UNIVERSITY OF WEST ATTICA

FACULTY OF ENGINEERING

DEPARTMENT OF INFORMATICS AND COMPUTER ENGINEERING

DIPLOMA THESIS

Creating an Educational Application using the principles of Responsive Web Design for optimal user experience

Author: **Dionisis Nikolopoulos**

Register Number: **18390126**

Supervisor: **Christos Troussas**

Athens, March 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημιουργία Εκπαιδευτικής Εφαρμογής με τις αρχές της Προσαρμοστικής Σχεδίασης για την βέλτιστη εμπειρία χρηστών

Συγγραφέας: **Διονύσης Νικολόπουλος**
Αριθμός Μητρώου: **18390126**

Επιβλέπων: **Χρήστος Τρούσσας**

Αθήνα, Μάρτιος 2024

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημιουργία εκπαιδευτικής εφαρμογής με τις αρχές της Προσαρμοστικής Σχεδίασης για την βέλτιστη εμπειρία χρηστών

Διονύσης Νικολόπουλος
Α.Μ. 18390126

Μέλη Εξεταστικής Επιτροπής, συμπεριλαμβανομένου και του Εισηγητή:

Χρήστος Τρούσσας
Ακριβή Κρούσκα
Παναγιώτα Τσελέντη

Christos Troussas Digitally signed by Christos Troussas Date: 2024.10.09 20:59:38 +03'00'

Akrivi Krouska Digitally signed by Akrivi Krouska Date: 2024.10.14 18:25:41 +03'00'

Panagiota Tselenti Digitally signed by Panagiota Tselenti Date: 2024.10.14 14:50:02 +03'00'

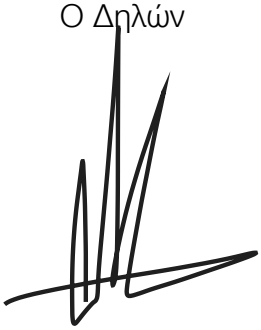
Ημερομηνία εξέτασης:
Σεπτέμβριος 2024

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος **Διονύσης Νικολόπουλος** του **Σπυρίδωνα**, με αριθμό μητρώου **ice18390126**, φοιτητής του Πανεπιστημίου Δυτικής Αττικής, της Σχολής Μηχανικών του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών, δηλώνω υπεύθυνα ότι:

«Βεβαιώνω ότι είμαι συγγραφέας αυτής της Διπλωματικής εργασίας και κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.
Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών



ΕΥΧΑΡΙΣΤΙΕΣ

Η εκπόνηση της παρούσας διπλωματικής εργασίας πραγματοποιήθηκε πάνω σε ένα θέμα το οποίο είναι τόσο ενδιαφέρον, όσο είναι και επίκαιρο, κοινωνικά αλλά και επαγγελματικά. Σημαντική αρωγή στην προσπάθεια αυτή αποτέλεσε ο επιβλέπων καθηγητής μου, Χρήστος Τρούσσας, τον οποίο θα ήθελα να ευχαριστήσω θερμά για τις άμεσες και κατατοπιστικές απαντήσεις του σε ζητήματα της εργασίας αλλά και της υλοποίησης της εφαρμογής.

Ακόμα, θα ήθελα να ευχαριστήσω τους φίλους και την οικογένειά μου για τη συμπαράσταση κατά τη διάρκεια των σπουδών μου, ιδιαίτερα τον πατέρα μου, που αποτελεί την έμπνευση αλλά και το πρότυπο μου για να επιλέξω τον προγραμματισμό ως επάγγελμα, όπως επίσης και την μητέρα μου, που πάντα υποστήριζε τις επιλογές μου. Τέλος, θα ήθελα να ευχαριστήσω την Έλλη, για την σημαντικότατη στήριξη που μου προσφέρει πρόθυμα και απλόχερα πάντοτε.

Abstract

The present thesis is about creating an educational Web Application following the modern guidelines of the Responsive Web Design pattern. This pattern allows for an easier customisation of the functionality of the Application on demand, according to the user's wishes and needs. Thus, it results on the optimisation of the user experience. The user experience therefore is unhindered by the type of device of the end user, or the size of the screen of the device in question. The creation of this Web Application also follows industry-standard methodology and utilises tools that are commonplace in the web development space nowadays.

Scientific Field: Web Applications
Keywords: Web Appication, Educational Application, React, JavaScript, Responsive Web Design

---

Περίληψη

Η παρούσα διπλωματική ασχολείται με την σύγχρονη προσέγγιση της αρχιτεκτονικής μιας διαδικτυακής εφαρμογής, ακολουθώντας τις αρχές της προσαρμοστικής σχεδίασης. Η προσαρμοστική σχεδίαση επιτρέπει την εύκολη εξατομίκευση της λειτουργίας της εφαρμογής ανάλογα με τις ανάγκες και επιθυμίες του χρήστη, οδηγώντας στην βελτιστοποίηση της εμπειρίας του. Η εμπειρία χρήστη, επομένως, δεν επηρεάζεται από τον τύπο της συσκευής του τελικού χρήστη ή το μέγεθος της οθόνης της εκάστοτε συσκευής. Κατά την διάρκεια της ανάπτυξης αυτής της διαδικτυακής εφαρμογής τηρήθηκαν μεθοδολογίες που θεωρούνται πλέον βιομηχανικά πρότυπα στον προγραμματισμό διαδικτυακών εφαρμογών και έγινε χρήση εργαλείων που είναι ευρέως διαδεδομένα στον γενικότερο χώρο της ανάπτυξης ιστοσελίδων των τελευταίων ετών.

Επιστημονική Περιοχή: Διαδικτυακές Εφαρμογές
Λέξεις Κλειδιά: Διαδικτυακή Εφαρμογή, Εκπαιδευτική Εφαρμογή, React, JavaScript, Προσαρμοστική Σχεδίαση

Table of Contents

List of Abbreviations

App: Application

CERN: European Organization for Nuclear Research (in French: Conseil européen pour la Recherche nucléaire)

CSS: Cascading Style Sheets

CVS: Concurrent Versions System

DOM: Document Object Model

FAQ: Frequently Asked Questions

HTML: HyperText Markup Language

HTTP: HyperText Transfer Protocol

JS: JavaScript

JSON: JavaScript Object Notation

ms: Milliseconds

NPM: npm is not an acronym (Literally, since 'npm' is a recursive bacronymic abbreviation)

OLED: Organic Light-Emitting Diode

PWA : Progressive Web Applications

Repo: Repository

RTE: Runtime Environment

SCCS: Source Code Control System

SVN: (Apache) Subversion

Tech: Technological (Used as "Technological Sector" or "Technology Industry")

TV: Television

UI: User Interface

UniWA: University of West Attica

UOM: University of Macedonia

UX: User Experience

VCS: Version Control System

Zsh: Z Shell

List of Figures

1. Introduction

1.1. Historical Retrospective

Amidst the recent decade's digital revolution's numerous protagonists, there has been a sector even more rapidly evolving than its peers. This sector is of course the Web. Only fifteen years ago, a typical website was a file written in simple HTML markup and served as a straightforward way, akin to a Word document or Plaintext file, to communicate information remotely, with little functionality available beyond text formatting. That all changed with the introduction of complex stylesheet and scripting languages like JavaScript and CSS, evolving to the point where today, we have what we call "Web Applications".

Web Applications now serve purposes so distant and so much more intricate than these simple HTML files, that currently Web Apps, as they are called, are replacing a plethora of native applications. Nobody has elucidated this point better and earlier than Apple's Founder, Steve Jobs, when in the presentation of the original iPhone, as early as 2007, mentioned:

> "The full Safari engine is inside of iPhone and it gives us tremendous capability, more than there's ever been in a mobile device to this date, and so you can write amazing Web 2.0 and Ajax apps that look exactly and behave exactly like apps on the iPhone!"[1]

For context, Safari is Apple's Web Browser. Since then, smartphones, tablets and computers are everywhere and accessing web applications is now part of everyday life, synonymous with browsing the web.

Due to this meteoric rise that followed the rise of the smartphone, advanced software tooling was needed, since now websites were taking the form of an application. Applications required programmers, or, as they were beginning to be known as, *developers.* Thus, the birth of the Web Applications, birthed *Web Developers*. The streamlining of the development of a Web App through the community of Web Developers brought about a diverse range of architectures, approaches and unique software solutions in order to supply for a newly found ever-growing demand. Web Frameworks, meaning software designed to support and accelerate a web app's construction, were now a staple among developers, allowing them to design software on a high level, grasping advanced design principles with relative ease. With the communities' programming experience of the past decades, web programming seemed to catch up to the polyphony of the tools, philosophies and languages of native application programming at a blistering pace.

1.2. Problem Statement

During the course of attending the University of West Attica in the department of information and computer engineering, signing up for classes and services on campus necessitated navigation upon a myriad of different websites. This was party caused by the relatively recent foundation of the Institute in 2018, through the merging process of the former Technological

[1] "**Steve Jobs iPhone Introduction in 2007** (Complete): Apple: Free Download, Borrow, and Streaming", **Internet Archive**, January 7, 2007, https://archive.org/details/original-iphone-keynote.

Educational Institute of Athens and Piraeus University of Applied Sciences. In 2019, the National School of Public Health joined the newly established university[2]. It is easy to imagine that a multitude of protocols, websites, databases and digital tools had to be altered or re-created to accommodate for this merging. The functionality of the existing infrastructure was not constructed with the same design in mind, especially as far as user experience and interface is concerned. Furthermore, the commonly agreed consensus among students was that the sheer number of sites with seemingly no centralisation made tracking time-sensitive information, like pending applications, grades, curriculums and timetables a challenge. In addition, a increasing number of sites had not been updated for some time. These were not following the principles of responsive web design, simply because they were not the standard at the time of the development, leading to a substandard user experience when viewing from a mobile or tablet device.

In response to this, there was an initiative of three students, under the guidance of professor Christos Troussas, for a centralised hub to be created in order to overcome this impediment. Apart from being mobile-friendly and thus responsive, this hub needed to be customised dynamically and on-the-fly by the department the user chooses on-site, since some websites are particular to a specific department of the university. The former mandated the development of a web application, built upon the modern principles and design philosophy of responsive web design.

1.3. Thesis Objectives

The main goal of the current thesis is to basically, simplify and streamline the process that the students go though when using the digital infrastructure, as outlined in Chapter 1.2. This entails the following specific objectives:

- Composition and organisation of a dedicated team of Web Developers with experience in designing and coding a Responsive Web Application, in order to speed up the development process.
- Software Architecture of the technologies used in the web app, taking into consideration multiple factors like the scaling of the project, the upgradeability and maintenance, as well as the compatibility with the already existing hardware whereupon the web application will be hosted.
- Coding the application with strict adherence to the principles of Responsive Web Design, following widespread industry code standards.
- Deploying the application on the already mentioned hardware and making sure it is maintainable, through the documentation and code commenting quality.

[2] "**History - University of West Attica.**" University of West Attica - **University of West Attica**, September 17, 2022. https://www.uniwa.gr/en/the-university/history/.

2. Technical Background

2.1. Programming & Markup Languages, the Building Blocks of the Modern Web

The changing environment in the tech sphere, now increasingly and aggressively including the Web, brought upon entirely new paradigms. It was only natural that new programming languages arise and existing ones would be moulded accordingly. On this chapter, the focus is on the ones used in this thesis.

2.1.1. HTML

HTML, or **H**yper**T**ext **M**arkup **L**anguage, is the standard markup language of the Internet. A markup language is a standard text-encoding system consisting of a set of symbols inserted in a text document to control its structure, formatting, or the relationship between its parts[3]. Specifically HTML is used to control the structure and formatting of documents, named *HTML Documents*, that are then parsed and rendered by Web Browsers. Its origins can be traced back to CERN, with the first proposal of the standard being in 1989[4].

```
<!DOCTYPE html>
<html>
<body>
<h1>A heading of level 1!</h1>
<p class="name-of-an-html-class">
      A paragraph with really interesting content.
</p>
<p id="nameOfAnHtmlElementID">
      A second paragraph with really boring content.
</p>
</body>
```

Figure 1: A simple HTML code snippet representing two paragraphs with a heading of level 1.

2.1.2. CSS

With plain HTML, very basic formatting and styling of HTML pages can be applied. However, very soon after people started to write HTML, there was a demand for more complicated display settings than headings, paragraphs and text alignment.

[3] "**Markup Language Definition**." Encyclopædia Britannica. Accessed May 26, 2024.https://www.britannica.com/technology/markup-language.

[4] "**Information Management: A Proposal.**" The original proposal of the WWW, HTMLized. Accessed May 26, 2024. https://www.w3.org/History/1989/proposal.html.

With the establishment of a non-profit organisation named World Wide Web Consortium in 1994[5], a standard to style web pages (plain HTML documents up until that point) was designed, based upon logic-less and simple documents named *style sheets*. These files, called CSS files, or Cascading Style Sheets files, contained CSS *rules*, meaning simple lists of attributes that were applied to specific HTML elements upon rendering on the browser. Users could now apply complex formatting on their web pages in a more organised manner. CSS could apply styling settings to multiple elements at a time, based on different kinds of identifiers that HTML elements have, like a *class* or an *id*.

```
/*Heading Level 1 HTML elements on HTML will have a font size of 35px */
h1 { font-size: 35px; }

/*HTML Elements with id “nameOfAnHtmlElementID” will have the color green*/
#nameOfAnHtmlElementID { color: green; }

/*HTML Elements with class “name-of-an-html-class” will have a cyan background
and padding of 20px*/
.name-of-an-html-class {
    background: cyan;
    padding: 20px;
}
```

Figure 2: A simple CSS code snippet styling some elements selected using different identifiers

2.1.3. JavaScript

One of the most poignant examples of an internet born and bred programming language is none other than JavaScript. No discussion of web-centered programming languages would be complete without mentioning it, being as it was originally created with the thought of adopting the newly-born, at the time, Java[6], for use in the Internet.

In the 1990s, the proprietary web browser *Netscape Navigator* was a pioneer and the first massively popular browser of its kind. Created by a company called Netscape in 1994[7], it quickly needed to include features that escaped concept of static HTML markup & CSS pages. Thus, in 1995, through a collaboration with Sun Microsystems, an effort was made by Netscape for the Java Programming language, as well as the Scheme[8] Programming Language (a Lisp-like language), to be embedded into Netscape Navigator.

---

[5] "**About Us - World Wide Web Consortium.**" W3C. Accessed May 26, 2024. https://www.w3.org/about/.

