



2024/2025

**ERUDITE**  
THE COLLABORATIVE IVLE

CHRISTOPHER FAGG (210189160)  
DR CHLOE BARNES  
[Company address]

**Declaration:**

I declare that I have personally prepared this assignment. The work is my own, carried out personally by me unless otherwise stated and has not been generated using Artificial Intelligence tools unless specified as a clearly stated approved component of the assessment brief. All sources of information, including quotations, are acknowledged by means of the appropriate citations and references. I declare that this work has not gained credit previously for another module at this or another University.

I understand that plagiarism, collusion, copying another student and commissioning (which for the avoidance of doubt includes the use of essay mills and other paid for assessment writing services, as well as unattributed use of work generated by Artificial Intelligence tools) are regarded as offences against the University's Assessment Regulations and may result in formal disciplinary proceedings.

I understand that by submitting this assessment, I declare myself fit to be able to undertake the assessment and accept the outcome of the assessment as valid.

## Abstract

**TODO!**

**Keywords:** topic1, topic2, topic3, topic4, topic5.

## List of Acronyms

<i>AWS</i>	Amazon Web Services
<i>CD</i>	Continuous Deployment
<i>CI</i>	Continuous Integration
<i>CLI</i>	Command Line Interface
<i>CMS</i>	Content Management System
<i>DNS</i>	Domain Name System
<i>EC2</i>	Elastic Cloud Compute
<i>ERB</i>	Embedded Ruby
<i>FOI</i>	Freedom of Information
<i>HE</i>	Higher Education
<i>HESA</i>	Higher Education Statistics Agency
<i>M365</i>	Microsoft 365
<i>MVC</i>	Model-View-Controller
<i>ORM</i>	Object-Relational Mapper
<i>OS</i>	Operating System
<i>RoR</i>	Ruby on Rails
<i>UX</i>	User Experience
<i>VLE</i>	Virtual Learning Environment
<i>SDK</i>	Software Development Kit

## Dedications

**TODO**

## Table of Contents

<b>DECLARATION:</b>	<b>1</b>
<b>ABSTRACT</b>	<b>2</b>
<b>LIST OF ACRONYMS</b>	<b>3</b>
<b>DEDICATIONS</b>	<b>4</b>
<b>TABLE OF FIGURES</b>	<b>7</b>
<b>TABLE OF TABLES</b>	<b>7</b>
<b>KEY</b>	<b>8</b>
<b>BACKGROUND RESEARCH / CONCEPTION/ANALYSIS</b>	<b>9</b>
EXISTING PRODUCTS	10
<i>Key Success Factors</i>	12
<b>PROJECT MANAGEMENT</b>	<b>13</b>
GITHUB PROJECTS	13
LUCIDCHART	14
BALSAMIQ CLOUD	14
CI/CD	15
SQLITE	16
<b>TOOLING BREAKDOWN</b>	<b>18</b>
<i>IPassword</i>	18
<i>Docker</i>	18
<i>Kamal</i>	18
<i>GitHub Actions</i>	19
<i>GitHubCLI</i>	20
SOFTWARE LIBRARIES	20
<i>Ruby on Rails 8</i>	20
<i>Vite-Rails</i>	27
HARDWARE	28
<i>Digital Ocean Droplet VPS</i>	28
<b>CONSTRUCTION</b>	<b>29</b>
ARCHITECTURE OF RELATIONSHIPS	29
<i>Course Structure</i>	29
<i>Message Model and Polymorphism</i>	30
EVOLUTION OF ROUTES	32
EVOLUTION OF CONTROLLERS	33
<i>Authentication</i>	33
<i>Migration to namespaces</i>	34
UI EVOLUTION	36
<b>TESTING</b>	<b>37</b>
ROUTE TESTING	37
ERB TESTING	37
<b>DEPLOYMENT</b>	<b>37</b>

ARCHITECTURE/RESOURCE DIAGRAM .....	37
<b>EVALUATION / MAINTENANCE .....</b>	<b>37</b>
<b>FURTHER WORK .....</b>	<b>37</b>
<b>BIBLIOGRAPHY .....</b>	<b>39</b>

## Table of Figures

Figure 1 - VLE Market Share .....	10
Figure 2 - Student Focused NoSQL Example .....	16
Figure 3- Messages NoSQL with Referencing.....	17
Figure 4 - Student NoSQL with Referencing.....	17
Figure 5 - Kamal Diff for Deployment .....	19
Figure 6 - Turbo Frame Implementation Example .....	22
Figure 7 - Implementation of Navigation Bar Clock .....	23
Figure 8 - HTML Navigation Bar Clock Uses .....	23
Figure 9 - Instance Two of Stimulus-based Clock .....	23
Figure 10 - Instance One of Stimulus-based Clock .....	23
Figure 11 - Example Migration Creating Enrollments .....	24
Figure 12 - Example Schema with Enrollment .....	24

## Table of Tables

Table 1 - resources verbs per Rails Guide .....	32
---	----



**Key**

**Reference**

**Expansion/Rephrase**

**Cross-Reference**

**Acronym**

**Code**

## Background Research / CONCEPTION/ANALYSIS

During the 2022/2023 academic year there were 660,420 individuals (*Who's studying in HE? | HESA, 2024*) who enrolled in a UK-based institute of higher education to commence the pursuit of their first degree. Such a commitment occurs worldwide and has been globally suggested to have the potential to negatively impact various aspects of a students' life, including: their mental health (Levine, Tabri and Milyavskaya, 2023), economic position (Aina *et al.*, 2022), and their social standing (Thompson, Pawson and Evans, 2021). As such, it can be proposed that the adaption to HE learning paradigms and routines stands to be more stress inducing than it otherwise would be as a result of the sum of itself and external stressors.

Due to this, new students are likely to imitate the behaviour of other students due to an implicit form of peer pressure (Laursen, 2013), coupled with the suggestion that frequent, complex behaviour can cause habit formation anywhere from “weeks” to “months” (Buyalskaya *et al.*, 2023), and that this maximum bound is close to the average term length at a UK-based institute of HE (Jackson, 2025), it is probable that the existing gap in study-habits between pre-university and university level education will be filled with the habits of the masses, rather than the habits most beneficial to the individual student.

From an empirical perspective, summarising these habits is complex due to varying methodological limitations, analysis of which falls outside the scope of this report, however, it can be said that in recent years students have been forced to adapt their working style due to external influences. Mainly, the impact of the COVID-19 lockdown in the UK (*Prime Minister's statement on coronavirus (COVID-19): 23 March 2020, 2020*) as all educational institutes were forced to adopt an online learning model as of March 2020 (Hubble and Bolton, 2021). In doing so, it became a necessity for students to utilise online resources for their education, including a newfound dependency on e-mail for communication, their institutions VLE for content, and digital facilities for notetaking as compensatory measures for the mandated absence of in-person discussion and library access. For HE institutions in the UK, the sanctions remained in some form until 2021, where select students could return on March 8<sup>th</sup>, with the majority returning on July 19<sup>th</sup> (Lewis, Bolton and Hubble, 2021), as such, the majority of students experienced ~16 months of online education. As a result, a number of students claimed they lost their ability to effectively study, suffered a reduction in teaching quality, and experienced a lack of support (McGivern and Shepherd, 2022).

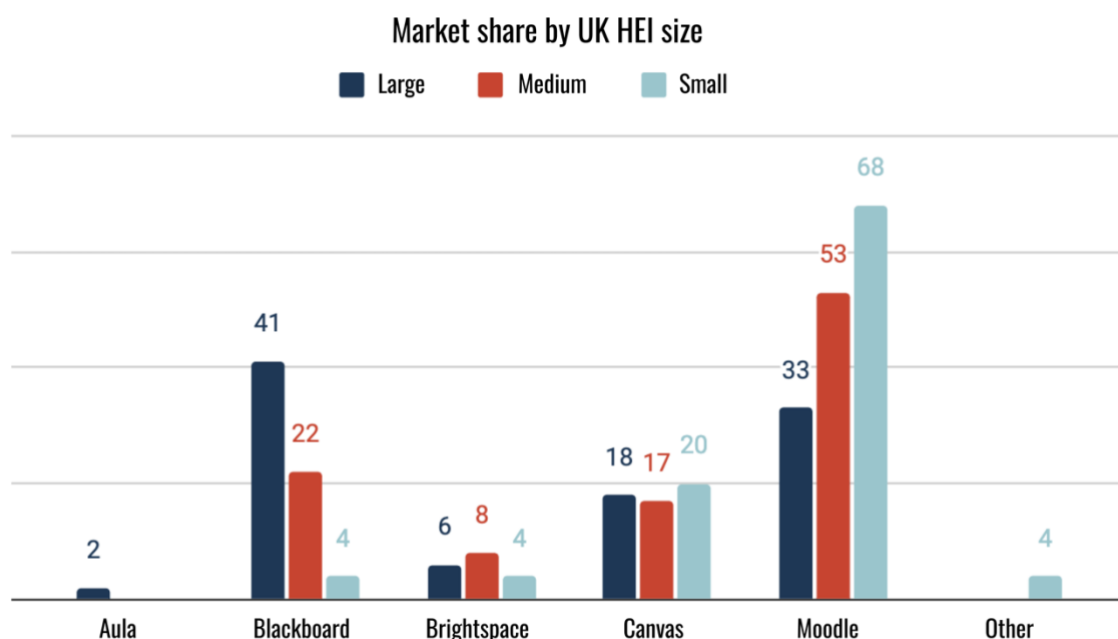
Despite the negative reception of online learning outlined by McGivern and Sheperd (2022), the results offered by Laursen (2013) and those by Buyalskaya *et al* (2023) enforce an idea that students and staff who experienced HE throughout lockdown may have retained these habits as a result of digital learning, unconsciously modelling them for post-COVID students to imitate. Thus creating a common study paradigm centred around digital technology, where individuals are “finding and making good uses of digital technologies that ‘work best’” for them in context, despite these uses not being “the most expansive, expressive, empowering, enlightening [...] ways that digital technologies *could* be used” (Henderson, Selwyn and Aston, 2017).

Recent market shares regarding technologies used by HE institutions and student opinion lends credence to the perspective that the modern learning experience in a digital zeitgeist is one of expected and accepted inefficiency, as there are a minimal number of viable, more efficient offerings. According to HESA there are 162 institutes of HE (*Where do HE students study? | HESA, 2024*), of which, 134 have publicly visible email addresses for FOI requests (Burgess, no date). Analysis of the respective DNS records, namely, against their Mail Exchange (MX) records, Autodiscover CNAME, and Sender Policy Framework (SPF) suggests that ~93% of these are associated with Microsoft in some form. **[Find this code for the appendix]**

While it is a technical possibility for these institutions to be accessing Microsoft servers outside of an M365 Education subscription, the associated cost and complexity of doing so makes this improbable, thus, it is the likely case that most institutions have adopted Microsoft 365 for Education, which means by default, students will be encouraged to utilise the Microsoft Office Suite, OneDrive, and Outlook by their institution.

