Project Title: “Tutors Hive”

Sorting and Social for Tutors Live

Final Report

Name: Grace Doyle

Student ID: 20002141

Higher Diploma in Computer Science

South East Technological University

Supervisor: Eamonn de Leastar

Abstract

Tutors Hive is based around the current SETU TutorsLive software. It is a prototype of the current Tutors Live software, with two apps, one for simulation of data, which writes to Firebase Realtime database, and secondly, a front-end prototype" (Tutors Hive) app, which builds features around online presence, to enhance the user experience. The two main features showcased are sorting the students by topic/lab in which they are studying, and also the slack communication feature, whereby each student card has individual message and call (huddle) buttons, with an additional group huddle button per topic.

Table of Contents

1. Introduction

1.1 Preface

This project is an independent project, based around adding features and new functionality to a current educational software, Tutors Live. Tutors Live is a section of the Tutors Open Source project at SETU, which was built and is used by SETU, for students and tutors across computer Science Modules.I will be learning new skills and technologies and utilizing the knowledge acquired from the course modules to build and integrate new features and functionalities to this project, so that future students can have an even more enhanced learning experience.

Tutors Live has been an integral part of my learning path throughout this course and has made my remote learning experience more enjoyable. I have experienced, first hand, the benefit of remote learning on a modern, advanced and easy to use platform. I have an interest in helping with this open source project, to further improve the learner and educator experience. There is already a large choice of learning platforms and coursework providers on the internet available, most notably for SETU students, Moodle. However, Tutors Live aims to provide students with a more sophisticated, modern and advanced, smoother platform with more up to date features. Tutors Live is also open source and free of charge.

1.2 Scope

The additional features I would like to add will help improve the Educator Experience by enabling the Educator to clearly see a visual of who is online, what they are studying, but also what group they are in based on what they are currently working on. They will also improve the Learners Experience by enabling the learner (and Educator) to easily start a chat with their fellow classmates (via a button), who are in the same group as them, and the option to also start a group huddle/chat. This gives the option to the other classmates to accept or reject the chat or call. The aim is that this chat feature will include the Slack API for messaging and hopefully video too.

Currently, Tutors live displays student cards and what they are working on with no order to it and no ability to reach out and chat. See Fig 1.1 below

Fig 1.1 Tutors layout mock up (at time of project proposal)

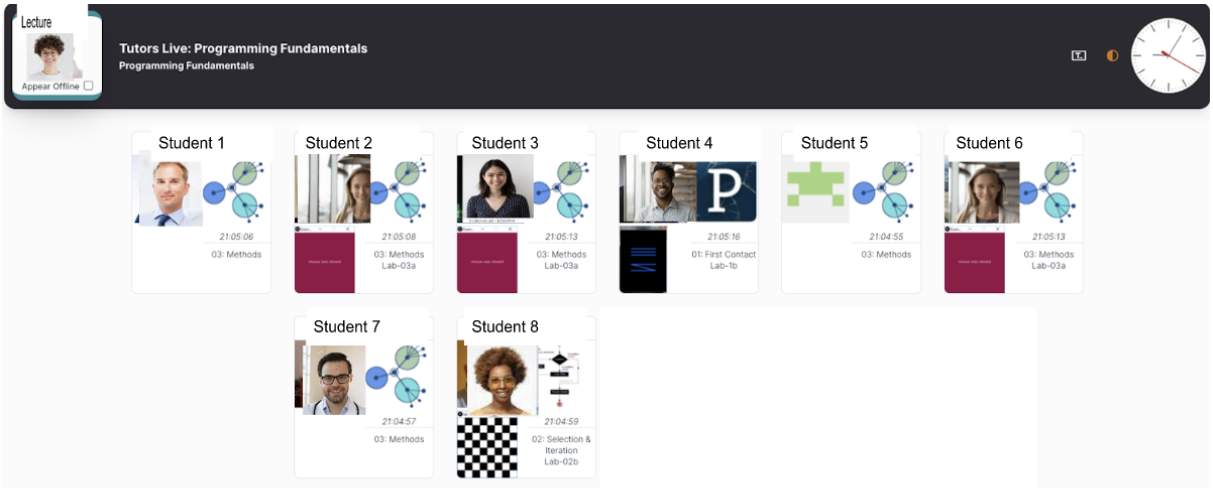


Fig 1.2 shows how I would envisage the sorting of the groups to look like based on what they are working on and a button to DM or call

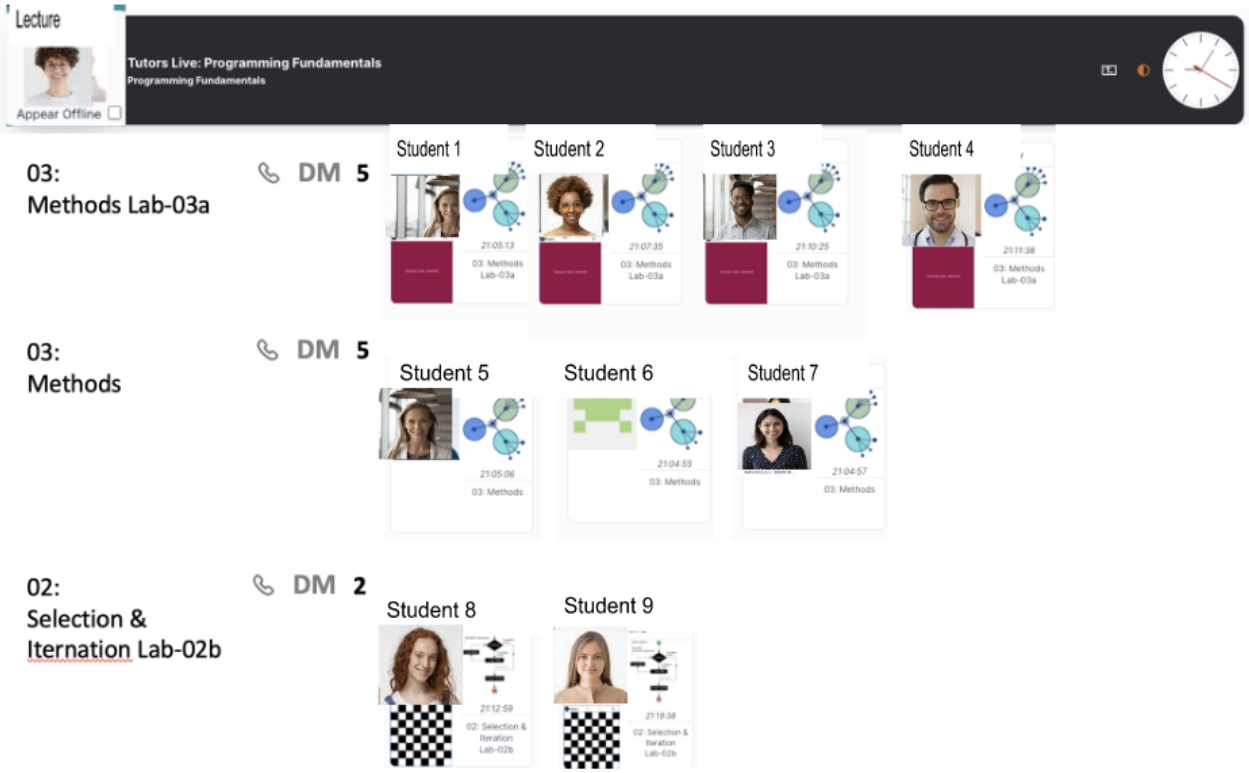


Fig 1.2 visual goal of sorting by topics with social buttons

This Final screenshot show my vision of how it might look.

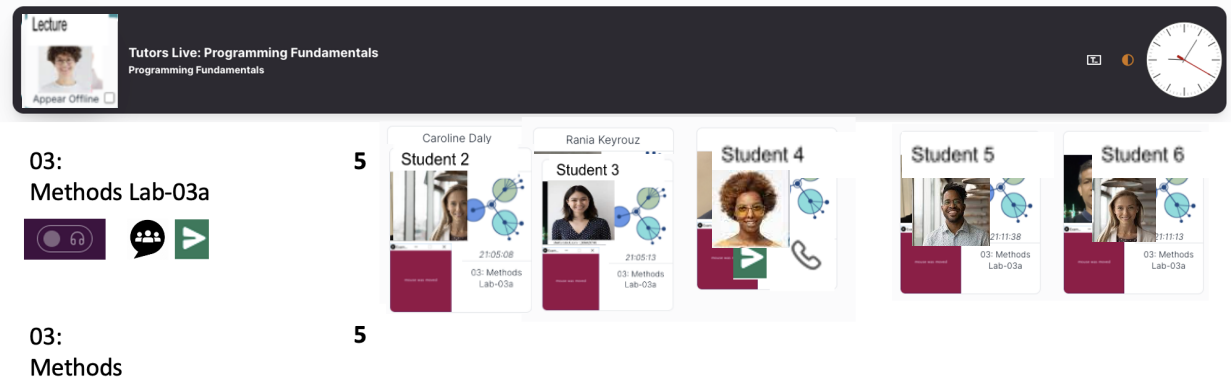


Fig 1.3 Final Proposal card layout with social aspect

1.3 Tutors Live

Tutors Live is an existing open source Project, that is a section of the Tutors Open Source Project. The creation of Tutors Open Source Project was started by Eamonn de Leastar around ten years ago as a python application, and is now using more modern technologies. The project consists of three sections, The Educator, The Learner and the Developer experiences. For this project I am focusing on the Tutors Live aspect of the open source project, which provides modern remote, transformative learning experience to HDip computer Science students in SETU Computer Science Courses. Tutors Live uses open source web standards (Tutors Open Source Project, 2023)

Tutors Live is built using JavaScript and SvelteKit, typescript and Firebase Database and can be accessed on Github. I hope to use these same technologies, and new technologies such as SlackAPI and a skeleton app to test and demo my features as the project progresses.

2. Motivation

2.1 Learning Online

2.1.1 Online Education Platforms

Online Education platforms offer rich, interactive learning experiences, innovative and adaptable learning environments, and the potential to transform education delivery. They are now such an integral component of the education sector, “offering a range of courses, programs and credentials that serve the needs of learners, employers, and the economy." (Katz & Green, 2017). Universities are able to utilize modern technologies to reach people all around the world which also means students can taster courses beforehand before deciding on a course. “It begins with embracing stackable, online learning, which provides flexibility and affordability that increases access to university curricula and allows students to engage in smaller chunks of learning before committing to larger degree programs”(Belsky 2019).

Not only have online education platforms opened up new avenues for collaboration and knowledge-sharing, providing students with opportunities to connect and learn from their peers on a global scale." (Manca & Ranieri, 2016) but they have also revolutionized the way we think about learning and have made education more accessible and affordable for people all over the world (Salmon, 2013)

There is quite a lot of learning platforms and coursework providers in the EdTech space. As De Leastar (2022) showed in *Fig 2.1* below.

Fig 2.1 Ed Tech Space, (De Leastar 2022)

2.1.2 The Role of Presence & community

Our experiences in virtual and physical environments are shaped by our sense of presence and community. As well as providing social interaction and support, they can make people feel connected and belong. "A sense of presence can enhance the feeling of connection and interaction between community members, leading to a more immersive and fulfilling experience" (Biocca, 1997). An online community gives users of any platform a place to ask questions, provide answers, discuss problems or anything related to the platform they are using.Online learners benefit from being aware of other students and what they are doing can contribute to presence

(Garrison, 2021) states that there are three aspects to an online teaching experience which are cognitive presence, teaching presence and online presence. "Teaching presence refers to the design, facilitation, and direction of cognitive and social processes for the purpose of realizing personally meaningful and educationally worthwhile learning outcomes" (Anderson & Dron, 2011). Social presence refers to “the sense of being in the company of others and is a key factor in determining the effectiveness of communication and the quality of relationships in online contexts." (Walther, 1996). Social presence is also described as the ability of an individual to project their personal characteristics, such as personality and emotions, into an online interaction, creating a sense of connection and community with others. (Tu, McIsaac, & Yi, 2002). "Cognitive presence is the extent to which individuals are able to construct their own understanding of a concept or issue through interaction and reflection with others in a community of inquiry." (Swan, Richardson, & Ice, 2007).

Cognitive presence promotes critical thinking, problem-solving, and sense-making in online learning environments. According to (Garrison & Arbaugh, 2007) Cognitive presence is “concerned with the quality of the meaning-making process of a community of inquiry and is determined by the extent to which learners are able to construct and confirm meaning through sustained communication and reflection."

2.2 The Tutors Platform

The motivation for Tutors platform is to improve the learner and educator experience. Tutors is a platform for building, managing, and delivering online education programs. According to Tutors Open Source Project(2023) Tutors provides a suite of tools for creating and delivering engaging and interactive learning experiences, as well as tools for managing student progress and analytics.

2.2.1 Structure of a course

A Tutors Course template 'tutors-starter' can be initially downloaded from the Setup section within the Tutors Documentation from Tutors Open Source Project(2023) From this a custom course can be built, which I completed as part of my research. The File Structure for a course includes a json folder, the various topic folders, which are the topics within a module. Each topic is represented by a card on the course web. See image 2.1 below of my personal customised demo course which showsEach topic is represented by a card on the course web and each card has a Lecture (unit)Fig 2.3. Each lecture would then open up a slide deck. Users can navigate back and forward between topics and lectures and labs from the navigation panel near the top left of the screen.

Fig 2.2 My personal customised tutors starter demo course main view

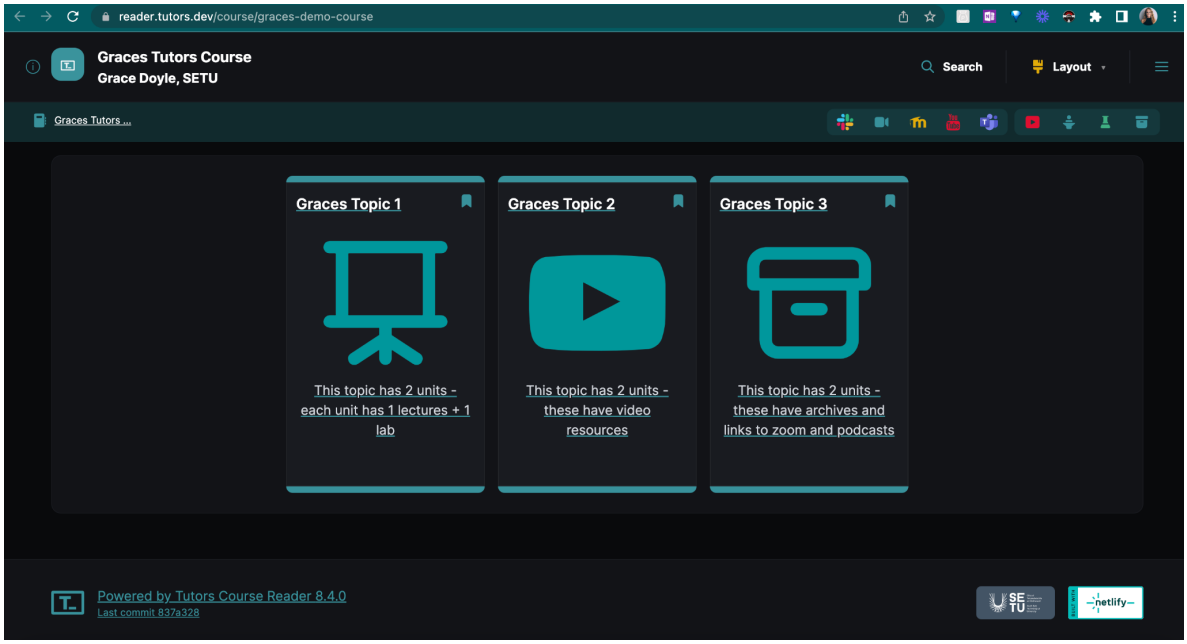
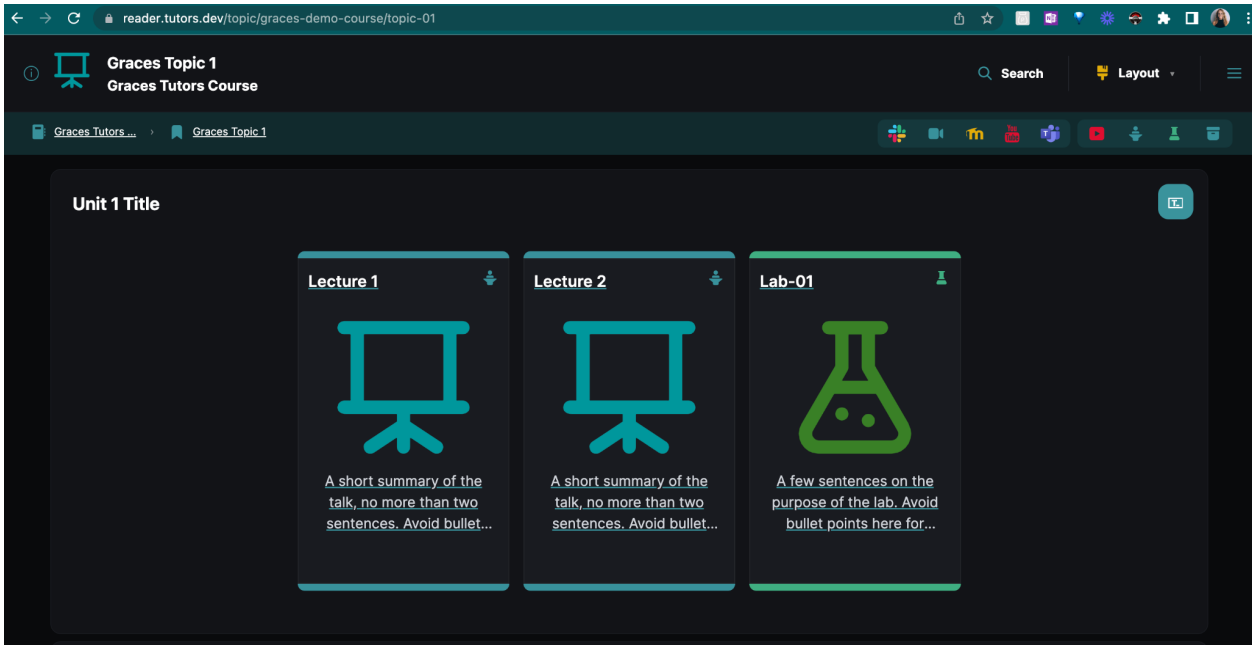


Fig 2.3 My personal customised tutors starter demo course topic view



More topics can be created by following the naming convention 'topic-**', replacing the "*" with digits. The same can be said for the units, which are the sub folders of the topic folders (*Fig 2.3*) The Folder Structure of the courses can be seen below in *Fig 2.4* and Fig *2.5.*

Fig 2.4 Structure of My demo course

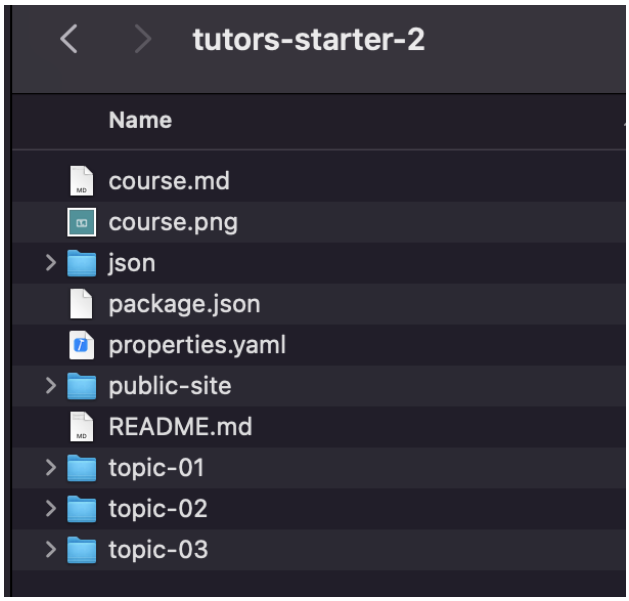
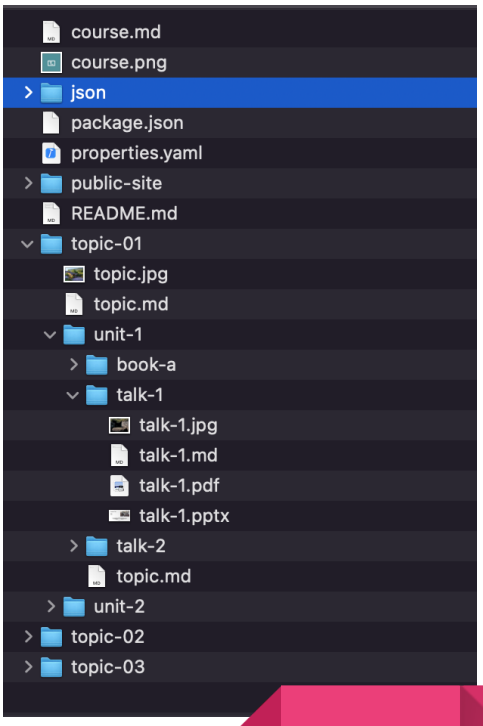
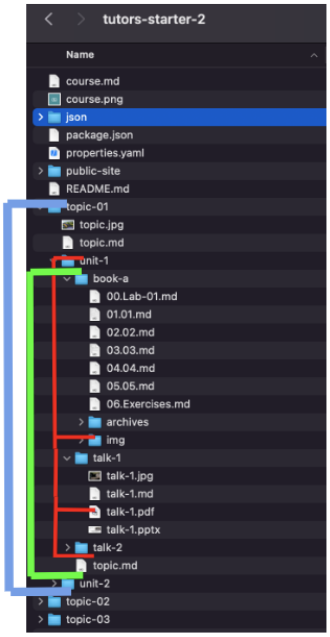


Fig 2.5 Structure of My demo course deeper level



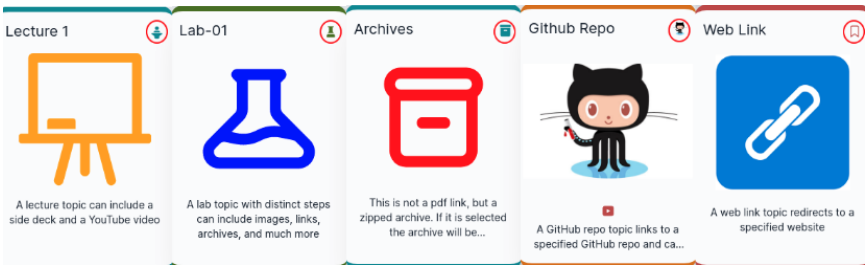
The Units is where the **talks** are embedded and are the pdf versions of the slide decks, rendered as a card also. See FIg 2.6 and Fig 2.3. The Folder Book-a is where the **labs** are embedded, whereby each lab has a series of steps that are represented by the '.md' files. The book-* folder can also contain 'img' and 'archives' folders. See Fig 2.6.