[6] "**What Is Java Technology and Why Do I Need It?**" Java.com. Accessed May 26, 2024. https://www.java.com/en/download/help/whatis_java.html.

[7] "**Netscape Communications Ships Release 1.0 of Netscape Navigator and Netscape Servers**", Netscape Communications News Release, December 15, 1994, https://web.archive.org/web/20050326152726/http://wp.netscape.com/newsref/pr/newsrelease8.html.

[8] "**Scheme**." The Scheme Programming Language. Accessed May 26, 2024. https://www.scheme.org/.

At first, JavaScript was named “Mocha”, pointing to its roots in Java. During the course of the development however, only the basic syntax of Java was integrated and much of the functionality was borrowed by Scheme, while the object-oriented elements were added from Self[9], another popular Smalltallk[10]-family language of the period. Fast-forward to today, JavaScript (or JS as it is often abbreviated as) is used in 99% of all webpages on the internet[11]. It is also the language upon which most Web Frameworks, as outlined in Chapter 1.1, are built. In current day, its popularity seems unwavering still, since 63% of developers in the Stack Overflow Developer Survey of 2023 noted they used JS, marking the eleventh year in a row JS is the most used programming language among programmers[12].

```
function countTheCounters () {
    // Get every counter on HTML file (elements with class name ‘counter’)
    const everyCounter = document.getElementsByClassName(‘counter’);
    // Iterate over every such element
    for (const counter of everyCounter) {
        // Get number contained in element
        let upto = parseInt(counter.innerHTML);
        if (upto) {
            let from = 0;
            // Store function that counts up by 1 into variable ‘updated’
            const updated = () => {
                let count = counter;
                count.innerHTML = ++from;
                if (from === upto) {
                    clearInterval(counts);
                }
            };
            // Execute function contained in variable “updated” every 50ms
            let counts = setInterval(updated, 50);
        }
```

Figure 3: A JS code snippet manipulating the DOM[13], in order to render a count up from 0 on a number contained in an element of class “counter”, every 50ms.

[9] "**Self Language.**" Self. Accessed May 26, 2024. https://selflanguage.org/.

[10] "**Squeak/Smalltalk**". Squeak.org. Accessed May 26, 2024. https://squeak.org/.

[11] "**Usage Statistics of JavaScript as Client-Side Programming Language on Websites.**" W3Techs. Accessed May 26, 2024. https://w3techs.com/technologies/details/cp-javascript.

[12] "**Stack Overflow Developer Survey 2023**." Stack Overflow. Accessed May 27, 2024. https://survey.stackoverflow.co/2023/#section-most-popular-technologies-programming-scripting-and-markup-languages.

[13] MozDevNet. "**Introduction to the DOM - Web Apis: MDN**." MDN Web Docs. Accessed May 27, 2024. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

The most common use case for JS, as highlighted above, is to manipulate the DOM, or Document Object Model, of a web document dynamically. The DOM is a programming interface that represents the page, so that programs can change the document structure, style, and content. It represents the document as nodes and objects; that way, programming languages can interact with the page. But JS is not only used on the user interface, or the *front-end*, as the industry calls it, of our apps. It is also used on the non-observable, from the user's perspective, processes that operate in the background of an app. This is called the *back-end*. JavaScript is a multi-paradigm language; meaning a language that not 'opinionated' or 'strict' as far as its structure (or even syntax on some cases) is concerned. It is object-oriented, but also can accommodate procedural, functional and imperative programming. This means that, by nature, it is extremely flexible insofar its use cases and the approach the user chooses.

Perhaps lacking originality, we chose JavaScript as our language of choice in building our Web App for University of West Attica too. Its versatility on building web software, the gigantic community that has been built around it and the seemingly never-ending supply of developers who are proficient in it are simply unparalleled.

2.2. Frameworks & Runtime Environments, the Foundation of Web-based systems

2.2.1. Web Frameworks

Constructing a small-scale web page by writing a few HTML, CSS and JS files and bundling them is a trivial process. However, things are not so simple constructing a web *application*. Significant thought must be put in dealing with purely programming matters, like the dynamic rendering of a user account page when fetching data from a database. Not unlike building a house, the programmer has to start from the ground up, while having infinite options for the architecture, the building blocks and the underlying functions of the project. Fortunately, no 'reinventing of the wheel' is necessary, since these infinite options have been condensed into a, admittedly vast, number of Web Frameworks.

Each Web Framework can have a radically different philosophy on the architectural approach and function of a web app. Most frameworks operate in the back-end, others in both the front and back-end, making them *full stack frameworks*. Some even utilise more than one programming language, like Ruby on Rails[14], that uses the Ruby Programming language along JS, or Django[15], that opts for using Python along JS instead.

Of course, choosing a framework is not mandatory; software tools nowadays might not even need a framework in order to make a complete application. For the purposes of this thesis, no tool explicitly labeled as a "framework" was necessary to be used.

[14] "**Getting Started with Rails**." Ruby on Rails Guides. Accessed May 27, 2024. https://guides.rubyonrails.org/getting_started.html.

[15] "**Django**." Django Project. Accessed May 27, 2024.https://www.djangoproject.com/.

2.2.2. Runtime Environments

Earlier, it was mentioned that JavaScript is used both in the front and on the back end of web apps. It was mentioned that the browser is the interpreter of JS; and since the end user interacts with the browser, that naturally places anything that is processed on it the front-end. So how can JS run on the back end? This is where runtime environments come in.

2.2.2.1. What are Runtime Environments in general?

A runtime environment (or runtime system) in programming is a generally lower-level system that enables the instructions written in a program to be executed, while dealing with concepts like parallelism and memory-management. The runtime environment is more complex the higher the level of the language is. Meaning, a high-level language like JavaScript, Python or Ruby has a more extensive runtime than a lower-level one, like C.

2.2.2.2. The Node.js runtime environment

In the case of JavaScript, the most popular runtime environment by far is Node.js[16]. It is a cross-platform and open-source RTE which enables the user to run JS without the need of a browser. This enables the use of JS for tasks like database querying, HTTP server creation and test software implementation, which were previously not possible due to the front-end only nature of vanilla JS.

```
// server.mjs
import { createServer } from 'node:http';

const server = createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello World!\n');
});

// starts a simple http server locally on port 3000
server.listen(3000, '127.0.0.1', () => {
    console.log('Listening on 127.0.0.1:3000');
});

// run with `node server.mjs`
```

Figure 4: A simple HTTP server code snippet on Node.js.

[16] "**Node.Js** - Run JavaScript Everywhere." Node.js -. Accessed May 27, 2024. https://nodejs.org/en.

2.2.2.3. The NPM package manager

The Node.js runtime environment is the de-facto standard when it comes to writing JS on the back-end. However, the default functionality of Node.js could not possibly (and shouldn't) cover every single use case of its nearly 10 million (as of 2018) users. Most of the advanced features used by modern web apps are relying on the use of additional code that extends the ecosystem, written by users, for users. The additional code, neatly organised into a format called a *package,* is available for download publicly by users of node.js.

At times, even institutional users, like companies, make and publish packages. Such is the case the subsequent React library, which is made by Meta[17]. Nevertheless, there have been cases where so much as two or a single programmer have published a package that is later used by millions of people around the world, for example the core-js[18] package. All these packages are shared through NPM, which is the default *package manager* of the Node.js ecosystem.

A package manager is a tool which stores and distributes the aforementioned packages to users who need to include them in their programs with the use of specific commands and automated install scripts. The package manager also takes care of the dependencies of downloaded packages on other packages in the NPM registry, while making sure the different software versions are compatible with each other. Such tasks would be tremendously difficult to maintain manually, especially when diving into the more advanced NPM features like software security vulnerabilities reports, dependency cycle resolving and even package funding.

The web app this thesis is based on is a package by its very nature. Every app built using the NPM package manager stores the packages it depends on, its name and other identifying info like the package creator name on a file named package.json, where all this info is available in JSON format.

```
{
  "name": "studentsapp",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@chakra-ui/icons": "^2.0.4",
    "@chakra-ui/react": "^2.2.4",
      …
    "react-router-dom": "^6.3.0",
    "react-scripts": "5.0.1",
    "react-social-icons": "^5.14.0",
    "web-vitals": "^2.1.4"
 …
}
```

Figure 5: A code snippet from a package.json

[17] "**Social Metaverse Company**." Meta. Accessed June 4, 2024. https://about.meta.com/.

[18] "**Core-JS**." npm. Accessed June 4, 2024. https://www.npmjs.com/package/core-js.

2.3. Libraries, the Power Tools of Web app Building

Just like power tools can be used in conjunction with one another to accomplish the task of building a house, so can software tools, as more than one is likely required to make a web app. Libraries are, essentially, collections of software tools that share similar functionality and use philosophy. The use of multiple libraries is also not uncommon. Also just like "power" tools, the exact definition of a "library" is up for interpretation. Where does a very extensive library become a framework? It is generally agreed throughout the industry that a framework must include all the necessary resources that a large-scale application would likely need. This, albeit trivial, definition does not automatically lead to the conclusion that multiple libraries can't be used together in order to facilitate a web app creation. This approach was used in the current thesis' web application, where the back-end of the app was not extensive enough to warrant the usage of a web framework.

The two main libraries upon which the app was built were *React*[19] and *Create React App*[20], on the Node.js runtime environment.

2.3.1. React

React is massively popular JavaScript library that is used to build complex user interfaces. It is part of the Meta Open Source[21] initiative, from Meta. The central idea of React is the concept of a *component*. A component is essentially a JS function that returns markup similar to HTML and custom to React, called JSX. This markup is not logic-less, like HTML, enabling it to conditionally and dynamically render UI elements on the client. Components are usually interactive too, which means their behaviour to user input can be programmed through the use of JSX, letting the "lower-level" functions be transpiled into vanilla JS. Thus, the programmer is not required to manual tweak and tinker with every single fine aspect of the UI, instead adopting a "higher-level" approach, which saves time, effort and makes the application scaling easier.

Lets assume, for example, that a list of videos needs to be placed in our web application. In React, the solution could be something similar to:

[19] "**React**" React Blog RSS. Accessed May 27, 2024. https://react.dev/.

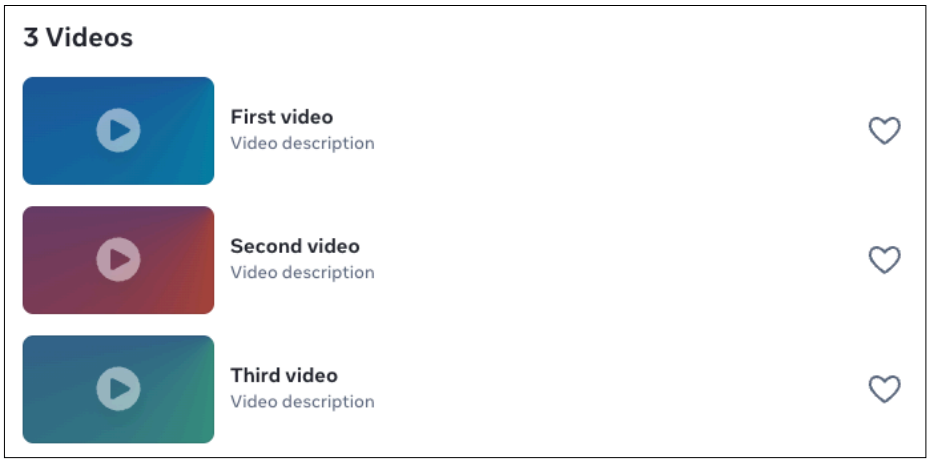
[20] "**Create React App**" Create React App. Accessed May 27, 2024. https://create-react-app.dev/.

[21] "**Projects**." Meta Open Source. Accessed June 4, 2024. https://opensource.fb.com/projects/.

```
function VideoList({ videos, emptyHeading }) {
    // Count the videos and form heading accordingly
    const count = videos.length;
    let heading = emptyHeading;
    if (count > 0) {
        const noun = count > 1 ? 'Videos' : 'Video';
        heading = count + ' ' + noun;
    }
    // Return JSX markup to render on UI, notice the multiple Video components.
    return (
        <section>
            <h2>{heading}</h2>
            {videos.map(video =>
                <Video key={video.id} video={video} />
            )}
        </section>
    );
}
```

Figure 6: A "*VideoList*" component that counts and displays "*Video*" components in React

The end result on the UI side would be something akin to:



But what does the "*Video*" in the return of the function render as? Of course, it is also a component, which would also be coded in a similar way to Figure 5. For example:

```
function Video({ video }) {
    return (
        <div>
            <Thumbnail video={video} />
            <a href={video.url}>
                <h3>{video.title}</h3>
                <p>{video.description}</p>
            </a>
            <LikeButton video={video} />
        </div>
    );
}
```

Figure 7: A "*Video*" component in React

Within our *Video* component, we also have components like *Thumbnail* and *LikeButton.* This captures the essence of the React library. Rather than trying to maintain and change a large codebase, which is a large problem to solve, the process is broken down into solving small problems by breaking down the codebase into manageable, streamlined and reusable pieces of code.

The process might seem over-engineered at first. After all, web apps were getting made long before React came around! However, when a team of developers need to cooperate and communicate, especially new developers are introduced to the codebase, it is greatly facilitating to have strict and universal standards which the code adheres to. This could be one of the reasons React is so widely used within the industry, where developers rapidly program apps without the need to study every single aspect of the, sometimes giant, codebase.

2.3.2. Create React App

It has been mentioned that React does not offer a coherent way to build and deploy a web application, since it is better categorised as a *library*, rather than a *framework*. Create-react-app is another library that offers pre-configuration of modern build setup for a web application. By downloading the library through an npm package and running some simple commands:

```
npm install create-react-app
npx create-react-app my-app
cd my-app
npm start
```

Figure 8: All the commands needed to create a React application through create-react-app

The application would be stored under the directory my-app/, where a pre-configured build system with other libraries and tools would provide everything a programmer needs to continue development without the need of any boilerplate code. Functionality like the bundling of static assets like client side JS files, CSS files, images and auto-fixes on compatibility of JS are already handled with no configuration needed.