## Existing Products

While the data suggests Microsoft holds a monopoly over the communicative tools used within the sphere of UK universities, this is not the case for the learning environments used. A consultancy report in 2023 suggested that Moodle is the most used VLE out of 247 institutes of higher education in the UK. (*2023 Review: What's the state of the VLE market in UK higher education?*, no date). However, this data considers all institutes of higher education, not just universities. The chart shown below is the representation of the data with consideration given to institution size and percent ownership per size, with large, medium, and small being categorised as greater than or equal to 15,000, between 15,000 and 5,000, and below 5,000 students respectively.



**Figure 1 - VLE Market Share**

Based on the UK having 162 universities, only 39 have a student population considered medium and 44 have a student population considered small. With this consideration it becomes more appropriate to suggest that within UK universities, it is Blackboard that holds the highest adoption numbers as there are 79 universities considered large by these metrics, of which blackboard has a 41% market share. Followed by Moodle and then Canvas. The exact factors contributing to adoption fall outside the scope of this report, though, it is worth noting Moodle's popularity across the board will be assisted by it being the only free system to implement and maintain as it is open source.

To begin designing Erudite, comparisons were drawn between Moodle, Blackboard and Canvas with attention being given to their similarities and differences regarding their student **UX** and functionality. For this, access to the sites respective administrator and teacher spaces proved unattainable so the comparisons are drawn exclusively from what a student user would experience with regular usage.

When using these apps all three offer an experience akin to that of a file-explorer or CMS with a litany of links available to various places with no emphasis on making use of the content within the bounds of the website. This is evident when the homepages are considered, figures \_\_\_\_ demonstrate how there is a noticeable absence of insightful content on the homepages though there are main navigation links to other parts and a schedule which the student would likely already be aware of due to the typically uniform nature of university timetabling.

### [HOMEPAGE FIGURES]

Furthermore, the pages responsible for displaying relevant content such as lecture slides are often nested through a series of non-uniform directories meaning accessing something as fundamental to the learning experience as lecture slides is not a habit a student is able to build. It becomes an almost gamified process of clicking into named directories with no guarantee the desired content is available. In some cases [BLACKBOARD FIGURE], it was found that the top-level directories in this process were not named with an indication of included content, rather, a week number. While convenient for chronological organisation, this acts to the detriment of everyone involved as without explicit memory of which information is associated with each week it becomes difficult to find on short notice. This weakness can extend to staff members too as it is normal for members of staff to have responsibility over numerous modules at any one time.

In addition, all three sites offer some form of ‘discussion board’ to permit discourse between students and faculty, while brilliant in theory, they are placed away from the source of content or disabled by default. Meaning, students would not feel inspired to utilise this as while communication is important, there are habitual forms of communication they could use instead such as Discord or Email. The visual disconnect, the knowledge other students often do not frequent those parts of the website for discussion and the lack of control as to whether they consistently have access to it means there is no drive to use it.

With these things considered a questionnaire was formed and distributed via social media to gather the opinions of post and current students on what they feel is performs well and requires improvement in the VLEs they have exposure with and what features they would value being implemented.

In total, the questionnaire received 16 responses offering an insight into the current uses and desires from students in their VLE.

Of the questions, 69% of respondents found their current VLE easy to use, though over 40% of respondents made some reference to information being cluttered, disorganised and varying between topics. Alongside this, 94% of participants responded confirming their VLE has built-in communication features, despite 50% then continuing to say they would rather book an office hour or send an email than make use of it. From this, they were asked how they access their emails, which returned a 94% response rate of either the Outlook application or the website equivalent with 42% confirming it is due to it being the default, supporting the earlier claim that students develop habits based on that which is provided to them.

When queried about whether the standard method of leaving a VLE to send an email is something they considered inefficient there was a 75% response rate in agreement. From this, 94% agreed that if they were able to directly ask staff for assistance without leaving the platform they would do so. It was also found that there is diversity associated with how the participants currently note-take, with 50% confirming pen and paper with the rest spread between various digital platforms such as Microsoft Office or Obsidian. Interestingly, the majority of responses associated with why they note-

take this way included it being an effect of habit and ease of use, further suggesting students develop habits throughout their schooling and rarely deviate.

[MAYBE ADD OBSERVATION INFORMATION?]

### Key Success Factors

The assessment of current offerings and the information gathered via the questionnaire formed the basis of the requirements for Erudite to be considered a successful project. Such a method was deemed appropriate as features consistent between the most popular VLEs could be considered as ‘fundamental’ for an application to be considered a virtual learning environment.

Said features were included alongside those deemed by students to be useful but absent from their current VLE to form the functional requirements dictating whether or not Erudite was a project success.

### Functional Requirements

User Type	Requirement
All	Will have their requests validated to ensure they have permission to access requested resources
	Will have their create/update/delete requests validated to ensure data integrity
	Will have access to their own profile information
	Will have access to <i>visible</i> content per their enrolments
	Will be able to send and receive messages on relevant pages
Admin	Will have CRUD for everything within the application
	Will be able to view an audit of User updates
	Will be able to view an audit of Topic updates
Staff	Will have CRUD access to pages they’re listed owner of
	Will be able to view an audit of Topic updates they’re owners of
	Will be able to upload PDFs for content display
Student	Will have access to a privacy feature meaning they are not directly contactable by other students

[ADD NOTEBOOK THING 2 BOTTOM OF ALL IF ADDED]

### Non-Functional Requirements

Requirement
All pages should load within 2 seconds
The project should be easily deployable for self-hosting or downtime fallback on major cloud providers
All traffic should be sent over HTTPS
The project should work consistently between the latest browser versions

## Project Management

Overall, Erudite is a product of the Waterfall approach (Petersen, Wohlin and Baca, 2009) to development, as a result, development was dependent the completion of stages before it. These stages being conception, analysis, design, construction, testing, deployment, and maintenance. With the notable exception of deployment, which was introduced throughout the construction phase, the intention of which is further explained within CI/CD. While atypical, this was done to better align with the development methodologies used within each waterfall phase.

Contained within each stage of development an attitude akin to the philosophy of “fail fast” was adopted, whereby core parts of that stage are rapidly implemented to a minimum functional standard. Inspired by the Agile Manifesto (*Manifesto for Agile Software Development*, no date), the work is reviewed for functionality and a decision is made regarding whether the code needs altered functionality before anything else is built onto it. The core motivation to “fail fast” is avoiding sunken-cost fallacy where resources are spent fixing a project that will likely not be used regardless. Accepting that there are multiple ways to implement something and choosing the most appropriate helps avoid unnecessary delays, this was especially important due to the deadline present against the project.

This approach of development in linear stages, with each stage being expected to iterate through rapid implementations was utilised due to this project being novel in nature while retaining several expected functions that would act as pillars to the project success.

## GitHub Projects

The source code for Erudite in production is stored, deployed and maintained through a GitHub Repository, consequently, all changes made to the source code can be viewed through commit histories and pull-requests.

Appropriately, the decision was made to use GitHub’s *Projects* offering as the primary project management platform. As seen in the official documentation “A project is an adaptable spreadsheet, task-board, and road map that integrates with your issues and pull requests on GitHub to help you plan and track your work effectively.”(*About Projects*, no date) , this permits the manipulation of data into multiple viewing formats without the need for porting information to more than once place. Meaning, if projects within a kanban board are supplied sufficient date constraints, they can be rendered into a Gantt chart to track progress in relation to time without any added effort manually transforming the data.

There are alternatives to GitHub projects for project management such as Trello, Jira, and Basecamp, all of which have been successfully implemented to manage projects that far exceed the complexity and scale of Erudite. As such, any of these platforms would have been suitable for use throughout the development lifecycle. In this project, GitHub Projects was chosen as it does not lack any required features offered by competitors while also being able to directly reference pull-requests and other metrics within the main repository.

## LucidChart

Erudite's data modelling has been subject to breaking changes throughout development due to the introduction of associations between models to ensure a logical flow of information within the application. As such, planning the intended data-flow between objects became essential to mitigate the risk of project delays due to a major design flaw in logic.

Said plans were visualised by LucidChart, a freemium platform used by major corporations such as Nvidia and Atlassian. This choice was made as while there are limits to the LucidChart's free-tier, these would not impede use of the platform for the intended purposes. From this, it was possible to use an enterprise-scale diagramming platform with the relevant industry standard iconography to create and maintain database diagrams, ensuring consistency in project artefacts.

These diagrams were then ported to the relevant GitHub Issue assigned to the Erudite GitHub Project, coupled with the relevant commits containing changes to `schema.rb`. This allowed a continuous visualisation of database development throughout the project.

## Balsamiq Cloud

Throughout the design phase of this project Balsamiq Cloud was used to create and maintain the wireframes that formed the visual prototypes and what the final implementation aims to be. The decision was made to use this platform over alternatives such as LucidChart and Figma as while all could perform the desired tasks, the design and experience of erudite has and always will be intended to be highly usable and visually clear, so the advanced complexities of alternatives compared to Balsamiq would go under-utilised while the increased complexity around performing basic tasks would offer an unnecessary learning curve when compared to the drag & drop components offered by Balsamiq.

This decision is further backed by the literature, with the absence of a measurable correlation between time spent prototyping and overall project success (Yang, 2005) and rapid prototyping being suggested to negate fixation (Viswanathan and Linsey, 2013). This is a positive considering the perceived absence of a significant relationship between the temporal investment into a prototype and its perceived value (Nelson and Menold, 2020). Collectively, adhering to the conclusions above allows the design of Erudite to be minimally impacted by the sunken-cost fallacy, a key intention outlined in Project Management

## CI/CD

As mentioned in Project Management, continuous integration and deployment was introduced earlier in the project cycle than it would have done had Erudite been developed under a strictly waterfall approach. The development of Erudite aimed to follow industry patterns, as such, a primary branch titled `main` was maintained with the intention of always being deployable while development branches were used to implement and debug new features.

The systems used to integrate this, and the overarching configuration is expanded upon in Docker, GitHub Actions, and **Error! Reference source not found.** This section is explicitly concerned with the motivations and shortcomings of such implementation.