Fig 2.6 Tutors course file embedded system



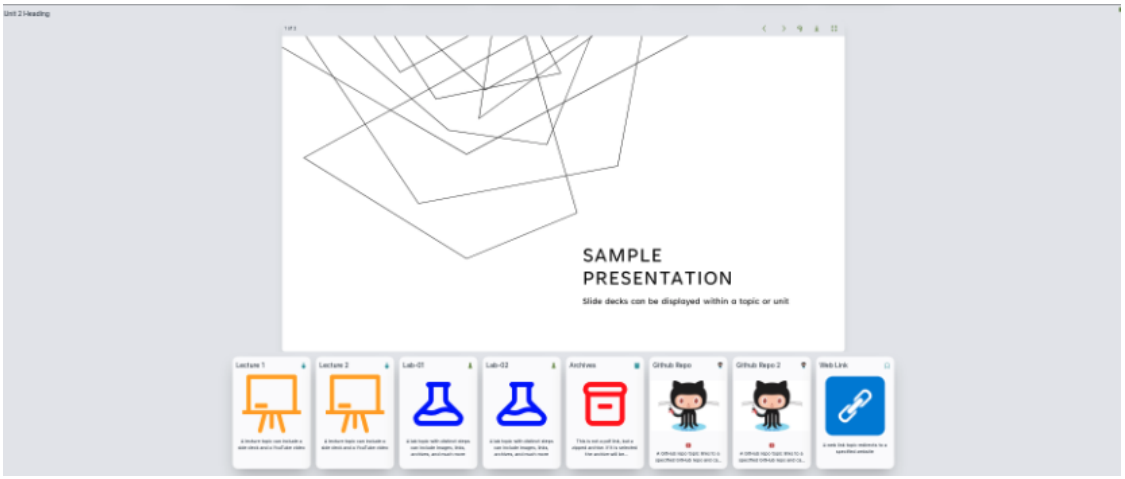
There are also Tutors Elements, which are represented by a card with each topic type having a corresponding icon *Fig 2.7)*.

Fig. 2.7 topics with corresponding icons



So a A typical sample presentation in Tutors could look like this (*Fig 2.8*)

Fig 2.8 Sample Presentation (Tutors Open Source Project, 2023)



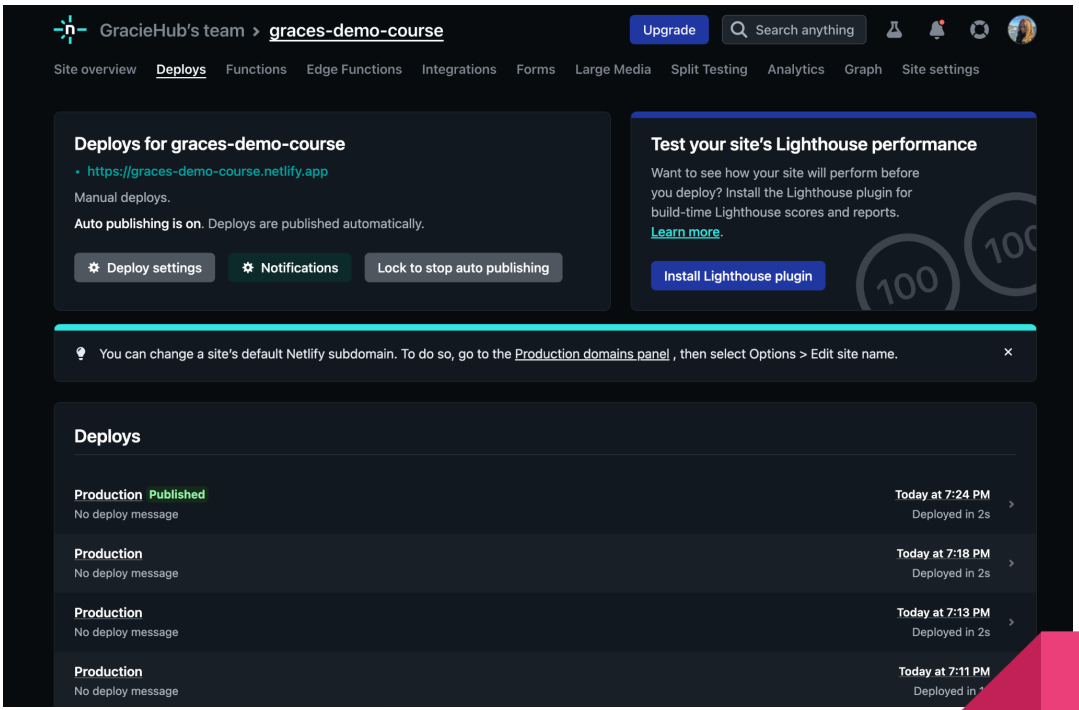
2.2.2. Publication process

I experimented by creating and publishing my own course, following the steps provided the Setup section within the Documentation from Tutors Open Source Project(2023). I changed some folder names and republishing the course again. The main steps involved in this are to firstly, download the template of the tutors readers course from the link above, install Node.js on your machine, use Sublime to navigate the folder structure, create an account on Netifly and login. You simply click and drag the Json folder over to your sites section of the Netifly account directly into the drop in area and the site will now be published (deployed) See Fig 2.9. And when you click the link at the top of the page you will be redirected to the site overview page. (Fig 2.2). You can republish a course by making changes to the folders in sublime, then compile the course (regenerate it) by using the command

```
npx tutors-json
```

from the command line while in the main course folder. This will produce an output which is a list of the course contents, so then you redrag the updated json file over to the Deploy drag and drop section on the Netifly account(*Fig 2.9*). The JSON directory will have been updated to the latest version. When you refresh your published site you will now see that your updates have been implemented.

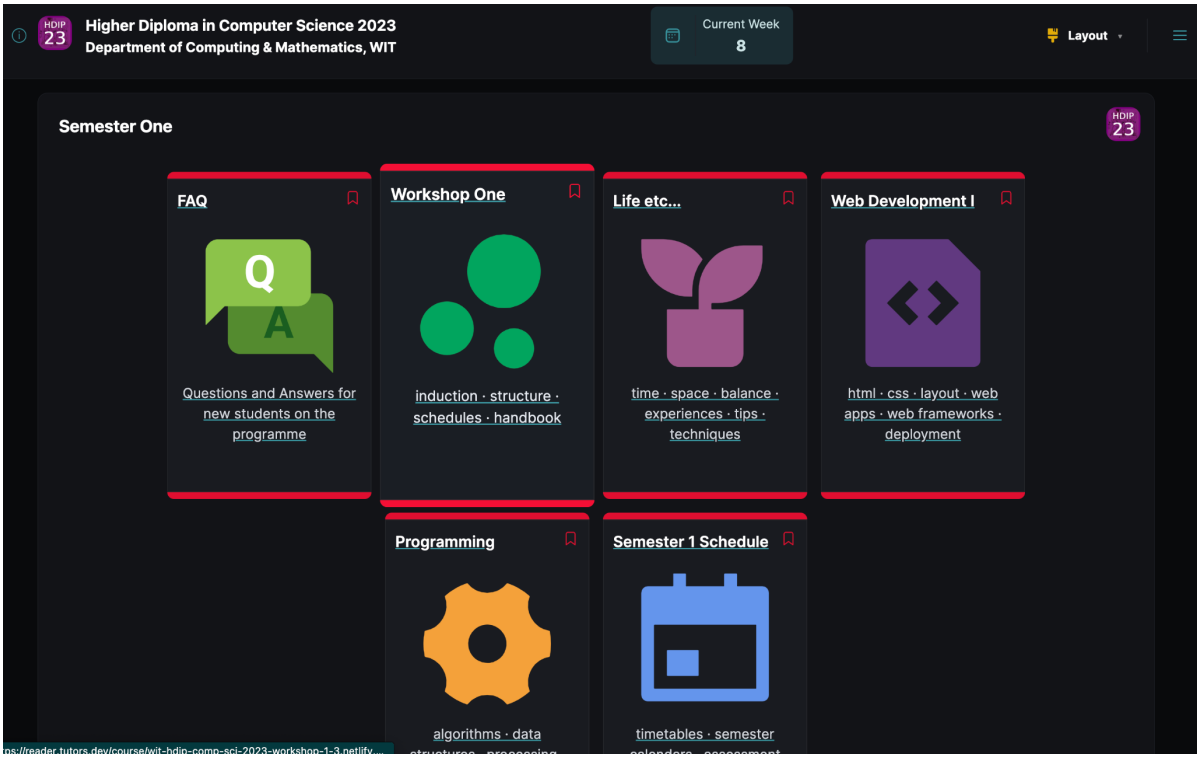
Fig 2.9 My deployment of tutors course via Netifly



2.2.3 The user experience

The Tutors platform gives a clear structure to the user who is consuming the course. They can view the different modules, lectures, labs and video material for a course. Fig 2.11 shows the tutors home page in dark mode.

Fig 2.11 Tutors Homepage in dark mode



The user can select dark or light mode, and also they can choose from different themes, or create their own custom theme (*Fig 2.12*) and *Fig 2.13*

Fig 2.12 Tutors Live theme selection

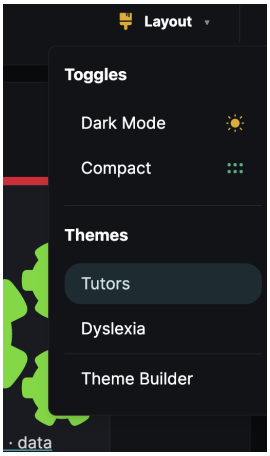
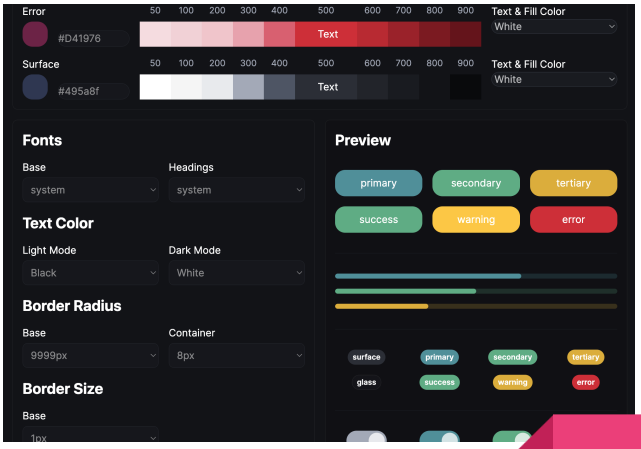
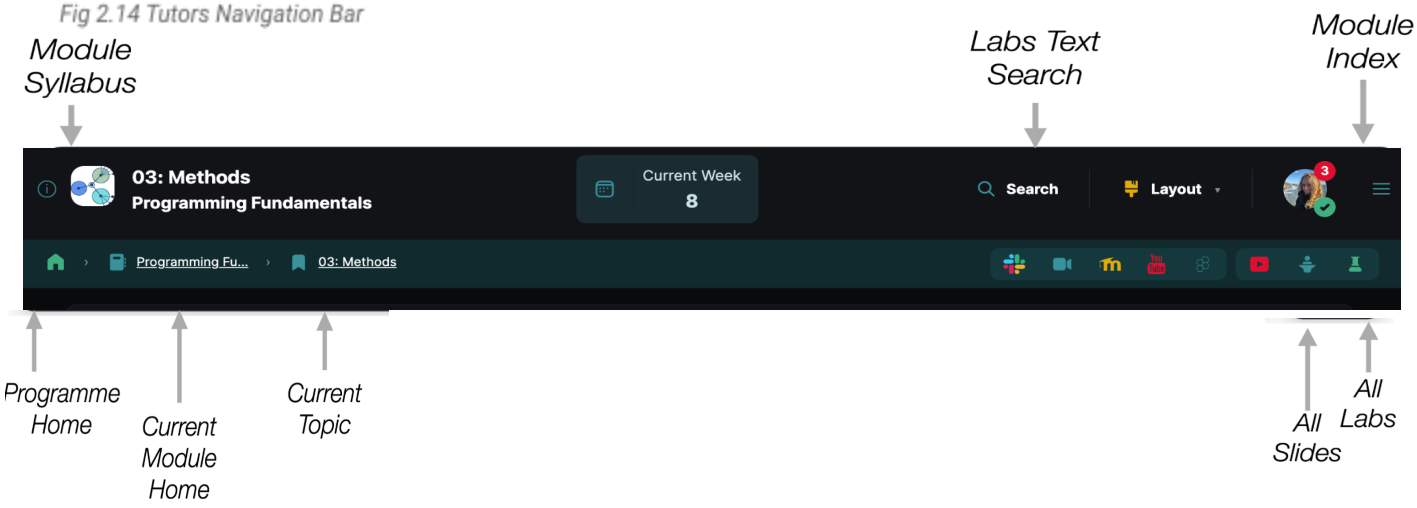


Fig 2.13 Tutors Live custom theme creator



The User can Navigate between courses using the Navigator bar, and they can also view what week they are on and the number of people currently online, along with links to slack, zoom, moodle, youtube, socrative and also the videos, talks and labs within this course so far (See *Fig 2.14* and Appendix 1 which was also referred to in this project)

Fig 2.14 Tutors Navigation Bar



The Avatar image of the user in the top right of the screen shows how many students are online (Fig 2.14) This section, has a dropdown (*Fig 2.15*), which enables the user to see more aspects of the tutors Reader, more specifically, the option to share their presence of being online with other Tutors Users at this time. Fig 2.16 shows the full page view of this, with the student cards and info on each student. This is the Tutors Live aspect of the Reader, which my project is based around, and I will go into more detail in the next section.

Fig 2.15 Tutors Live Dropdown menu presence view

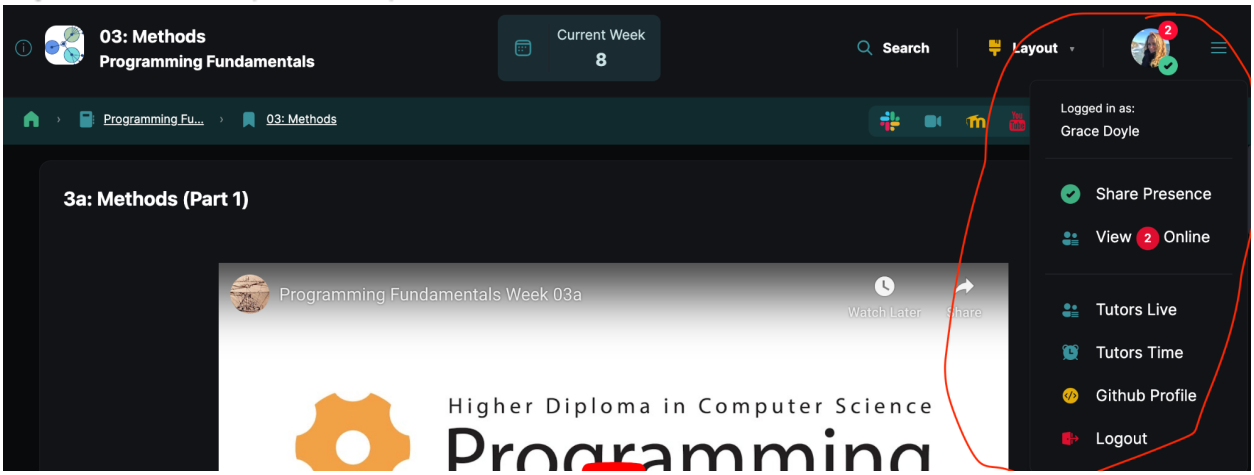
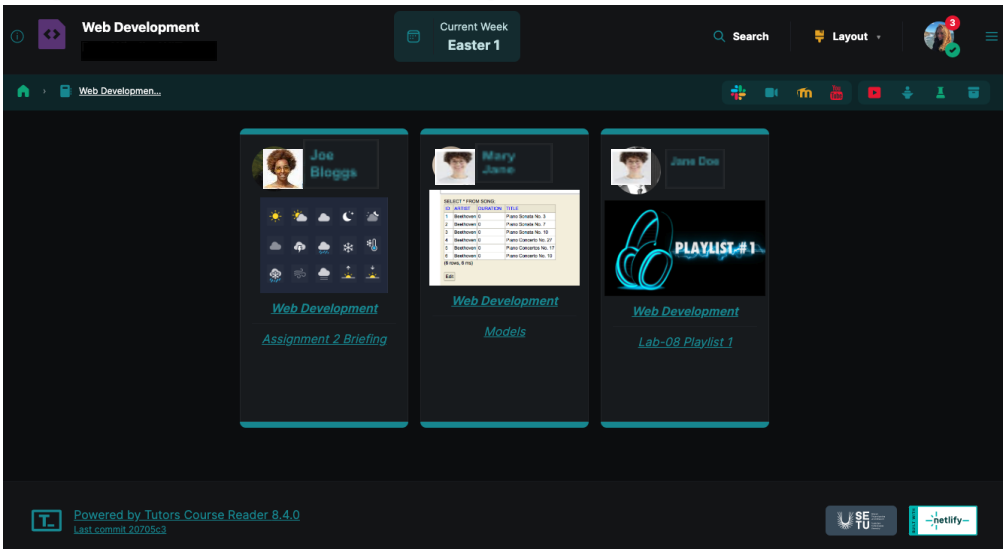


Fig 2.16 Tutors Live - Online Presence Full Page view



When the User is in a module, they see a card with a topic for each week *(Fig 2.17)*

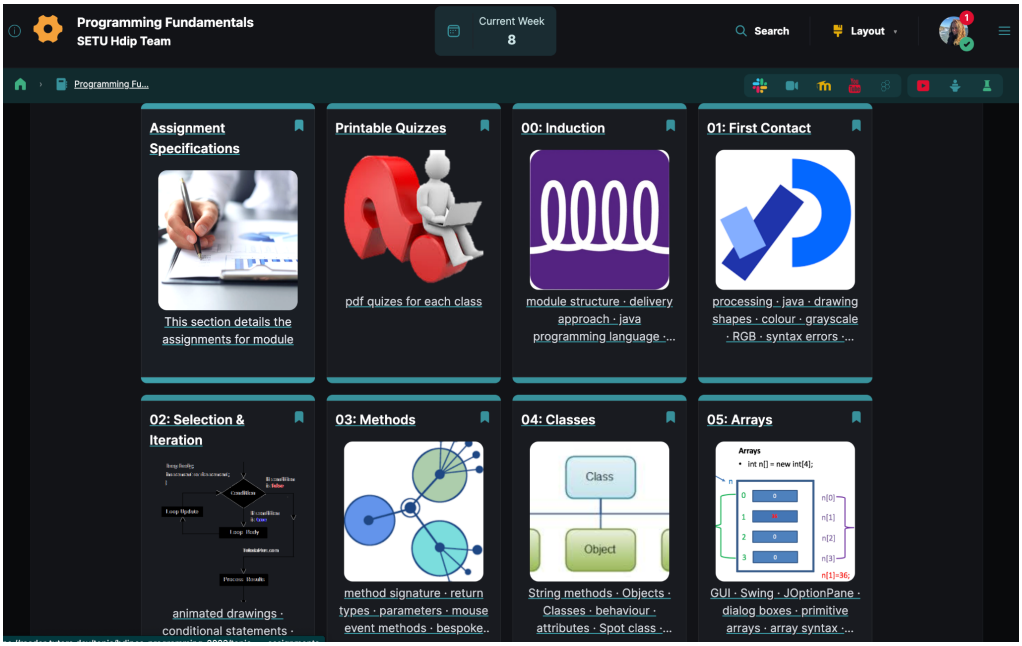


Fig 2.17 Student Weekly Topic view

When a User is within a Topic, each topic has Sub topics with lecture videos and subtopic cards that include sections of the lecture, and sometimes one or more labs *(Fig 2.18)*. Users can Navigate into a Topic, and navigate and select different sub topics and labs within these subtopics.