The previous commands in Figure 8, would result in the downloading of the following files:

```
my-app
├── README.md ← Information about the project in MarkDown format.
├── package.json ← The dependencies on external libraries and tools this app has.
├── .gitignore ← Selectors of files that need to be ignored from version control
system.
├──  node_modules ← The external dependencies that are downloaded.
├──  public ← Images, thumbnail files, metadata files, etc. that are going to be
publicly available.
│    ├──  favicon.ico
│    ├──  index.html
│    ├──  logo192.png
│    ├──  logo512.png
│    ├──  manifest.json
│    └──  robots.txt
└── src ← Source code of the application, JS and CSS files mostly.
     ├──  App.css
     ├──  App.js
     ├──  App.test.js
     ├──  index.css
     ├──  index.js
     ├──  logo.svg
     ├──  serviceWorker.js
     └── [?] setupTests.js
```

Figure 9: The general structure of the simple React app made with create-react-app

This is the general layout of the application too, since create-react-app was used to initialize its fundamentals.

3. Configuration of the Application

3.1. Forking from myUOM project

The demand for a centralised point, on which students can access University resources, is understandably not unique to the University of West Attica. During research for the design of the application, a project by the University of Macedonia[22] surfaced. It is an *Open Source* Project by the open source team of UOM built on a React stack with create-react-app, so it constituted a perfect fit for the purposes of this thesis. Furthermore, the project was hosted on GitLab with the use of git, which simplified the forking process greatly.

3.2. What is Open Source?

There are two types of programs in this world. Proprietary, or “non-free” software and (free and) open source software. The important distinction here is that the word “free” does not refer to the *monetary cost* of the software, rather the fact that the licensing permits more *freedoms* to the user. Or as a popular saying goes on the open source community: “Free as in freedom, not free beer”[23]. This open source software can be licensed in many different ways, but the common attribute is, without exception, unhindered access to the source code of the program.

[22] "**University of Macedonia**". Accessed June 13, 2024. https://www.uom.gr/en.

[23] "**What Is Free Software? - GNU Project - Free Software Foundation**," https://www.gnu.org/philosophy/free-sw.en.html.

3.2.1. Source Code, Machine Code, Bytecode

The source code of a program is all the files containing programming language scripts and instructions that constitute the program as it is released by developers. Access to the source code enables anyone who possesses it to use tools to rebuild or run the program using their own computer. In interpreted languages, source code is just ran by a tool named the *interpreter* and no translation to machine or bytecode needs to happen. In contrast to this, a compiled language is first translated to bytecode or machine code, usually into a binary file, meaning a file that is not a text file[24]. Machine code is encompassed by instructions that can be ran directly by the computer, whereas byte code is either directly ran by an interpreter, or sometimes further translated to machine code and ran on. Programming languages like C[25] and C++[26] use machine code, while languages like Java[27] and Lua[28] use bytecode to run.

These distinctions are relevant because the specific instructions of the machine code differ from CPU to CPU, making the binary files sometimes inefficient or even incompatible when running on computers different than the one they were originally compiled in. Bytecode, on the other hand, is higher level than machine code but lower level than source code, making the translation to machine code or interpretation more efficient. So why do compiled languages even exist if they make portability so complex? Could the world just run on interpreted languages?

Of course interpreted languages are more easily portable, needing no intermediate compilation or installation, but that is not the whole story. There are certain tradeoffs to consider too. The interpreter that operates on source code, like the one on Python and JavaScript, which are both considered interpreted languages, still needs to translate the instructions of human-readable form to a machine-readable one. This process looks remarkably similar to a compilation from afar. It just happens on demand, while the compiled languages just have ready-to-run files from the get-go. However, this process takes time and it not as optimised and as closely tailored as specific machine code instructions for ones specific CPU. While this additional time is imperceptible to humans when writing simple programs, when the complexity of the algorithms written begins to grow, the choice of a specific language is of paramount importance. That is not to say all compiled languages are quicker than interpreted ones in all use cases, but that a fair amount of nuance must be present when considering which is the right tool for the job. The web app of this thesis is built on JavaScript exclusively, which falls squarely in the realm of interpreted languages.

---

[24] "**Binary File Definition by the Linux Information Project (LINFO),**" https://www.linfo.org/binary_file.html.

[25] Kernighan, Brian W., and Dennis M. Ritchie. "**The ANSI C Programming Language: Brian W. Kernighan, Dennis M. Ritchie**" Internet Archive, January 1, 1978. https://archive.org/details/the-ansi-c-programming-language-by-brian-w.-kernighan-dennis-m.-ritchie.org.

[26] Stroustrup, Bjarne. "**The C Programming Language: Bjarne Stroustrup**" Internet Archive, January 1, 1997. https://archive.org/details/cprogramminglang00stro_0.

[27] Lindholm, Tim, Frank Yellin, Gilad Bracha, and Alex Buckley. "**The Java® Virtual Machine Specification**." The Java® Virtual Machine specification, February 13, 2015. https://docs.oracle.com/javase/specs/jvms/se8/html/.

[28] "**The Implementation of Lua 5.0.**" Journal of Universal Computer Science, 2005. https://www.jucs.org/jucs_11_7/the_implementation_of_lua/jucs_11_7_1159_1176_defigueiredo.html.

3.3. What is a git & what is a fork?

Since programming started to pick up steam during the end of the 70s and in the start of the 80s, teams of programmers started to increase in size, making working in a complex application with a cornucopia of disparate files increasingly harder too. It is of no surprise that these programmers were required to work on the same source code and sometimes even in the same file simultaneously. Each one of them needed to copy the file in their own computer in order to make the changes and then communicate to merge the changes onto the same location. Such process was beginning to grow extremely tedious and difficult, since conflicting changes could sometimes be entangled in a way that took up precious development time to resolve. Naturally, organisational tools to simplify and optimise this process arose, satisfying this demand, named *Version Control Systems*.

3.3.1. Version Control Systems

Versions of software are distinct states on which the program or application has changed from its previous state, apropos of its source code and usually features added and removed or bugs resolved. Each version of the source code is a state on which the version control system references in order to detect intermediate or later changes by developers and if necessary, merge these concurrent changes or revert back to a previous state in part or as a whole.

3.3.1.1. Early VCS: Local & File-centric

One of the earliest and most common tools that are used for this purpose was the Source Code Control System (SCCS)[29] from Bell Labs[30], which implemented a simple system, contrasted with contemporary implementations. Each change was stored in a discrete *delta,* which is then applied to every new iteration of the source code. This core premise is still present on modern Version Control Systems. Building and improving upon the principles of SCCS, other version control systems were surfacing, like RCS in 1982, or Revision Control System[31]. Here, the idea of *reverse-deltas* was cultivated. A *reverse*-delta, in comparison with a *forward*-delta, previously mentioned as plain *delta*, is a record that instead of storing information about what *has changed* (i.e. forward), stores information about *how to go back to the previous state* (i.e. reverse) the reverse-delta describes. Thus, reverting source code to a previous version becomes a trivial operation.

3.3.1.2. Middle VCS: Networked & Project-centric

The internet brought massive changes to all software, VCS were not an exception. Developers were now managing *projects,* not a selection of files. The earliest VCS that had

[29] Rochkind, Marc J. "**Source Code Control System**." Internet Archive, IEEE Transactions on Software Engineering, Vol. SE-1, No. 4, December 1975. https://web.archive.org/web/20110525193926/http://basepath.com/aup/talks/SCCS-Slideshow.pdf.

[30] "Bell Labs Site - 1997." **History of Bell Laboratories**, 1997. https://web.archive.org/web/19961030004652/http://www.bell-labs.com/geninfo/history/.

[31] Tichy, Walter F. **RCS—a system for version control,** January 3, 1991. http://www.cs.umsl.edu/~schulte/cs2750/docs/rcs.pdf.

project-centric features was the notorious Concurrent Versions System (CVS)[32] in 1986. CVS was heavily reliant on RCS, using it to introduce changes and handling all the project-centric features with concepts like branching (independent string of versions) which are still used today. A bit later and with a number of commercial projects developed over the years to improve upon CVS, like the even more notorious Subversion (SVN). SVN was released in 2004[33], integrating with the Apache Suite of products[34,35] later on.

3.3.1.3. Modern VCS: Distributed & Fully Commercial

Through the course of history of VCS, it is apparent that these tools were beginning to grow more commercial and enterprise grade with each new tool released. Dozens of commercial products were released before and after Subversion, like ClearCase of IBM[36] origins and Perforce (now named Helix Core)[37]. All of the successful ones were rapidly commercialised, due to the even more rampant need of smooth collaboration of developers that were now not only working with multiple people, but were working on multiple projects. Thus, the developer to project ratio was shrinking and the market had to respond. But the problem with the aforementioned tools was that they were providing a solution to an issue of booming complexity which was now, additionally, changing form. Distribution and Scaling was of utmost importance now, something that was not designed in the time period that these tools were created.

This was the motivator for the creator (and benevolent dictator) of Linux, Linus Torvalds, to create Git[38], released in 2005[39], the tool that overwhelmingly is the standard for VCS nowadays.

3.3.2. Git

A part of the motivation of Linus Torvalds to even start to make Git was the echoed sentiment of growing resentment with certain difficulties CVS and by extension Subversion were causing programmers working in the modern landscape. Linus was in the midst of managing one of the most prolific and complex repositories of a project at the time, the Linux Kernel[40], which he was the creator of.

---

[32] **CVS - Concurrent Versions System**. Accessed June 15, 2024. https://cvs.nongnu.org/.

[33] Zeiss, Benjamin. "**Subversion 1.0 Is Released**." Subversion 1.0 is released [LWN.net] (Mailing List), February 23, 2004. https://lwn.net/Articles/72498/.

[34] Hyrum. "**Subversion Is Now Apache Subversion**." Wayback Machine, February 18, 2010. https://web.archive.org/web/20110512171259/http://subversion.wandisco.com/component/content/article/1/43.html.

[35] "**Apache® Subversion®**." Subversion (SVN). Accessed June 15, 2024. https://subversion.apache.org/.

[36] "**DevOps ClearCase**." IBM. Accessed June 15, 2024. https://www.ibm.com/products/devops-code-clearcase.

[37] "**Perforce Helix Core**." Perforce. Accessed June 15, 2024. https://www.perforce.com/products/helix-core.

[38] **Git**. Accessed June 15, 2024. https://git-scm.com/.

[39] Torvalds, Linus. "**Initial Revision of 'Git'**, the Information Manager from Hell · Git/Git@e83c516." GitHub, April 8, 2005. https://web.archive.org/web/20151116175401/https://github.com/git/git/commit/e83c5163316f89bfbde7d9ab23ca2e25604af290.

[40] "**The Linux Kernel Archives - About**." Accessed June 15, 2024. https://www.kernel.org/category/about.html.

Other developers were beginning to see the weaknesses of already existing VCS as early as 1997[41], as evident by the numerous changes the VCS of the Linux Kernel underwent, as well as Linus's less than favourable comments about CVS and SVN in 2007[42,43]:

> "When I say I hate CVS with a passion, I have to also say that if there any SVN users (Subversion users) in the audience, you might want to leave. Because my hatred of CVS has meant that I see Subversion as being the most pointless project ever started, because the whole slogan for the Subversion for a while was 'CVS done right' or something like that. And if you start with that kind of slogan, there is nowhere you can go. It's like, there is no way to do CVS right."

Linus' instinct, although absolutist in nature, proved to be correct. As early as 2010, git was taking up a significant share of nearly 11.3% of all repositories publicly available, while SVN was taking up 60.8% (SVN and SVNSync combined share)[44]. Since then, the tide has shifted, with Git gradually thriving more and more. In 2016, Git took up 39% and SVN 47%[45], while in the present day, SVN takes up 22%, while Git takes up 74%[46]. Even though the number of repositories grew in number from 142k in 2010 to roughly 350k in 2024, Git repositories went from 116k to 1 million in the same time period. In fact, SVN's growth seems to have been largely stagnant since 2014, where it hit the milestone of 325k[47], while Git's has skyrocketed. Of course, these are only the publicly available repositories, but a clear enough conclusion can be drawn from these results. Git is the de-facto standard of the industry in regards to VCS. New software seems to be developed mostly using Git, while SVN is left for maintaining large projects whose migration is not feasible. 93% of Stack Overflow developers seem to favour Git over other versioning systems, a percentage which does not seem to be changing any time soon[48].

Git has become so instrumental for the development of modern software, that entire companies have been founded on the basis of providing hosting for the central point on which the current state of the source code is stored, called the *repository*. Companies like the one with the leading market share, called GitHub, that was acquired by Microsoft in 2000 for $7.5

---

[41] McVoy, Larry. "**A Solution for Growing Pains.**" lkml.org, September 30, 1998. https://lkml.org/lkml/1998/9/30/122.

[42] "**Linus Torvalds on Git (Tech Talk): Google Talks**." Internet Archive, May 14, 2007. https://archive.org/details/LinusTorvaldsOnGittechTalk.

[43] "**Linus Google Tech Talk Transcript.**" Gist. Accessed June 15, 2024. https://gist.github.com/dukeofgaming/2150263.

[44] **Compare Repositories - Ohloh**, August 2011. https://web.archive.org/web/20100821122603/http://www.ohloh.net/repositories/compare.

[45] **Openhub.net.** Compare repositories - open hub, September 3, 2016. https://web.archive.org/web/20160903120733/https://www.openhub.net/repositories/compare.

[46] **Openhub.net**. Compare repositories - open hub. May 15, 2024 Accessed June 15, 2024. https://web.archive.org/web/20240515114715/https://openhub.net/repositories/compare.

[47] **Openhub.net**. "Tools." Compare Repositories - Open Hub, September 7, 2014. https://web.archive.org/web/20140907051024/https://www.openhub.net/repositories/compare.

[48] Donovan, Ryan. "**Beyond Git: The Other Version Control Systems Developers Use.**" Stack Overflow, January 9, 2023. https://stackoverflow.blog/2023/01/09/beyond-git-the-other-version-control-systems-developers-use/.

billion[49], or GitLab, another publicly traded company, with a $7 billion market cap as of June 2024[50].

3.3.3. Fork

A project fork in software development is the act of starting to develop a project while starting with source code of another project as a basis. In open source software, this practice is commonplace, since the licensing encourages programmers to customise or even make a different product based on code of the given licensed software. Using git in conjunction with repository-hosting sites like GitHub and GitLab makes forking a project a matter of making an account and clicking a button that reads “Fork Project”. The application is a fork project itself as mentioned previously, with changes that make the original project (myUoM[51]) and this application significantly different structurally and functionally, albeit familiar to each other.

Countless applications and programs have been initially launched as forks. One of the most compelling examples is Apple’s open source[52] browser engine, called WebKit[53], which was forked[54] from KHTML[55] (and KJS[56]), a now discontinued project for a browser engine in KDE, a popular Linux Desktop Environment[57].