Since `main` intended to maintain a state of functionality, it was chosen as the deployment source with the intention of being available for passive testing through use and demonstration purposes for interested parties and the project supervisor. Throughout development changes were made on a branch forked from `main` and then once completed, integrated back into `main` via a pull request. These pull requests then triggered the redeployment of the Erudite `main` branch with new changes integrated.

This also ensured that should anything go catastrophically wrong during development, there was always a functional commit the project could be reverted to which would act as an efficient checkpoint compared to sifting through development history via `git bisect` or an equivalent tool.

However, there were limitations to this method. Mainly, the absence of a ‘staging’ environment between ‘development’ and ‘production’. While deployment is powered by Docker, development was not done within a Docker container. Thus, if there was an addition to Erudite that necessitated an update to the Environment and this failed to reflect within the Dockerfile then the deployment build would fail. When this did occur, there was no down-time to the current site due to the presence of Kamal-proxy, responsible for ensuring the new build is healthy prior to directing traffic to it and stopping the old container. Though this did not solve the deployment problems associated with not having a ‘staging’ environment or containerised development to begin with.

An in-depth explanation of Kamal-proxy and its function can be found within Docker.



## SQLite

During the conception period of Erudite it was found that the project would contain a sizable amount of relational data, ranging from the tracking/retention of session information per user to the degree courses contained and their associated sub-content including units, topics, and assignments. This level of relational data was a core motivator for the choice to use an SQL database such as SQLite or PostgreSQL over a NoSQL database such as MongoDB.

While it may have been feasible to integrate a NoSQL equivalent database, the related nature of the data would force the document-based storage into a significant level of nesting. For example, if a NoSQL equivalent had top-level documents focusing on a **Student** model, the nest to access the **Messages** associated with an assignment would be similar to Figure 2.

This would quickly become unmanageable if the course data was to continue to grow or support for new features such as notification preferences was integrated against the **Student** model.



```
1 {
2   "student_id": "123",
3   "enrolled_course": {
4     "course_id": "C001",
5     "name": "BSc Computer Science",
6     "units": [
7       {
8         "unit_id": "U001",
9         "name": "Data Structures and Algorithms",
10        "assignments": [
11          {
12            "assignment_id": "A001",
13            "title": "Reversing Linked Lists",
14            "messages": [
15              {
16                "message_id": "M001",
17                "sender": "John Doe",
18                "text": "I need help with question 3",
19                "timestamp": "2025-04-03T12:00:00Z"
20              }
21            ]
22          }
23        ]
24      }
25    ]
26  }
27 }
```

**Figure 2 - Student Focused NoSQL Example**

Figure 3 and Figure 4 demonstrate the NoSQL support for referencing, which would permit identical function to Figure 2 with less nests per document.

```

● ● ● Student NoSQL w/Reference
1 {
2   "student_id": "123",
3   "enrolled_courses": {
4     "course_id": "C001",
5     "name": "Math 101",
6     "units": [
7       {
8         "unit_id": "U001",
9         "name": "Algebra",
10        "assignments": [
11          {
12            "assignment_id": "A001",
13            "title": "Homework 1"
14          }
15        ]
16      }
17    ]
18  }
19 }

```

**Figure 4 - Student NoSQL with Referencing**

```

● ● ● Message NoSQL w/Reference
1 [
2   {
3     "_id": ObjectId("6618f8e9a1b2c45e9c6d1234"),
4     "message_id": "M001",
5     "assignment_id": "A001",
6     "sender": "John Doe",
7     "text": "I need help with question 3",
8     "timestamp": "2025-04-03T12:00:00Z"
9   },
10  {
11    "_id": ObjectId("6618f8e9a1b2c45e9c6d5678"),
12    "message_id": "M002",
13    "assignment_id": "A001",
14    "sender": "Teacher",
15    "text": "Try using the quadratic formula.",
16    "timestamp": "2025-04-03T12:05:00Z"
17  }
18 ]

```

**Figure 3- Messages NoSQL with Referencing**

It would be possible to remove the nests entirely, introducing a table per object. However, NoSQL fails to enforce integrity across the references by default. As such, what is native to SQL via foreign key constraints requires manual implementation and validation in a NoSQL environment. This integration, while matching the needs of the project, would effectively be a re-implementation of a relational database in breach of the paradigms associated with the technology used.

## Tooling Breakdown

### 1Password

Security considerations are an integral part of building a web application. Such considerations range from ensuring only necessary ports are exposed on hardware to keeping privileged machines safe from theft. Arguably, the most important security consideration in modern-day web development is credential storage and injection. Published by Verizon Business, the 2024 Data Breach Investigations Report details the origin of various forms of data breaches, suggesting that within the past “10 years, stolen credentials have appeared in almost one-third (31%) of breaches” (Hylender *et al.*, 2024). While this report goes on to discuss the ways in which individuals acquire these details, it is often the case that developers make them publicly available without the need for malicious parties to have to search for them. GitGuardian, a company responsible for detection and management of secret leaks within codebases released a report defending the claim that these secrets can end up publicly available. The State of Secrets Sprawl 2025 details that throughout 2024 there were 23,770,171 secrets detected within public GitHub commits in 2024, with 70% of valid secrets found in 2022 remaining active in 2024.

Clearly, secrets exposure is a problem when it comes to development, likely down to the fact software developers are not security experts. Due to this, throughout the development of Erudite all environment variables, SSH Keys, and encryption keys were stored, managed and injected by 1Password to ensure no leaks from poor management. 1Password’s services were leveraged via the local application, their GitHub Actions library, and the Kamal extension with support for password manager extraction.

While there is a debate surrounding the reliance on a third-party for security, in this instance there is little credence to the debate as “150,000 businesses trust 1Password” (*Password Manager & Extended Access Management | 1Password | 1Password*, no date).

### Docker

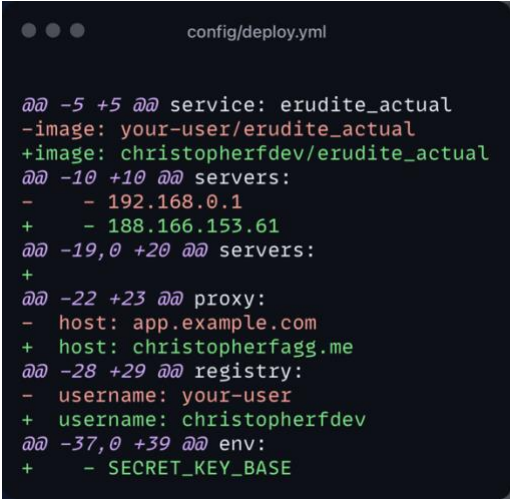
In the context of Erudite and its development, Docker was leveraged as a tool to facilitate deployment via Kamal. Fundamentally, Docker permits the separation of applications and the infrastructure they are built on to make the development process more efficient (*What is Docker?*, no date). It does this by sharing the host computers operating system kernel but within a separate user-space, this allows Docker to exclusively run based on the instructions for system configuration as provided by the Dockerfile. This happens without the overheads associated with virtual machines due to effectively piggybacking the host hardware and OS so is often used to create replicable environments for applications to run within.

Erudite is hosted and maintained on a Digital Ocean Droplet running Ubuntu LTS, however, it could be deployed on any Linux distribution and run in an identical environment due to the containerisation of Docker. Furthermore, Ruby on Rails ships by default with a Dockerfile that supports a standard project running in the ideal environment which made it a sensible choice for use. As a result, the only modifications to the Dockerfile were the libraries necessary to support the integration of Vite and Vue. This assisted in debugging build failures as the logic was isolated to the Vite/Vue additions due to Rails shipping in a buildable state for a production environment.

### Kamal

Kamal is an SSH based, container-powered deployment tool that “offers zero-downtime deploys, rolling restarts, asset bridging, remote builds” (*Deploy web apps anywhere*, no date) and a selection of other tools to enable web app deployment to most environments. The decision to implement Kamal into Erudite was made as Ruby on Rails provides support for it by default, with the relevant files existing in

a boilerplate project meaning minimal configuration is required. The configuration required was simply the addition of an IP address of the server, the proxy address and the container registry username. Kamal offers support for various container registries with a default of DockerHub, which was used for Erudite. Figure 5 shows the diff output between the boilerplate Rails project and the additions that permitted the first deployment of Erudite.



```
config/deploy.yml

@@ -5, +5 @@ service: erudite_actual
- image: your-user/erudite_actual
+ image: christopherfdev/erudite_actual
@@ -10, +10 @@ servers:
-   - 192.168.0.1
+   - 188.166.153.61
@@ -19,0 +20 @@ servers:
+
@@ -22, +23 @@ proxy:
-   host: app.example.com
+   host: christopherfagg.me
@@ -28, +29 @@ registry:
-   username: your-user
+   username: christopherfdev
@@ -37,0 +39 @@ env:
+   - SECRET_KEY_BASE
```

**Figure 5 - Kamal Diff for Deployment**

Kamal effectively does 11 things on behalf of the developer (*Installation*, no date):

1. Connects to the listed server via SSH
2. Installs Docker if not already present
3. Authenticates against DockerHub
4. Builds the project against the source Dockerfile
5. Pushes said image to DockerHub
6. Pulls the newly hosted image from DockerHub to the server
7. Ensures kamal-proxy is running on the relevant ports
8. Starts a new container of the app based on the most recent Git hash
9. Instructs kamal-proxy to route traffic to the new container once it returns a healthy status
10. Stops the old container running the now outdated version
11. Prunes obsolete images and containers to maintain space

As a result of this process, once deployed the site will not suffer downtime due to a failed deployment. Should the newest version fail to build, traffic is not routed there.

This was an incredibly efficient choice for Erudite as services such as Vercel that offer default deployment pipelines via a Git repository were unsuitable for use based on their incompatibility with Rails outside of its API mode. Had Kamal not been used, deployment would have likely been managed by Capistrano, an SSH manager that performs similar SSH commands to Kamal but lacks the proxy and cleaning methods without extensive manual configuration.

### GitHub Actions

Following on from CI/CD, GitHub Actions was used as the deployment base for triggering Kamal. While there are many alternatives available for CI/CD pipelines, namely, Jenkins and GitLab, Erudite's source code is stored and maintained via GitHub which labels GitLab unusable without migrating the entire source code to their systems. Jenkins could have been used as it is an established, mature, and

well-maintained open-source option, but was decided against due to its self-hosted nature and steep learning curve. Due to the low complexity of Erudite's structure as a direct result of utilising tools native to Rails, the extent of the requirements needed for the CI/CD software effectively became the ability to trigger the `kamal deploy` command and accept authentication environment variables.