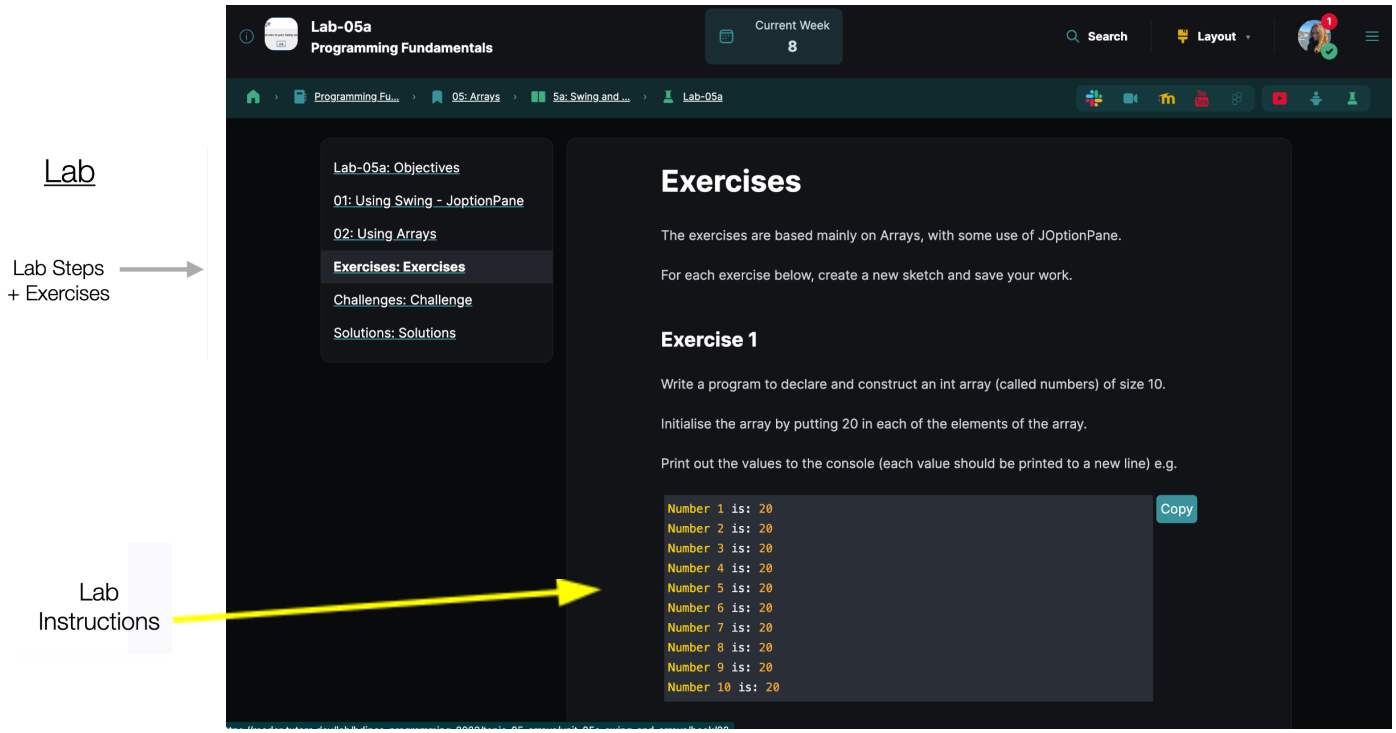
Topic Video

Sub Topic 1

Fig 2.18 Topic view of Video and Sub Topics

When a user clicks on a lab, they can navigate through the tabs on the left that portray the information, instructions, exercise or solutions on the right hand side as seen below in *Fig 2.19*

Fig 2.19 Tutors Lab instructions View



2.2.4 Tutors Live user experience

As *Fig 2.15* in the previous section shows, users on Tutors can view how many are online at any given moment. *Fig 2.21* shows one student online right now from the dropdown menu. Fig 2.22 shows the alternative full page view of the students online presence.. The user has the option not to appear online if they wish and only students who opt to be online will be shown in the cards here.

Fig 2.21 Student online right now from dropdown menu

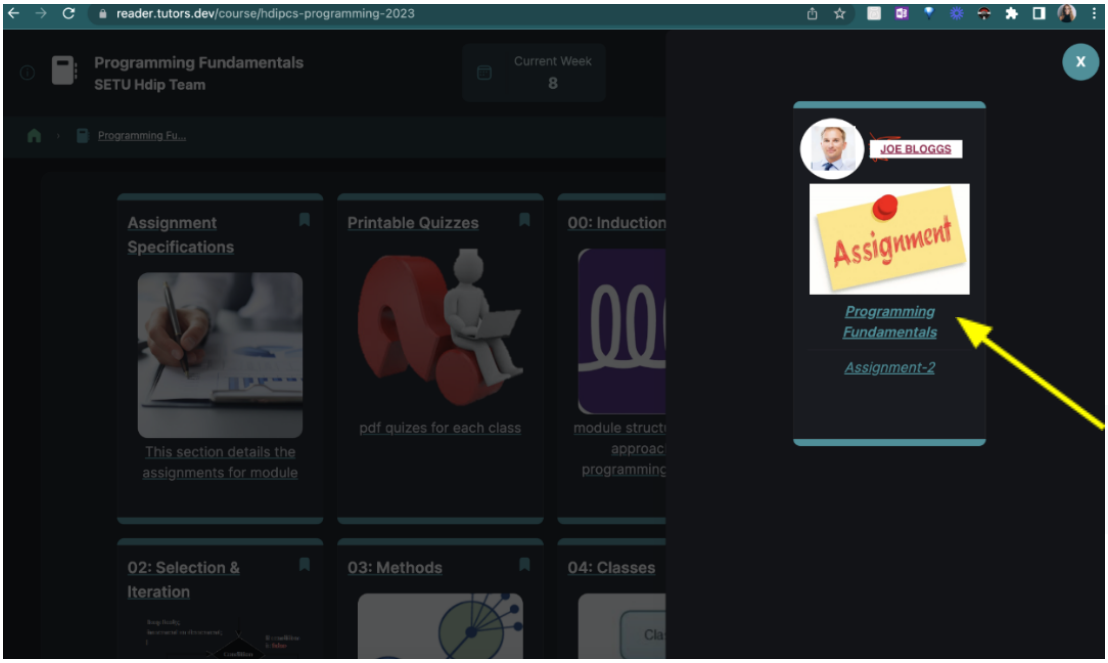
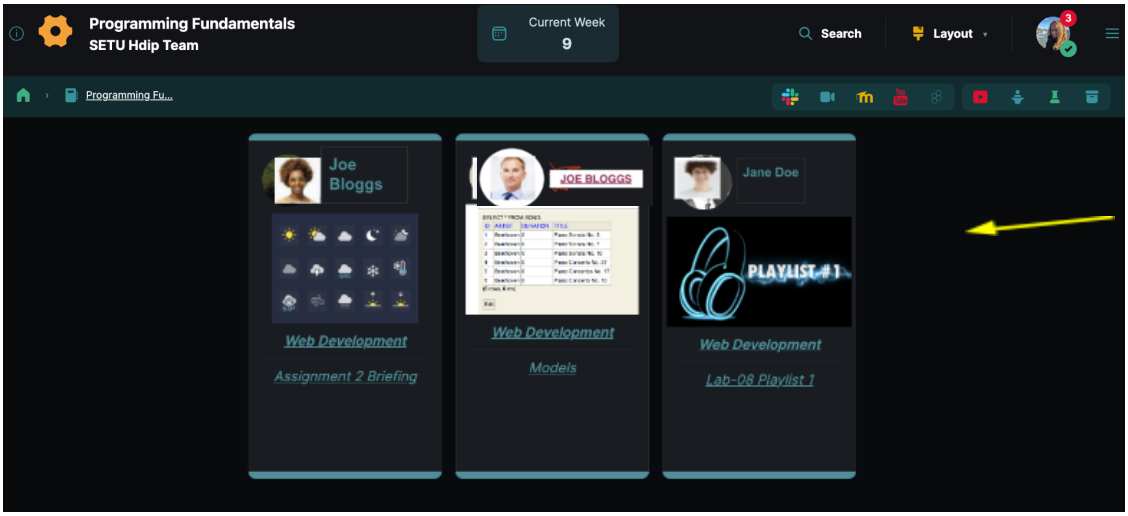


Fig 2.22 Student online right now full page view



This project focuses on this aspect of Tutors, Tutors Live, and aims to improve the online presence capabilities with some changes and additional features with the aim to enhance the learner and educators online educational experience.

3. Tutors Components

3.1 Core Technologies

3.1.1 TypeScript

TypeScript is a syntactic superset of JavaScript which adds static typing. This basically means that TypeScript adds syntax on top of JavaScript, allowing developers to add types (w3schools, 2023).TypeScript is fully compatible with NodeJS and provides several benefits over traditional JavaScript. It exists along with JavaScript code, allowing developers to gradually adopt TypeScript without rewriting their entire codebase. TypeScript enables developers to catch errors early in the development process, reducing the likelihood of bugs and errors in production.TypeScript uses compile time type checking. Which means it checks if the specified types match before running the code, not while running the code. (Monesi, 2021) states that "TypeScript intrinsically encourages us to code cleanly, making the code more scalable. However, projects can contain as much plain JavaScript as we like, so using TypeScript is not an all-or-nothing proposition. (Monesi, 2021) also explains how many IDEs can process information from the TypeScript type system, providing reference validation as we code, while also delivering relevant, at-a-glance documentation (e.g., the arguments a function expects), as we type, for any reference and suggest contextually correct variable names.

3.1.2 Svelte Kit

SvelteKit is a Javascript framework for building high-performance web apps. Whereas Svelte is a component framework, SvelteKit is a meta-framework built on top of Svelte which is an app framework that solves the tricky problems of building something production-ready: Routing, Server-side rendering, Data fetching, Service workers, TypeScript integration, Prerendering, Single-page apps, Library packaging, Optimised production builds, Deploying to different hosting providers (SvelteKit, 2023). The difference between Svelte and SvelteKit is that Svelte is a language, a compiler and a frontend framework, while SvelteKit is the full-stack framework built on top of Svelte.The Svelte language is an association of HTML , CSS and JavaScript. Svelte can also be considered as a component framework, like React or Vue.

SvelteKit apps are **server-side rendered** by default for speed and SEO (search engine optimization) but once the page is loaded the client-side router kicks in which makes your app feel like a Single Page App to avoid reloading the page between page navigation.

SvelteKit uses the web platform which means using web standards like the Fetch API, Request and Response objects, Headers and FormData for working with forms. SvelteKit also helps you create and publish packages if you're working on component libraries using svelte-package.A component library is similar to a SvelteKit app in structure but it uses src/lib as the root and running svelte-package takes the content of src/lib and generates a package directory you can publish on npm (Matija, 2023).

The Svelte team developed SvelteKit as a new approach to building web applications. Developers can leverage Svelte, a popular JavaScript framework for building user interfaces, along with modern build tools and frameworks to achieve a seamless and efficient development process. "SvelteKit enables developers to write components and functionality once and reuse them across different platforms, reducing the time and effort required to develop and maintain web applications." (Svelte Team, 2020)

SvelteKit is what was chosen already as the best fitting technology for the most recent version of Tutors, that has being built on top of Svelte, and provides features, such as server-side rendering and a built in file-based routing system. Eamonn and is responsible for the currently designed Tutors Course Reader to SvelteKit. I will be working on adding my new features to the current design, firstly practicing these builds on my own skeleton demo app.

3.1.3 Skeleton

Skeleton is a small collection of CSS files that can help you rapidly develop sites that look great. As a UI library, Skeleton integrates tightly with Svelte and Tailwind CSS, allowing users to quickly and easily customize and extend each component's appearance and feel.Tutors Course Reader design is created in Skeleton. The tutors -ui library has been built in Skeleton, whereby the components have been used to build out the Svelte application. Skeleton integrates directly with Svelte and SvelteKit's best features, including components, stores, actions, and more. Offering interactive components, image filters, and much more (Skeleton UI, 2023). This was a good choice of UI pattern by Eamonn De Lester, as it improves performance and provides a basic visual structure to a page while the content is loading, helping to improve perceived

performance. The Benefits of using Skeleton is that it is free and open source and it's community owned by contributors, so you are able to contribute if you wish, it has a theme system that allows you to control things like border radius, and apply those universally throughout your application (the theme are also built using CSS variables, which makes themes simple to configure and reuse (Skeleton, 2023). Another benefit include the design tokens that enables you to reuse your theme settings so that your UI elements can automatically adjust to theme changes.

3.1.4 Firebase

Firebase is a platform containing multiple databases, authentication, file storage and more. "Firebase is a comprehensive mobile app development platform that provides a range of tools and services for building high-quality, scalable mobile apps quickly and efficiently." (Google, 2020). With Firebase, developers can focus on creating great user experiences, without having to worry about server-side infrastructure." (Brennan, 2018), An integral part of this current enhanced Tutors Live project is to store user information and course data (such as who is logged in, and what module and topic they are studying), so this Project utilizes the Realtime Database within Firebase, so that my prototype tutors live app has a way of listening for event updates that are written in my simulator app.

In Realtime Database, Firebase data is retrieved by attaching an asynchronous listener to a firebase.database.Reference. The listener is triggered once for the initial state of the data and again anytime the data changes. For basic **write** operations, you can use set() to save data to a specified reference, replacing any existing data at that path. For example in my project I use set() in my writeObj method (*Fig 3.1*) which is then called within handleClick method that is gathering data (*Fig 3.2*), and therefore writes the CourseData variables to Realtime Database which is called Tutors Simulator App *(Fig 3.3)*

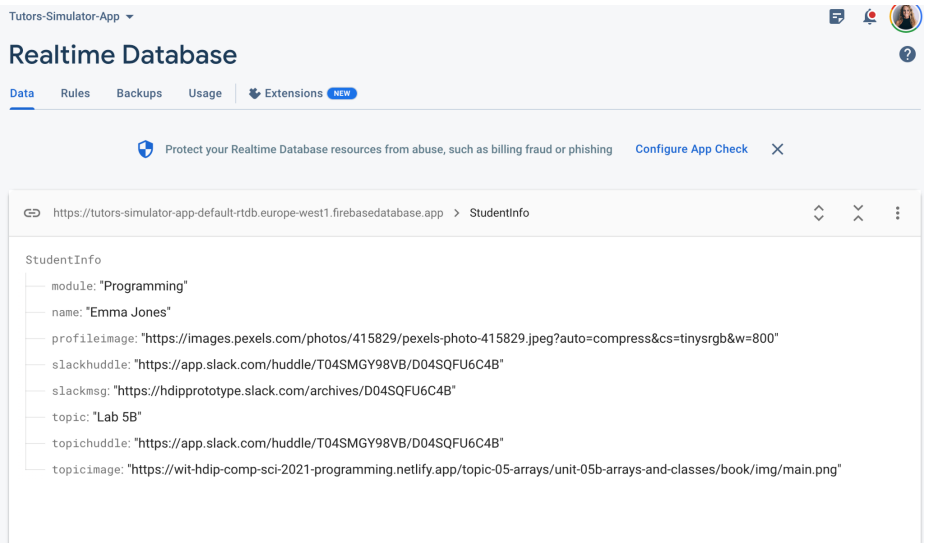
```
export function writeObj(key: string, obj: any) {
    try {
        const db = getDatabase();
        const dbRef = ref(db, key);
        void set(dbRef, obj);
    } catch (error: any) {
        console.log(`TutorStore Error: ${error.message}`);
    }
}
```

Fig 3.1 Write Object Method

```
function handleClick() {
    const studentObj = studentData.find((student) => student.name == nameSelected);
    const topicObj = topicData.find((topic) => topic.topic == topicSelected);
    const courseData = {
        name: nameSelected,
        module: moduleSelected,
        topic: topicSelected,
        topicimage: topicObj.topicimage,
        profileimage: studentObj.profileimage,
        slackmsg: studentObj.slackmsg,
        slackhuddle: studentObj.slackhuddle,
        topichuddle: topicObj.topichuddle
    };
    console.log(courseData);
    writeObj("StudentInfo", courseData);
}
```

Fig 3.2 writeObj method called from handleClick method

Fig 3.3 Date stored inTutors Simulator App (Realtime Database)



To **read data** at a path and listen for changes, use `onValue()` to observe events. This method is triggered once when the listener is attached and also every time the data changes. The event callback is passed a snapshot containing all data at that location, including child data. In my project I listen for changes in my tutors live prototype app (Tutors Hive) and implement this *(Fig 3.4)*

Fig 3.4 Tutors Hive listening for changes

```
onMount(async () => {
    initFirebase(getKeys().firebase);
    let statusRef = ref(getDatabase(), 'StudentInfo');
    onValue(statusRef, async (snapshot) => {
        if (snapshot) {
            // a new event has been seen
            const studentEvent = snapshot.val();
            console.log('student info received');
            console.log(studentEvent);
```

3.1.5 Tailwind

Tailwind is a utility-first CSS framework that provides a set of pre-designed classes to quickly style HTML elements (Barden, 2021). A utility-first approach means using a set of predefined utility classes to build a user interfaces. They are small, CSS classes that are designed to do one thing well. Then, a set of utility classes that can be combined to efficiently and quickly build UI components without writing custom CSS code. Tailwind CSS is a popular utility-first CSS framework that provides a large set of pre-defined utility classes for common styles such as colors, typography, spacing, for example. Instead of writing custom CSS for each component, you can use these utility classes to style your components. The utility-first approach offers several advantages over traditional CSS approaches such as improved consistency, reduced code duplication and file sizes which together can make it a valuable tool for building modern, responsive web applications and makes it easier to create consistent designs across your entire application. Tutors live uses Tailwind CSS framework, which is easy to customize and adapt to any design. Personally I like that it is still CSS but you are using shortcuts like flex and grid so it's much quicker to write and maintain. Tailwind allows developers to build custom user interfaces with ease, offering a comprehensive set of CSS classes that can be combined to achieve almost any design (Smith, 2022).

3.1.6 Components

The following components are some of the components used in Tutors Live that can be found in package.json files of tutors-lib and tutors reader lib;

- **Iconify** is a large set of open source icon components with thousands of validated and up to date high quality icons from 100+ icon sets.
- **Auth0 for authentication** - A flexible way to add authentication and authorization services to your applications.
- **Pdfjs dist -** A Portable Document Format (PDF) generation library. It enables downloading a PDF file from a web server to parse its contents. Once prepared, content is then rendered onto an HTML5 <canvas> element using canvas drawing commands.

These are three of the main add-ons that are being used in Tutors Live, but there are others associated with code quality, and linting. (e.g Eslint-config-custom)

3.2 Tutors Generators

The Tutors Generator *(Fig 3.5)* is reading the Tutors Course. Tutors courses are built up by the **tutors-lib " abstract syntax tree"**. Here it is compiling an entire folder structure into an in-memory model (instead of compiling a single file). Courses and their topics and subtopics are all learning objects.. Topics contain lectures, labs and labs contain sections/chapters. This is the structure of the model that is developed & assembled in-memory by the tutors-lib library *Fig 3.5*

Fig 3.5 Tutors Generator Architecture Diagram (De Leastar, 2022)

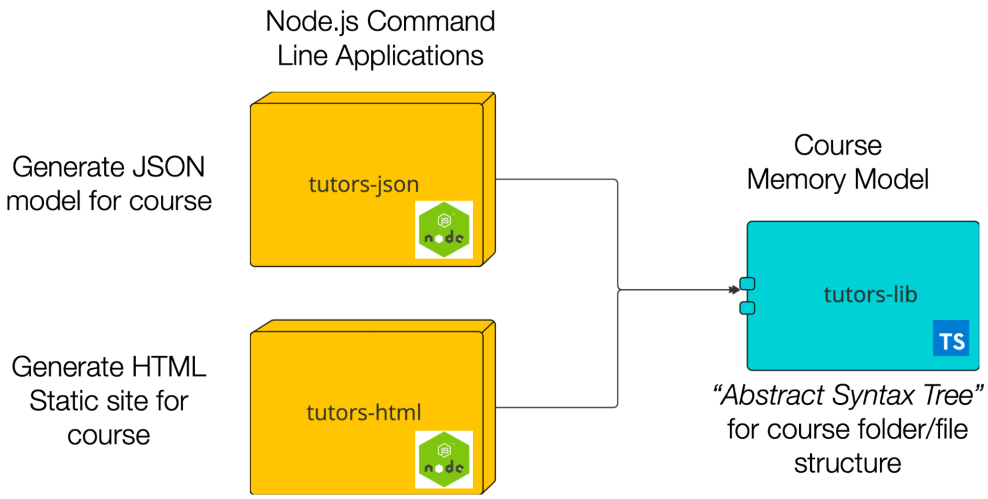
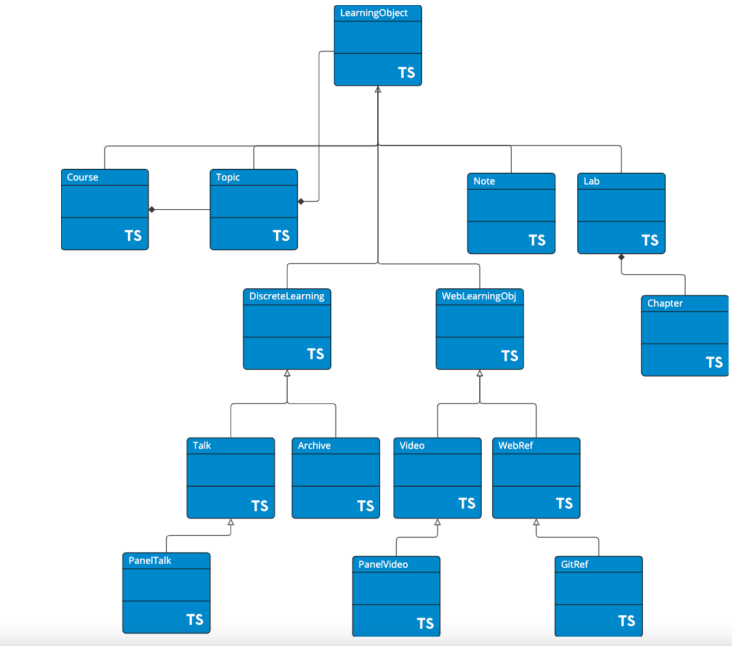


Fig 3.6 *Tutors-lib "Abstract Syntax Tree" (De Leastar, 2022)*



3.2.1 Tutors JSon

Tutors-json writes a JSON representation of all the information on the course such as Module, topics, sub topics, labs, lectures etc and copies it out into a JSON folder, while maintaining the same structure of the course. Tutors-json is the main course generator that transforms the course into the tutors.json file. This file is the latest version of the course itself. It's a short typescript library which follows the abstract tree & produces a representation in a single json file.