[49] "**Microsoft Acquires GitHub.**" Microsoft Announcements, September 3, 2020. https://news.microsoft.com/announcement/microsoft-acquires-github/.

[50] "**GTLB**." Nasdaq. Accessed June 16, 2024. https://www.nasdaq.com/market-activity/stocks/gtlb.

[51] "**MyUoM**." myUoM. Accessed June 15, 2024. https://my.uom.gr/.

[52] **WebKit**. "WebKit/Webkit: Home of the Webkit Project, the Browser Engine Used by Safari, Mail, App Store and Many Other Applications on Macos, IOS and Linux." GitHub. Accessed June 16, 2024. https://github.com/WebKit/WebKit.

[53] **WebKit**. Accessed June 16, 2024. https://webkit.org/.

[54] Melton, Don. "'(FWD) **Greetings from the Safari Team at Apple Computer**.'" MARC, January 7, 2003. https://marc.info/?m=104197092318639.

[55] Kde. "**KDE/KHTML** at KF5." GitHub. Accessed June 16, 2024. https://github.com/KDE/khtml/tree/kf5.

[56] Stachowiak, Maciej. [KDE-darwin] javascriptcore, **Apple’s JavaScript framework based on KJS**, June 13, 2002. https://web.archive.org/web/20070310215550/http://www.opendarwin.org/pipermail/kde-darwin/2002-June/000034.html.

[57] "**KDE - Home**." KDE Community. Accessed June 16, 2024. https://kde.org/.

4. Implementation of the Application

4.1. Initial Setup

After forking the project of myUoM, the initial step was to clone the repository of the project locally. Then, installing the necessary files with the help of NPM was necessary. Finally, when the installation finishes, the app is ready to be ran. This process carried out with four simple shell commands (on zsh[58], under MacOS):

```
git clone git@gitlab.com:nionios/my-uniwa.git
npm install
cd my-uniwa
npm start
```

Figure 10: Process of cloning repo locally, installing dependencies via NPM and running the app.

4.2. Coding

The source code was ready to be edited freely now! The actual transformation of the original myUoM project to fit UniWA's specific requirements and services was underway.

4.2.1. index.js, Context and Provider Concepts

This file is the root point of the application. Every other file written as source code, excluding dependencies is related to this file, or is related to a descendant of it.

```
import {StrictMode} from "react";
import {ChakraProvider, ColorModeScript} from "@chakra-ui/react";
import ReactDOM from "react-dom/client";
import "./index.css";
import theme from "./theme/theme";
import App from "./App";
import reportWebVitals from "./reportWebVitals";
import {BrowserRouter} from "react-router-dom";
import {DepartmentProvider} from "./contexts/departmentContext";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <StrictMode>
    <BrowserRouter>
      <DepartmentProvider>
        <ChakraProvider theme={theme}>
          {/*ColorModeScript needed to set the theme according to OS*/}
          <ColorModeScript initialColorMode={theme.config.initialColorMode}/>
          <App/>
        </ChakraProvider>
      </DepartmentProvider>
    </BrowserRouter>
  </StrictMode>
);
```

Figure 11: The index.js file, the "root" of our application.

[58] **Zsh**. Accessed June 16, 2024. https://www.zsh.org/.

As is evident, a method of ReactDOM is called, which is the part of the react ecosystem responsible for the interaction with the Document Object Model of the client. This method is *createRoot*[59], that spawns the first markup element of the application onto the virtual DOM. In this case, an element named "<root>".

At this point it is important to note how dedicated React apps are to the concept of components that were previously mentioned in Section 2.3.1. Even the "strict mode" helper is a component, named *StrictMode*[60] that is first imported and then wrapped around the app, in order to facilitate warnings and errors during development regarding best React practices. *BrowserRouter*[61] closely follows suit, in order for the app to store the current location in the browser's address bar and navigate using the browser's built-in history stack. Then, the first *Provider* comes up, *DepartmentProvider*. After that provider, another one wraps around following components, named *ChakraProvider*, relating to the UI library this web app uses, named ChakraUI[62]. Finally, a *ColorModeScript* component is placed, relating to the detection and switching of light and dark UI mode within the application, along with the *App* component, which will be explained later in detail. But, this begs the question, what *is* a provider?

4.2.1.1. Providers and Contexts, the case of DepartmentProvider

A provider is a component that utilises React's Context API to assist with data sharing while managing a deeply nested component structure. Due to the components' inherent ambiguity when observing them by themselves, a context is a component that can be wrapped around components to provide the necessary data for their functionality. Minimal changes are necessary when readjusting them to use their strengths like portability (each component is a file or a small selection of files) and reusability, if this approach is followed.

To showcase the importance of Providers in the app and in React at large, the DepartmentProvider is a succinct example. This provider lives inside a file named DepartmentContext.jsx, which houses the following contents:

[59] **React Documentation - createRoot**. Accessed June 16, 2024. https://react.dev/reference/react-dom/client/createRoot

[60] **React Documentation - StrictMode**. Accessed June 16, 2024. https://react.dev/reference/react/StrictMode.

[61] "**BrowserRouter v6.23.1.**" BrowserRouter v6.23.1 | React Router. Accessed June 16, 2024. https://reactrouter.com/en/main/router-components/browser-router.

[62] "**Chakra UI** - a Simple, Modular and Accessible Component Library That Gives You the Building Blocks You Need to Build Your React Applications." Accessed June 17, 2024. https://v2.chakra-ui.com/.

```
import {createContext} from "react";
import {useLocalStorage} from "../hooks/useLocalStorage";

export const DepartmentContext = createContext({
  depName: "",
});

export const DepartmentProvider = ({children}) => {
  const [depName, setDepName] = useLocalStorage('depName', null);

  function changeDepartmentName(departmentToBeSet) {
    setDepName(departmentToBeSet);
  }

  const value = {depName, changeDepartmentName};
  return (
    <DepartmentContext.Provider value={value}>
      {children}
    </DepartmentContext.Provider>
  );
};
```

Figure 12: The DepartmentProvider component code.

The first function being imported here is createContext[63]. With this function, a JSON object is being created, that has a object key named **depName**, signifying a department name. When initialised, it's value is an empty String, as evident by the "". Practically, the objective of this provider is to supply the nested components with this department name that the user will select on the UI of the application later. This name will mutate certain department-specific information on numerous components of the application, such as links to the given department's site or links to the personnel pages of the department. The available department names are loaded using the *useLocalStorage* function, which is a *hook,* another especially noteworthy concept of the React Ecosystem.

4.2.1.2. Hooks in general, the case of useLocalStorage Hook

One of the overarching concepts of React is the management of *state*. It is most accurately and succinctly explained through examples. When a user of the application selects their desired department from the UI, the *state* of the program changes, describing that a department is now selected and the name of the department. The components of the app *react* to the state of the application and immediately mutate according to the change. This management of "stateful logic" happens through the use of multiple tools, one of them being Hooks. Hooks are just like most tools in react. At their core, they are functions. What makes them a "hook" is their design and handling of the stateful logic. There exist only two rules for Hooks in React. "Only call hooks on the Top level", meaning not inside conditions, loops, try/catch blocks etc, and "Only call hooks from React functions", which is self-explanatory[64].

There are several built-in hooks in React and their use is more than commonplace. So commonplace, in fact, that even custom hooks include built-in ones in their functions. For example, the useLocalStorage custom hook from the codebase:

[63] **React Documentation - createContext.** Accessed June 16, 2024. https://react.dev/reference/react/createContext

[64] **React Documentation - Rules of Hooks.** Accessed June 16, 2024. https://react.dev/reference/rules/rules-of-hooks

```
import {useState, useEffect} from "react";

export const useLocalStorage = (storageKey, fallbackState) => {
  const [value, setValue] = useState(
    JSON.parse(localStorage.getItem(storageKey)) ?? fallbackState
  );

  useEffect(() => {
    localStorage.setItem(storageKey, JSON.stringify(value));
  }, [value, storageKey]);

  return [value, setValue];
};
```

Figure 13: The useLocalStorage hook code.

We have two built in hooks being used in the code of Figure 13, highlighted in yellow. Hooks are by convention named with the combination of the word "use" and the word of the concept affected by their usage. Here, we see *useState* and *useEffect* being employed in the implementation of the custom hook:

- **useState**[65] is a function with that returns an array of two variables. The first is the stateful variable, meaning the variable that signifies the state, followed by a second variable, which stores a function through which the first stateful value is changed. In JS, variables can hold functions. The parameter of useState is the initial state that will be stored into the first variable in index 0 of the returned array. In this case, it is the json object that is fetched using the storage key while calling the hook in the first place.

- **useEffect**[66,67] is a bit more complex. It's usage is necessary when a React component needs to exchange information with something that is not part of the React Ecosystem. In this case, the JSON file stored on disk with the department information. It is a function that always returns **undefined** and has two parameters, of which the second is optional. The first parameter is a function that implements the logic of the Effect. This function is ran on each render of the component if the second parameter is missing. The second parameter is an array of *dependencies* of the Effect. The dependencies are "reactive" values, meaning values defined in the component. If they were to be changed, the code of the setup function (the first parameter of useEffect) will be re-run. In this case, the inputted key (storageKey variable) is used to load into the memory the specific information under the key with the same name on the department JSON file on disk. These JSON files are loaded through React's localStorage, which edits the browser's Storage object[68].

[65] "**useState**." React. Accessed June 17, 2024. https://react.dev/reference/react/useState.

[66] "**useEffect**." React. Accessed June 17, 2024. https://react.dev/reference/react/useEffect.

[67] "**Synchronizing with Effects**." React. Accessed June 17, 2024. https://react.dev/learn/synchronizing-with-effects.

[68] MozDevNet. "**Window: Localstorage Property - Web Apis: MDN**." MDN Web Docs. Accessed June 17, 2024. https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage.

4.2.2. App.js & Routing Concepts

The root point of the application might be index.js, but the part that might be more welcoming to a (human) eye is the the component of App.js. Here, the routing of the application is set up with the help of React Router, allowing for a quick overview of all the accessible pages of the application, coded in the “React way”, as nested components.

```
import HomePage               from "./pages/HomePage";
import Header                 from "./components/Header";
import RestaurantPage         from "./pages/RestaurantPage";
import { Routes, Route }      from "react-router-dom";
import ServicesPage           from "./pages/ServicesPage";
import GraduationPage         from "./pages/GraduationPage";
import LibraryPage            from "./pages/LibraryPage";
import FAQSettingsPage        from "./pages/FAQSettingsPage";
import AboutSettingsPage      from "./pages/AboutSettingsPage";
import SemesterSchedulePage   from "./pages/SemesterSchedulePage";
import ExamsSchedulePage      from "./pages/ExamsSchedulePage";

function App() {
  return (
    <>
      <Header />
      <Routes>
        <Route index                 element={<HomePage />} />
        <Route path="/services"      element={<ServicesPage />} />
        <Route path="/restaurant"    element={<RestaurantPage />} />
        <Route path="/graduationpage" element={<GraduationPage />} />
        <Route path="/librarypage"   element={<LibraryPage />} />
        <Route path="/faq"           element={<FAQSettingsPage />} />
        <Route path="/about"         element={<AboutSettingsPage />} />
        <Route path="/semesterschedule" element={<SemesterSchedulePage />} />
        <Route path="/examsschedule"  element={<ExamsSchedulePage />} />
      </Routes>
    </>
  );
}
```

Figure 14: The App.js component code with all the routes (comments removed for brevity).

As is visible in Figure 14, every JS file that contains a page component is passed as a parameter to a Route component, accompanied by the path URL that leads to the given page. The Header component is the navigation bar and the Route with the **index** parameter passed is the index (or “home”) page of the application.

4.2.3. HomePage.jsx, Rendering & Configuration Concepts

A concise demonstration of Page creation in React can be had through the homepage. Following is the code of HomePage.jsx:

```
const stagger = {
  inView: {
    transition: {
      staggerChildren: 0.1,
    },
  },
};

export default function HomePage() {
  // Get the current department name set from settings
  const { depName } = useContext(DepartmentContext);
  // Get the page that lists the academic personnel for specific department
  const academicPersonnelSourceIndex = academicPersonnelPages.findIndex((i) => i.department ===
depName);
  // Replace the link on the categories json with the appropriate one each time
  const academicPersonnelIndex = Categories.findIndex((i) => i.title === "Ακαδημαϊκό Προσωπικό");

  const announcementsSourceIndex = departmentAnnouncements.findIndex((i) => i.department === depName);

  const announcementsIndex = Categories.findIndex((i) => i.title === "Ανακοινώσεις");

  if (academicPersonnelIndex !== -1 && academicPersonnelSourceIndex !== -1) {
    Categories[academicPersonnelIndex].route =
academicPersonnelPages[academicPersonnelSourceIndex].pageUrl;
  }

  if (announcementsIndex !== -1 && announcementsSourceIndex !== -1) {
    Categories[announcementsIndex].route = departmentAnnouncements[announcementsSourceIndex].pageUrl;
  }

  const [categoriesListForSearch, setCategoriesListForSearch] = useState(Categories);

  useEffect(() => {
    window.scrollTo(0, 0);
  }, []);
  return (
    <Flex
      id="main-grid"
      direction="column"
      alignItems="center"
      justifyContent="start"
      gap="3rem"
      h={{ sm: "100%", "2xl": "calc(var(--available-height)*0.9)" }}
      px={{ sm: "1rem", lg: "3rem" }}>
      <Grid
        as={motion.section}
        initial="initial"
        animate="inView"
        variants={stagger}
        className="home-grid"
        marginTop="20px"
        gap={{ sm: 2, md: 3, lg: 4, xl: 5, "2xl": 6, "3xl": 7 }}
        templateColumns={{
          sm: "repeat(3, 1fr)",
          md: "repeat(3, minmax(0, 1fr))",
          xl: "repeat(5, calc((var(--available-width) - 5rem) / 5))",
          xxl: "repeat(5, calc((var(--available-width) / 1.5) / 5))",
          "2xl": "repeat(5, calc((var(--available-width) / 1.75) / 5))",
          "3xl": "repeat(5, calc((2*var(--available-width)/3) / 5))",
        }}>
        {categoriesListForSearch.length === 0 ? (
          <Heading        gridColumnStart={1}       gridColumnEnd={3}
                 fontSize={{ sm: 11.95, md: 16, lg: 26, xl: 32 }}
                 w="100%"
                 fontWeight={500}>
            Η αναζήτηση δεν επέστρεψε αποτελέσματα.
          </Heading>
        ) : null}
        {categoriesListForSearch.map((category) => (
          <MenuBox category={category} key={category.title} />
        ))}
      </Grid>
    </Flex>
```

<- "This search yielded no results."