Based on the above, the most logical decision was to undertake the learning curve associated with GitHub Actions as configuring it to trigger based on behaviour in the Erudite repository would be a simpler effort than alternatives.

### GitHubCLI

As Erudite was a solo-developer endeavour most interactions with GitHub were done via the GitHub API CLI tool and the Git command line. This was done as the use of Git was relatively straightforward with a large portion of interactions being standard commits, branching and pushing.

While tools such as GitTower, GitHub Desktop and GitKraken are available their use could be considered overkill in this context. Made more relevant when exposure prior to the development of Erudite is considered. Using these tools would have included learning these GUI tools for the first time compared to leveraging prior exposure to the available CLI tools

## Software Libraries

### Ruby on Rails 8

Ruby on Rails, commonly referred to as Rails, is an MVC based web-framework powered by Ruby, with a motto of "Compress the complexity of modern web apps" (*Ruby on Rails*, no date). Rails prides itself on being incredibly powerful and straightforward to work with, championing the phrases "Provide sharp knives" and "Convention over Configuration" (*The Rails Doctrine*, no date). Meaning, there is nothing in the framework that developers are forbidden from using while offering strong conventions that naturally stop developers causing breaking changes due to misuse. As a result, if said conventions are adhered to it is possible to do incredibly powerful things with very little effort, making Rails an incredibly accessible and useful framework for both novice and experienced developer.

Furthermore, the first instances of Rails were concerned with making integrated systems, referred to by the founder as "Majestic monoliths" (*The Rails Doctrine*, no date), allowing Rails to be responsible for the entirety of an application from databases to front-end content display and everything in between. Constructed with the intention "to equip generalist individuals to make these full systems. Its purpose is not to segregate specialists into small niches and then require whole teams of such in order to build anything of enduring value."

Being Ruby based, Rails also carries support for third-party libraries referred to as 'gems', these offer added functionality to applications not provided by default. These additions often allow the developer to avoid manual configuration. For example, the inclusion of gem `aws-sdk-s3`, a wrapper enabling easier interaction with the AWS S3 service.

Erudite likely could have been built using other frameworks, the most likely competitor being Laravel based on its similar MVC model. However, there would have come a notable moment in development where technology such as an ORM for database interaction or native support for file-control would be lacking, swaying the architecture into a spread of microservices. Whereas Rails carries support for ActiveRecord and ActiveStorage by default, meaning Erudite does not have to rely on a third-party or juvenile self-built library for core functionality.

Due to Rails core motivations being to empower singular developers to develop a full application at scale, it was the logical choice for Erudite given Erudite is a full application requiring support for a myriad of features. The following sub-sections discuss choices made within the Rails eco-system, heavily demonstrating the benefits of adhering to the conventions under the “majestic monolith”.

### **Turbo**

As Erudite is intended to be used throughout an individual’s experience at university, the user experience of the platform is integral to project success. Part of this experience is perceived load times, with the common reference being load times should be between 0.1 and 1 second to retain engagement (*Response Time Limits: Article by Jakob Nielsen*, no date). As JavaScript has developed, single-page applications became popular, as these only load a single web document, it offers performance gains over conventional websites. (*SPA (Single-page application) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*, 2024). However, Erudite is not reliant on JavaScript, so such an implementation is not viable.

Described as “several techniques for creating fast, modern, progressively enhanced web applications without using much JavaScript”, (*Turbo Handbook*, no date) turbo aims to offer “The speed of a single-page web application without having to write any JavaScript.” (*Turbo: The speed of a single-page web application without having to write any JavaScript.*, no date). Erudite leverages three of these techniques: Turbo Drive, Turbo Frames, and Turbo Streams.

### **TurboDrive**

If websites are viewed as documents, standard navigation entails a requestor and a responder with the former being the current page and the latter being the desired page. Typically, the responder would replace the requestor, triggering an entire page reload while re-downloading all assets and re-executing all JavaScript regardless of whether these assets are at all different.

Turbo Drive takes this process and streamlines it in multiple ways. Firstly, Turbo Drive pre-fetches the responder page when a clickable link is hovered on acting to obfuscate any form of network latency. Due to the speed of modern networks, the page is often cached before the user clicks. Drive then selectively updates the page with the response, merging the `<head>` tags and replacing the `<body>` content while preserving the loaded assets from the initial page load.

Negating the need for full-page reloads means there is less data transferred per request after the first due to caching, ergo, the average speed will reduce, paralleling how single page applications offer faster navigation between pages via loading only immediately required resources.

### **TurboFrames**

In pursuit of more efficient navigation, Turbo offers Turbo Frames, a way to exclusively update specific aspects of a page without requiring the server to respond to a request with a full document. Turbo Frames form much of the navigation in Erudite, with the key navigation links targeting a main, central turbo frame visualised in UI Evolution. While Turbo Drive ensures that aspects of a web page such as the loaded CSS/JS are not unnecessarily reloaded on every page request, Turbo Frame permits developers to exclusively update content within a pre-defined `<turbo-frame>` with the use-case of scoping interaction to a specific section of the page.

Using Figure 6 as an example, the presence of the `<turbo_frame_tag>` means that when the contained link is pressed, the request response will populate between the `turbo_frame_tag` and `end`, leaving the `<h1>` unchanged.

A screenshot of a code editor window titled 'turbo\_frame.html.erb'. The code is as follows:

```
1 <h1> This Header Will Persist Forever </h1>
2
3 <%= turbo_frame_tag "frame_path" do %>
4   <%= link_to "Load Content", frame_path %>
5 <% end %>
```

**Figure 6 - Turbo Frame Implementation Example**

This contributes to the user experience as Turbo Frames update rapidly, surgically replacing exclusively the desired content, this effectively draws attention to the change and ensures that nothing outside of the Turbo Frame is impacted by the server response.

### Turbo Streams

Turbo Streams offer fragmented page updating by wrapping HTML in a `<turbo-stream>`. In Erudite, the most prominent uses are to permit real time messaging between users and a live notification feed for when individuals are mentioned within messages. While supported by HTML responses, Erudite leverages its asynchronous capability, utilising WebSockets as opposed to polling the database for new messages or alternatives. Typically, turbo streams are used in response to create, read, update, delete (CRUD) operations to show changes across instances of the application as opposed to forcing page refreshes. Specific to Erudite's use-case, they are used to populate messages as they are sent into a given topics associated message section.

### Stimulus

Building on the above, it is proposed that JavaScript is used in 99% of websites globally (*Usage Statistics of JavaScript as Client-side Programming Language on Websites, April 2025*, no date), with developers demonstrating a majority intention to continue working heavily with it (*Technology | 2024 Stack Overflow Developer Survey*, no date). This is often the case as databases return JSON and then one of many libraries will parse this JSON into a viewable webpage. However, a major reliance on JavaScript is not obligatory in web development, as is the case with Erudite for most of its functions and navigation relies on the Turbo offerings from Hotwire.

As such, Erudite implements Stimulus, coined "A modest JavaScript framework for the HTML you already have" (*Stimulus: A modest JavaScript framework for the HTML you already have.*, no date). Stimulus works on the inverse of JavaScript paradigms, using HTML as the source of truth, permitting interaction via augmentation rather than complete ownership and manipulation. As such, the behaviour in Erudite that requires JavaScript is implemented discreetly and powerfully, with minimal associated overhead. The exception here is given to the **Vue** components leveraged, though this is assessed in its sub-heading.

For example, the clock contained within the navigation bar is created and updated by JavaScript. While it could be implemented within a `<script>` element in the view file, this would look untidy, be hard to maintain and breach separation of concerns. In the case the same clock needs to be displayed elsewhere,



the code would either require refactoring or moving to a place it can be referenced globally. Whereas, with stimulus it can exist wherever the element target is defined.

```
clock_controller.js

1 export default class extends Controller {
2   static targets = ["display"]
3
4   connect() {
5     this.updateTime()
6
7     this.intervalId = setInterval(() => {
8       this.updateTime()
9     }, 1000)
10  }
11
12  // extraction of h/m/s from now omitted for space
13  updateTime() {
14    const now = new Date()
15    const h, m, s
16
17    this.displayTarget.textContent = `${h}:${m}:${s}`
18  }
19 }
```

**Figure 7 - Implementation of Navigation Bar Clock**

```
_navbar.html.erb

1 <div class="stat-value" data-controller="clock">
2   <span data-clock-target="display">00:00:00</span>
3 </div>
```

**Figure 8 - HTML Navigation Bar Clock Uses**

Figure 7 & Figure 8 demonstrate the relevant parts of how the clock functions. The stimulus controller looks for the `data-controller` attribute matching the start of the controller file name, in this case, `clock`. From here it seeks the target `display`, assigned to a child element of the `data-controller`, it then continues to set the content of the `display` attribute based on the logic in `updateTime()`.

Integrating Stimulus also makes expansion easier. Should Erudite need to display a clock elsewhere, the HTML from `_navbar.html.erb` can be extracted into a `_clock.html.erb` partial and rendered as seen in Figure 10 and Figure 9, maintaining adherence to the principle of “Don’t Repeat Yourself”. (Hunt and Thomas, 2011)

```
foo.html.erb

1 <%= render "clock" %>
```

**Figure 10 - Instance One of Stimulus-based Clock**

```
bar.html.erb

1 <%= render "clock" %>
```

**Figure 9 - Instance Two of Stimulus-based Clock**

### ***ActiveRecord***

Active record as a pattern is defined as “an object that wraps a row in a database table, encapsulates the database access, and adds domain logic to that data.” (Fowler, 2013). This is implemented in Rails’ Object Relational Mapper (ORM) as `ActiveStorage`. The intention of `ActiveRecord` is to minimise the amount of database specific access code a developer must write; in the case of Rails this is often SQL, due to the default configuration being SQLite. The presence of this ORM permits developers to interact



with objects by writing ruby code, rather than enduring the mental fatigue associated with switching between ruby and SQL. ActiveRecord has support for the following ruby transformations: representing models, representing associations between models, represent inheritance through relations, validate models prior to persistence, perform database operations in object-orientated fashion. (*Active Record Basics*, no date)

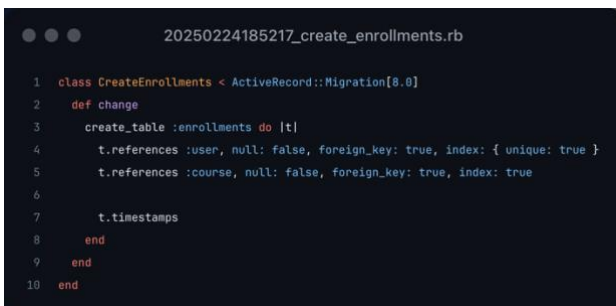
ActiveRecord is arguably the most integral part of Erudite as without any way to interact with, or create data, there would be no functions to the application. It also was and continues to be responsible for generating migrations that amend the database schema and all CRUD operations against the SQLite database.