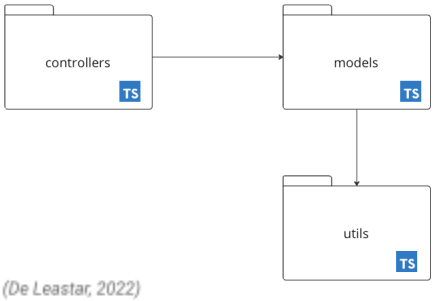
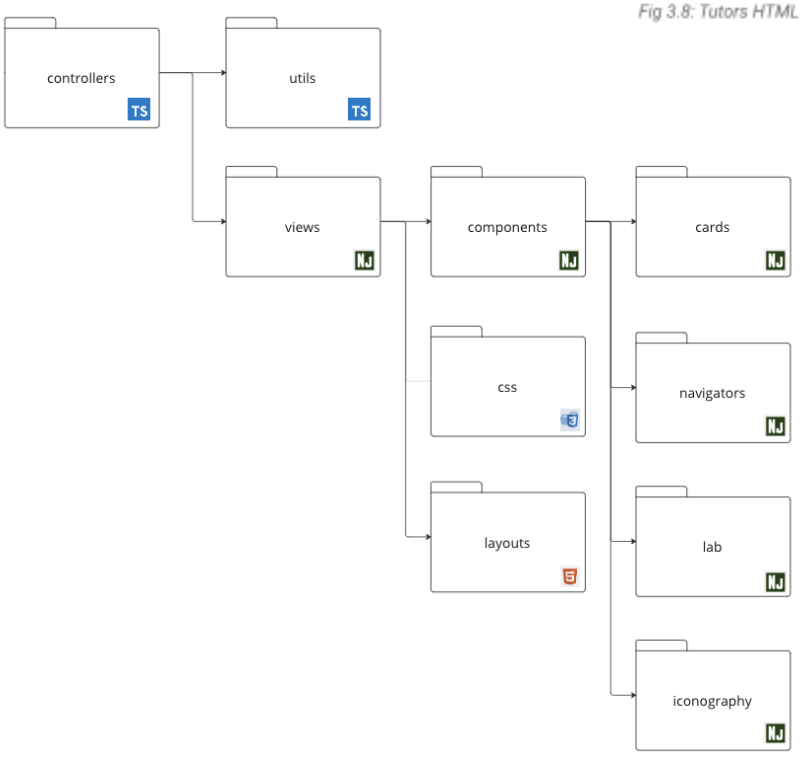


Fig 3.7: Tutors JSON Generator File structure (De Leastar, 2022)

3.2.2 Tutors HTML

At the same time, the Static HTML site generator is doing a similar implementation of the tutors JSON, by creating same copy of all the components/elements of the course, except it is generating a HTML file instead of JSON, as it is a static site generator. The structure of tutors HTML is a little more complex than Tutors JSON, as it has all the styles incorporated with each page and it generates more content as it has to generate the user experience as a static standalone site.

Fig 3.8: Tutors HTML Generator File structure (De Leastar, 2022)



3.3 Tutors Reader

Tutors Reader is a software development kit (SDK) for the Tutors platform. The Tutors Reader SDK is a set of libraries and tools that developers can use to interact with the Tutors platform and build custom applications that integrate with it. Developers can access data and functionality from the Tutors platform, such as course content, assessments, student progress, and more, and use this information to build custom applications that meet the specific needs of their online learning platform. The tutors-reader-lib package is where the course is downloaded with information such as the course, topic, module, presence, authentication, images and icons per student card. THe tutors reader is essentially the main front-end application of the Tutors project, that renders a course for the student from the generated JSON complied from the tutors-json package.

The Tutors Reader SDK is open source and available on GitHub , allowing developers to contribute to its development and extend its functionality as needed.

3.3.1 Architecture of the reader

The Tutors Course Reader is what the user sees, i.e, the front-end application of the whole tutors project. This is a web application hosted on Netifly and renders the student and course info of those students that are present on-screen. The Tutors Course Reader is built with SvelteKit using the two components - **tutors ui** (the UI design elements that render the core elements of the reader) and **tutors-reader-lib** (a typescript library that is the logic behind the Tutors Course Reader (*Fig 3.9)*. It is responsible for loading the course info from the remote server and how the metrics (used to track student progress) and presence engine model(which is used to track online status) is rendered from *(Fig 3.11)*

Fig 3.9 Tutors Course Reader (Overview 1), (De Leastar, 2022)

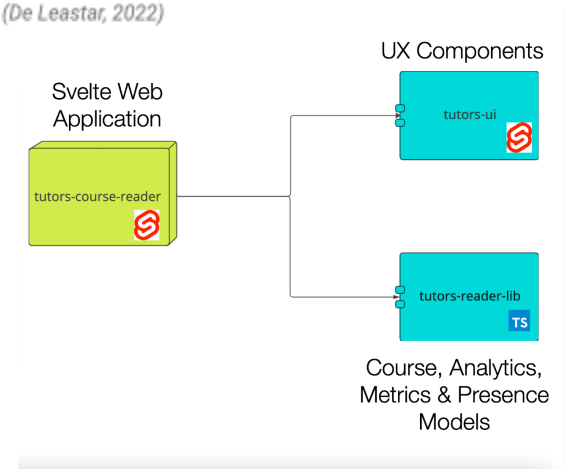
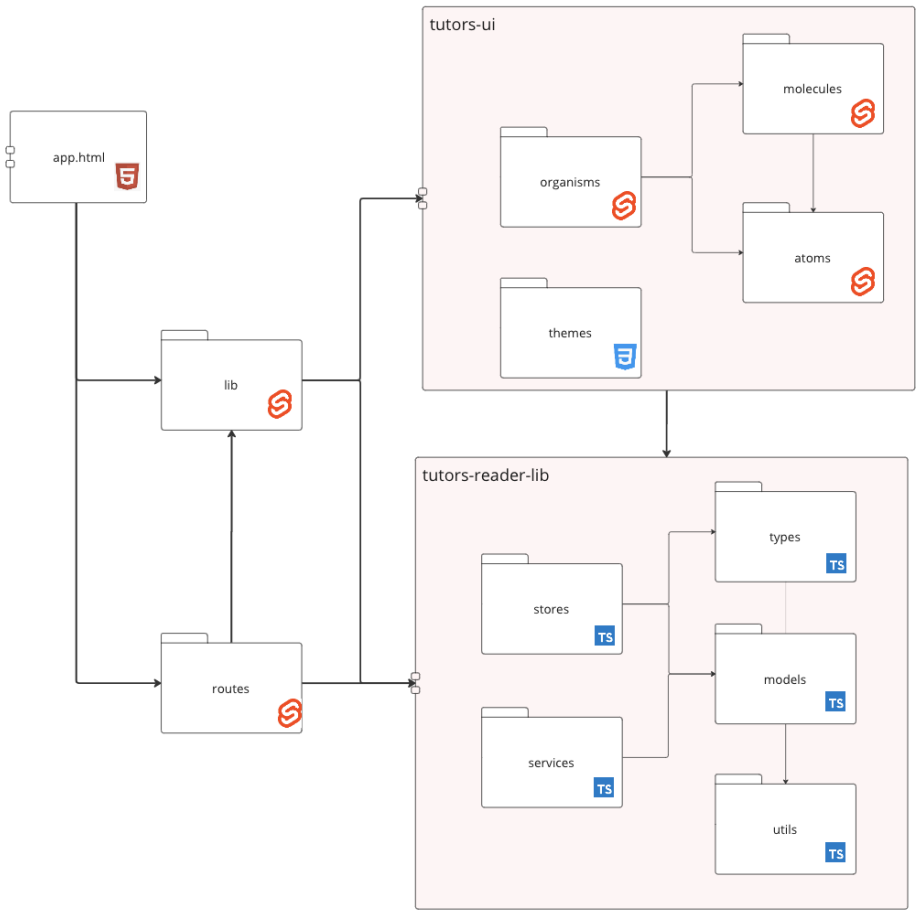


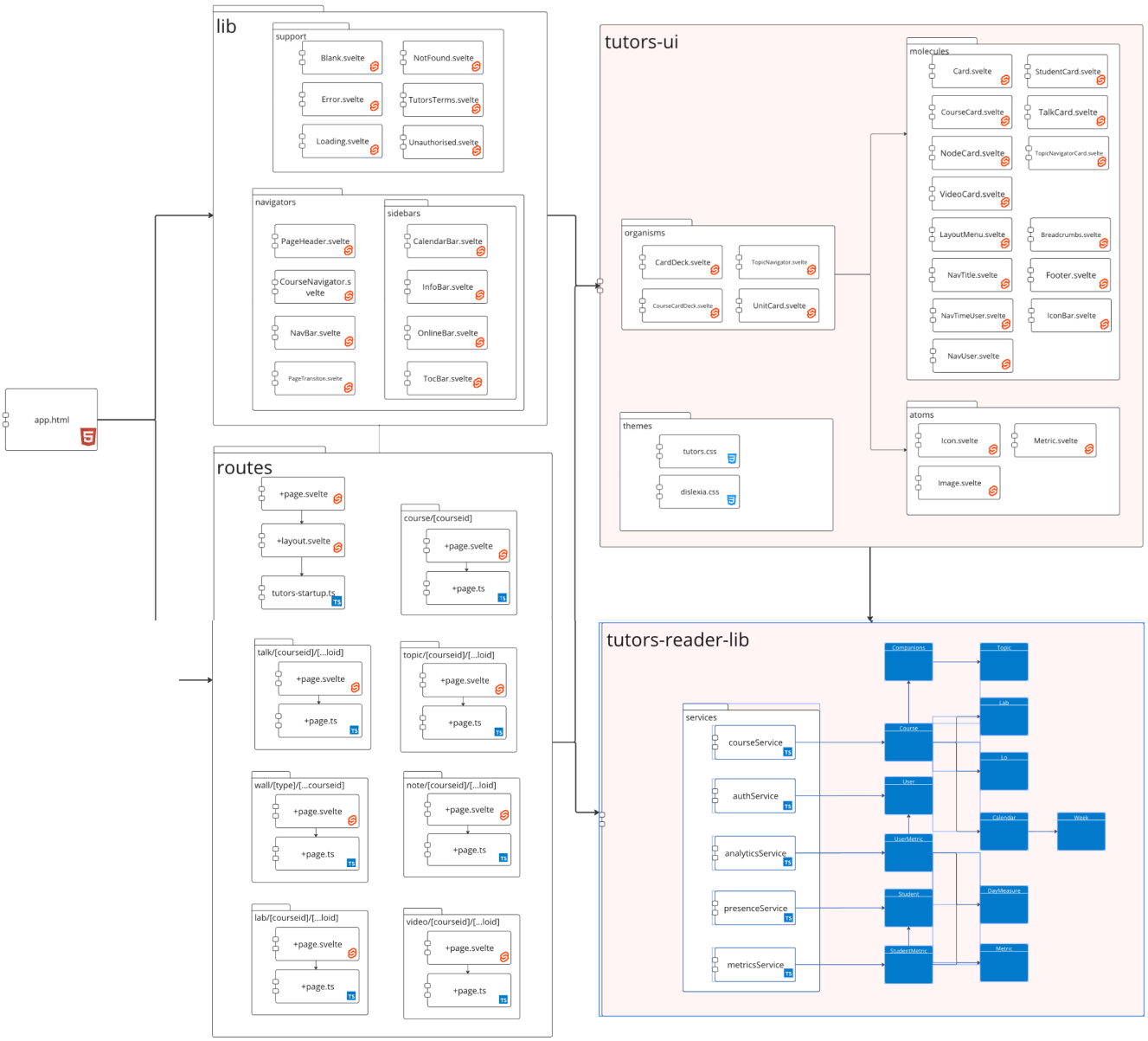
Fig 3.11 Tutors Course Reader breakdown (Overview 2), (De Leastar, 2022)



The tutors-ui package is the design system for the Tutors project. It is built on top of Skeleton, a UI library built specifically for SvelteKit. Skeleton is based on TailwindCSS, and provides a lot of components, utilities & stores which we use to build out the Tutors Course Reader.

Inside the course-reader, we have Pages and Routes *(Fig 3.12)*. Inside the routes folder we have a component for each user interface element on the screen (talk, topic, wall, note,lab, course and video.

Fig 3.12: Tutors Course Reader detailed structure (Overview 3), (De Leastar, 2022)

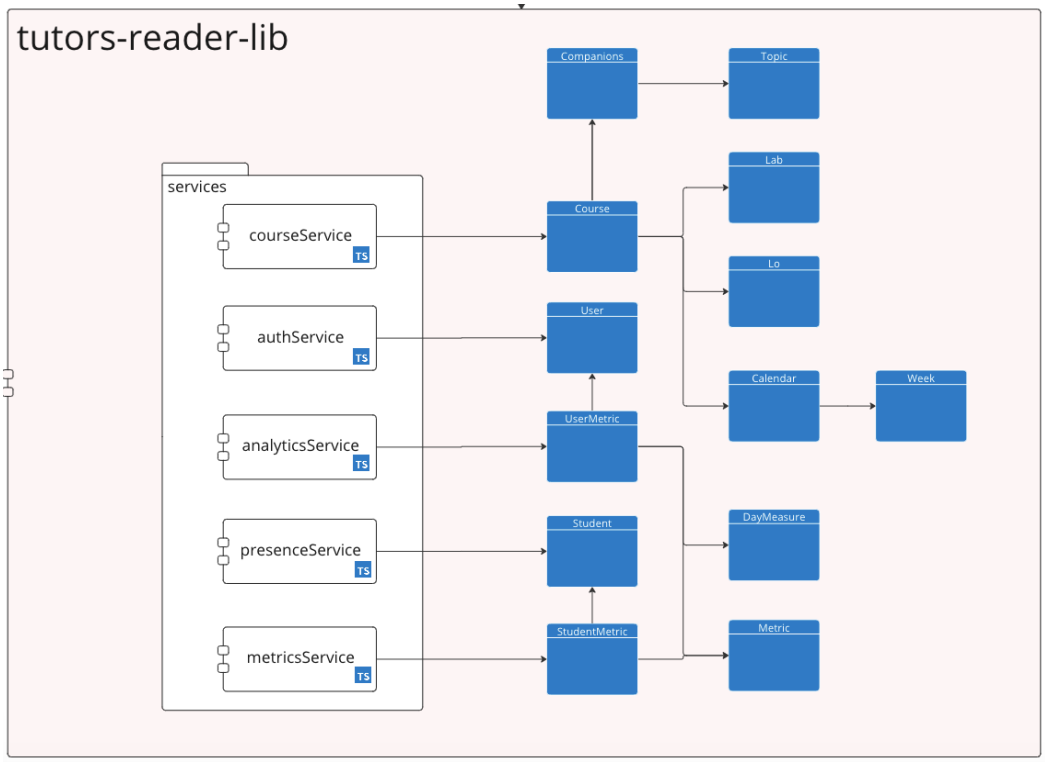


3.3.2 Tutors Reader Lib

Tutors-reader-lib is a typescript library that is the logic behind the Tutors Course Reader. The Tutors Reader Lib is a package that downloads the course from the remote server and generates, to the front-end browser, a model of the course. This breaks down into the course module, topic/lab, authentication, metrics, and presence along with stores for the course, icons ,

learning objects, metrics and stores *(Fig 3.13)*. The focus here is on analyticsService and the presenceServiceI for the scope of this project. I used the analytic-service section of the **tutors-readers-lib** to guide me in building my own prototype of tutors live presence engine.

Fig 3.13: tutors-reader-lib architecture, (De Leastar, 2022)



3.3.3 Tutors Analytics / Datastore structure

One of the two main parts of the current Tutors Reader that I focused on for this project was the analytic-services file within the services folder in **tutors-reader-lib**. *(Fig 3.14)*

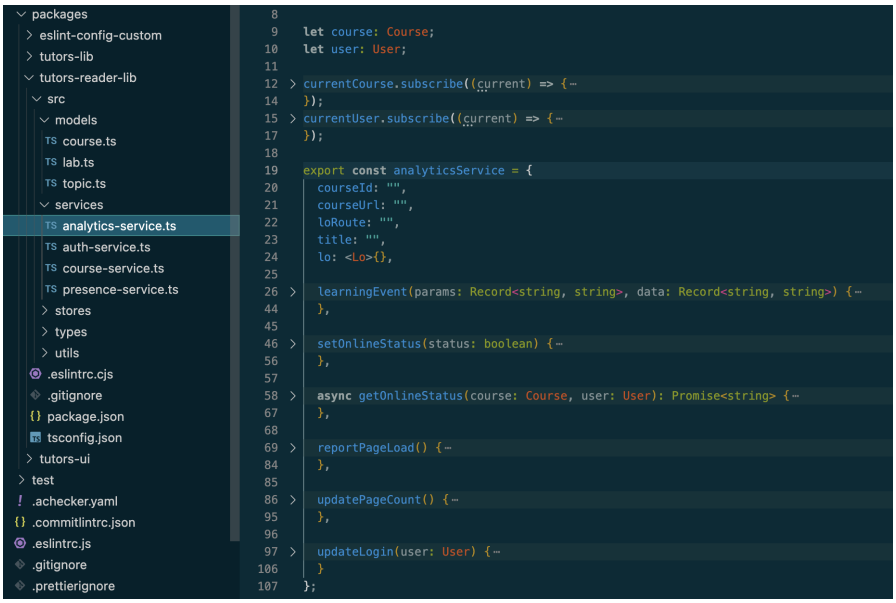


Fig 3.14: analytics-service.ts

The overall purpose of this part of the code is that whenever the user clicks on a link, clicks on a card or clicks on a step in a lab for example, it generates a learning event. (See *Fig 3.15 learningEvent method*). The reader calls this function from the main application *(Fig 3.16)* whenever something happens and the very last line of the code **reportPageLoad()** is called anytime the user changes anything, which ultimately calls **updateLO** and update LO is writing a course, (an event) to the data store.

Fig 3.15: learningEvent method

```
26    learningEvent(params: Record<string, string>, data: Record<string, string>) {
27      this.courseUrl = params.courseid;
28      if (this.courseUrl.includes(".")) {
29        this.courseId = params.courseid.substring(0, params.courseid.indexOf("."));
30      } else {
31        this.courseId = params.courseid;
32      }
33      this.loRoute = "";
34      if (params.loid) {
35        this.loRoute = sanitise(params.loid);
36      }
37      this.lo = data.lo;
38      if (this.lo) {
39        this.title = this.lo.title;
40      } else {
41        this.title = course.lo.title;
42      }
43      this.reportPageLoad();
44    },
```

Fig 3.16: main application calls the learningEvent

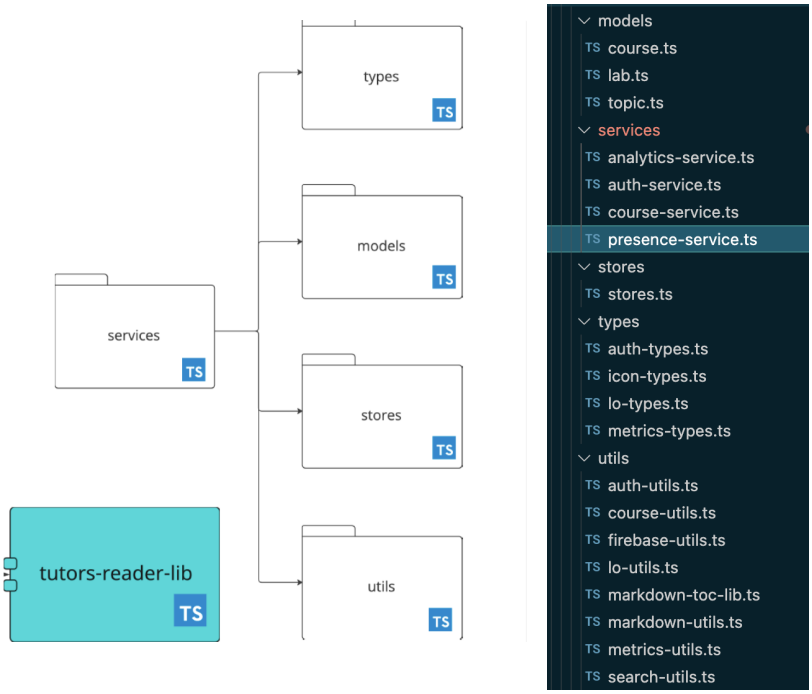
```
50        analyticsService.learningEvent(path.params, path.data);
```

Fig 3.17 line 76 shows that if any user clicks on a card/lab/topic, this action then translates into a call to this updateLo method. This method is updating the learning object. This is ultimately what my simulation entails, as when I write a courseEvent, the same thing happens as here, except here, a learning object is created (as opposed to my courseEvent).