Figure 15: The HomePage.jsx component code as a whole (imports removed for brevity).

4.2.3.1. Preparing and processing the data

The first part that will be explained in this chapter is the code snippet before the return of the JSX code, excluding imports for brevity:

```
export default function HomePage() {
  // Get the current department name set from settings
  const { depName } = useContext(DepartmentContext);
  // Get the page that lists the academic personnel for specific department
  const academicPersonnelSourceIndex = academicPersonnelPages.findIndex((i) => i.department ===
depName);
  // Replace the link on the categories json with the appropriate one each time
  const academicPersonnelIndex = Categories.findIndex((i) => i.title === "Ακαδημαϊκό Προσωπικό");

  const announcementsSourceIndex = departmentAnnouncements.findIndex((i) => i.department === depName);

  const announcementsIndex = Categories.findIndex((i) => i.title === "Ανακοινώσεις");

  if (academicPersonnelIndex !== -1 && academicPersonnelSourceIndex !== -1) {
    Categories[academicPersonnelIndex].route =
academicPersonnelPages[academicPersonnelSourceIndex].pageUrl;
  }

  if (announcementsIndex !== -1 && announcementsSourceIndex !== -1) {
    Categories[announcementsIndex].route = departmentAnnouncements[announcementsSourceIndex].pageUrl;
  }

  const [categoriesListForSearch, setCategoriesListForSearch] = useState(Categories);

  useEffect(() => {
    window.scrollTo(0, 0);
  }, []);
…
```

Figure 16: HomePage.jsx snippet in function body and before return (imports removed for brevity).

Here, the application of theoretical background of hooks is applied. First, the DepartmentContext is used through the built-in hook *useContext*, which returns the department name if the variable **depName**. This keeps in memory the (reactive) name of the department the user clicked on, on the dropdown menu on the UI, which will be showcased later. At any point, the user can now click another name. The variable **depName** will then change accordingly. After importing the JSON files with the links of the announcement pages of each department, as well as the JSON file with the links for the personnel pages (in the import section of the file), highlighted in yellow here:

```
import { useState, useEffect, useContext } from "react";
import { Categories } from "../assets/categories";
import academicPersonnelPages from "../assets/academicPersonnelPages";
import departmentAnnouncements from "../assets/departmentAnnouncements";
import MenuBox from "../components/MenuBox";
import { Flex, Grid, Heading } from "@chakra-ui/react";
import { motion } from "framer-motion";
import Search from "../components/Search";
import { DepartmentContext } from "../contexts/departmentContext";
```

Figure 17: HomePage.jsx snippet from the import section of the file.

The information is stored into the variables named: **academicPersonnelPages** and **departmentAnnouncements**. Then, a search is performed on the objects stored into these variables for the information pertaining to the chosen department name, which is stored into **depName**, with the help of *findIndex*[69]:

```
const academicPersonnelSourceIndex = academicPersonnelPages.findIndex((i) =>
i.department === depName);
// Replace the link on the categories json with the appropriate one each time
const academicPersonnelIndex = Categories.findIndex((i) => i.title === "Ακαδημαϊκό
Προσωπικό");
const announcementsSourceIndex = departmentAnnouncements.findIndex((i) => i.department
=== depName);
const announcementsIndex = Categories.findIndex((i) => i.title === "Ανακοινώσεις");
const [categoriesListForSearch, setCategoriesListForSearch] = useState(Categories);
```

Figure 18: HomePage.jsx snippet about the department information indices.

The *Categories* object, which is another imported JSON file with information about the categories of links in the Homepage, is also searched for the indices where the academic personnel and the announcements info resides.

Here, if the search is successful, then the index of the result will be returned into these four variables individually. Else, the number "-1" will be returned instead:

```
if (academicPersonnelIndex !== -1 && academicPersonnelSourceIndex !== -1) {
  Categories[academicPersonnelIndex].route =
        academicPersonnelPages[academicPersonnelSourceIndex].pageUrl;
}

if (announcementsIndex !== -1 && announcementsSourceIndex !== -1) {
  Categories[announcementsIndex].route =
        departmentAnnouncements[announcementsSourceIndex].pageUrl;
}
```

Figure 19: HomePage.jsx snippet about the population of the URLs into the Categories object.

Is the indeces of the announcements and academic personnel info are found both on the Categories object and on the other info objects (**academicPersonnelPages** and **departmentAnnouncements** respectively), the information pertaining to the URL of the page searched is injected into the Categories object, which will then be used to display the homepage UI, after being loaded as a state. This will be the subject of the next chapter.

Acknowledgement: At this point, a case could (and should) be made, that the usage of a database would be preferable to reading off of filesystem-stored JSON files. However, a database structure was not implemented due to *non-technical* limitations on university databases and their creation and management. The JSON files are only used to read information and never edited, so a static implementation such as this is functional for the time being, but could be greatly improved by directly querying a central database for information like academic personnel and academic calendars. No such central database is available at the time of authoring this thesis.

[69] MozDevNet. "**Array.Prototype.findIndex() - Javascript: MDN**." MDN Web Docs. Accessed June 18, 2024. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/findIndex.

4.2.3.2. Rendering the processed data to the UI

After gathering and processing of all vital information, the return statement which contains the JSX markup needs to be returned, utilising the aforementioned info through the UI.

```
return (
  <Flex
    id="main-grid"
    direction="column"
    alignItems="center"
    justifyContent="start"
    gap="3rem"
    h={{ sm: "100%", "2xl": "calc(var(--available-height)*0.9)" }}
    px={{ sm: "1rem", lg: "3rem" }}
  >
    <Grid
      as={motion.section}
      initial="initial"
      animate="inView"
      variants={stagger}
      className="home-grid"
      marginTop="20px"
      gap={{ sm: 2, md: 3, lg: 4, xl: 5, "2xl": 6, "3xl": 7 }}
      templateColumns={{
        sm: "repeat(3, 1fr)",
        md: "repeat(3, minmax(0, 1fr))",
        xl: "repeat(5, calc((var(--available-width) - 5rem) / 5))",
        xxl: "repeat(5, calc((var(--available-width) / 1.5) / 5))",
        "2xl": "repeat(5, calc((var(--available-width) / 1.75) / 5))",
        "3xl": "repeat(5, calc((2*var(--available-width)/3) / 5))",
      }}
    >
      {categoriesListForSearch.length === 0 ? (
        <Heading gridColumnStart={1}
                 gridColumnEnd={3}
                 fontSize={{ sm: 11.95, md: 16, lg: 26, xl: 32 }}
                 w="100%" fontWeight={500}>
          Η αναζήτηση δεν επέστρεψε αποτελέσματα.
        </Heading>
      ) : null}
      {categoriesListForSearch.map((category) => (
        <MenuBox category={category} key={category.title} />
      ))}
    </Grid>
  </Flex>
);
```

<- "This search yielded no results."

Figure 20: HomePage.jsx snippet with the returned JSX markup.

The usage of several ChakraUI-specific components like *Flex*, *Grid* and *Heading* is on display here. The exact implementation of these components is not relevant in this thesis. What is relevant, however, is the showcasing of the flexibility React offers through the use of any components, especially in accordance with Responsive Design Principles. In the code snippet of Figure 20, multiple parameters passed to components relate to how the component will change form upon the transformation of the users *viewport*. The viewport is the user's visible area of a web page. The area in question is understandably different when viewing the page from a mobile device, a laptop, or a TV screen. In the snippet, the font size, gaps, columns of the elements change *responsively*, meaning as a response to the changes in viewport dimensions, in accord with the parameters provided.

For a telling illustration, the focus is now placed on this segment of the code snippet on Figure 20:

```
 <Heading gridColumnStart={1}
                  gridColumnEnd={3}
                  fontSize={{ sm: 11.95, md: 16, lg: 26, xl: 32 }}
                  w="100%" fontWeight={500}>
…
```

Figure 21: HomePage.jsx segment of the returned JSX markup .

Judging solely based on this segment and depending on what is present in the parameter *fontSize*, the *Heading* component's return value can be safely assumed to be influenced. The exact influence, in this case, is that the {{ sm: 11.95, md: 16, lg: 26, xl: 32 }} part is setting the font size to 11.95, 16, 26 and 32 in accordance with the viewport size. When the viewport dimensions change, if the present dimensions are in the "zone" of what is configured to be, for example, size "sm" (shorthand for "small"), the font size changes to 11.95. If they are in the zone of "md" (shorthand for "medium"), the font size changes to 16. These specific zones can be configured in another file in the project directory. In this instance, it is the file theme.js:

```
import { extendTheme } from "@chakra-ui/react";

const config = {
  // The initial color mode is inherited from the OS, try to get it from the
  // localStorage also (for when the user has no cookies)
  //initialColorMode: localStorage.getItem('chakra-ui-color-mode') || 'system',
  initialColorMode: 'system',
  useSystemColorMode: true,
}

const theme = extendTheme({
  colors: {
    darkprimary: "#06273E",
    primary: "#094169",
    secondary: "#7FB6D5",
    lightBlue: "#DFE6EB",
    dark: "#1d2021",
    light: "#FEFEFE",
  },
  fonts: {
    heading: `'Comfortaa', sans-serif`,
    body: `'Comfortaa', sans-serif`,
  },
  fontSizes: {},
  breakpoints: {
    sm: "280px",
    md: "768px",
    lg: "960px",
    xl: "1200px",
    xxl: "1400px",
    "2xl": "1600px",
    "3xl": "1921px",
  },
  config
});

export default theme;
```

<- The breakpoints for viewport sizes!

Figure 22: The theme.js file contents, where some constants related to UI of the app are set .

The theme.js file is part of ChakraUI's functionality that allows developers to customise and tweak the "out-of-the-box" theme with specific constants which can be used to globally change attributes like fonts, colours on specific colour modes (dark and light mode). Here, the *breakpoints* key in the theme JSON object stores the breakpoints set by developers to manipulate components based on various screen sizes. The "zone" where a screen would be considered to have triggered the "sm" breakpoint would be 280-767px in width, for example.

The substantial takeaway from this exhibit of Figure 22 is that, instead of being forced to edit multiple files stored in various directories, the control of the basic of the cohesion of the UI is centralised a handful or even a single file, potentially saving hours of development time and testing.

Another concept that is important to focus on is the one displayed in the following lines of Figure 20:

```
{categoriesListForSearch.length === 0 ? (
        <Heading gridColumnStart={1}
                 gridColumnEnd={3}
                 fontSize={{ sm: 11.95, md: 16, lg: 26, xl: 32 }}
                 w="100%" fontWeight={500}>
          Η αναζήτηση δεν επέστρεψε αποτελέσματα.
        </Heading>
) : null}
{categoriesListForSearch.map((category) => (
        <MenuBox category={category} key={category.title} />
))}
```

<- Conditional Rendering

<- Iterative Rendering

Figure 23: HomePage.jsx segment that is responsible for the category rendering on homepage .

We can see that within the brackets "**{**", JS code is being executed, returning JSX markup in the process.

- Firstly, with the code highlighted in blue, the JSX is being returned *conditionally,* by checking whether or not the **categoriesListForSearch** variable (which has the aforementioned **Categories** object stored into it as state (see Figure 20) has a length of 0 or not. If it does, **null** is returned, thus nothing is displayed there in the UI. If the length is indeed 0, then a heading component is rendered, with a message informing the user that their search has yielded no result.

- Secondly, with the code highlighted in yellow, the JSX is being returned *iteratively*, (and conditionally, since if **categoriesListForSearch.length === 0**, the JSX would not be returned at all). Using the *map*[70] function too iterate over the object in the variable, for every element (here named "category") a *MenuBox* component is being rendered, with the category as a parameter, as well as the title as a key. It is good practice to have a key parameter while rendering multiple components in react. In turn, the MenuBox component returns JSX that is rendered with each individual category's information onto the UI of the homepage.

[70] MozDevNet. "**Array.Prototype.Map() - Javascript: MDN.**" MDN Web Docs. Accessed June 18, 2024. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map.

5. Functionality of the Application

The fine-grained control React provides over the development of the application, as well as the modularity of every component of it should hopefully be clear by now. The following chapter showcases the result of all the coding and data processing, the part that the end user experiences.

5.1. Home Page

The first screen that the user sees upon navigating onto the site is the following:

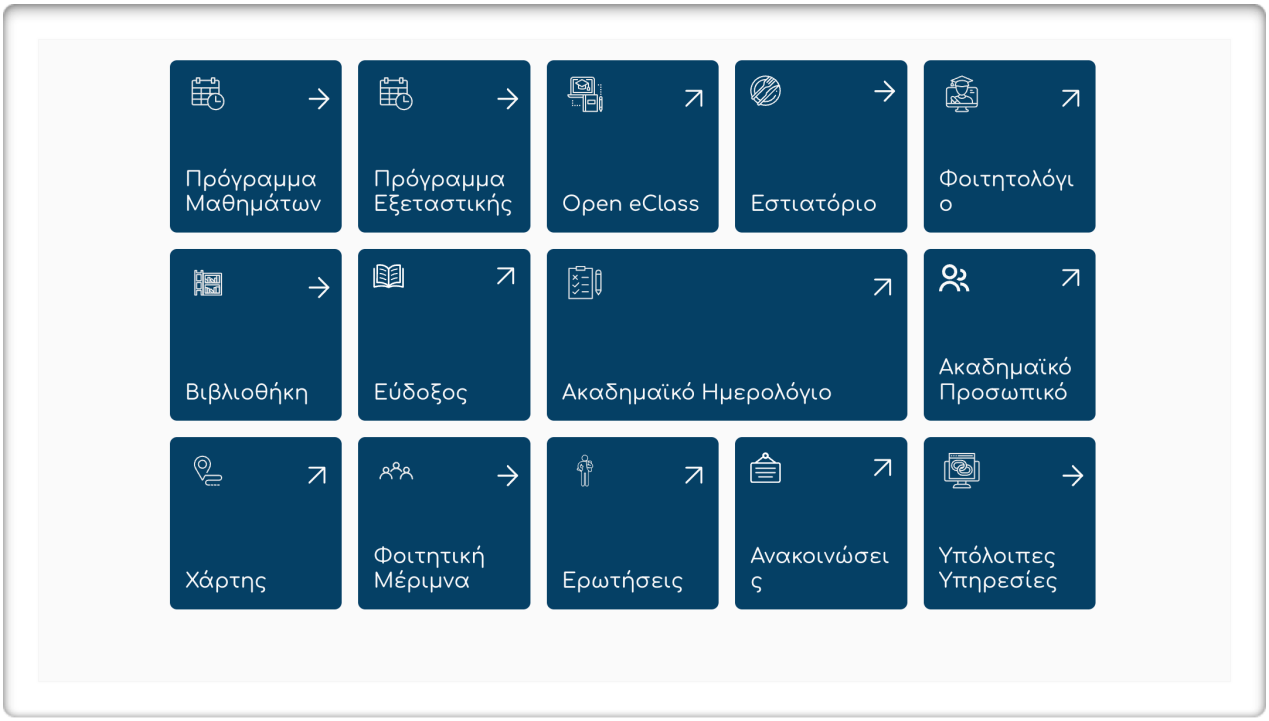


Figure 24: The Homepage of the application upon navigation.