In line with the Rails way of “convention over configuration”, there was little consideration given to alternative ORM’s prior to or throughout the development process. While they do exist, with alternatives such as `orm_adapter` and `sequel` being available, their usage within the community is significantly lower than that of ActiveRecord, with 224,302,381 and 58,259,959 downloads respectively compared to ActiveRecord’s 638,890,106. If ignorance is given to the idea that using an ORM different to ActiveRecord would be against one of the Pillars of Rails, ActiveRecord would likely still be the de-facto ORM choice based on the significantly higher level of users and the resultant higher frequency and standard of maintenance.

ActiveRecord has a range of capabilities, including, but not limited to database migrations, data validations and associations. All of which are explored in the relevant sub-sections.

## Migrations

ActiveRecord migrations are an incredibly powerful feature that facilitate the creation, amendment and deletion of databases and their associated records within a Rails project. Written in a domain specific language, this permits migrations to be database agnostic, meaning, they will run against any supported database including but not limited to: MySQL, PostgreSQL and SQLite (*Active Record Migrations*, no date). Alongside this, written migrations are persisted in the project directory under `db/migrate` prepended with the timestamp at time of creation with a `schema.rb` file being automatically maintained, allowing developers access to the database structure at any time while also allowing an effective log of changes. Thus, ActiveRecord is incredibly powerful regarding accurate database structure and maintenance.



```
20250224185217_create_enrollments.rb

1 class CreateEnrollments < ActiveRecord::Migration[8.0]
2   def change
3     create_table :enrollments do |t|
4       t.references :user, null: false, foreign_key: true, index: { unique: true }
5       t.references :course, null: false, foreign_key: true, index: true
6
7       t.timestamps
8     end
9   end
10 end
```

Figure 11 - Example Migration Creating Enrollments



```
schema.rb

1 ActiveRecord::Schema[8.0].define(version: 2025_03_25_174943) do
2   create_table "enrollments", force: :cascade do |t|
3     t.integer "user_id", null: false
4     t.integer "course_id", null: false
5     t.datetime "created_at", null: false
6     t.datetime "updated_at", null: false
7     t.index ["course_id"], name: "index_enrollments_on_course_id"
8     t.index ["user_id"], name: "index_enrollments_on_user_id", unique: true
9   end
10
11   # Table info for courses & users omitted
12   add_foreign_key "enrollments", "courses"
13   add_foreign_key "enrollments", "users"
14 end
```

Figure 12 - Example Schema with Enrollment

Alongside this, it will automatically generate the appropriate columns should associations be needed between two objects. For example, it is possible to create migrations that associate `Users` and `Courses` via `Enrollments`, with Rails navigating the creation and enforcement of foreign keys on behalf of the developer.

### Validations

As ActiveRecord falls under the Model aspect of MVC, its model system permits model-level validation, the Rails recommendation for most circumstances (*Active Record Validations*, no date), while Erudite does not solely rely on model level validations with consideration being given to client-side, database and controller level validation, validation at the model level helps ensure that validation logic is centralised to its related object which is essential for long-term growth and maintenance.

For example, the User model in Erudite has validations as shown in Figure 13 ensuring that a `User` object always has an email address unique within the database and both a first and last name.



```
user.rb

1 validates :email_address, presence: true, uniqueness: true
2 validates :first_name, :last_name, presence: true
```

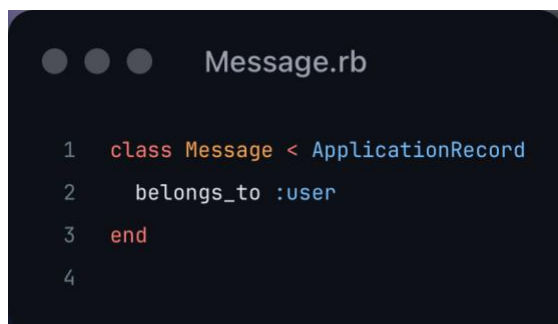
Figure 13 - Validations on User Model

These validations are essential for data integrity, as Erudite experienced rapid development from a data perspective. Such enforcements allowed a more efficient debugging experience as objects grew and associations changed.

### Associations

The presence of associations in Rails permits the normalisation of data and removes a significant amount of manual work required without it. In the example of Erudite, `Users` are encouraged to send `Messages`. In the absence of associations, the process of deleting all `Messages` sent by a `User` would involve looping through all `Messages`, extracting those who were authored by the `User`.

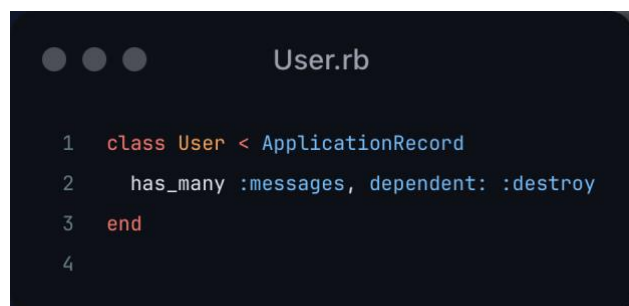
Rails offers a straightforward syntax to create an association, shown in Figure 14 & Figure 15.



```
Message.rb

1 class Message < ApplicationRecord
2   belongs_to :user
3 end
4
```

Figure 15 - Message Association with User



```
User.rb

1 class User < ApplicationRecord
2   has_many :messages, dependent: :destroy
3 end
4
```

Figure 14 - User Association with Message

From this, it then becomes possible to access `Messages` directly through the `User` object, rather than having to cherry-pick the author from all messages. The presence of `destroy` also ensures that should a `User` be destroyed; their `Messages` are too. Removing the computational overhead of having to loop every `Message` to destroy the relevant ones.

### *ActiveStorage*

As Erudite is fundamentally a content management and digestion site, emphasis was placed on the service used to host and maintain the documents uploaded. Continuing with the theme of “Majestic Monoliths”, the decision was made to leverage ActiveStorage, Ruby on Rails native service permitting the direct association of files against Models. ActiveStorage supports many `BLOB` storage systems including but not limited to `AWS` S3, Azure Storage and Google Cloud Storage, the choice was made to use an S3 bucket due to familiarity with the AWS eco-system compared to the other offerings.

Supporting local storage and major cloud service providers including `AWS` and Azure, ActiveStorage removes the need for the manual configuration that would otherwise be required to permit file uploads via a platforms SDK. Configuring this entailed creating the bucket via AWS services, adding the `aws-sdk-s3` gem to the projects Gemfile, adding the access credentials to Erudite’s environment and then referencing the details of both in `config/storage.yml`. Beyond this, ActiveStorage is responsible for architecting the code that permits the creation, removal and reference of stored files.

In addition, ActiveStorage permits file variations, allowing erudite to include features such as dynamically sized user avatars based on the page requesting them.

Similarly to ActiveRecord, there are well maintained alternatives to ActiveStorage, namely Shrine. The primary difference between the two projects exists within the actual relation it forms against the model to store attachments and Shrine allowing modular importing, permitting the developers to only include the parts of the project required for use. ActiveRecord relies on each attachment model having a `belongs_to` reference to the parent Model, thus, ensuring every access to the attachment is via a `JOIN` table. Shrine on the other hand, requires a new column in the parent Model keeping the relevant data within the Model table directly. As a result, Shrine technically performs better at scale due to the reduced computation required for file interaction, however, the granular approach to inclusion and the existence outside of the Rails eco-system naturally means the use of such would incur a significant learning curve. At this scale, pursuit of the learning curve would reap little tangible benefit over the use of ActiveStorage.

### *Thruster/Puma*

Within the deployed container, Erudite in production uses a combination of Puma and Thruster, a “Ruby based web server built for parallelism” (`puma/puma`, 2025) and a “HTTP/2 proxy for simple production-ready deployments of Rails applications”. (`basecamp/thruster`, 2025).

Maintaining the theme of “Majestic Monoliths”, both Puma and Thruster are the default for a Ruby on Rails project. There was minimal consideration given to alternatives when it came to application server as puma is “the most popular Ruby webserver” (`puma/puma`, 2025) with 443,605,613 downloads (`puma` | *RubyGems.org* | *your community gem host*, no date) and thruster “aims to be as zero-config as possible”. (`basecamp/thruster`, 2025).

While alternatives such as Nginx are capable and widely used, the deviation from the defaults offered by Rails would introduce added configuration requirement with a detriment to system cohesion as Puma/Thruster are compatible by default with Kamal and its attached kamal-proxy.

### ***Faker***

Due to the copyright and intellectual property rights associated with academic content, a seeding library was used for the default content used to demonstrate Erudite as external sourcing was not viable. Faker facilitates automatic data “such as names, addresses, and phone numbers.” (‘faker-ruby/faker’, 2025), this was used in Erudite’s `seeds.rb` file to ensure that throughout development and in production, a pre-populated database could be generated that would allow the app to function correctly.

The implementation permitted rapid availability of consistently high-quality faux data, permitting presentation akin to that of a production environment. Thus, demonstrating Erudite in a form close to intended rather than a litany of “*test user*”, “*test assignment*”, “*test unit*” etc.

### ***Vite-Rails***

Erudite leverages Vite.js via a `vite_rails` gem as a front-end bundler, in breach of the “Majestic Monolith” but with fair reason. Two of Erudite’s required functions are the ability to view an uploaded PDF and view a schedule from a calendar. The libraries native to ruby’s ecosystem proved lacking for this purpose, as such, Vue was integrated to permit the use of Vue Components.

Rails ships with import maps as default due to the belief they have the “potential for reducing complexity, improving developer experience, and delivering performance gains” (*Working with JavaScript in Rails*, no date) and while this remains a valid opinion, it is speculative in nature and not common-place in the industry. Therefore, there are more incompatibilities with import map use compared to traditional JavaScript bundling.

The gem `vite_rails` was chosen to enable Vite integration with Rails as Vite is framework-agnostic, with effective plug & play support for Vue, `vite_rails` permits plug & play support for Vite and Rails. As such, `vite_rails` is the ideal avenue for integrating Rails and Vue due to it acting as a direct bridge with the entire chain having compatibility.