Fig 3.17: analytics-service.ts Line 76

```
69    reportPageLoad() {
70      updateLastAccess(`${this.courseId}/usage/${this.loRoute}`, this.title);
71      updateVisits(this.courseId);
72
73      if (!user || (user && user.onlineStatus === "online")) {
74        updateLastAccess(`all-course-access/${this.courseId}`, this.title);
75        updateVisits(`all-course-access/${this.courseId}`);
76        updateLo(`all-course-access/${this.courseId}`, course, this.lo);
77      }
78
79      if (user) {
80        const key = `${this.courseId}/users/${sanitise(user.email)}/${this.loRoute}`;
81        updateLastAccess(key, this.lo.title);
82        updateVisits(key);
83      }
84    },
85
```

3.3.4 Architecture of the Presence Engine



As Tutors Live is so large, my main focus is on particular parts of the course, mainly the analytic-service in the previous section and this presence-service section. It is essentially the presence engine of tutors live that generates the tutors live experience for the users. This file can be found in **tutors-reader-lib** > **services > presence-service** (*Fig 3.18)*

Fig 3.18: Architecture of the Presence Engine (De Leastar, 2022)

As Tutors live website has clickable aspects/links, these links/clicks are being transferred into a database consisting of currentEvents and the presence engine is listening for those events and trying to figure out what happened and then updates on the screen with the updated or new card information. So in other words, **presence-service.ts** is a part of Tutors Live where it is listening

for any updates to the eventStore for this course. In this section of the code you can see how the startListening, stopListening and updateListener are all part of the presenceService function within **presence-service.ts.**

Fig 3.19: Presence-engine is listening for eventStore for this course

```
export const presenceService = {
  db: {},
  user: <User>{},
  userSummaryCache: new Map<string, UserSummary>(),
  course: Course,
  lastCourse: Course,
  students: new Map<string, StudentLoEvent>(),
  los: new Array<StudentLoEvent>(),
  listeners: new Map<string, StudentLoUpdate>(),

  startListening(key: string, listener: StudentLoUpdate) {
    this.listeners.set(key, listener);
  },

  stopListening(key: string) {
    this.listeners.delete(key);
  },

  updateListeners(kind: string, event: StudentLoEvent) {
    this.listeners.forEach((listener, key, map) => {
      listener(kind, event);
    });
  },
```

The **visitUpdate** function of the presence-service.ts file is a function whereby, as the courses are updated, the online presence is updated and updated on the student card also.

In line 49 *(Fig 3.21)* we are getting the Learning Object (lo) that has just happened. In this method we are getting more information on the user, creating a new learning object that is then

going to write to the current learning object (name, id, title, image etc) and then these are kept in an array and updated here. This is effectively what I rebuild, in simpler terms, when building my own presence engine, which is explained in the next section of the project..

Fig 3.21: presence-analytics: visitUpdate function breakdown

```
47  async visitUpdate(courseId: string) {
48    const lo = await (await get(child(ref(this.db), `all-course-access/${courseId}/lo`))).val();
49    if (lo) {
50      const userId = decrypt(lo.tutorsTimeId);
51      if (userId && this.user.email !== userId) {
52        const user = await readObj(`${courseId}/users/${sanitise(userId)}`);
53        if (user) {
54          const event: StudentLoEvent = {
55            studentName: user.name,
56            studentId: userId,
57            studentImg: user.picture,
58            courseTitle: lo.courseTitle,
59            loTitle: lo.title,
60            loImage: lo.img,
61            loRoute: lo.subRoute,
62            loIcon: lo.icon,
63            timeout: 2
64          };
65          const studentUpdate = this.students.get(userId);
66 >        if (!studentUpdate) {…
70 >        } else {…
77          }
78          studentsOnlineList.set([...this.los]);
79          studentsOnline.set(this.los.length);
80        }
81      }
82    }
83  },
```

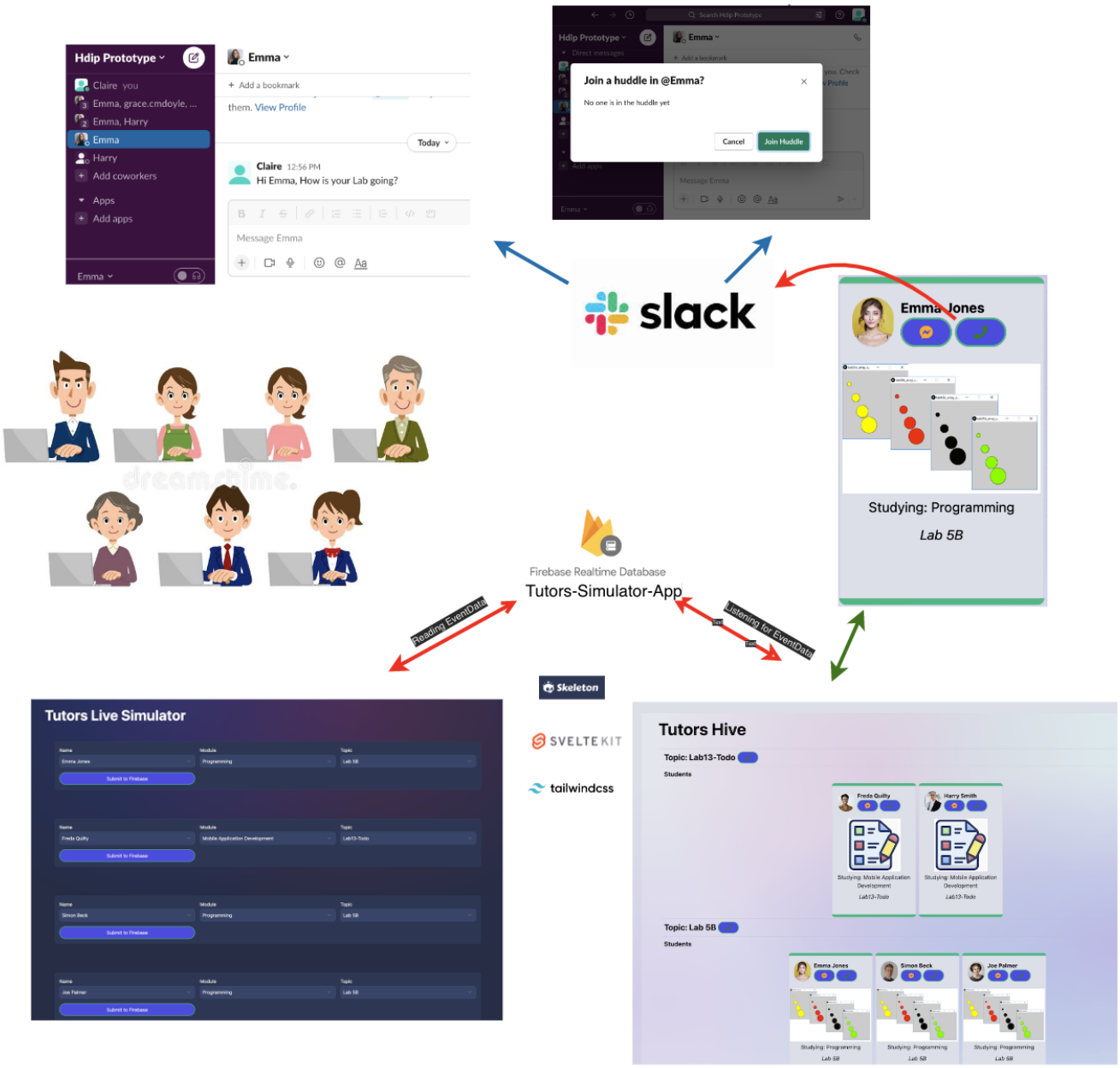
4. Project Architecture

4.1 Presence Engine Overview

As mentioned in my introduction, the aim of this project is to enhance the Tutors live online educational experience for both the educator and the learner by building my own prototype presence engine. The reason I wanted to build my own presence engine and did not work on the current one, is because Tutors Live is a very large system with a complex file structure, with about 10,000 lines of code which is refreshed and updated regularly, not to mention that there are seven hundred students using it all the time. I had my own innovations.idea for this project that I wanted to experiment with, which included regrouping the card structure on the front-end based on the topic the students were studying at any given time, linking to slack huddle and to slack messenger, therefore I felt it best to build a simulator app to simulate student data and I also built my own presence-engine (Tutors Hive). These two apps are my own sandbox which builds a simulator/replica of what I wanted in an independent app at first, and then eventually Tutors Hive could be merged into the main tutors reader down the line once I have evolved the kind of patterns that I think I am going to need to build and the overall architecture that I am going to evolve. Fig 4.1 shows the overall overview of Tutors Hive. It shows that the simulation app inputs the student Info, which includes, name ,profile image, Module and Topic they are currently studying, Topic Image, and their slack links for messages and huddles.

Fig 4.1 explains how students who are studying the same topic are categorized into the topic group. If they switch topics, their topic info will be updated and they will be moved to the relevant topic group. The student cards in Tutors Hive have additional Call and Message buttons, which, when clicked, opens up a direct slack message, or a direct huddle call to that person. There is also a Topic group huddle button per topic, which means the user can call the entire group of people that are studying that topic. The people in the group can opt whether they want to answer or not.

Fig 4.1 Tutors Hive Project Overview

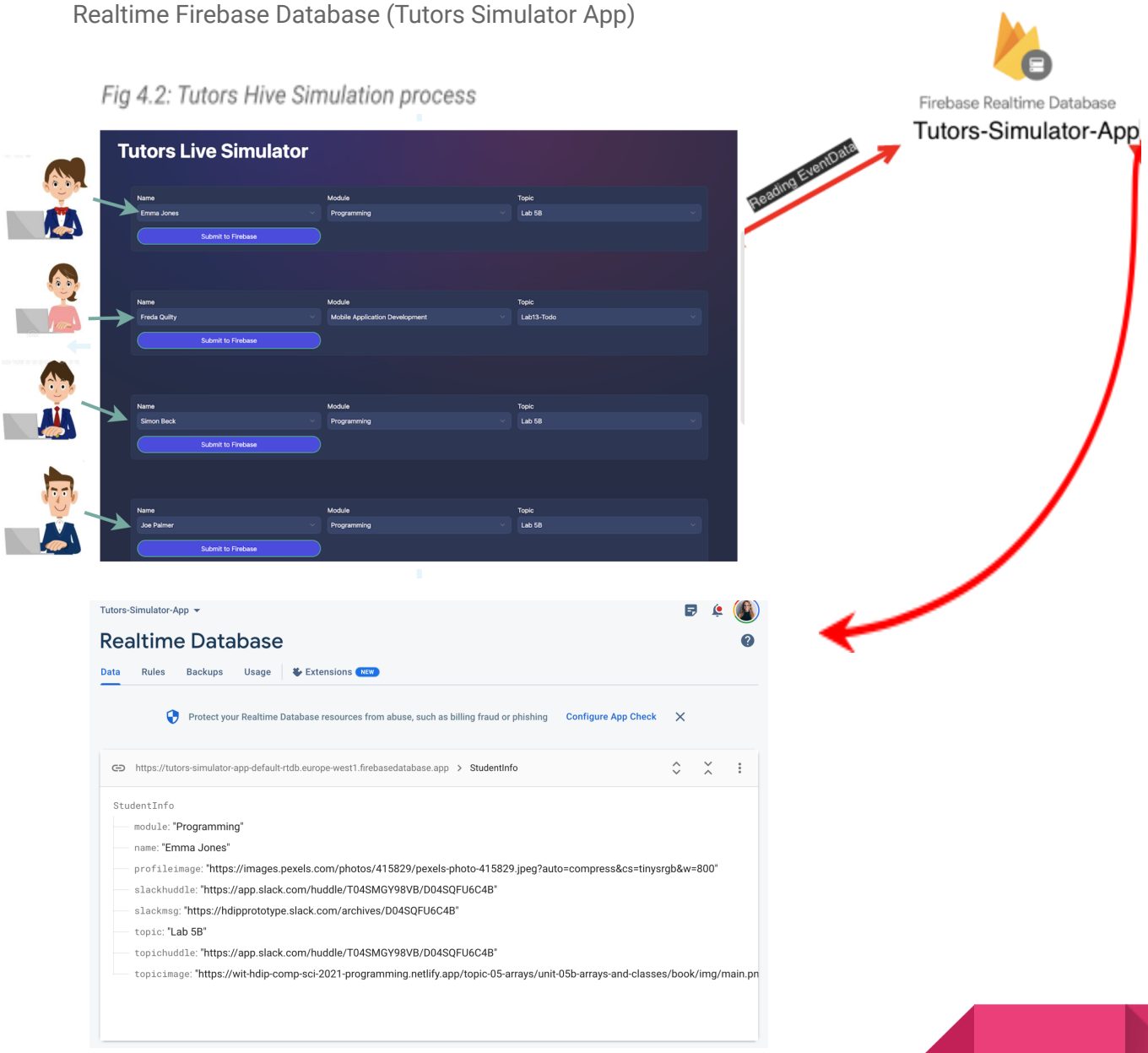


4.2 Tutors Live Simulator

The Tutors Live Simulator was built for this project, with the purpose of easily simulating learning events (courseData) which includes student card details such as, what module and topic they are studying and the corresponding images. These learning events to the Tutors Simulator App (Realtime firebase database).

The students info is inputted into the simulator app, including Student name, Module and Topic (Fig 4.2) with each entry being a new courseEvent. Each input of data into the simulator app, is pulling the appropriate data from the JSon data in the simulator app, which is all put together as a courseEvent. This courseEvent is then wrote to the
Realtime Firebase Database (Tutors Simulator App)

Fig 4.2: Tutors Hive Simulation process



Tutors Live Simulator data is stored within the **data.js** file in the simulator app, and each piece of data has other relevant data linked to it such as their slack messenger and huddle links and images (Fig 4.3).The slack links are links that consist of the individual's slack ID and that with the relevant links with IDs attached, slack messages and slack huddles can be launched externally. When the user clicks on the link. This opens the slack app in a new browser tab as seen in Fig 4.1.

Fig 4.3: Simulator App - data.js

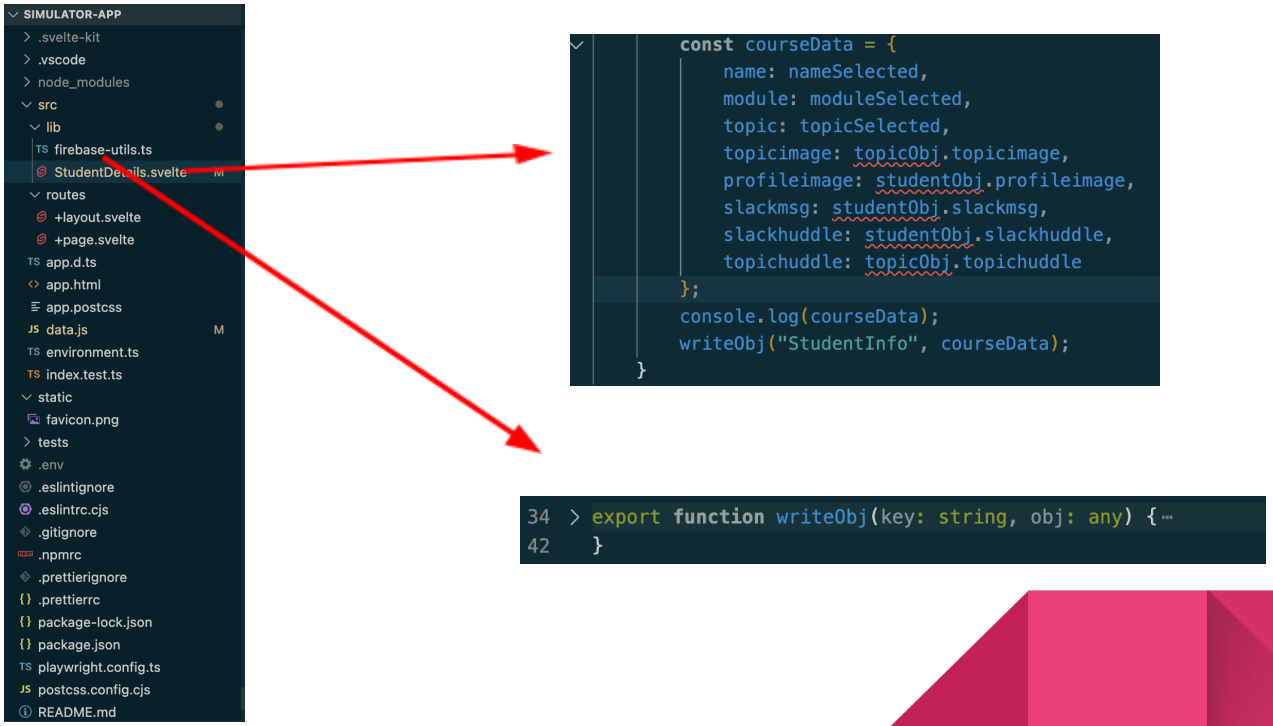
```
{
    name: 'Emma Jones',
    slackmsg: 'https://hdipprototype.slack.com/archives/D04SQFU6C4B',
    slackhuddle: 'https://app.slack.com/huddle/T04SMGY98VB/D04SQFU6C4B',
    profileimage: 'https://images.pexels.com/photos/415829/pexels-photo-415829.jpeg
},
```

```
export const moduleData = [
    'Programming',
    'Mobile Application Development',
    'ICT Skills2',
    'Full Stack Web Development'
];
```

```
{
    topic: 'Lab 5B',
    topicimage:
        'https://wit-hdip-comp-sci-2021-programming.netlify.app/topic-05
    topichuddle: 'https://app.slack.com/huddle/T04SMGY98VB/D04SQFU6C4B'
},
```

The StudentDetails page in the Tutors Live Simulator App gathers all of this student data into an object called courseData and then writes the object to the Database, as "Student Info". The writeObj function is called from the **firebase-utils.ts** (line 34). Fig 4.4 shows these along with the folder structure of the Tutors Live Simulator App.

Fig 4.4: Tutors Simulator App structure and code that writes student data to Realtime Firebase



Also, in the studentDetails page, the input values are selected from a drop pre-made list of data which is being fed in from the data.js file. The For loop here (Fig 4.5) is meaning that Each time a new entry is input, a new set of CourseData is writing that new “Student Info” object to the Tutors-Simulator App Database (See Database in Fig 4.2)

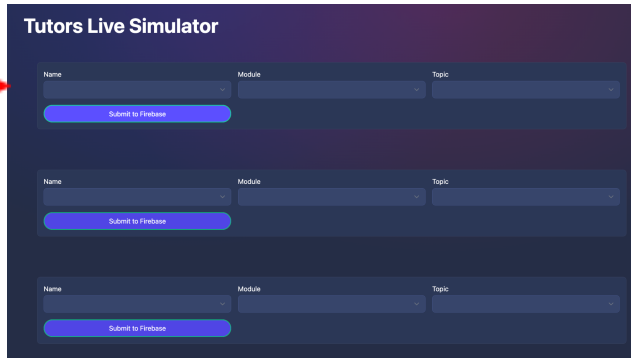
Fig 4.5: studentDetails page - For Loop

```
<span> Name</span>
<select id="name" bind:value={nameSelected} on:change={() => (name = "")} class="select">
    {#each studentData as student}
        <option value={student.name}>
            {student.name}
        </option>
    {/each}
</select>
```

The Page.svelte file in the Tutors Live Simulator is then calling multiple of these same cards at once on the homescreen of the app, *(Fig 4.6)* so that multiple objects can be written quickly and therefore multiple studentCards can be created on the front-end Tutors Hive.

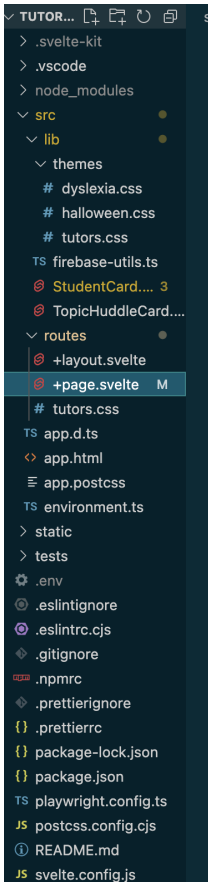
Fig 4.6: Simulation page creating multiple students at once to be displayed on Tutors Hive

```
<div class="container mx-auto p-8 space-y-8">
    <h1>Tutors Live Simulator</h1>
    <StudentDetails />
    <StudentDetails />
    <StudentDetails />
    <StudentDetails />
    <StudentDetails />
    <StudentDetails />
    <StudentDetails />
    <StudentDetails />
</div>
```



4.3 Tutors Hive

Fig 4.7 Tutors Hive Folder Structure



Tutors Hive is the front-end user interface for this project that was built to replicate the Tutors Live presence engine feature. This app is a simpler version, so that it is easier to experiment with new code and experiment building new features. The three main files being dealt with in this architecture *(Fig 4.7)* is the **page.svelte** file, StudentCard and TopicHuddleCard.

