users found the user experience confusing or frustrating. For example, the Module catalogue does not provide the user with any hints, and for a Lecturer, the “Join Module” and “Add Module” buttons are confusing. For future work, guidance could be provided, and, a wizard could also be provided to speed up the time it takes to join modules after first joining the website.

Inclusivity is a crucial design pillar for the solution, and more work is needed to ensure the Recap client meets the expectations of all users and usage scenarios. Conducting a usability study which includes participation by both Lecturers and Students will go some way to achieving this. Accessibility Testing showed that this is another area where the solution can be improved.

Accessibility Testing

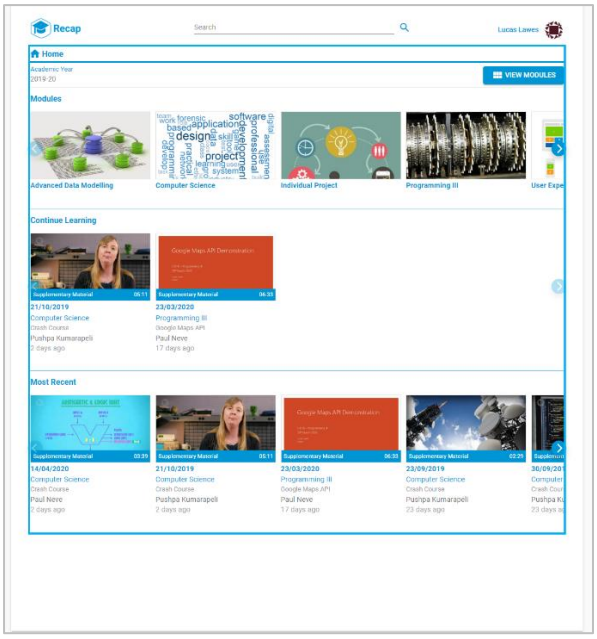
One of the user stories specifically required lecture recordings to have captioning, and although system-generated transcripts are by no means perfect, the feature appears to work.

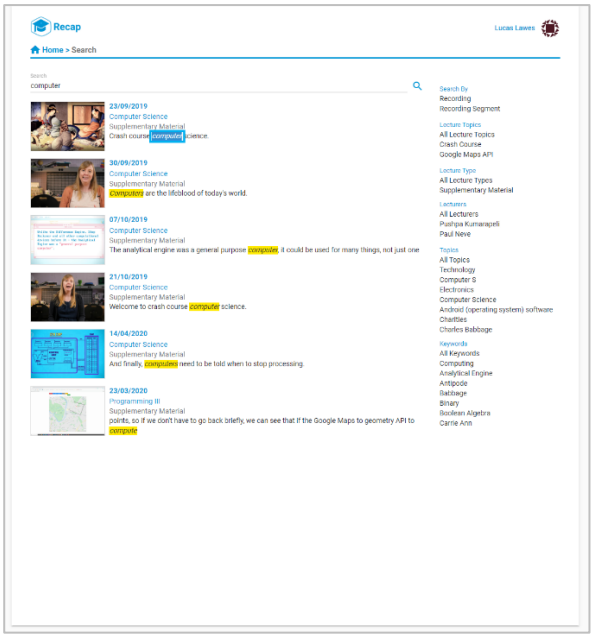


Testing showed there is room for improving accessibility in support for screen readers and support for those with a visual disability. The user interface design was intended to be minimal to make it informative yet easy to use. However, when using a screen reader, the underlying HTML is not structured particularly well for meeting these aims.

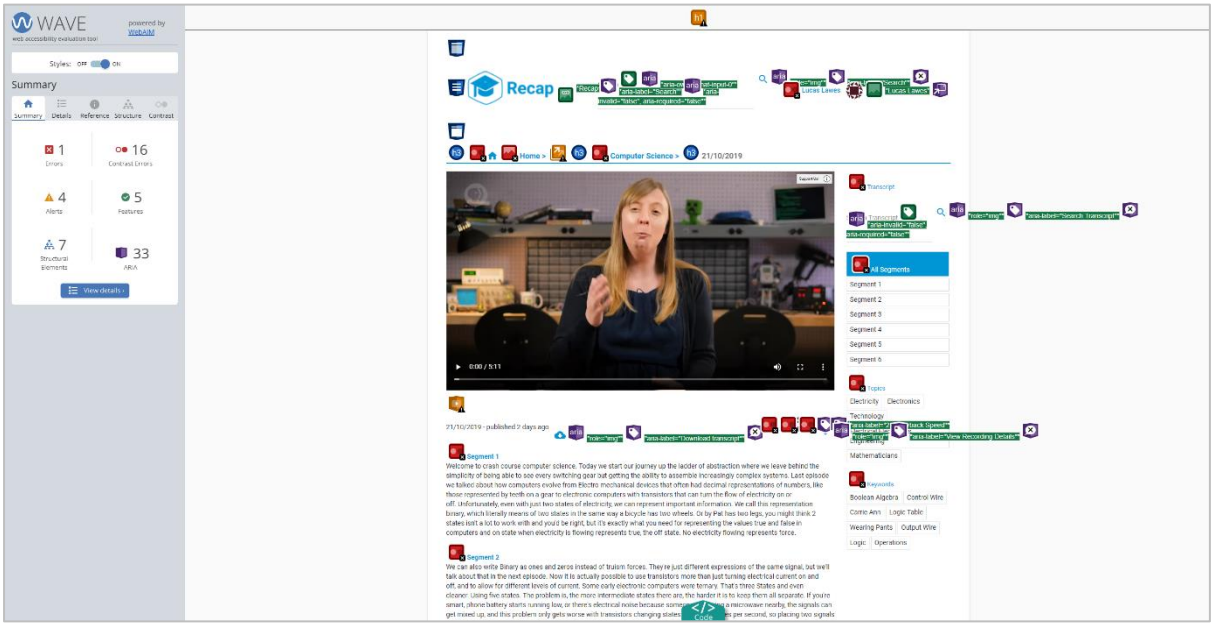
Testing showed that the screen reader did not understand that lecture recordings are an essential part of the navigation. Instead, the screen reader reads aloud the top navigation and then went silent.