Furthermore, as `vite_rails` is a multi-purpose front-end tool, it allowed the separation of concerns for the styling libraries used in Erudite with TailwindCSS and DaisyUI being imported and bundled via Vite. Effectively separating the domain between information to be rendered and how said information is rendered.

### ***Vue library***

The Vue libraries leveraged to power Erudite are VuePDF and the VuetifyJS calendar component.

These were chosen due to their open-source nature, permitting free-use within the project and their ease of use. While possible to build both using ERB and TailwindCSS, this would have been inefficient compared to passing data to a component library that has already accounted for the edge cases associated with either component.

Initially, Erudite aimed to use the Syncfusion component library due to its common use in the industry. However, the development process was persistently hindered by a bug whereby Syncfusion would nullify its license key, rendering the pages unusable. While initial debugging was done to resolve the issue permanently, the decision was made to migrate elsewhere as the implementation

suffering the problem was basic in nature. Offering minimal confidence in the stability of the implementation if Erudite was to scale in complexity.

## **Hardware**

### **Digital Ocean Droplet VPS**

Initially Erudite was deployed to an AWS EC2 t2.micro instance running Ubuntu Server LTS. However, the 1gb of available RAM meant deployment via Kamal would often crash the server due to requiring more resources than available. To solve this, `deploy.yml` was updated to target a Digital Ocean Droplet with 2gb of RAM, this level of available resource allowed deployment with an average of 40% memory usage.

While more compute power is available via self-hosting, at its current usage the extra power is unnecessary and would introduce the associated risks regarding security and availability as Digital Ocean owns a far more mature infrastructure purpose-built to ensure uptime and image persistence.

## Construction

Fundamentally Erudite follows MVC design paradigms, a direct result of Rails being an MVS framework. Resultingly, the average request can be said to have the lifecycle shown in Figure 16 (*Getting Started with Rails*, no date).

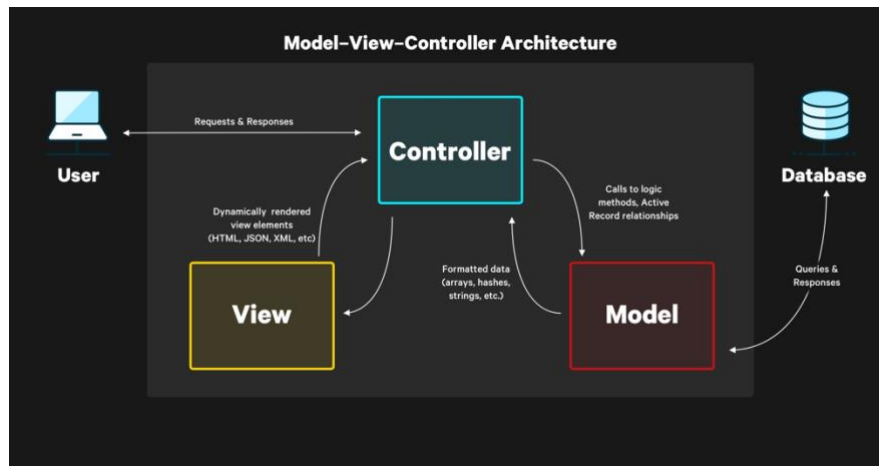


Figure 16 - MVC Architecture Overview

Meaning, each request received by Erudite is routed to a controller action, with said controller then querying a model if required and returning some format of the data.

The following sections delve into some of the major changes throughout the construction of Erudite regarding its Models, Controllers and Views respectively as the project grew and a better understanding of the domain was gained.

## Architecture of Relationships

### Course Structure

The first addition to the Erudite schema was that of the authentication generator, offering a **User** model and a **Session** model. While the key properties within these have not changed outside of **User** being given a *first\_name*, *last\_name* and *account\_type*, the associations **User** holds as a Model has grown significantly.

The first of which was the association between a **User** and a **Course** as individuals need to be enrolled on their relevant course. During planning, this was to be a **Course** *has\_many* Modules, however, this ended up not feasible as Module is a keyword in Ruby. It was also realised that to have content attached to a Module directly rather than as a child of makes the data tightly coupled, requiring more transformation should the feature ever be implemented where content can belong to more than one Module. Erudite in its current form makes content exclusive to one object, however, isolated data models mean that should this feature require developing, it would require less work to implement.

The associations then continued to evolve into a cascading **Course**, **Unit**, **Topic** structure, with each **Unit** containing multiple Topics relevant to the **Course**. Using computer science as an example **Course**, 'data structures and algorithms' would be a **Unit**, with Topics such as 'HashMaps' or 'FIFO principles' being attached to the **Unit** and by association, the **Course**. An implementation example can be seen in Figure 18 & Figure 17.

```

1  class Course < ApplicationRecord
2    has_many :units, dependent: :destroy
3    has_many :topics, through: :units
4  end
5
6  class Unit < ApplicationRecord
7    belongs_to :course
8    has_many :topics, dependent: :destroy
9  end
10
11 class Topic < ApplicationRecord
12   belongs_to :unit
13   has_one :course, through: :unit
14 end

```

**Figure 18 - Example Associations for Course/Unit/Topic**

```

1  # c/u/t ⇒ CompSci, DSA, HashMaps
2  $ c.units.first.title # ⇒ DSA
3  $ u.course # ⇒ CompSci
4  $ u.topics.first # ⇒ HashMaps
5  $ t.unit # ⇒ DSA
6  $ t.course # ⇒ CompSci

```

**Figure 17 - Example Access to Course/Unit/Topic Associations**

This implementation permits the separation of concerns between models that are fundamentally responsible for information in differing contexts while retaining the usability and maintainability from a development perspective.

### Message Model and Polymorphism

The messaging facility in Erudite is a major indicator of project success so a maintainable and functional implementation was essential. As a result, the implementation of the model evolved over time ensuring it could remain maintainable from a single-source yet usable in multiple parts of the application.

Initially there was many-to-one relationship between a **Message** and a **Topic** with an attached **User** as an author. This permitted a single **User** to attach multiple **Message** objects to a **Topic** for display, however, this lacked any form of user tagging and was far from a viable, permanent implementation for the application.

```

1  class Topic < ApplicationRecord
2    has_many :messages, dependent: :destroy
3  end
4
5  class User < ApplicationRecord
6    has_many :messages, dependent: :destroy
7  end
8
9  class Message < ApplicationRecord
10   belongs_to :user
11   belongs_to :topic
12 end

```

**Figure 19 - Initial Message Implementation**

The implementation evolved to use a *mentions* table, effectively a join table between a **User** and a **Message**. When a new message is created it is checked for the presence of other **User** ID's, if found a new **Mention** is created, allowing the tagged **User** to not only be notified of the messages existence but also permitted direct navigation to it as the association offered full access to the **Message** object.

This version of the messaging system became the foundation for the notification system added to Erudite but remained locked to a **Topic**, which was not sustainable due to the requirement specifying messaging should be a feature enabled between multiple parts of the website, be it **Topics**, **Assignments** or **User** to **User**. To bring the project closer to meeting requirements the **Message** model was made polymorphic and updated accordingly, permitting new **Message** objects against any model by implementing *messageable* against the required model as shown in \_\_ which permits **Messages** against

```
1 class Topic < ApplicationRecord
2   has_many :messages, as: :messageable, dependent: :destroy
3 end
4
5 class Assignment < ApplicationRecord
6   has_many :messages, as: :messageable, dependent: :destroy
7 end
8
9 class User < ApplicationRecord
10  has_many :messages, dependent: :destroy
11 end
12
13 class Message < ApplicationRecord
14   belongs_to :user
15   belongs_to :messageable, polymorphic: true
16 end
```

**Figure 20 - Example of Message Polymorphism**

**Assignments and Topics.**

The polymorphic implementation of Messages in this form supports:

1. The tagging of Users with the *taggable* profile setting enabled
2. The relative ease of implementation into new Models
3. Straightforward maintenance due a single **Message** model
4. Click-through behaviour on a Mention/Notification for ease of access

Overall, this implementation meets the projects functional requirement regarding the ability to send and receive messages on relevant pages.



## Evolution of Routes

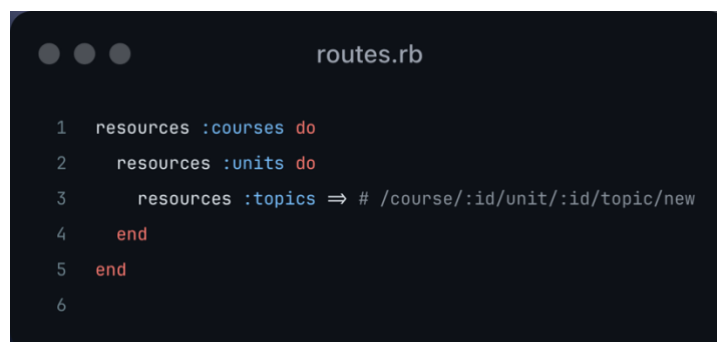
Routing in an MVC application is responsible for ensuring the requests go to the correct controller to be processed according to expectation. In a Rails project, this is mainly done automatically if the expected conventions are followed. For example, given the controller `CoursesController`, adding “resources :courses” to Rails’ main route file `routes.rb`, would create the following routes:

HTTP Verb	Path	Controller#Action	Used to
GET	/courses	courses#index	Get all
GET	/courses/new	courses#new	Get form for new Course
POST	/courses	courses#create	Create new course
GET	/courses/:id	courses#show	Show specific course
GET	/courses/:id/edit	courses#edit	Edit specific course
PATCH/PUT	/courses/:id	courses#update	Update specific course
DELETE	/courses/:id	courses#destroy	Delete specific course

**Table 1 - resources verbs per Rails Guide**

It then becomes possible to namespace and nest these, so each route path is prepended with the namespace belonging to the Controller or the parent resource.

The major routing development for Erudite was the refactoring of a deeply nested association relating to the Model associations. As `Courses` have `Units` & `Units` have `Topics` the initial route for creating a new topic became:



```
1 resources :courses do
2   resources :units do
3     resources :topics => # /course/:id/unit/:id/topic/new
4   end
5 end
6
```

**Figure 21 - Initial Route Implementation**

Such an implementation very quickly became incredibly difficult to work with and maintain due to everything required to access the innermost object. In this form, the link to edit a topic would be:



```
1 <%= link_to "Edit Topic" edit_course_unit_topic_path(@course, @unit, @topic) %>
```

This meant great attention had to be paid to all three variables when trying to access a Topic as opposed to only leveraging the model-level associations.