The Tutors Hive app has a more complicated set of code, as the goal was to make sure that when a new studentCard appears, it is grouped based on topics in a StudentMap, whereby there could be multiple studentMaps within a topics Map. Fig 4.9 visually shows that as a student appears online, depending on what Topic they are studying, they are being filtered into the relevant studentMap, which is sitting within the topics map. Fig 4.8 shows the snippet of code within **page.svelte** that relates to this. Here we can see a nested for loop, whereby the outer for loop is searching through the topics map and making a new topic if it doesn't already exist and adding the topicHuddle card for each new topic (topicName), and the inner loop is showing how within this topics map, for each topic, studentCards are added to the studentMap within this topic. Each topic being studied has a studentMap and each topic is part of the topics map

*Fig 4.8: Code snippet within **page.svelte***

```
58      <h1>Tutors Hive</h1>
59      {#key reload}
60          <section class=" space-x-2">
61              {#each [...topics] as [topicName, topicMap]}
62                  <hr />
63                  <h3 class="p-2">Topic: {topicName}
64                      <TopicHuddleCard {topicMap} />
65                  </h3>
66                  <hr />
67                  <h5 class="p-2">Students</h5>
68                  <div class="flex justify-center">
69                      {#each [...topicMap] as [studentName, studentRecord]}
70                          <Card courseEvent={studentRecord} />
71                      {/each}
72                  </div>
73              {/each}
74          </section>
75      {/key}
```

Fig 4.9: Visual of students being filtered by topic into relevant studentMap, sitting within the topics map

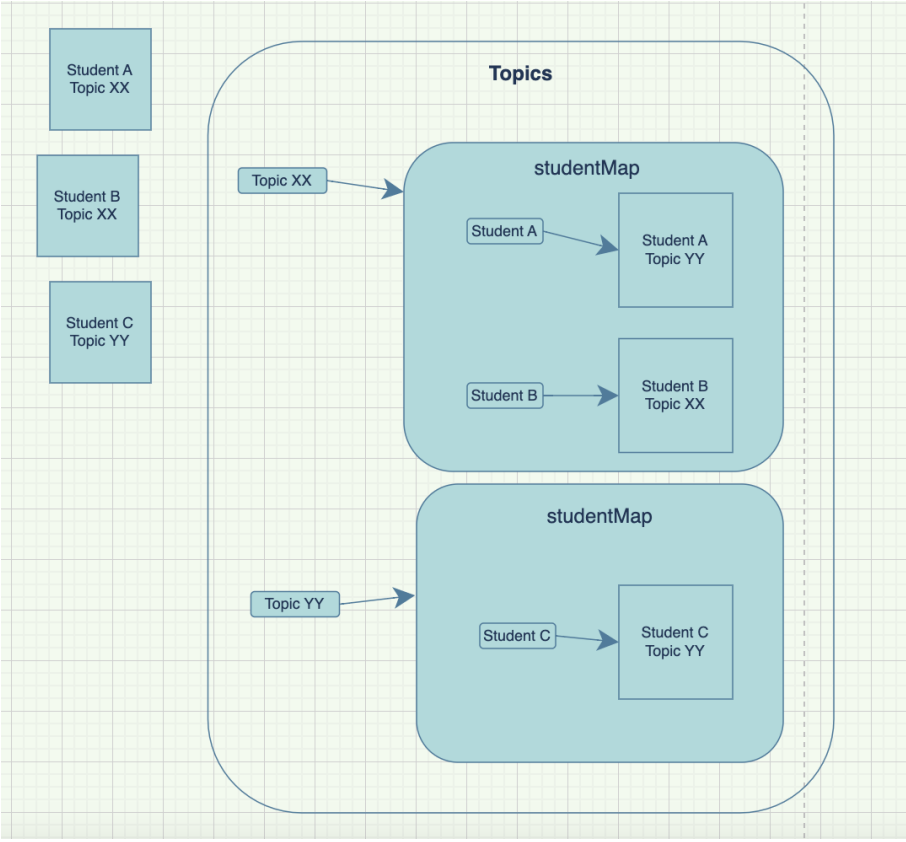


Fig 4.11: Visual of when a student changes topic

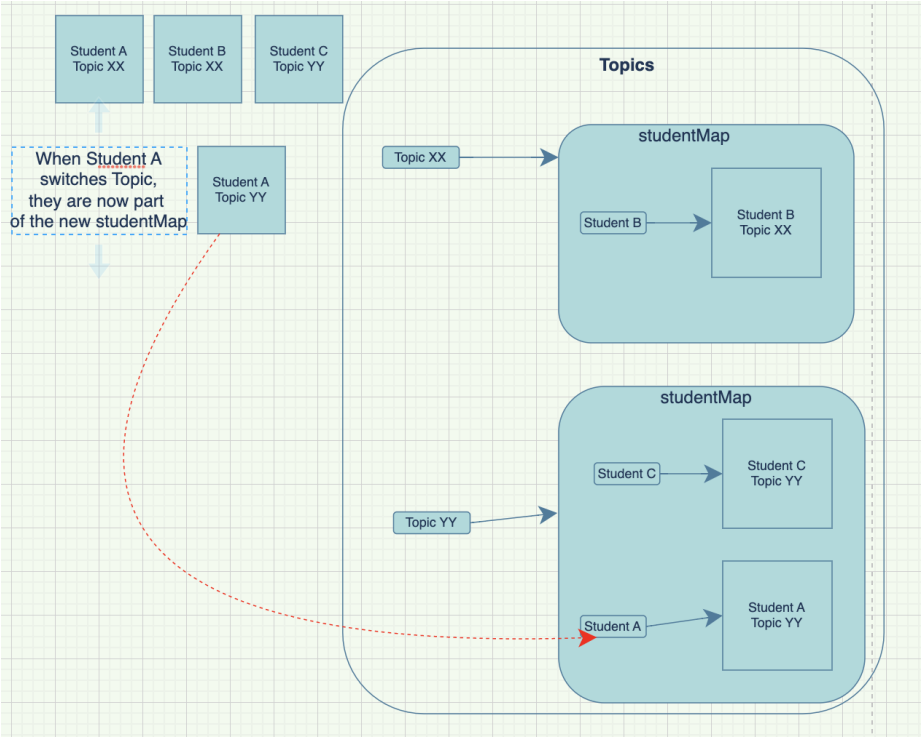


Fig 4.11visually shows how, if a student changes topic, the Tutors Hive app is checking to see if the student already exists, and if it does, then update the topic and move the student to the relevant studentMap within the topics map.

The function is also checking if the topic already exists, then add the student there, if it is a new topic, create a new studentMap within a new topic.

In the code breakdown in page.svelte Fig 4.12, we can see how the Tutors Hive app performs the way that it does. Firstly Tutors HIve is listening to the Tutors Simulator App Firebase database for courseData event which is called “Student Info”, and this information is stored into a new studentEvent, whereby the name is added to the relevant studentMap (Line 27) within the topics Map, unless the student already exists, then it is deleted. (Line 28).

Fig 4.12: Tutors Hive page.svelte code breakdown

```
17    let topics = new Map();
18    onMount(async () => {
19        initFirebase(getKeys().firebase);
20        let statusRef = ref(getDatabase(), 'StudentInfo');
21        onValue(statusRef, async (snapshot) => {
22            if (snapshot) {
23                // a new event has been seen
24                const studentEvent = snapshot.val();
25                console.log('student info received');
26                console.log(studentEvent);
27                topics.forEach((studentMap, topicName) => {
28                    studentMap.delete(studentEvent.name);
29                });
30                // Now, see if we have seen topic before
31                const studentMapForTopic = topics.get(studentEvent.topic);
32                if (!studentMapForTopic) {
33                    // Nope, this is a new topic. Lets create a new studentMap for this topic
34                    const studentMap = new Map();
35                    //  put the student into this map
36                    studentMap.set(studentEvent.name, studentEvent);
37                    // put the map into the topics mao
38                    topics.set(studentEvent.topic, studentMap);
39                } else {
40                    // existiing topic, put the student in
41                    studentMapForTopic.set(studentEvent.name, studentEvent);
42                    // if the student was already in aother topic, we have removed it (see earlier)
43                }
44                // we have changed the topic map, reload this part of the page
45                reload = !reload;
46                // touch all sutdent records, so that if any have changed they will be refreshed in screen
47                courseEvents = [...courseEvents];
48            }
49        });
50    });
```

Here the code is also checking if we have seen the Topic before and if it does not already exist, then it creates a new studentMap for this topic *(Fig 4.9 shows shows this visually),* and then puts the student into this studentMap, otherwise, if the topic already exists, the student is put into

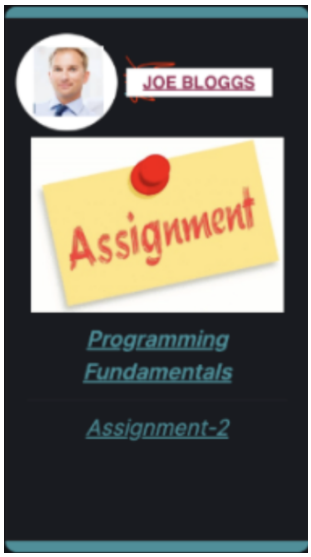
this studentMap for this topic. If the student was already in another topic, we have removed it and are putting it into the correct studentMap *(Fig 4.11 shows this visually)*

4.4 Design changes

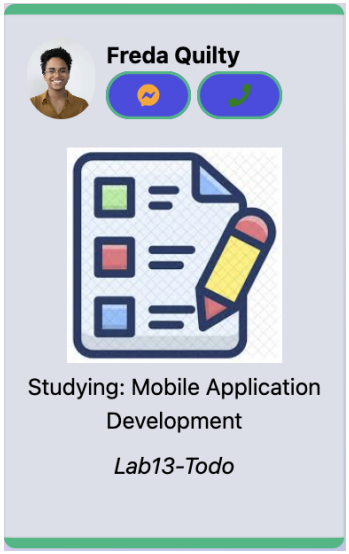
The new Tutors Hive Card design is not too dissimilar to the current Tutor Live card design, apart from the addition of the slack message and call buttons.

Fig 4.13: Tutors Live Card Design Vs Tutors Hive Card Design

Current Tutors Live Student Card



Tutors Hive Student Card (buttons added)



The Buttons are added as clickable buttons that link to the appropriate slack channel (message or huddle) for that specific student (Fig 4.14)

Fig 4.14: New Slack linked buttons (code sample)

```
<a href="{courseEvent.slackhuddle}" target="_blank">
<button
    class="btn variant-filled-secondary btn-base ring-2 ring-primary-500 ring-inset text-filled-500"
>
    <div class="icon" style="color: green; width: 20px; height:16px">
        <FaPhone />
    </div>
</button>
</a>
iv>
```

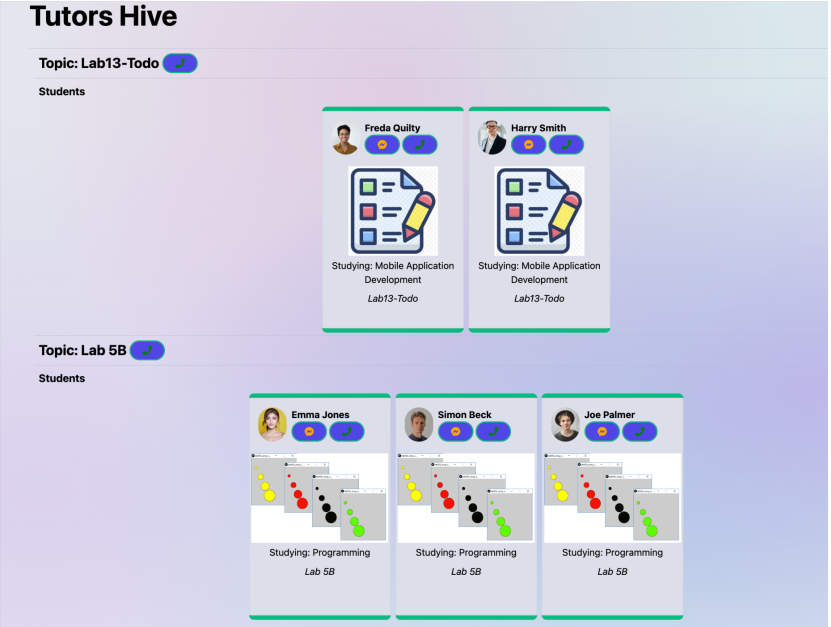
Fig 4.15 Shows the design of the card layout in TutorsHive Cards

```
<div class="card w-60 h-[24rem] overflow-x-hidden m-1 border-y-8 border-primary-500">
    <div class="flex">
        <header class="card-header inline-flex items-center">
            <div class="inline-flex w-full">…
            </div>
        </header>
    </div>
    <figure class="flex justify-center object-scale-down mt-4 p-1 h-40">
        <img src= "{courseEvent.topicimage}" alt="topic Image"/>
    </figure>
    <div class="card-body text-center">
        Studying: {@html courseEvent.module}
    </div>
    <footer class="card-footer text-center">
        <p class="mt-2 italic">
            {@html courseEvent.topic}
        </p>
    </footer>
</div>
```

Fig 4.15: Design layout of Tutors Hive Cards

There is also a New group huddle button for Topic groups, which can be seen here in Fig 4.16

Fig 4.16: Tutors Hive group huddle button visual



This button is the TopicHuddle Card within the outer for loop that we already saw in Fig 4.8 (line 64). Each topicHuddleCard is linked to the group huddle call for that topic, and anyone in the group can choose whether to answer the call or not. Fig 4.17 shows how each card calls for the topicHuddle for that topic

Fig 4.17: Each Student Card calls for TopicHuddle for that topic

```
<a href={firstTopic?.topichuddle} target="_blank" rel="noreferrer">
    <button
        class="btn variant-filled-secondary btn-base ring-2 ring-primary-500 ring-inset text-filled-500"
    >
        <div class="icon" style="color: green; width: 20px; height:16px">
            <FaPhone />
        </div>
    </button>
</a>
```

5. Project Plan

Tutors Hive Project plan had the following goals to begin with;

Goals	Description	Due Date
Draft Proposal	Project Proposal and concept	6th of Nov 22
Final Proposal	Final project proposal in more detail	4th Dec 22
Interim Report	Project Final Report underway	12th Feb 23
Final Report	Project Submission	2nd April 23

5.1 Schedule of sprints

The sprints for this project were set out across the span of the project and each sprint had some mini milestones that I wanted to achieve. Fig 5.1 shows the Trello board that I used to plan out my milestones and my sprints. These changed as I progressed to suit the progression of the project, and as ideas formulated.

Sprint phase	Preliminary Sprint due date
Sprint 1	20th Jan
Sprint 2	20th Feb
Sprint 3	1st March
Sprint 4	10th March
Sprint 5	15th March
Sprint 6	25th March

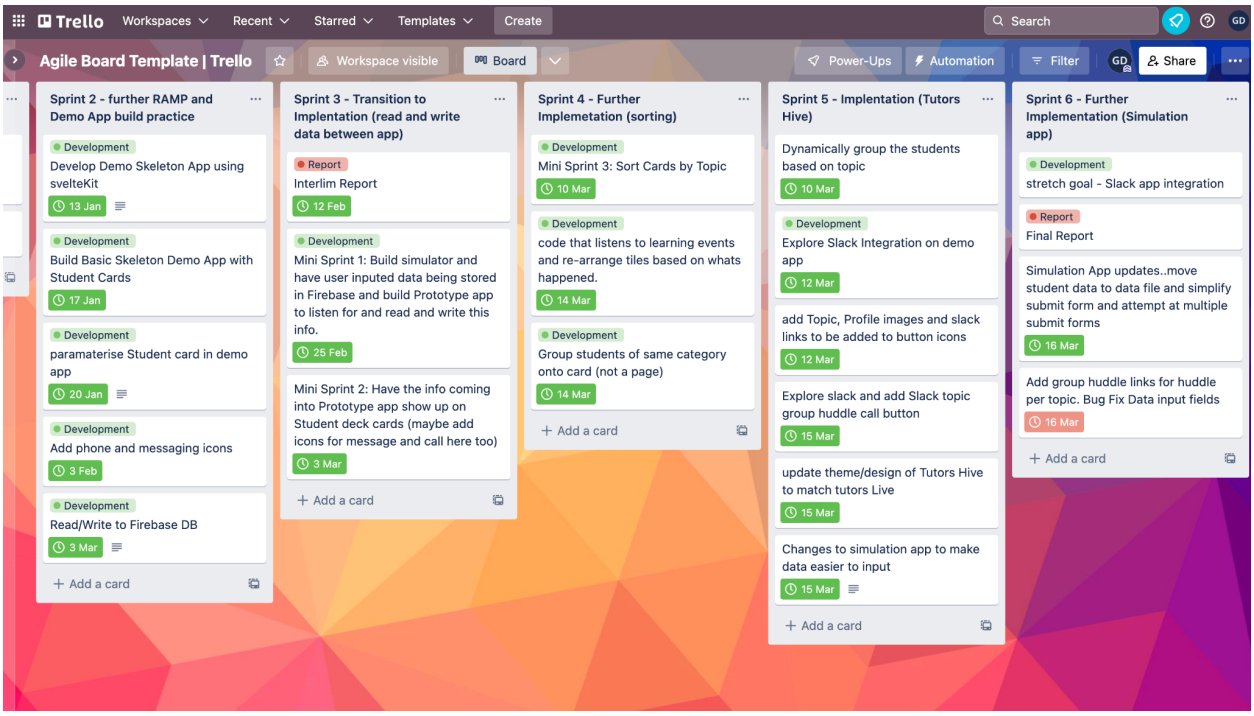


Fig 5.1 Trello board - Status of sprints and mini goals within these sprints

5.2 Implementation details

5.2.1 Sprint 1: Begin Research and Proposal/Ideas

During my research and preparing for the project, I learnt that the tools I would need to use include Javascript, SlackAPI, Github, typescript, Sveltekit, Git & GitHub. I wrote my project proposal and researched into the reasons why online presence in learning is beneficial which has been documented above. Once I had done more research and revised some tools, I was feeling ready for the next stage.

5.2.2 Sprint 2: RAMP and Demo App Practice

I spent some time researching these tools and making sure I was up to speed on them. Before starting any development work for the project. project I built a demo SvelteKit app *(Fig 5.2)* that displays simple cards, like in Tutors live, with some info and an image. This was very basic, just to get a feel for the way I wanted to start building the prototype app, and a starting step.

Fig 5.2 Demo SvelteKit app displaying simple cards with info and image

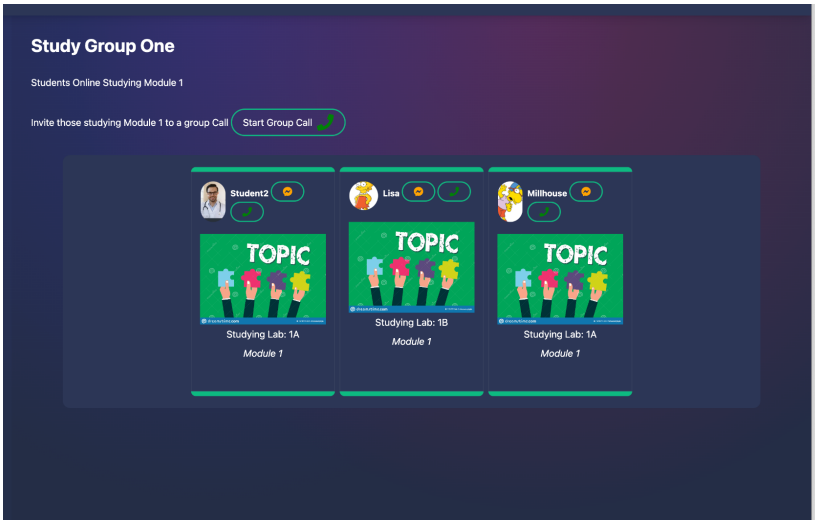
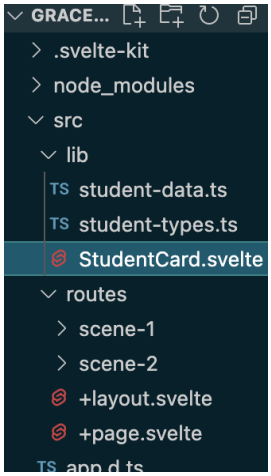


Fig 5.3 Basic Student Card code snippet




```
<Avatar src={student.image} alt={student.name} class="mr-2" />
<h6>{student.name}
    <button class="btn variant-filled-secondary btn-base ring-2 ring-primary-500 ring-inset text-filled-5
        <div class="icon" style="color: orange; width: 16px; height:16px">
            <FaFacebookMessenger/>
        </div>
    </button>
    <button class="btn variant-filled-secondary btn-base ring-2 ring-primary-500 ring-inset text-filled-5
        <div class="icon" style="color: green; width: 16px; height:16px">
            <FaPhone/>
        </div>
    </button>
</h6>
```

Fig 5.3 shows how I used Skeleton and svelteKit to build a basic version of what I was envisaging for my project. I studied the code from Tutors Live and used some of the same design elements to recreate some profile cards and then I even added some call and message buttons that weren't doing anything yet, but I wanted to have the vision of what it would look like.

I also in this phase, created a Realtime Firebase Database called Tutors-Simulator-App and I wrote up some simple code to experiment with writing any data to the database. In Fig 5.4 you

can see the first time that I achieved this, where the info inputted into the simulator app was being stored in my new Realtime Firebase Database (Tutors Simulator App) and then very printed onto the tutors live prototype app.