Here, every category of link to the services of the specific user's department within the University of West Attica is visible. The buttons that display an arrow slanted to the top right (↗) are signifying the presence of an *external* link, while the ones displaying a middle right arrow (→) are signifying an *internal* link, meaning a link that leads to another page *within* the web application.

The links on this page are subject to change when the user selects a department from the dropdown menu that is visible when clicking on the settings button ( ) on the top left of the screen. When the settings button is pressed, the user is faced with the following screen:

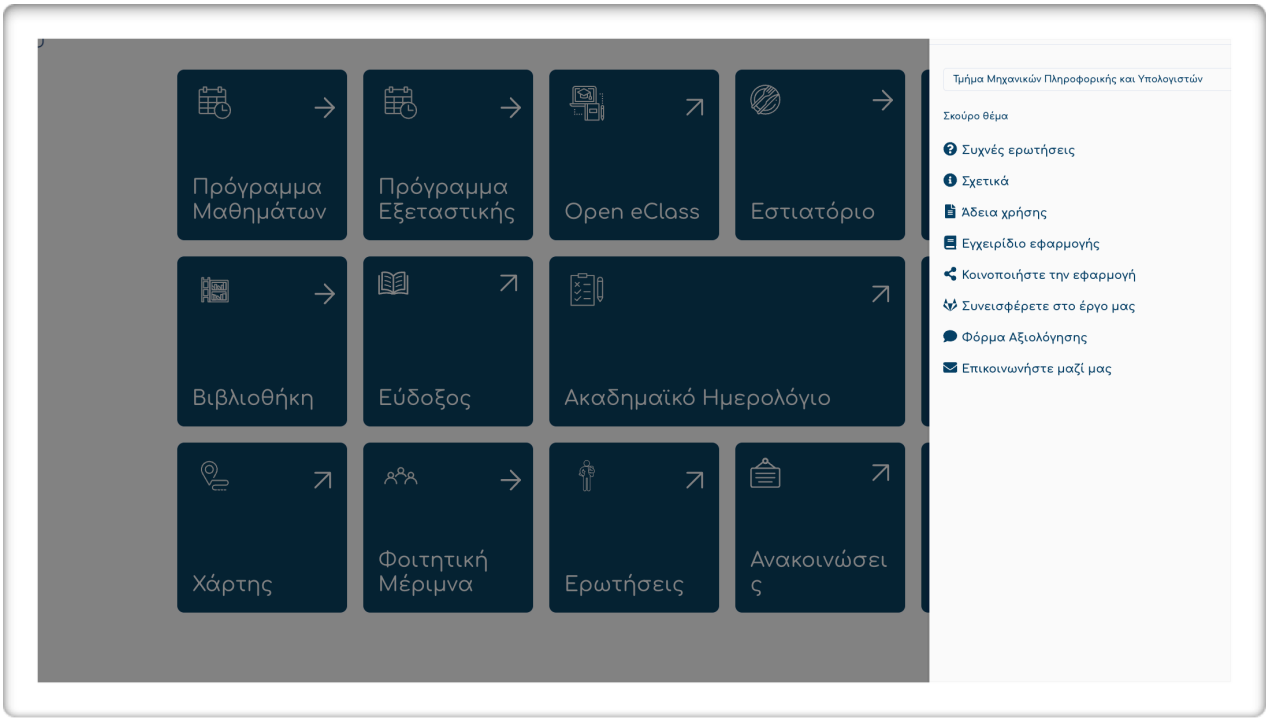
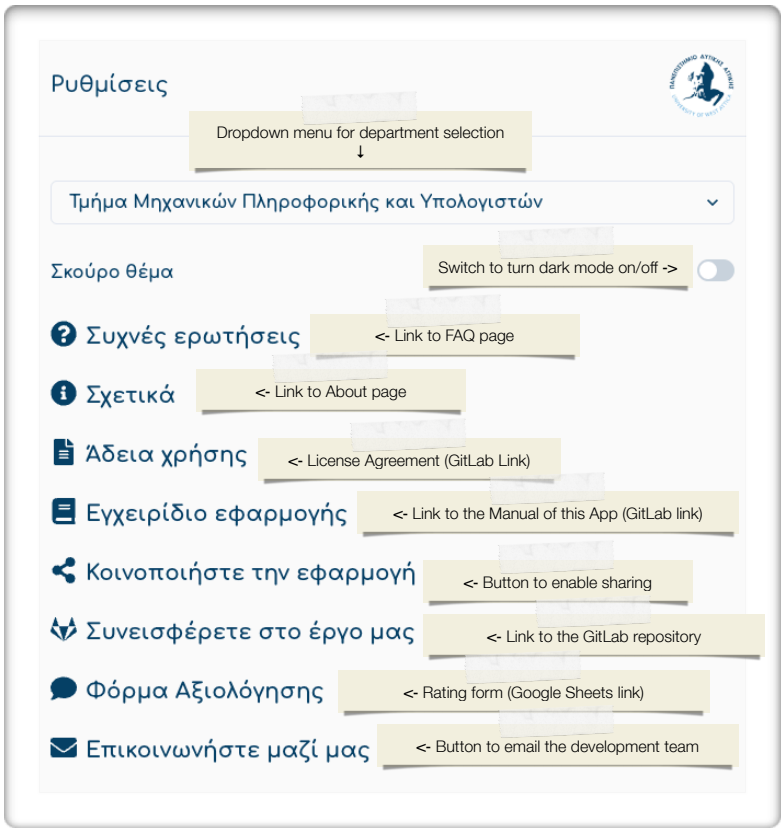


Figure 25: The Homepage of the application upon clicking on the settings button.

Zooming in to analyse the options one by one:



5.2. Internal Pages accessible through the settings menu

There are two internal pages accessible through navigation upon the settings menu, which is in turn accessible through every internal page of the application. The FAQ and the About page.

5.2.1. FAQ Page

Figure 26: The FAQ page of the application, accessible through the settings menu.

The FAQ (Frequently asked questions) page of the application is a simple page with an accordion, answering seven simple questions about things the user might be interested in. For example, to whom the application caters to, whether or not it is free, and who developed the application.

5.2.2. About Page

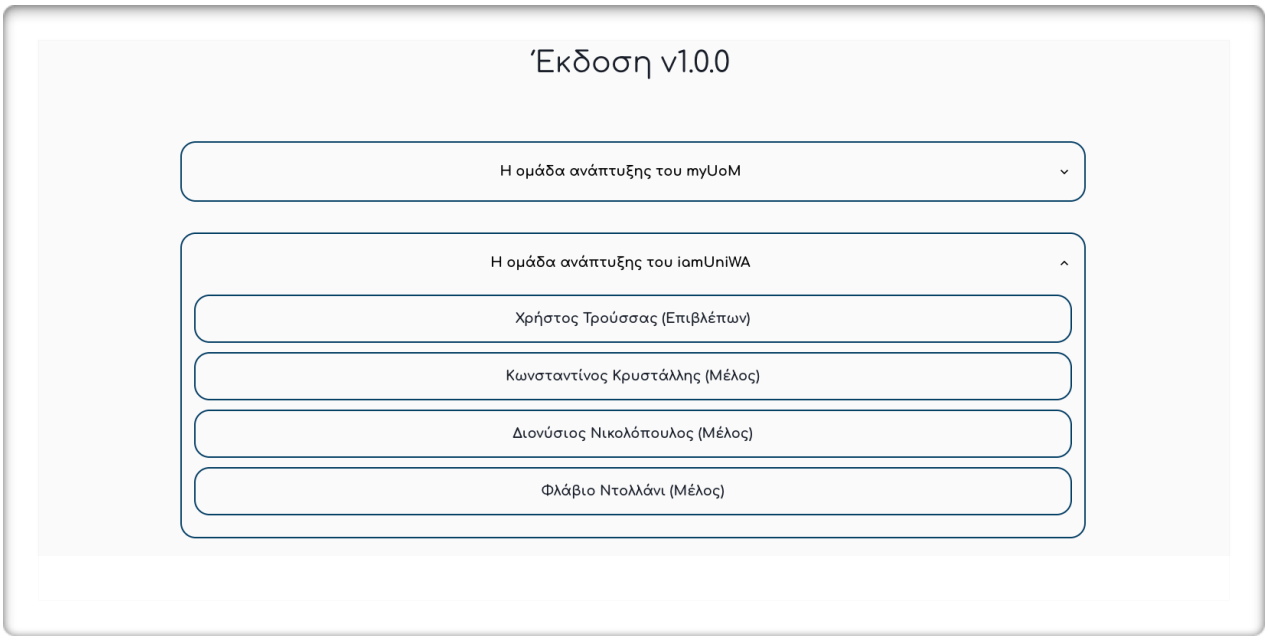


Figure 27: The Information page of the application, accessible through the settings menu.

The information page is another rather simple page with two accordions, that mentions by name everyone involved with the development of the application, both the original project (myUoM) and the UniWA fork.

5.3. Semester Schedule Page

To navigate to the semester page, the user must click upon the first category button on the Home page, after having selected a department:

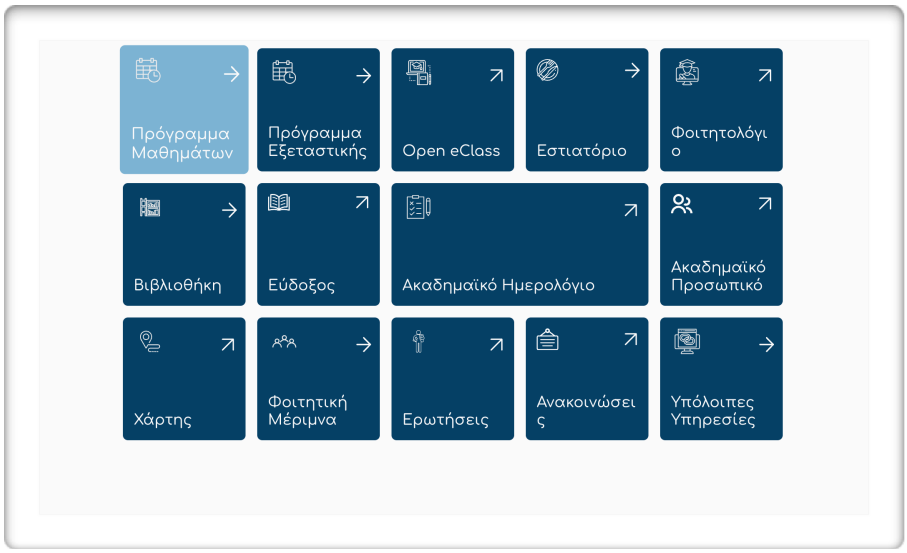


Figure 28: Navigation to the Semester Schedule page.

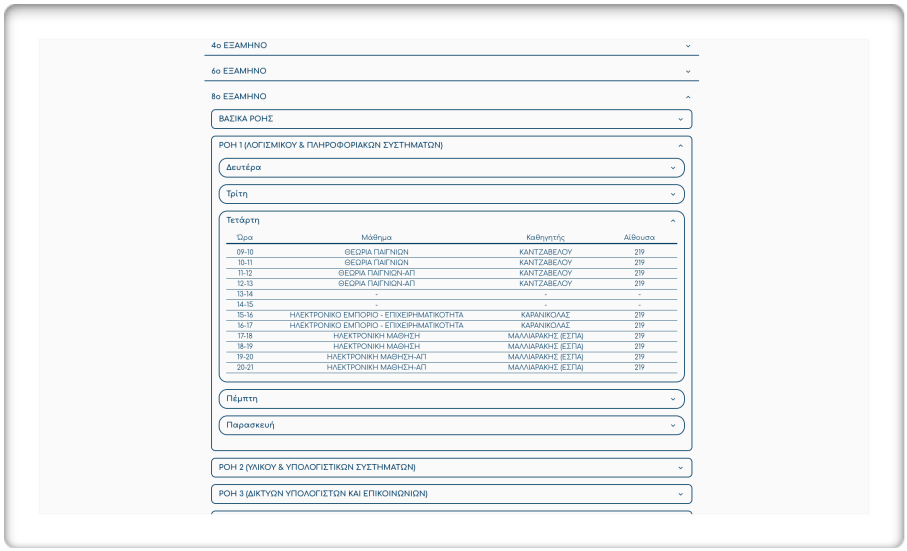


Figure 29: The Semester Schedule page upon navigation through the Home page.

If a semester schedule is available for the selected department, then the schedule will be displayed on the page using a list format and clearly formatted using a table view. Information relating to the class will also be displayed, like the professor who oversees it, the time, day of the week and location where the lectures take place.