After refactoring, a conscious effort was made to ensure Erudite nesting stopped at one level of nesting, to avoid this happening with other data models.

## Evolution of Controllers

Following a successful request routing, the “controller is responsible for processing the request and generating the appropriate output” (*Action Controller Overview*, no date). This means the controllers within an application are effectively the bridge between the user and their input permitting interaction with the database and application Models.

Since every request is routed through a controller, if they are not maintained adequately or separated appropriately it can become easy to lose track of what happens through the lifecycle of a request, sacrificing maintainability for proposed technicality. To avoid this Erudite’s controllers grew iteratively, with core functionality being implemented and then abstracted or encapsulated accordingly as opposed to implementing a complex pre-planned system before core functionality is ensured. This decision was inspired by the founder of Rails suggesting that sub-resources should warrant a new controller, rather than adding methods to a controller that are not inherently create/read/update/delete as “Every single time I’ve regretted the state of my controllers, it’s been because I’ve had too few of them. I’ve been trying to overload things too heavily.” (Dalbert, 2016).

The primary evolutions of Erudite’s controller logic were the implementation of authentication; the migration to namespaces and the extension of `BaseController` for authentication

Building on the development of the Model objects within the project, the controllers rarely shared responsibility, due to this, the changes made to the controllers were primarily a result of controlling access to them based on User account status.

## Authentication

Once implemented, the Rails authentication system implements a “Deny by Default” policy which means everything within the app requires authentication to access. While such a philosophy can cause inconvenience for users, there is less risk associated with misconfiguration hiding information than exposing it to unauthorised parties.

Deny by Default meant pages that should be publicly available such as the sign-in page had to be implemented as:

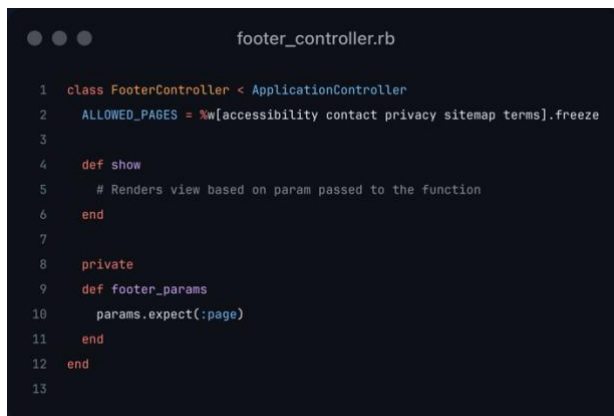
```
1 class SessionsController < ApplicationController
2   allow_unauthenticated_access only: %i[ new create ]
3
4   def new
5     # Omitted for space
6   end
7
8   def create
9     # Omitted for space
10  end
11 end
```

Figure 22 - Allowing Unauthenticated Controller Actions

The above implementation means that the routes targeting `SessionsController#new` and `SessionsController#create` within the application can be accessed without the user being signed in.

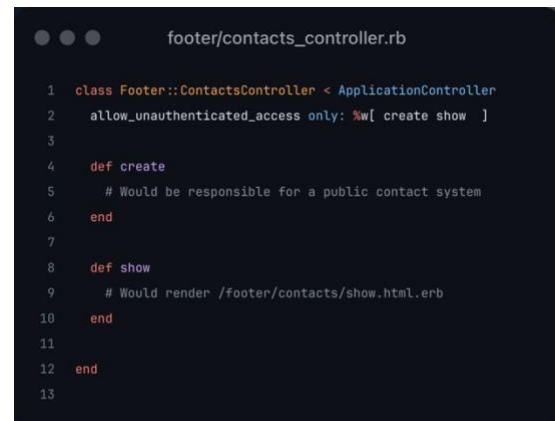
In context this was not a major problem to overcome as the project requirements specified that the majority of the pages available would only be accessible by those with permission to do so. Furthermore, the `allow_unauthenticated_access` method can be included without options, permitting blanket access against all controller methods. This method was used on `footer_controller.rb`, as while against the separation of concern principle discussed earlier, usually enforced by multiple controllers, the linked pages are display-only information pages that are unlikely to change.

Should it become appropriate to expand upon these pages, the relevant methods would be refactored into a `Footer::` namespace with the associated page. For example, the implementation seen in Figure 24 would become Figure 23 :

A screenshot of a code editor showing the `footer_controller.rb` file. The code defines a `FooterController` class that inherits from `ApplicationController`. It includes a constant `ALLOWED_PAGES` with a list of page names, a `show` method that renders a view based on the page parameter, and a private `footer_params` method that expects the page parameter.

```
1 class FooterController < ApplicationController
2   ALLOWED_PAGES = %w[accessibility contact privacy sitemap terms].freeze
3
4   def show
5     # Renders view based on param passed to the function
6   end
7
8   private
9   def footer_params
10     params.expect(:page)
11   end
12 end
13
```

Figure 24 - Current Footer Controller

A screenshot of a code editor showing the `footer/contacts_controller.rb` file. The code defines a `Footer::ContactsController` class that inherits from `ApplicationController`. It includes the `allow_unauthenticated_access` method and two methods: `create` and `show`. The `show` method has a comment indicating it would render a specific view.

```
1 class Footer::ContactsController < ApplicationController
2   allow_unauthenticated_access only: %w[ create show ]
3
4   def create
5     # Would be responsible for a public contact system
6   end
7
8   def show
9     # Would render /footer/contacts/show.html.erb
10  end
11 end
12 end
13
```

Figure 23 - Scalable Footer

While the integration of default authentication did not warrant major changes to the controllers within Erudite, it is still considered a significant event throughout development as it impacts the implementation of every controller created thereafter.

### Migration to namespaces

The addition of the Rails 8 authentication system allowed separation of pages between someone with a valid Session compared to those without, however, it did not add functionality that supports the separation of pages that require authorisation beyond that of a standard user. Per the project requirements, access should be role-based, meaning more than one `User` account. For Erudite this meant the introduction of `Student`, `Teacher`, and `Admin`.

Rails' is by default incredibly advanced regarding processes being readily available and nearly automatic if convention is adhered to, in line with their motto "Convention over Configuration". Due to this, any site behaviours that had an equivalent action attached to a different `User` type, for example, a student and an admin having different views when accessing a `Course` index, was encapsulated within the relevant namespaced controller. From this example, it would have been possible to build `Course.rb` as three separate methods within the same controller per Figure 26

```

courses_controller.rb

1  class CoursesController < ApplicationController
2
3    def index
4      # index for student
5    end
6
7    def teacher_index
8      # index for teacher
9    end
10
11   def admin_index
12     # index for admin
13   end
14 end
15

```

**Figure 26 - Courses Controller with Three Index Methods**

However, this would make Rails unable to generate the default connections between data for `teacher_` and `admin_`, creating further technical debt for the developer and forces validation to be added to any methods beyond this to ensure they're interacting with the correct version of their intended method. `Course.rb` could also be implemented as a case statement, checking the Users account type and

```

courses_controller.rb

1  class CoursesController < ApplicationController
2
3    def index
4      case user.account_type
5      when student
6        render student_index
7      when teacher
8        render teacher_index
9      when admin
10       render admin_index
11      else
12        "Unexpected error"
13      end
14    end
15

```

**Figure 25 - Courses Controller with Switch Statement**

returning the view based on that information as shown in Figure 25.

Consequently, this implementation would ensure that the varying account types would exclusively be able to see their pages and no others. Long-term this could lead to a disconnect between users, impeding future communication as students would be forced to explain what they see to teachers and administrators, rather than privileged users being able to look at the pages directly. Furthermore, this also would not permit Rails to generate its automatic associations.

As such, Rails support for namespacing was introduced, such implementation meant that code of similar function, but different result could be written without requiring added work associated with checking the user account or manually configuring the routing to non-RESTful controller actions.

```
1 class CoursesController < ApplicationController
2   def index
3   end
4 end
5
6 class Admin::CoursesController < ApplicationController
7   def index
8   end
9 end
10
11
```

**Figure 27 - Namespaced Index Methods**

Such implementation would enable the path features native to rails, meaning each index could by default be accessed via `course` or `admin_course` depending on the target controller. Building Erudite's controllers out in this fashion not only contributes to enhanced maintainability in the future based on the awareness that anything erroring from an `/admin` route will be caused by code within an `Admin::` controller but also reduces developer workload to begin with due the support offered by Rails to autonomously configure the routes in accordance with desired actions.

The decision to integrate namespaced controllers in Erudite carries minimal downside as while it could be argued namespaces contribute to a more complex codebase, the consistent theme of Erudite requiring the same pages with varying visible data would inevitably require a shift into namespaces as complexity grew.

## UI Evolution

The aim of Erudite was to build an easy to navigate, clear and predictable UI allowing students to interact with their degree without the decision fatigue associated with a busy or hard to navigate interface. Fundamentally, Erudite is a navigation bar with a clock, the closest deadline and a sign-out button with a tab-based homepage permitting navigation into course pages without each section having varying layouts to learn, understand, and remember.

From the perspective of a student, each page within this is generally split down the middle with a vertical half of the page for content, and another for notes/messaging. Erudite was implemented in such a way to ensure students are not required to frequently switch between tabs or split application windows across their display to view all relevant content.

Past the initial planning stage, the layout of Erudite was subject to little change besides moving the profile button from the navigation bar to a home tab to maintain navigation consistency.

As such, there was little visual development besides the addition of new pages as development continued. To construct the design the decision was made to use DaisyUI, a TailwindCSS based component library offering pre-configured styling for essential parts of the application.

The implementation of an open-source component library meant less time was required to build the necessary pages as use of the library naturally ensured the pages had a consistent and expected aesthetic. This removed the need for development time being spent on constructing CSS classes for use throughout the project, had this time been spent, the component styling likely would have ended up similar to that of DaisyUI so it was a decision made with efficiency in mind.

Due to the lack of complexity in the components many other libraries could have been used to build a near equivalent display. Libraries such as Shoelace and FlyonUI were considered as they both contained the necessary components, however, familiarity with DaisyUI meant that the integration of other libraries would introduce an unnecessary learning curve given the relatively uncomplicated use of the components.

## **Testing**

### **Route Testing**

[DO THE ROUTES GO WHERE THEY ARE SUPPOSED TO BASED ON AUTH/AUTH]

### **ERB Testing**

[DOES THE SITE RETURN THE EXPECTED HTML BASED ON CERTAIN BEHAVIOURS]

## **Evaluation / Maintenance**

### **Conclusion of Project**

[QUERY OPTIMISATION]

### **Self-reflection (1<sup>st</sup> person)**

### **Further Work**

[SUB-DOMAINS FROM A MAJOR DEPLOYMENT CENTRE FOR MUTLI-TENANCY/MORE THAN ONE INSTITUTION]



## Bibliography

2023 Review: *What's the state of the VLE market in UK higher education?* (no date) Neil Mosley Consulting. Available at: <https://www.neilmosley.com/blog/2023-review-whats-the-state-of-the-vle-market-in-uk-higher-education> (Accessed: 16 April 2025).