When developing a website with Angular, it is possible to focus too much on the components and not enough on the structure of the HTML page. Future work will have to improve the structure of each page and ensure headings are read aloud.





The Web Accessibility Evaluation Tool (WAVE) by WebAIM was a Chrome Extension used to evaluate the HTML markup and visual contrast. The ARIA warnings were due to a lack of support for older browsers, which was not an issue. However, the brand blue on a white background raised 16 contrast errors, and future work should include the provision of a high-contrast theme. The screen reader struggled to understand the breadcrumb navigation, and this scored poorly in the test. Future work will also have to reconsider the navigation and titling approach to make it more mobile-friendly as well as for accessibility reasons.



Appendix G – Usability Test Pack

Student Task Instruction Sheet

Please complete the following tasks in order, one at a time. At the end of each task, please say wait to answer some post tasks questions before continuing to the next task.

Task 1

Sign in to the Recap website.

Task 2

Find and follow these five Computer Science Modules:

- *Computer Science*
- *Individual Project*
- *Advanced Data Modelling*
- *Programming III*
- *User Experience*

Task 3

Find the presentation on Recap and watch the segment on educational technology.

Task 4

Find the information on writing a critical review for a Final Year Project report.

Task 5

Find the revision material for Programming III Worksheet 10.

Task 6

Find the link to download the Worksheet 10 source code from Gitlab.

Lecturer Task Instruction Sheet

Please complete the following tasks in order, one at a time. At the end of each task, please say wait to answer some post tasks questions before continuing to the next task.

Task 1

Sign in to the Recap website.

Task 2

Find and join the Computer Science and Individual Project Modules.

Task 3

Create a new lecture recording for the Individual Project module using QuickStart and Screen Capture.

Task 4

Find the presentation on Recap, edit the transcript and publish the lecture recording.

Task 5

Add a colleague to be a contributor to the Individual Project Module.

Task 6

Find the lecture recordings on "interfaces" which relate to "Java Programming".

Observation Sheet

Participant No. _____

Task	Outcome C/I/A/F	Time on Task	Errors	Assists	Frustrations
1					
2					
3					
4					
5					
6					

Task	Mental Demand	Difficulty Level	Frustration Level	Satisfaction Level	Confidence Level
1					
2					
3					
4					
5					
6					

Post-Test Questionnaire

Satisfaction Survey

For each for the following statements, place a mark on one box that best describes your impressions of the Recap website.

		Strongly Disagree				Strongly Agree
1.	I think that I would like to use this website frequently.	☐	☐	☐	☐	☐
2.	I found this website unnecessarily complex.	☐	☐	☐	☐	☐
3.	I thought this website was easy to use.	☐	☐	☐	☐	☐
4.	I think that I would need assistance to be able to use this website.	☐	☐	☐	☐	☐
5.	I found the various functions in this website were well integrated.	☐	☐	☐	☐	☐
6.	I thought there was too much inconsistency in this website.	☐	☐	☐	☐	☐
7.	I would imagine that most people would learn to use this website very quickly.	☐	☐	☐	☐	☐
8.	I found this website very cumbersome/awkward to use.	☐	☐	☐	☐	☐
9.	I felt very confident using this website.	☐	☐	☐	☐	☐
10.	I needed to learn a lot of things before I could get going with this website.	☐	☐	☐	☐	☐

Customer Experience

For each for the following questions, please circle an answer that best describes your customer experience.

I like the Recap brand	YES | NO
The website made a good first impression	YES | NO
I would consider using Recap in the future	YES | NO
I would recommend Recap to a friend or colleague	YES | NO

User Experience Questionnaire

For each for the following statements, place tick one circle that best describes your impressions of the Recap website.

	1	2	3	4	5	6	7		
annoying	○	○	○	○	○	○	○	enjoyable	1
not understandable	○	○	○	○	○	○	○	understandable	2
creative	○	○	○	○	○	○	○	dull	3
easy to learn	○	○	○	○	○	○	○	difficult to learn	4
valuable	○	○	○	○	○	○	○	inferior	5
boring	○	○	○	○	○	○	○	exciting	6
not interesting	○	○	○	○	○	○	○	interesting	7
unpredictable	○	○	○	○	○	○	○	predictable	8
fast	○	○	○	○	○	○	○	slow	9
inventive	○	○	○	○	○	○	○	conventional	10
obstructive	○	○	○	○	○	○	○	supportive	11
good	○	○	○	○	○	○	○	bad	12
complicated	○	○	○	○	○	○	○	easy	13
unlikable	○	○	○	○	○	○	○	pleasing	14
usual	○	○	○	○	○	○	○	leading edge	15
unpleasant	○	○	○	○	○	○	○	pleasant	16
secure	○	○	○	○	○	○	○	not secure	17
motivating	○	○	○	○	○	○	○	demotivating	18
meets expectations	○	○	○	○	○	○	○	does not meet expectations	19
inefficient	○	○	○	○	○	○	○	efficient	20
clear	○	○	○	○	○	○	○	confusing	21
impractical	○	○	○	○	○	○	○	practical	22
organized	○	○	○	○	○	○	○	cluttered	23
attractive	○	○	○	○	○	○	○	unattractive	24
friendly	○	○	○	○	○	○	○	unfriendly	25
conservative	○	○	○	○	○	○	○	innovative	26

13,687/18,957