5.4. Exam Schedule Page

To navigate to the semester page, the user must click upon the second category button on the Home page, after having selected a department:

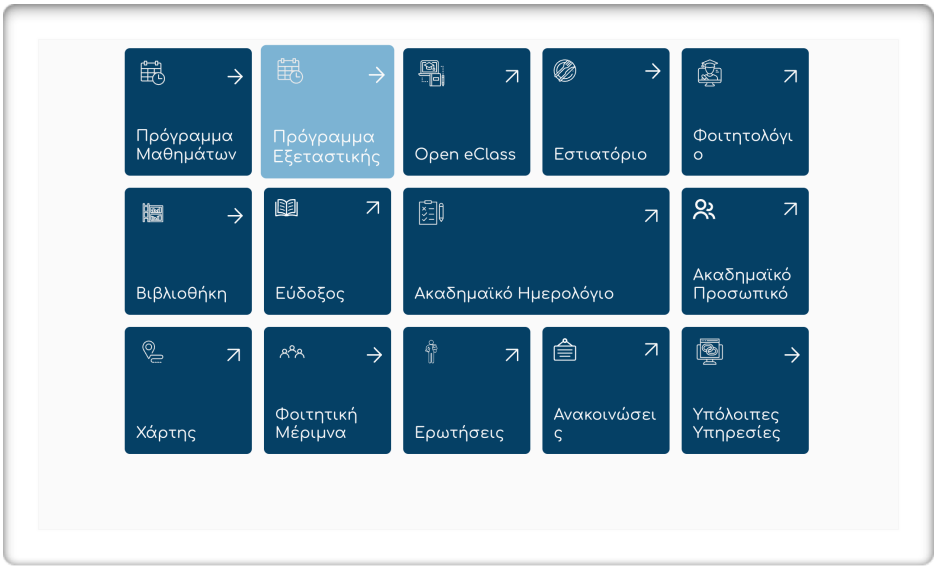


Figure 30: Navigation to the Exam Schedule page.

Similar to the semester schedule page, if an exam schedule is available for the selected department, an exam schedule will be displayed, on which all information pertaining to the exam will be shown, like the location of the exam, the week number of the exam period, the time and date of the exam.

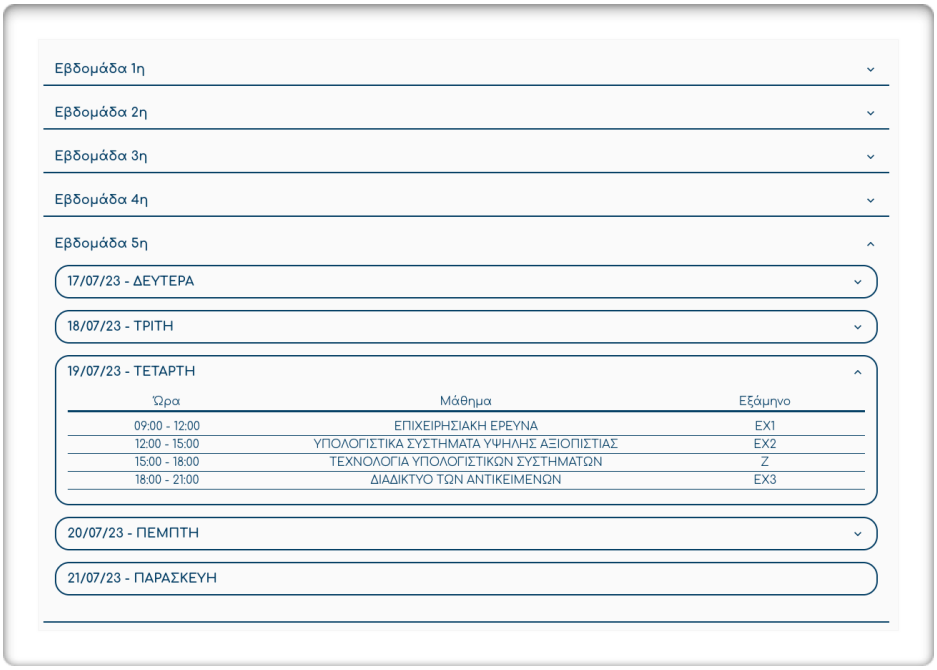


Figure 31: The Exam Schedule page, zoomed in to the exam information table.

5.5. Restaurant Page

The University of West Attica provides breakfast, lunch and dinner meals to students that fulfil certain economic requirements. These meals are provided in the restaurants of the UniWA campuses, following a weekly menu that alternates every week. The Restaurant pages outlines this menu in a concise manner, so that students are able to quickly see what is on the menu and whether or not breakfast, lunch or dinner is *currently* being served or not.

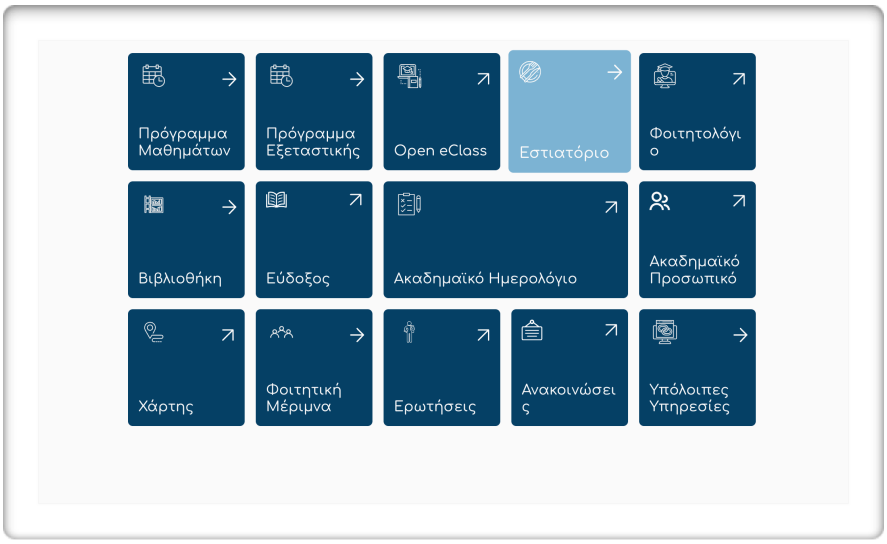


Figure 32: Navigation to the Restaurant page.

Navigation on this page is available through clicking the fourth button on the home page, the one that houses an icon with a plate and a knife and fork on the top left.

Figure 33: The Restaurant, zoomed in to the menu information table.

Noteworthy insights on this page are the features of seeing which menu is on rotation this week and whether or not breakfast/lunch/dinner is being served currently. If nothing is being

served at the time the user navigates this page, the menu for tomorrow's lunch is being highlighted on the bold heading on top of the page. The lunch is displayed instead of breakfast or dinner by default here, since it is by far the meal most students choose to use the restaurants. Below the header, every day of the 2-week menu is displayed and able to be viewed.

5.6. Library Page

The University of West Attica also provides libraries within its campuses, in which people can read and borrow books, as well as study in peace and use the facilities' computers.

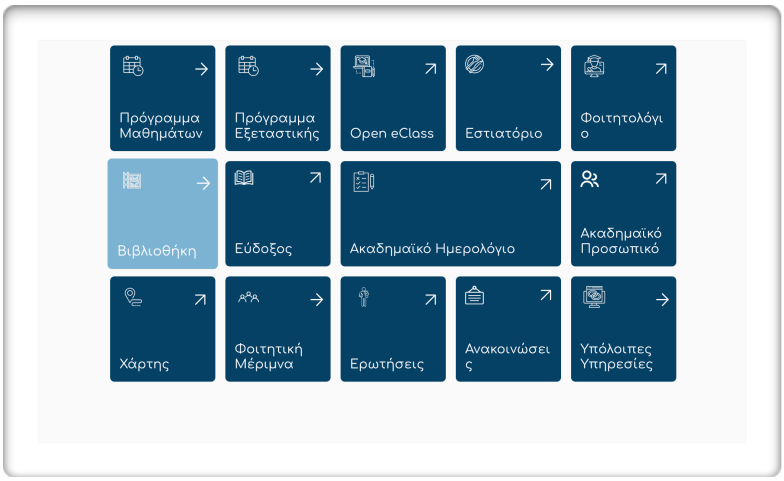


Figure 34: Navigation to the Library page.

Navigation on this page is available through clicking the fifth button on the home page, the one that houses an icon of a bookshelf on the top left.

Figure 35: The library page, zoomed in to the information elements.

There are three distinct libraries that are part of UniWA, each with it's own contact phone numbers and schedule. In the library page, all of this information is formatted and displayed to the user, as well as the external link for the official library webpage (by UniWA) on the bottom of the page.

5.7. Student Care Page

The student care page pertains to special benefits and programs some students are entitled to by law, usually depending on their social, economic or health status. There exist specific pages for these benefits and programs, which are all aggregated in the application on the Student Care Page.

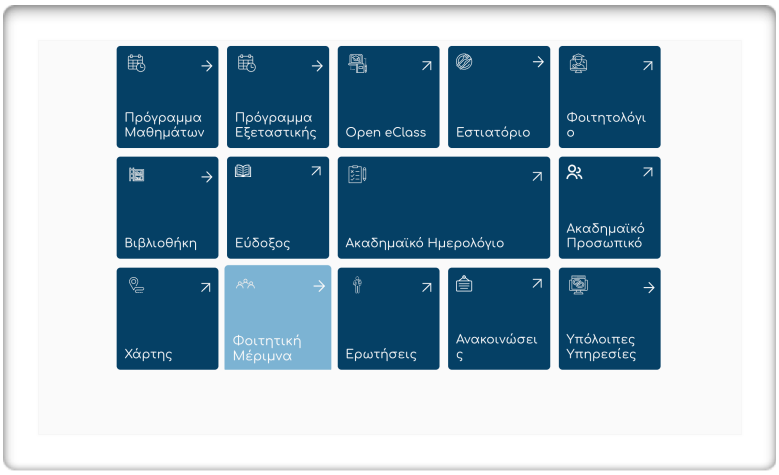


Figure 36: Navigation to the Student Care page.

Navigation on this page is available through clicking the eleventh button on the home page, the one that houses an icon of the outlines of three people on the top left.

Figure 37: All the Student Care page external links, zoomed in.

The links displayed here are all external, leading to the official websites of each program/ facility.

5.8. Services Page

There are some services which cannot be organised based on the categories established until this point. Each one has an official webpage, whose external link is provided in.

Figure 38: Navigation to the Services page.

Navigation on this page is available through clicking the eleventh button on the home page, the one that houses an icon of the outlines of three people on the top left.

These aforementioned service external links are housed on the Services Page, where they are grouped into three sub-categories: *Offices*, *Student Groups* and *Rest of Services.*

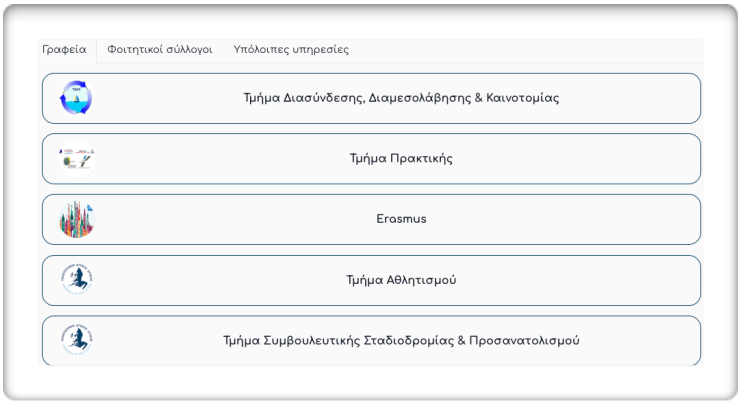


Figure 39: "Offices" tab on the Services Page, zoomed in.

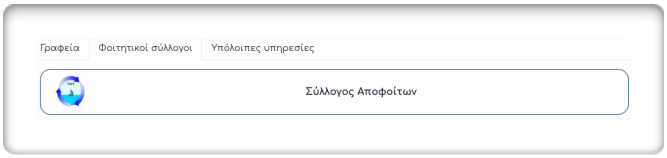


Figure 40: "Student Groups" tab on the Services Page, zoomed in.

Figure 41: "Rest of Services" tab on the Services Page, zoomed in.

5.9. External Pages & Homepage Map

In this chapter, the following figure is provided with annotations on links to external pages from the home page:

Figure 42: All the links in the homepage explained.

5.10.Additional Responsive Features

Apart from the features outlined until this point in Chapter 5, there are some additional responsive and modern features that the web application includes too.

5.10.1. Responsive Dark and Light mode theme

During the last decade, the prevalence of the feature of “dark mode” has been observable in nearly every device, from mobile phones to desktop computers. Dark mode is a paradigm of interface design, where instead of light backgrounds contrasting against dark details and lettering, the opposite it true, having light lettering and details on dark backgrounds. For example, on macOS, the two modes side-by-side, on each half of the window at a time (edited representation) would look like:

Figure 43: Light and Dark mode and the visual change their enabling typically makes. Image Credit:[71]

Tech giants like Microsoft[72] and Apple[73] have both integrated what is, effectively, dark mode into their operating systems, with Microsoft since 2016[74] and Apple since 2018[75]. While there are lot of options for[76] and against[77] ease-of-use and eye-strain prevention, one thing is for certain. There is a demand for this feature on the market. Most OSes even have a setting where

[71] **Sydney CBD Repair Centre**. "How to Enable Light Theme on Mac OS." Pinterest, December 28, 2018. https://pin.it/1hSBZowPb.

[72] "**Using Dark Mode in Windows 11: Windows Learning Center**." Windows. Accessed June 19, 2024. https://www.microsoft.com/en-us/windows/learning-center/when-to-use-dark-mode.

[73] "**Dark Mode.**" Apple Developer Documentation. Accessed June 19, 2024. https://developer.apple.com/design/human-interface-guidelines/dark-mode.

[74] Paul, Ian. "**The Anniversary Update’s Most Exciting Features: Windows 10 Users Weigh In.**" PCWorld, July 26, 2016. https://www.pcworld.com/article/415733/the-anniversary-udates-most-exciting-features-windows-10-users-weigh-in.html.

[75] Clover, Juli. "**MacOS Mojave: Dark Mode, Stacks, & More**." MacRumors, March 18, 2020. https://www.macrumors.com/roundup/macos-10-14/.

[76] Cummins, Eleanor. "**Dark Mode Is Easier on Your Eyes-and Battery.**" Popular Science, November 21, 2018. https://www.popsci.com/night-dark-mode-design/.

[77] Clarke, Laurie. "**Dark Mode Isn’t as Good for Your Eyes as You Believe**." Wired, July 30, 2019. https://www.wired.com/story/dark-mode-chrome-android-ios-science/.

after sundown (tracked with geolocation) or at a certain user-set time, dark mode is enabled, only for it to be switched to light mode on when the sun rises (or another user-set time). The positive effect on battery life dark mode, however seems to be backed by actual fact, but only on screens with technologies like OLED, which disable backlighting when displaying dark colour[78].

Regardless, a dark and light mode on the app is available. This feature's development was greatly aided by the centralised control of React that was mentioned in Chapter 4.3.2.3. The Web Application automatically tracks the state of the dark or light mode on the user's device and appropriately sets the theme on the website as well. The user also has the option to manually disable or enable dark mode from the settings menu, as visible in Figure 25.