*About Projects* (no date) *GitHub Docs*. Available at: <https://docs-internal.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects> (Accessed: 13 March 2025).

*Action Controller Overview* (no date) *Ruby on Rails Guides*. Available at: [https://guides.rubyonrails.org/action\\_controller\\_overview.html](https://guides.rubyonrails.org/action_controller_overview.html) (Accessed: 15 April 2025).

*Active Record Basics* (no date) *Ruby on Rails Guides*. Available at: [https://guides.rubyonrails.org/active\\_record\\_basics.html](https://guides.rubyonrails.org/active_record_basics.html) (Accessed: 9 April 2025).

*Active Record Migrations* (no date) *Ruby on Rails Guides*. Available at: [https://guides.rubyonrails.org/active\\_record\\_migrations.html](https://guides.rubyonrails.org/active_record_migrations.html) (Accessed: 18 April 2025).

*Active Record Validations* (no date) *Ruby on Rails Guides*. Available at: [https://guides.rubyonrails.org/active\\_record\\_validations.html](https://guides.rubyonrails.org/active_record_validations.html) (Accessed: 9 April 2025).

Aina, C. *et al.* (2022) 'The determinants of university dropout: A review of the socio-economic literature', *Socio-Economic Planning Sciences*, 79, p. 101102. Available at: <https://doi.org/10.1016/j.seps.2021.101102>.

'basecamp/thruster' (2025). Basecamp. Available at: <https://github.com/basecamp/thruster> (Accessed: 14 April 2025).

Burgess, M. (no date) *FOI List of Emails for UK Universities | FOI Directory*. Available at: <http://www.foi.directory/uk-universities-list-foi-emails/> (Accessed: 23 January 2025).

Buyalskaya, A. *et al.* (2023) 'What can machine learning teach us about habit formation? Evidence from exercise and hygiene', *Proceedings of the National Academy of Sciences*, 120(17), p. e2216115120. Available at: <https://doi.org/10.1073/pnas.2216115120>.

Dalbert, J. (2016) *How DHH organizes his Rails controllers*. Available at: <https://jeromedalbert.com/how-dhh-organizes-his-rails-controllers/> (Accessed: 15 April 2025).

*Deploy web apps anywhere* (no date) *Kamal*. Available at: <https://kamal-deploy.org/> (Accessed: 7 April 2025).

'faker-ruby/faker' (2025). Faker. Available at: <https://github.com/faker-ruby/faker> (Accessed: 14 April 2025).

Fowler, M. (2013) *Patterns of enterprise application architecture*. Nineteenth printing. Boston San Francisco New York Toronto Montreal London Munich Paris Madrid Capetown: Addison-Wesley (The Addison-Wesley Signature Series).

*Getting Started with Rails* (no date) *Ruby on Rails Guides*. Available at: [https://guides.rubyonrails.org/getting\\_started.html](https://guides.rubyonrails.org/getting_started.html) (Accessed: 14 April 2025).

Henderson, M., Selwyn, N. and Aston, R. (2017) 'What works and why? Student perceptions of "useful" digital technology in university teaching and learning', *Studies in Higher Education*, 42(8), pp. 1567–1579. Available at: <https://doi.org/10.1080/03075079.2015.1007946>.



Hubble, S. and Bolton, P. (2021) 'Coronavirus: Higher and further education back to campus in England in 2020/21?' House of Commons Library. Available at: <https://researchbriefings.files.parliament.uk/documents/CBP-9030/CBP-9030.pdf> (Accessed: 23 January 2025).

Hunt, A. and Thomas, D. (2011) *The pragmatic programmer: from journeyman to master*. 26. print. Boston: Addison-Wesley.

Hylender, D. et al. (2024) *2024 Data Breach Investigations Report*. Verizon Business. Available at: [https://www.verizon.com/business/resources/reports/dbir/?CMP=OOH\\_SMB\\_OTH\\_2222\\_MC\\_20200501\\_NA\\_NM20200079\\_00001](https://www.verizon.com/business/resources/reports/dbir/?CMP=OOH_SMB_OTH_2222_MC_20200501_NA_NM20200079_00001).

*Installation* (no date) *Kamal*. Available at: <https://kamal-deploy.org/docs/installation/> (Accessed: 9 April 2025).

Jackson, P. (2025) 'What is the academic year like at a UK university?', *Belong Blog*, 15 January. Available at: <https://belong.ncl.ac.uk/blog/the-academic-year-at-a-uk-university> (Accessed: 22 January 2025).

Laursen, B. (2013) 'The good and bad of peer pressure, with Brett Laursen, PhD'. (Speaking of Psychology:). Available at: <https://www.apa.org/news/podcasts/speaking-of-psychology/peer-pressure> (Accessed: 22 January 2025).

Levine, S.L., Tabri, N. and Milyavskaya, M. (2023) 'Trajectories of depression and anxiety symptoms over time in the transition to university: Their co-occurrence and the role of self-critical perfectionism', *Development and Psychopathology*, 35(1), pp. 345–356. Available at: <https://doi.org/10.1017/S0954579421000626>.

Lewis, J., Bolton, P. and Hubble, S. (2021) *Coronavirus: HE/FE return to campus in England 2021*. Available at: <https://commonslibrary.parliament.uk/research-briefings/cbp-9142/> (Accessed: 23 January 2025).

*Manifesto for Agile Software Development* (no date). Available at: <https://agilemanifesto.org/> (Accessed: 17 April 2025).

McGivern, P. and Shepherd, J. (2022) 'The impact of COVID-19 on UK university students: Understanding the interconnection of issues experienced during lockdown', *Power and Education*, 14(3), pp. 218–227. Available at: <https://doi.org/10.1177/17577438221104227>.

Nelson, J. and Menold, J. (2020) 'The Value of Prototyping: An Investigation of the Relationship Between the Costs of Prototyping, Perceived Value, and Design Outcome', in *Volume 8: 32nd International Conference on Design Theory and Methodology (DTM). ASME 2020 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Virtual, Online: American Society of Mechanical Engineers, p. V008T08A041. Available at: <https://doi.org/10.1115/DETC2020-22104>.

*Password Manager & Extended Access Management | IPassword | IPassword* (no date). Available at: <https://1password.com/> (Accessed: 7 April 2025).

Petersen, K., Wohlin, C. and Baca, D. (2009) 'The Waterfall Model in Large-Scale Development', in F. Bomarius et al. (eds) *Product-Focused Software Process Improvement*. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Business Information Processing), pp. 386–400. Available at: [https://doi.org/10.1007/978-3-642-02152-7\\_29](https://doi.org/10.1007/978-3-642-02152-7_29).

*Prime Minister's statement on coronavirus (COVID-19): 23 March 2020* (2020) GOV.UK. Available at: <https://www.gov.uk/government/speeches/pm-address-to-the-nation-on-coronavirus-23-march-2020> (Accessed: 23 January 2025).

*puma | RubyGems.org | your community gem host* (no date). Available at: <https://rubygems.org/gems/puma> (Accessed: 14 April 2025).

'puma/puma' (2025). Puma. Available at: <https://github.com/puma/puma> (Accessed: 14 April 2025).

*Response Time Limits: Article by Jakob Nielsen* (no date) Nielsen Norman Group. Available at: <https://www.nngroup.com/articles/response-times-3-important-limits/> (Accessed: 9 April 2025).

*Ruby on Rails* (no date) *Ruby on Rails*. Available at: <https://rubyonrails.org/> (Accessed: 9 April 2025).

*SPA (Single-page application) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN* (2024). Available at: <https://developer.mozilla.org/en-US/docs/Glossary/SPA> (Accessed: 9 April 2025).

*Stimulus: A modest JavaScript framework for the HTML you already have.* (no date). Available at: <https://stimulus.hotwired.dev/> (Accessed: 9 April 2025).

*Technology | 2024 Stack Overflow Developer Survey* (no date). Available at: <https://survey.stackoverflow.co/2024/technology> (Accessed: 9 April 2025).

*The Rails Doctrine* (no date) *Ruby on Rails*. Available at: <https://rubyonrails.org/doctrine> (Accessed: 9 April 2025).

Thompson, M., Pawson, C. and Evans, B. (2021) 'Navigating entry into higher education: the transition to independent learning and living', *Journal of Further and Higher Education*, 45(10), pp. 1398–1410. Available at: <https://doi.org/10.1080/0309877X.2021.1933400>.

*Turbo Handbook* (no date). Available at: <https://turbo.hotwired.dev/handbook/introduction> (Accessed: 9 April 2025).

*Turbo: The speed of a single-page web application without having to write any JavaScript.* (no date). Available at: <https://turbo.hotwired.dev/> (Accessed: 9 April 2025).

*Usage Statistics of JavaScript as Client-side Programming Language on Websites, April 2025* (no date). Available at: <https://w3techs.com/technologies/details/cp-javascript> (Accessed: 9 April 2025).

Viswanathan, V.K. and Linsey, J.S. (2013) 'Role of Sunk Cost in Engineering Idea Generation: An Experimental Investigation', *Journal of Mechanical Design*, 135(12), p. 121002. Available at: <https://doi.org/10.1115/1.4025290>.

*What is Docker?* (no date) *Docker Documentation*. Available at: <https://docs.docker.com/get-started/docker-overview/> (Accessed: 7 April 2025).

*Where do HE students study? | HESA* (2024). Available at: <https://www.hesa.ac.uk/data-and-analysis/students/where-study> (Accessed: 23 January 2025).

*Who's studying in HE? | HESA* (2024). Available at: <https://www.hesa.ac.uk/data-and-analysis/students/whos-in-he> (Accessed: 20 January 2025).

*Working with JavaScript in Rails* (no date) *Ruby on Rails Guides*. Available at: [https://guides.rubyonrails.org/working\\_with\\_javascript\\_in\\_rails.html](https://guides.rubyonrails.org/working_with_javascript_in_rails.html) (Accessed: 14 April 2025).

Yang, M.C. (2005) 'A study of prototypes, design activity, and design outcome', *Design Studies*, 26(6), pp. 649–669. Available at: <https://doi.org/10.1016/j.destud.2005.04.005>.