Fig 5.4 Initial read/write to Realtime FIrebase

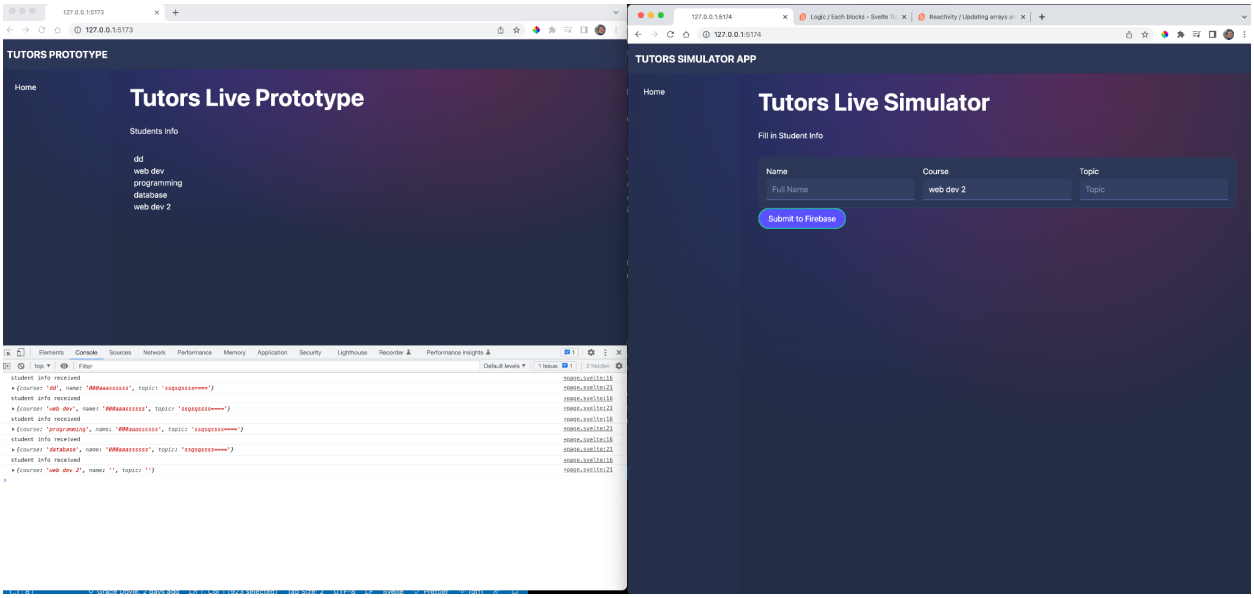


Fig 5.5 Code snippet capturing Data from the Realtime Database

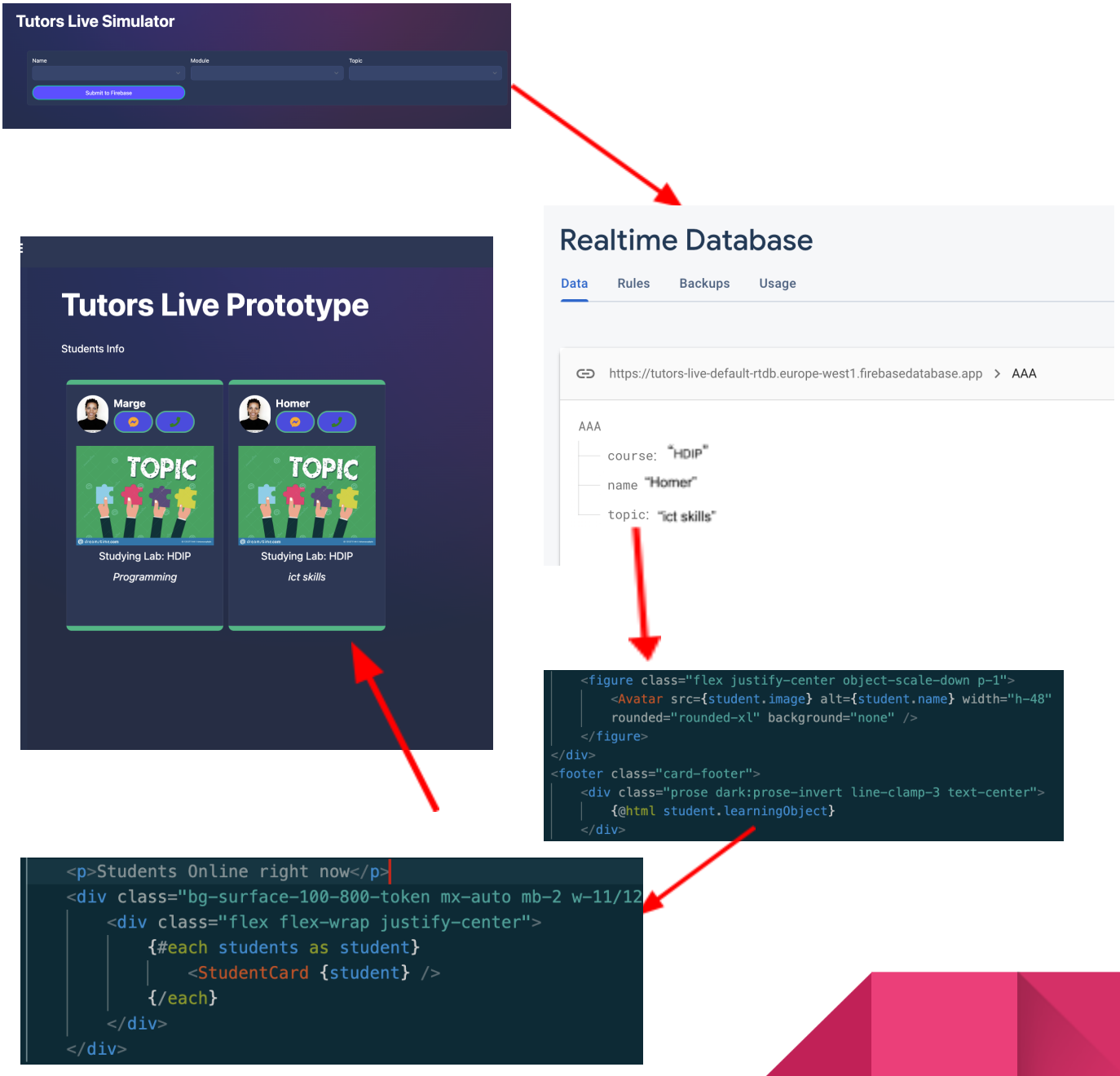
```
onMount(async () => {
    initFirebase(getKeys().firebase);
    let statusRef = ref(getDatabase(), 'StudentInfo');
    onValue(statusRef, (snapshot) => {
        console.log('student info Received');
        const data = snapshot.val();

    });
});
```

Fig 5.5 here shows the code snippet that captures the data from the Realtime Database (that has been sent from the simulator app) and then logs this the console as “Student Info Received” which can be seen in the console in *Fig 5.4.* This Sprint needed to be achieved, in preparation for the next phase , sprint 3, where I wanted to, next, display the info in a better way and more in line with tutors live, which would mean taking the data that is being read into the prototype app, and writing the code to capture this information and print onto a student card.

5.2.3 Sprint 3: Write Data from Simulator app and Read from other Prototype app to Student Cards

In this phase, two apps were developed. A Simulator app that simulates imputed data/learning events (courseData) that are stored in Firebase, and then a second prototype app (Tutors Hive) that reads this data from the database and now displays the data in a particular way on student cards like in tutors live and added message and call buttons too (Fig 5.6). In this phase, the Student Cards are not in any order, and each new data entry point creates a new card with the relevant student image, name and learning object.

Fig 5.6 *Sprint 3 overview visual*



5.2.4 Sprint 4: Sorting

The goal of this sprint was to write code that listens to the courseEvents being read from the Simulator App Database and re-arrange the tiles into groups based on the topics that the student is studying,. Firstly, I rewrote the code in Tutors Hive page.svelte to rearranged the cards statically, using a for loop, which can be seen in Fig 5.7

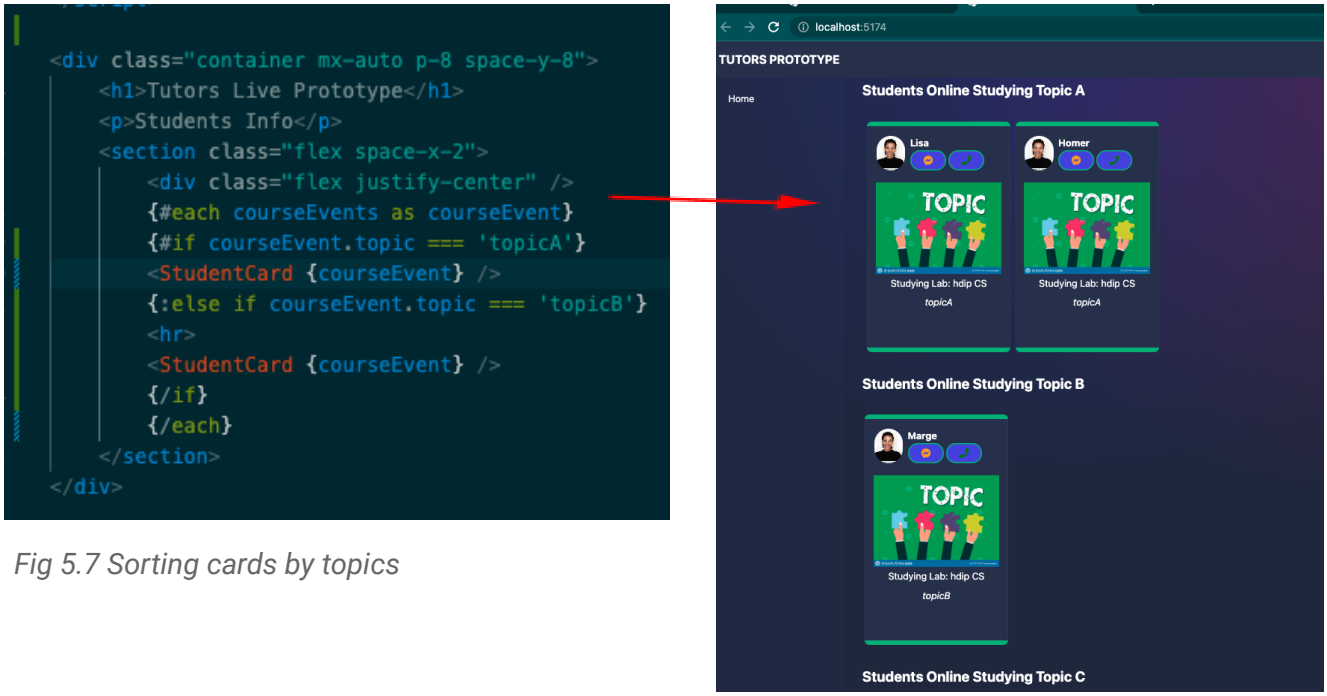


Fig 5.7 Sorting cards by topics

I then added more data such as images for the topics and an image for each person, which were hardcoded in the simulator app so that when simulation occurred from the dropdown options, the relevant info was attached each time (Profile image and topic image) Fig 5.8

The final stage of this sprint was to edit to code to show that, if a student changes topic, the Tutors Hive app is checking to see if the student already exists, and if it does, then update the topic and move the student to the relevant studentMap within the topics map. The function is also checking if the topic already exists, then student is added there, if

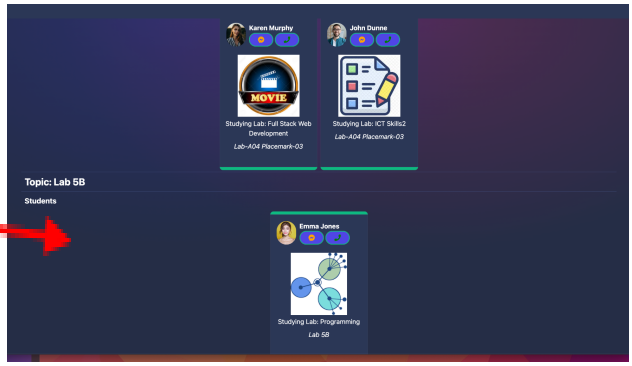


Fig 5.8 Profile and Topic images added

it is a new topic, create a new studentMap within a new topic.This implementation and architecture was already fully outlined on pages 38-40. The other goal of this sprint was to add a call button for each topic group. This for now, was just a button. In the next sprint, the slack integration was explored.

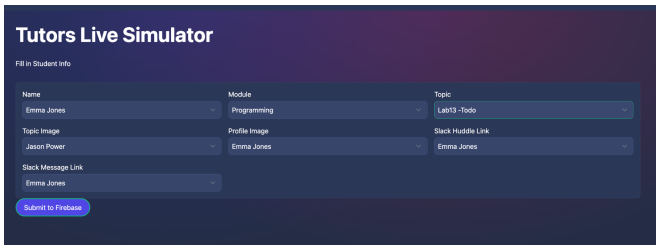
5.2.5 Sprint 5: Implementation (Tutors HIve)

The goal of this sprint was to add functionality to the huddle and message buttons, (both individually and the group topic huddle) to Tutors Hive.

When exploring ways in which I could connect to slack, I realised that each individual, and each huddle had their own slack ID's, so I felt that this was the best way to get a connection for the buttons to open up the relevant slack channel for either messaging or a huddle. The individual links were attached to the data in simulator app, so that if a person's name was selected, their slack id could also be selected, like in Fig 5.9

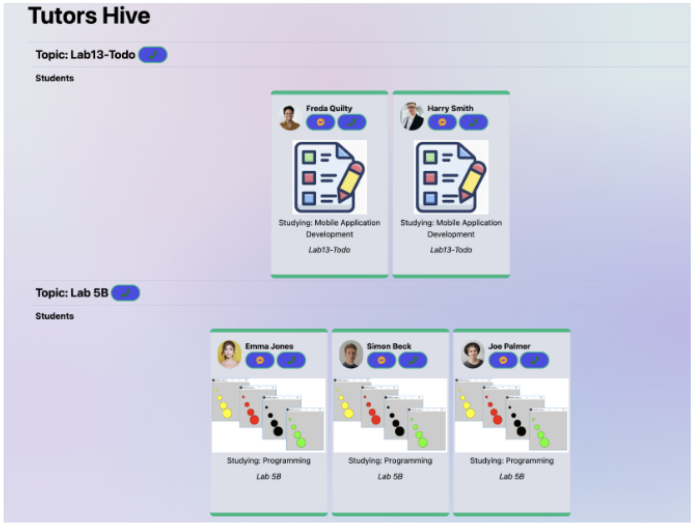
Fig 5.9 Slack ID's added to data.js

```
let studentData = [
    {
        name: "Emma Jones",
        slackmsg: "https://hdipprototype.slack.com/archives/D04SQFU6C4B",
        slackhuddle: "https://app.slack.com/huddle/T04SMGY98VB/D04SQFU6C4B",
        profileimage: "https://images.pexels.com/photos/415829/pexels-photo-415829.jpeg?auto=compress&cs=tinysrgb&w=800"
    },
```



This was done by adding the TopicHuddleCard for each topic, which opens up a huddle for anyone to accept/not accept. (See Fig 4.16 and Fig 4.17 page 42). This button is the TopicHuddle Card within the outer for loop that we already saw in Fig 4.8 (line 64). Each topicHuddleCard is linked to the group huddle call for that topic, and anyone in the group can choose to answer the call or not. These groups were pre-made for this experiment to be able to show the group calls in action.

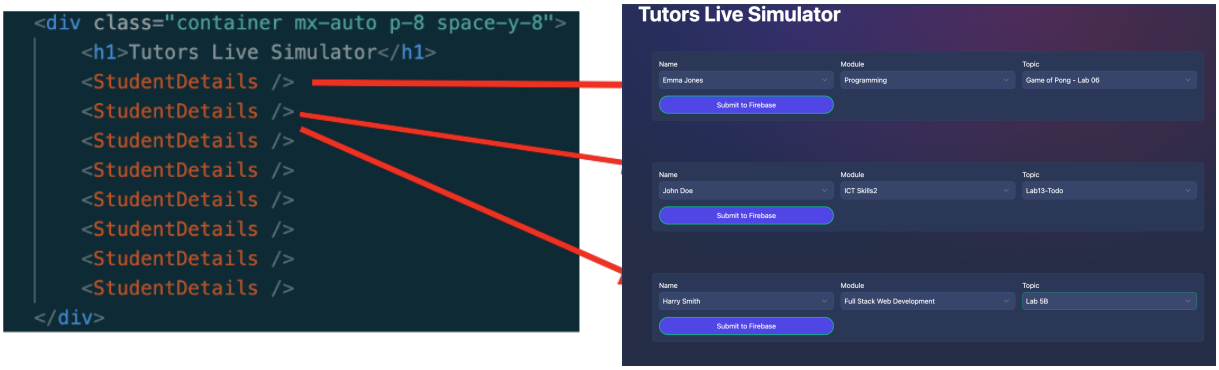
Fig 4.16: Tutors Hive group huddle button visual



5.2.6 Sprint 6: Further Implementation (Simulator App)

This stage was the final stage where I wanted to enhance the Simulator App for easier demoing. I moved the student data to data file and simplified the submit form and then I was able to add multiple submit forms on the one page so therefore I could demonstrate the presence aspect of Tutors Hive more efficiently. See Fig 5.11 below.

Fig 5.11 Multiple simulations at once



6. Conclusion & further work

6.1 Conclusion

This project (Tutor Hive) has shown that Tutors Live can be enhanced to improve the experience of the learner and educator, by both adding a sorting element to the student cards based on what topic they are studying, and also by adding a social option to the users, whereby they can click on a button to message or call their individual classmates, or group study groups (via links to slack).

6.2 Reflection and Challenges

This was a complex project,with regards to the coding involved in the two applications. I would have had the ambition to work directly in the reader itself but the further I got into it and studied it, the more I realized that it was too large of a task to just simply just add on a new feature like this to an already up and running system, with a large complex file structure. So instead, I picked out the pieces that I needed to focus on and built my own prototype of a presence engine to mimic tutors' lives as much as possible and then work on building on the feature. In the future I would like to bring this into the main reader to work on.

There were also challenges with choosing the best way to approach tasks at times, as I was always trying to keep the code as simple as possible, but the more the project progressed, the more complicated the code needed to be, while I was also conscious to keep in line with the DRY coding principle (Don't Repeat Yourself) and sometimes, trial and error led me to take a different route to achieve a task, than what I may have started with. This came into play when I wanted to make several entry forms for the simulator app, so I decided to move the code from page.svelte for the form, onto its own svelte page so that I could re-use it as many times as needed on the simulator page. Also, as more data was needed to be collected (slack links and images etc) the data was hardcoded into the simulator app, some data linked to each other, and I built the form in such a way so that when the data is being simulated for testing and a demo, the user can select just three pieces of info the data from the drop down list to generate the full learning object (i.e courseEvent)

6.3 Future Work

Future roadmaps could include Slack API integration. Slack can be accessed from the web via simple links, which can be demonstrated, but in the future, It could be further looked into to see if the Slack API could be more tightly integrated to the Tutors Hive (or tutors Live later on) so that perhaps the presence of the users on slack, will be visible inside tutors. This is a difficult integration and would require some further research and trialing. Slack presence could be a pop up on the side and potentially videos and message chats don't make you leave the page to go to the slack app, but rather, you're already signed in so the message and calls are either pop ups on the same page, or perhaps integrated directly into the page.

Another future idea would be that when categorizing the Student Cards into groups via topics, maybe each topic could have its own card, or its own with a tab on the side to allow you to navigate between topics. This means the page would be less crowded and users could perhaps see more clearly, or only see the Student Cards of those students who are studying the same topic as them.

Another idea would be to integrate a digital space called "Gather" which would enable students to have some control over where their cards appear on the Tutors Live. Gather is a digital space make virtual interactions more human (Gather Presence, Inc, 2022).

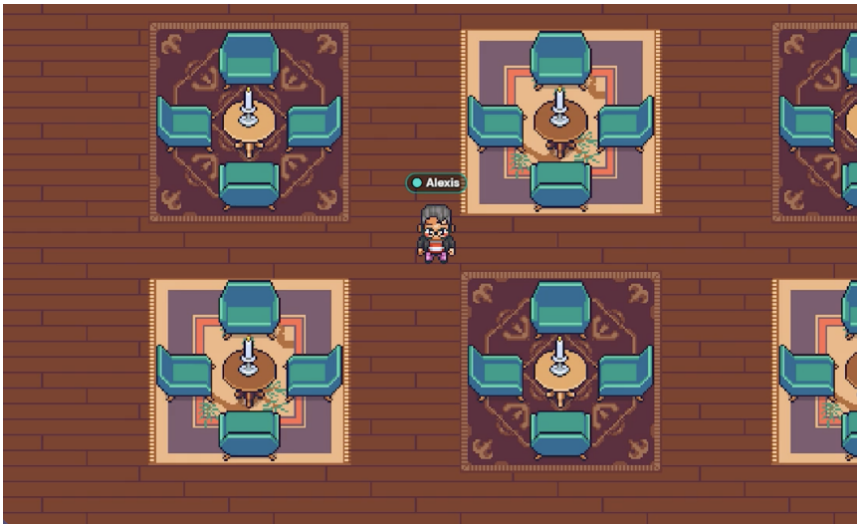


Fig 6.1: Gather - Online virtual interactions

An alternative idea again, would be to have each Category (Topic) as a separate tab like in this mock up in Fig 6.2, or each module as a separate tab with the topics categorised within the modules. the Modules into tabs.