Figure 44: Dark and Light mode and the visual change in the Homepage

Figure 45: Dark and Light mode and the visual change in the Restaurant Page

[78] Welch, Chris. "**Google Confirms Dark Mode Is a Huge Help for Battery Life on Android**." The Verge, November 8, 2018. https://www.theverge.com/2018/11/8/18076502/google-dark-mode-android-battery-life.

5.10.2. Disabled Options on Missing Selection of Department

Upon the user's first navigation upon the application, the web app understandably has no cookies stored in regards to the user's department and cannot determine which department the user is interested in. Thus, upon first navigation, the user will be faced with the following screen:

Figure 46: Screen upon first navigation (or) with no cookies. Notice disabled buttons.

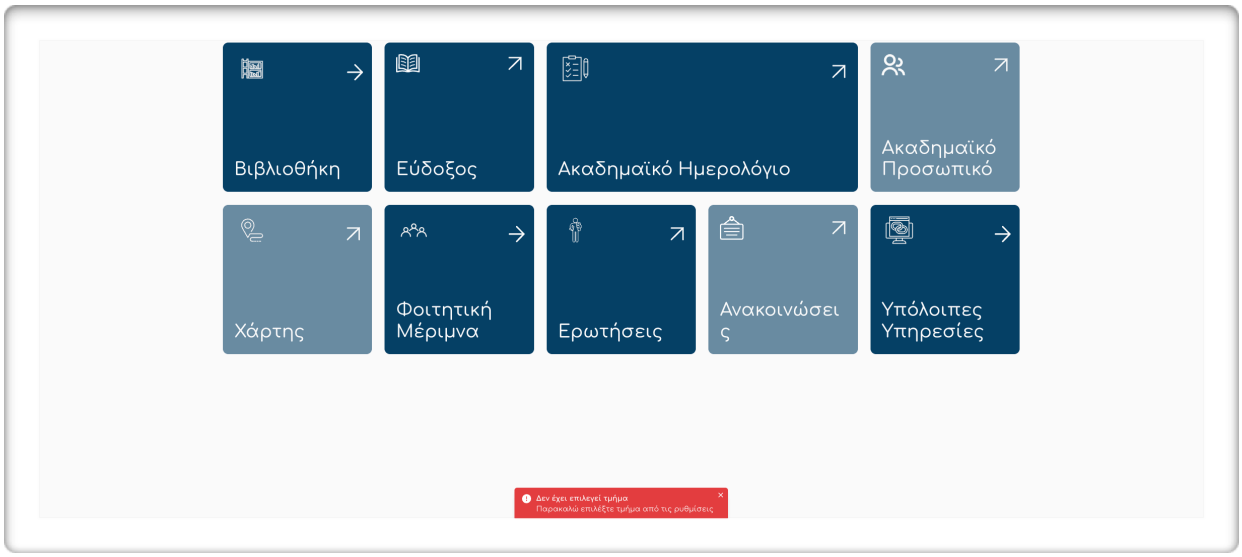


Figure 47: Warning upon clicking disabled button.

When and if a user clicks on a disabled button without having selected a department first, a warning will be displayed on the bottom of the screen informing them to select a department, so that the app load the necessary info for some department specific buttons and to then click the desired button again.

5.10.3. General Responsiveness on Smaller Screens

The web application was from the ground up designed to enable easy usage with mobile devices by transforming the interfaces, sometimes completely, in order to accommodate smaller screen sizes and mobile-centric UX paradigms. For example, this is the homepage and restaurant page of the application when viewed by a mobile devices:

Figure 48: Restaurant and Home Page when viewed by a smaller screen.

But this is not the only feature that is mobile centric in the application. When the application detects that the user has entered the application through a mobile device (and not a browser window, even a smaller one), there are some more UI elements present that will be showcased in the following chapter.

5.10.4. Additional Features on Mobile Devices

As mentioned previously, when the user is entering the web app with a mobile device, some more features appear on-screen. In the homepage, a bottom navigation bar with three distinct buttons appears.

Figure 49: Home Page when viewed by a mobile device, notice the navbar at the bottom.

The first button, displaying a home icon leads back to the homepage. The navigation bar is always present on the bottom of the screen on mobile devices, so it can be used from multiple pages. The second button, displaying a gear, opens up the side menu of the settings, which is now full screen to accommodate for the smaller screen size:

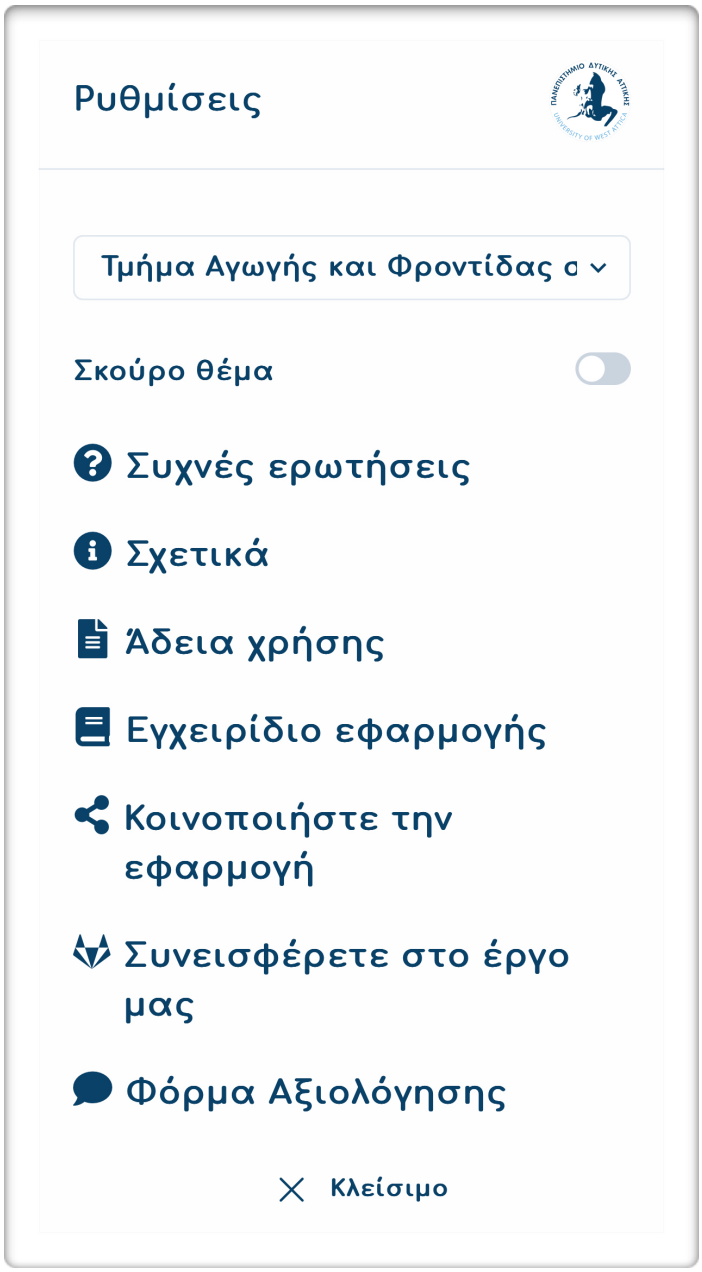


Figure 50: Settings Menu when Settings button is clicked on a mobile device

Lastly, there is the red phone button on the rightmost side of the bottom navigation bar. This button is the emergency call button, which is only useful is the user's device is able to receive and make calls. Hence, it is an exclusive mobile device feature that is not visible with a desktop computer or laptop. When the user clicks on the button, they are redirected to the call app on their mobile device with a predetermined phone number ready to be called on their screen. This number is the specific number of the campus security. The application determines the phone number depending on the department the user selected. The behaviour following a click of the button will be akin to this screenshot:

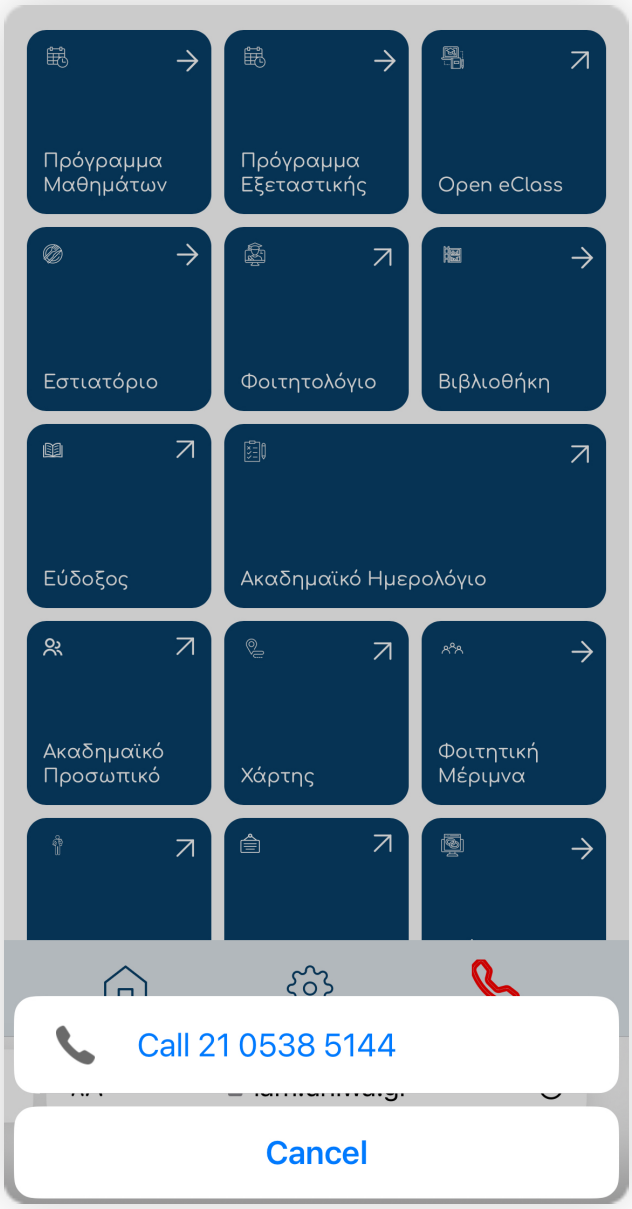


Figure 51: Screen right after clicking the emergency button on iOS

When the user clicks to confirm the call on the pop-up responsible for confirming on their device's OS, then the call to the campus security starts immediately. This functionality is important to be quick for the user to utilise even under stress, due to the inherent increased possibility of the campus security being called on times of crisis. A user who has previously selected their department on the site does not have to reselect it, since the site keeps the cookie of the selection for the specific device.

This is significantly useful, since it removes the burden from the user to have to search the phone number of the campus security of their campus, which would be less than ideal to have to deal with when in a time of need.

Appendix

In this appendix, the complete tree of the source code of the application is presented:

```
.
├── LICENSE
├── README.md
├── package-lock.json
├── package.json
├── public
│   ├── index.html
│   ├── logo192.png
│   ├── logo192_old.png
│   ├── logo512.png
│   ├── logo512_old.png
│   ├── manifest.json
│   ├── robots.txt
│   └── service-worker.js
└── src
    ├── App.js
    ├── App.test.js
    ├── assets
    │   ├── DepNames.js
    │   ├── FirstYearInfo.js
    │   ├── Graduation.js
    │   ├── ScheduleLink.js
    │   ├── UniWALogo.png
    │   ├── UniWALogo_alt.png
    │   ├── academicPersonnelPages.js
    │   ├── categories.js
    │   ├── departmentAnnouncements.js
    │   ├── departmentExamsSchedules.js
    │   ├── departmentSemesterSchedules.js
    │   ├── dinner.png
    │   ├── fonts
    │   │   ├── Comfortaa
    │   │   │   ├── Comfortaa-Variable.ttf
    │   │   │   ├── OFL.txt
    │   │   │   ├── README.txt
    │   │   │   └── static
    │   │   │       ├── Comfortaa-Bold.ttf
    │   │   │       ├── Comfortaa-Light.ttf
    │   │   │       ├── Comfortaa-Medium.ttf
    │   │   │       ├── Comfortaa-Regular.ttf
    │   │   │       └── Comfortaa-SemiBold.ttf
    │   │   ├── Syne-Bold.woff2
    │   │   ├── Syne-ExtraBold.woff2
    │   │   ├── Syne-Medium.woff2
    │   │   ├── Syne-Regular.woff2
    │   │   └── Syne-SemiBold.woff2
    │   ├── libraryData.js
    │   ├── mapData.js
    │   ├── menus
    │   │   ├── FirstWeekMenu.json
    │   │   └── SecondWeekMenu.json
    │   ├── professors.js
    │   ├── projectMembers.js
    │   ├── questionsForFAQ.js
    │   ├── secretaries.js
    │   ├── services.js
    │   └── Εγχειρίδιο.pdf
    ├── components
```

```
    │   ├── FAQCard.jsx
    │   ├── FoodMenuList.jsx
    │   ├── Header.jsx
    │   ├── InfoCard.jsx
    │   ├── LibraryCard.jsx
    │   ├── MapCords.jsx
    │   ├── Menu.jsx
    │   ├── MenuBox.jsx
    │   ├── MenuButton.jsx
    │   ├── ProfCard.jsx
    │   ├── ProfList.jsx
    │   ├── ProjectMembersCard.jsx
    │   ├── ProjectMembersList.jsx
    │   ├── Schedule.jsx
    │   ├── Search.jsx
    │   ├── SecrCard.jsx
    │   ├── ServicesCard.jsx
    │   └── TodaysMenu.jsx
    ├── contexts
    │   └── departmentContext.jsx
    ├── hooks
    │   └── useLocalStorage.js
    ├── index.css
    ├── index.js
    ├── pages
    │   ├── AboutSettingsPage.jsx
    │   ├── ExamsSchedulePage.jsx
    │   ├── FAQSettingsPage.jsx
    │   ├── FirstYearInfoPage.jsx
    │   ├── GraduationPage.jsx
    │   ├── HomePage.jsx
    │   ├── LibraryPage.jsx
    │   ├── MapPage.jsx
    │   ├── ProfInfoPage.js
    │   ├── RestaurantPage.jsx
    │   ├── SchedulePage.jsx
    │   ├── SemesterSchedulePage.jsx
    │   └── ServicesPage.jsx
    ├── reportWebVitals.js
    ├── setupTests.js
    └── theme
        └── theme.js
```