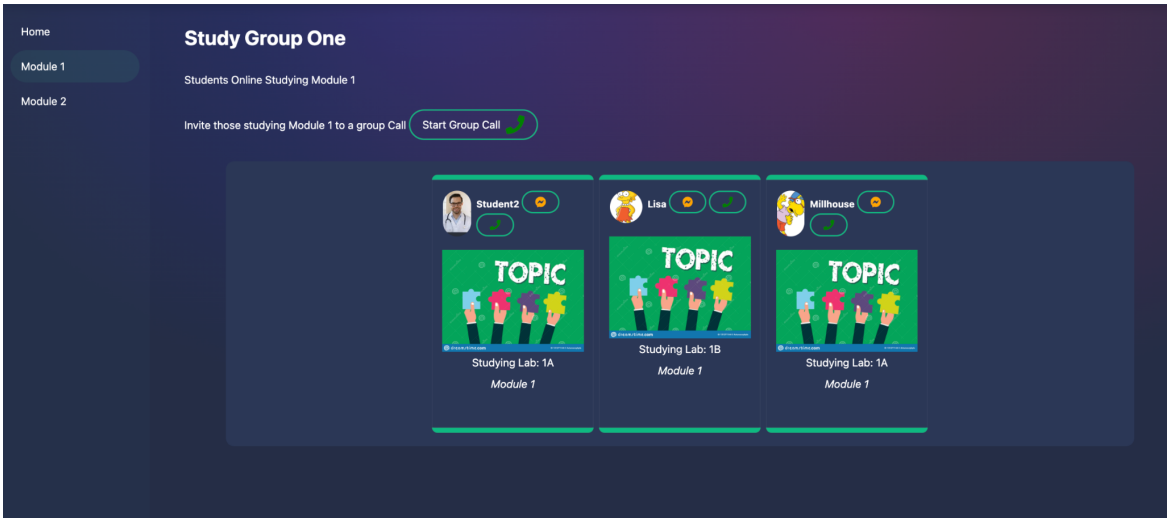


Fig 6.2: Topics/Modules catagorised by tabs

7. Bibliography

Anderson, T., & Dron, J. (2011). Three generations of distance education pedagogy. The International Review of Research in Open and Distance Learning, *Academia* [online] 12(3), 80-97, Available at: https://www.academia.edu/80865569/Three_generations_of_distance_education_pedagogy?sm=b [Accessed 11 Feb. 2023].

Barden, J. (2021). Tailwind CSS: A Rapid Development Framework. Packt Publishing.

Belsky, L. (2019). Where Online Learning Goes Next. Harvard Business Review, [online] Available at: https://hbr.org/2019/10/where-online-learning-goes-next [Accessed 12 Feb. 2023].

Biocca, F. (1997). The cyborg's dilemma: Progressive embodiment in virtual environments. Journal of computer-mediated communication, *Academia* [online] 3(2), 1-26, Available at: https://www.academia.edu/676179/The_cyborgs_dilemma_Embodiment_in_virtual_environments [Accessed 11 Feb. 2023].

Biocca, F. (1997). The cyborg's dilemma: Surviving the contradictions of technologically mediated social relationships. In J. Ferrell & M. Anspach (Eds.), Beyond postmodern politics: New directions for social theory (pp. 113-131). Routledge.

Eamonn de Leastar (2022) Tutors Architecture. *Waterford Tech Meet Up*, Waterford, Ireland, Dec.

Eamonn de Leastar. (2022). *TutorStack: Tutors Engagement Dashboard & Media Production Enhancement*. [Online]. Teaching and Learning. Available at: https://www.teachingandlearning.ie/project/tutorstack-tutors-engagement-dashboard-media-production-e [Accessed 4 April 2023].

Garrison, R. (2021). *The Community of Inquiry: about the Framework*. [online] The Community of Inquiry. Available at: https://www.thecommunityofinquiry.org/framework [Accessed 16th Jan. 2023].

Garrison, D. R., & Arbaugh, J. B. (2007). Researching the community of inquiry framework: Review, issues, and future directions. The Internet and Higher Education, ResearchGate [online] 10(3), 157-172, Available at: https://www.researchgate.net/publication/248540829_Researching_the_community_of_inquiry_framework_Review_issues_and_future_directions [Accessed 11 Feb. 2023].

Gather Presence, Inc. (2022). *Gather*. [Online]. Gather. Available at: https://www.gather.town/ [Accessed 3 April 2023].

Google. (2020). Firebase. [Online]. Available at: https://firebase.google.com/ [Accessed: 11th February 2023]

Gofine M, Clark S.(2017). Integration of Slack, a cloud-based team collaboration application, into research coordination. Journal of Innovations in Health Informatics, 24(2), 252. Retrieved from https://informatics.bmj.com/content/24/2/252 [Accessed 03 Feb 2023].

Katz, R., & Green, D. (2017). The rise of the new education sector. Harvard Business Review, 95(6), 22-25.

Manca, S., & Ranieri, M. (2016). Preparing teachers to design, deliver, and evaluate online learning. Journal of Educational Technology Development and Exchange, 9(1), 1-17.

Matija. (2023). *What Is SvelteKit?*. [Online]. Joy Of Code. Available at: https://joyofcode.xyz/what-is-sveltekit#best-of-both-worlds-server-side-rendering-with-client-side-n [Accessed 4 April 2023].

Mones, D. (2021). TypeScript vs. JavaScript: Your Go-to Guide. [Online]. toptal. Available at: https://www.toptal.com/typescript/typescript-vs-javascript-guide [Accessed 29 March 2023].

Moodle. (2023). Moodle - Open-source learning platform | Moodle.org. [online] Available at: https://moodle.org/ [Accessed 12 Feb 2023].

Walther, J. B. (1996). Computer-mediated communication: Impersonal, interpersonal, and hyperpersonal interaction. Communication Research, ResearchGate [online] 23(1), 3-43, Available at: https://www.researchgate.net/publication/284193126_Computer-mediated_communication_Impersonal_interpersonal_and_hyperpersonal [Accessed 03 Jan. 2023].

Salmon, G. (2013). E-Moderating: The Key to Teaching and Learning Online (3rd ed.). Routledge.

Skeleton. (2023). *Introduction*. [Online]. Skeleton. Available at: https://www.skeleton.dev/docs/introduction [Accessed 4 April 2023].

Skeleton. (2023). *The UI toolkit for Svelte and Tailwind.*. [Online]. Skeleton UI. Available at: https://www.skeleton.dev/ [Accessed 4 April 2023].

Skeleton.(2022). *Why Skeleton?*. [Online]. Skeleton. Available at: https://www.skeleton.dev/docs/why [Accessed 10 March 2023].

Smith, S. (2022). Tailwind CSS: The Complete Guide. O'Reilly Media, Inc.

Svelte Team. (2020). SvelteKit. Svelte. [Online]. Available at: https://svelte.dev/docs#sveltekit [Accessed: 11th February 2023]

SvelteKit. (2023). *What is SvelteKit?*. [Online]. SvelteKit. Available at: https://learn.svelte.dev/tutorial/introducing-sveltekit [Accessed 4 April 2023].

Swan, K., Richardson, J. C., & Ice, P. (2007). Synchronous text-based computer conferencing: A model for distance education. The American Journal of Distance Education, 21(2), 121-139.

Tutors Open Source Project. (2023). *Tutors Open Source Project*. [Online]. Tutors. Available at: https://tutors.dev/ [Accessed 10 March 2023]

Whiteside, AL, Garrett, DA, & Swan, K (eds) 2017, Social Presence in Online Learning : Multiple Perspectives on Practice and Research, Stylus Publishing, LLC, Herndon. Available from: ProQuest Ebook Central. [11 February 2023]..

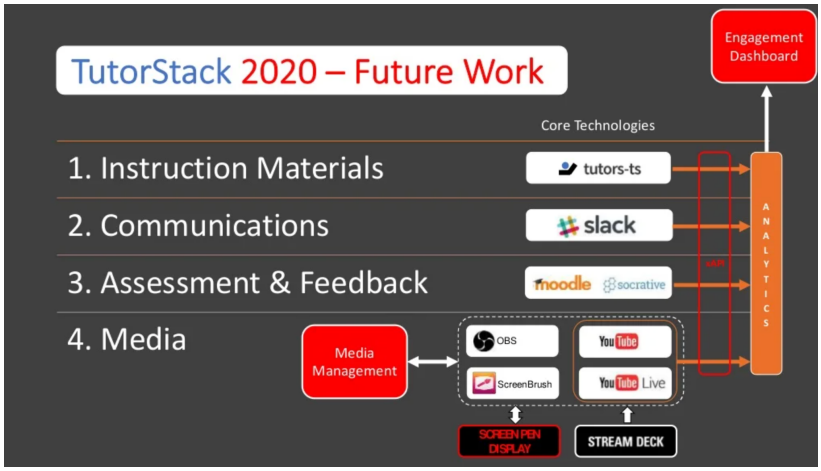
w3schools. (2023). What is TypeScript?. [Online]. w3schools. Available at: https://www.w3schools.com/typescript/typescript_intro.php [Accessed 29 March 2023].

8. Appendices

Appendix 1:

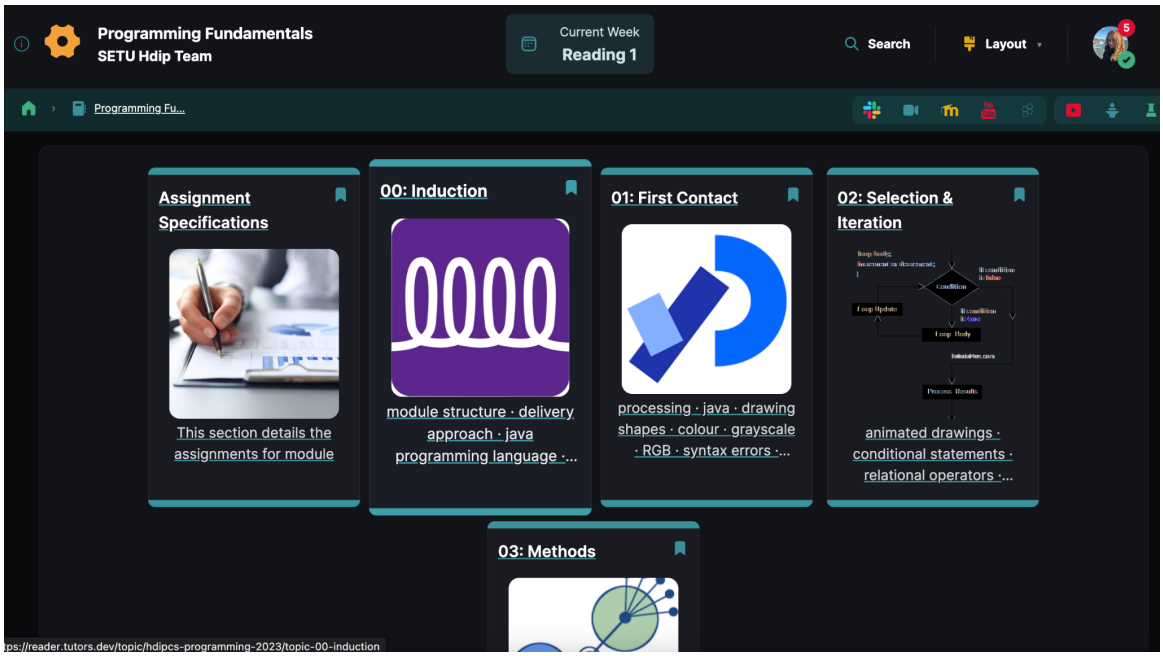
TutorsStack

Tutors Stack is an integrated collection of software for online delivery. There are 4 layers in the online delivery with Tutors Stack which are; 1. Instructional Materials, 2. Communications, 3. Assessment and Feedback and 4. Media. See FIG



Tutors

Under the first layer we have Tutors which holds all of the material for the educator and learner such as labs, lectures and videos. This is the landing page for the entire programme whereby both lecturers can create and add labs and courses and videos and students can select what they wish to study via the cards on display. Each card has a module or section or lab on it for the student to navigate through the course. See FIG



Slack

Slack is a communication hub for social and community interaction and is used across many educational institutes and workplace and is the leading messaging and communications application for teams used in industry. Slack is used to improve professional communication in the field (Gofine & Clark, 2017). All communication for the Hdip Computer Science Course programme (and many other programmes in the SETU) is on Slack. This means that students and lecturers can communicate via text, voice, video and communication on this platform. This is great for facilitating active learning and discussion during live classes, students are able to easily collaborate and gain lab support. Peer to peer support and group discussions via text or video are all possible on Slack. Screen sharing is another advantage for labs and project interviews in video calls. A goal of this project is to integrate Slack messenger, and possibly video calling, into Tutors Live so that students will be able to message or call other students, or group call other students within their study group, i.e call other students that are studying the same lab/lecture.

Moodle

Moodle is a free and open-source learning management system that allows educators and trainers to create, manage, and deliver online educational content. It is widely used by schools, colleges, universities, and other organizations that offer online learning. In Tutors, Moodle is used only for assessment and feedback and sometimes quizzes.

Moodle provides a range of features that help educators to create and manage online courses, including course management tools, activity modules, assessment tools, and collaboration features. Educators can use Moodle to create content, upload files, set up quizzes, manage discussions, and track student progress. Students can access the course materials, participate in activities, submit assignments, and track their own progress (Moodle 2023).

Zoom

Zoom enables individuals and organizations to connect and collaborate remotely through video conferencing and online meetings. Online education, virtual events, and virtual meetings are all common uses of Zoom. Video and audio meetings can be joined or hosted with Zoom, screens can be shared, and group chats can be held. Zoom offers a range of features to enhance the online meeting experience, such as virtual backgrounds, screen sharing, and meeting recording. In addition to real-time collaboration, Zoom offers features such as virtual whiteboards and screen annotations. As a cloud-based service, Zoom can be accessed from any computer, laptop, smartphone, or tablet with an internet connection. It is user-friendly and easy to use, with a range of tutorials and resources available to help users get started.

Appendix 2:

Table of Figures

Appendix 3: Declaration of Authenticity

I declare that the work which follows is my own, and that any quotations from any sources (e.g. books, journals, the internet) are clearly identified as such by the use of 'single quotation marks', for shorter excerpt and identified italics for longer quotations. All quotations and paraphrases are accompanied by (date, author) in the text and a fuller citation is the bibliography. I have not submitted the work represented in this report in any other course of study leading to an academic award.

Student..... Grace Doyle ..... Date 04/04/2023

Work Place Mentor..... Date .....

Appendix 4: Code of Ethics

School of Science and Computing Research Ethics Committee

Your response

Respondent: **Grace Doyle** (Group: CM-HDIPCS) Submitted on: Wednesday, 7 December 2022, 2:17 PM

Ethics Checklist for Undergraduate, Taught Postgraduate and Research Projects in the School of Science and Computing

All students in the School of Science and Computing who are either (1) in the final year of an undergraduate/BSc degree, or (2) on a taught postgraduate/MSc programme **must complete this Ethics Checklist before conducting their project** regardless of the project type or discipline. The Checklist should also be completed by anyone (whether staff member or student) conducting a **research project** (whether programmatic or not) within the School.

The purpose of this Ethics Checklist is to **identify projects that will require formal ethical approval** from the School Research Ethics Committee, or the SETU Research Ethics Committee, before they can proceed.

Students/applicants should note that this Ethics Checklist is a **formal declaration**, and great care must be taken to **answer all questions accurately**. Students should consult with their project supervisors/advisors regarding any aspects or questions that they are unsure of before completing and submitting the Ethics Checklist.

Students/applicants must **answer all questions** presented to them until the Checklist questionnaire is completed.

Feedback Report

No human experimentation issues (TPG/RP).

No animal experimentation issues (N/A).

No issues regarding the use of human tissues.

No animal tissue or biological fluids issues.

No ionising radiation issues.

No primary data collection issues (N/A).

No underage/vulnerable people issues (TPG/RP).

No issues regarding existing/secondary data use (N/A).

No controversial data issues.

No issues related to the collection of rare or protected plants.

No issues regarding the use of genetically modified (GM) plant material.

Instructions:

1. If the above feedback is **entirely green** then, based on your answers, there is **no need to apply for ethical approval** for your project.
2. If **any** part of the above feedback is **yellow/amber**, then there is at least one issue with your project that needs to be reviewed and **you must apply for ethical approval** to continue your project.
3. If **any** part of the above feedback is **red** then there is a serious ethical issue and **you cannot continue your project** as currently planned.

It is recommended that you print this Feedback Report to a PDF file for your records. You should also forward and discuss this Feedback Report PDF with your project supervisor. They will be able to advise if you have any further questions or if you need to apply for ethical approval.

1 * Are you a student on a **final year undergraduate** programme, a **taught postgraduate** programme, or are you conducting a **research project**?

- ○ Final Year Undergraduate
- ◉ Taught Postgraduate
- ○ Postgraduate Research Project
- ○ Other Research Project

2 * What is the **working title** of your project?

Social and Sorting for Tutors

3 * Who are the project **supervisors/advisors/principal investigators**?

Colm Dunphy , Peter Windle and TBC

4 * Does your project involve **human experimentation**? (e.g. clinical trials)

Yes No

5 * Does your project involve **animal experimentation**?

Yes No

(6) * Is the planned animal experimentation limited to **non-invasive procedures only** (such as feeding, weighing, or taking hair samples), and does **not** involve any invasive procedures (such as taking blood) from live animals?

Yes No

7 * Does your project involve the use of **human** remains/cadavers/tissues/cells/biological fluids/embryos/foetuses?

Yes No

(8) * Do you intend to only use established **commercial human cell lines**, and no other **human** remains/cadavers/tissues/cells/biological fluids/embryos/foetuses in your project?

Yes No

9 * Does your project involve the use of **animal tissues** or **biological fluids**?

Yes No

(10) * Do you intend to only use (1) **established commercial animal cell lines**, or (2) **slaughterhouse-derived tissues/fluids**, or (3) **fluids collected as part of routine animal husbandry** (e.g. milk) and no other animal tissues or biological fluids in your project?

Yes No

11 * Does your project involve the **collection of rare or protected plants**?

Yes No

12 * Does your project involve the generation or use of **genetically modified (GM) plant material**?

Yes No

(13) * Do you agree to (1) only use **established genetically modified (GM) plant cell lines, seeds, or plant products** in your project, (2) **not generate new plant mutations** using chemical or other means, and (3) follow specified SETU **containment and use protocols** for GM plant materials at all times?

Yes No

14 * Does your project involve the use of **ionising radiation**? (e.g. use of gamma ray spectrometry)

Yes No

(15) * Do you agree to carefully **follow the instructions** of the SETU designated **Radiation Protection Officer (RPO)**, and **adhere to all legal requirements** as set out in the Radiological Protection Act 1991 (Ionising Radiation) Regulations (2019), regarding the use of ionising radiation materials and equipment?

Yes No

16 * Does your project involve the **collection of any new (or primary) data** from **individual people or groups**?

Yes No

(17) * Does your project involve the **collection of any new (or primary) individual or group data** that is **personally or uniquely identifying**? (e.g. data about people or organisations/companies/groups that could be used to identify those individuals or groups; data collection might take any form, including internet and social media data, etc.)

○ Yes ○ No

(18) * Will you ensure that participants who you are collecting data from are provided with **fair warning** and must provide **explicit informed consent** for any data collected?

○ Yes ○ No

(19) * Will you ensure that any project-related data collection, data storage, and data use is in **full compliance** with the **EU General Data Protection Regulation (GDPR)** and the **Data Protection Act (2018)**?

○ Yes ○ No

(20) * Does any of the data that you intend to collect include **sensitive or private personal information** about individuals, or **commercially sensitive information** about organisations/companies/groups?

○ Yes ○ No

21 * Does your project involve **persons under the age of 18 years** (i.e. minors), or **any vulnerable groups**? (e.g. prisoners, refugees, those in care, addiction service users, etc.)

○ Yes ◉ No

22 * Does your project involve the use of **existing (or secondary) data**? (i.e. data originally collected for another purpose)

○ Yes ◉ No

(23) * Is the existing or secondary data you intend to use either (1) **anonymous/non-personally identifying** and in the **public domain**, or (2) available with **explicit and specific informed consent or permission** for the data to be **legally** reused in the way you intend?

○ Yes ○ No

(24) * Are any aspects of the primary/secondary data you intend to use for the project **controversial** in nature?

○ Yes ○ No

25 * Before you submit the Ethics Checklist, you must **confirm all of the following**:

☑ I understand that the Ethics Checklist is a formal declaration.
☑ I have answered all questions on the Ethics Checklist carefully and truthfully.
☑ The supervisor/advisor (or principal investigator) for the project is present as the Ethics Checklist is being submitted, or they have given me explicit permission to submit it in their absence.
☑ I have had adequate ethics training and/or instruction prior to completing the Ethics Checklist.
☑ I understand, and agree to abide by, the general ethical principle of "do no harm" for this project.
☑ I will follow the instructions given in the Feedback Report.

26 * Authentication Code (ask your project supervisor/advisor for this code)

Enter Student Number:

Enter the Authentication Code below and click "Verify Code" Verify Code

Note: If an INVALID authentication code is used then this submission is NULL and VOID

9951

◀ Announcements

Jump to...

Ethics Checklist Feedback Questionnaire (***optional***) ▶