OULU UNIVERSITY OF
APPLIED SCIENCES

Ngoc Phuong Quynh Dao

FRONTEND FOR PYTHON ONLINE LEARNING PLATFORM

FRONTEND FOR PYTHON ONLINE LEARNING PLATFORM

Ngoc Phuong Quynh Dao
Bachelor's Thesis
Autumn 2021
Bachelor's in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree of Information Technology

Author: Quynh Dao
Title of the thesis: Frontend for Python online learning platform
Supervisor(s): Kari Laitinen, Lasse Haverinen
Term and year of thesis completion: Autumn 2021 Pages: 74

Interjektio is a software company which doing consulting in web and cloud environments with Python. They had been growing and demanding a system for purpose of helping new team members. Therefore, this thesis is aimed at an online tutorial platform for Python programmers as an internal environment.

The process of implementing the platform had started from designing, building the view of the application, and connecting to the backend logical server written with Python in Pyramid. The design was using Figma and technologies for frontend implementation used were focused on React, Redux, TypeScript.

The project result is a system of web application of Python tutorial platform with a variety of exercises and lessons. The platform contains a list of mock data exercises and the code editor. This is only the early-staged prototype and still requires future development. Continuous Integration and Automation setup and users' right to update and review coding exercises, and integrations into cloud environment will be developed in the future.

Keywords: Design, Figma, React, Redux, TypeScript

PREFACE

First of all, I would like to send a thank you to my teammate, Long Nguyen, for developing Pymestari with me, supporting and encouraging me while working together. I am also thankful for introducing me to Interjektio and thanks to Interjektio for inspiring us with the idea of a Python learning platform. Yet, I would like to say apologize for my delay.

Hence, I would like to send my deepest appreciation to my supervisor Kari Laitinen for the patience, supports, and guidance for me to finish my thesis. I also would like to appreciate my second supervisor - Lasse Haverinen for his guides and feedback on my thesis. He helped me so much to complete with the content and structure of my thesis. Then, I would like to apricate the Head of my Degree Program, Susanna Kujanpää, for her active follow-up and for helping me in steps in my study time and also during the thesis writing time. And next is my sincere thanks to my teacher Kaija Posio for spending her time checking and guiding me through my thesis. Without their help and encouragement, my thesis could not have been finished successfully.

Lastly, I would like to extend my gratitude for my family and all of my friends, who have been going with me through my study, for their eternal and boundless love to give me the motivation and lift my spirit during the hard and dark time in my life and during my thesis time. Without their faithful understanding and sympathy, my today achievement would not have gone this far. Thank you very much.

Helsinki, 14.12.2021

Quynh Dao

TABLE OF CONTENTS

VOCABULARY

CSS	Cascading Style Sheets
DOM	Document Object Model
ES6	ECMAScript 6
HTML	Hypertext Markup Language
UI	User Interface
UX	User Experience
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
JSX	JavaScript XML
props	Properties
px	pixels (unit)
TSX	TypeScript XML
XML	Extensible markup language

1 INTRODUCTION

1.1 Thesis overview

There are many programming courses, playgrounds, and learning platforms for programmers to learn Python available on the Internet. Yet, none of those met the demand of Interjektio for their internal use for new programmers of the team. That raised the requirement of a platform that is suitable for training their programmers.

Inspired by this realistic demand, the aim of this project was to produce a web application form of the Python learning platform. This project was developed by two students who were assigned to take part in designing and implementing the frontend part of the website. My responsibility was to design and build the web interface. My partner was taking care of the backend part which includes building the server, environment, setting up the database, performing APIs and logic to connect features from the frontend. Team members were working remotely, delivering their work to Github, and making discussions on Slack.

The objective of this thesis was to create an early-stage application which contains two main modules: explore and challenge. The explore is given to help beginners to get started with Python by theory and exercises while the challenging part provides a library of questions for the purpose of testing and improving users' skills in Python. This application consists of two parts: frontend and backend. The frontend role includes rendering the front page, dialogs, and modules, while the backend takes care of the processing after the action has been made from end-users.

In order to achieve the objective of this solution, the works had been divided into different parts for front-end tasks. Firstly, the investigation on creating a good UI, which means giving the similar experience as using other similar online study programming platforms, such as Leetcode and Edabit. Second is about the study

and research on how elements in designing could effect on the product's view. Then is about the study to choose the set of tools and languages for implementing the work. And last is about the work of implementing the functionalities of the platform, to provide a validated method environment for submitting the solution for the challenges, which is a combination work of checking from server, then get the results and display on the client side.

1.2 Pymestari workflow

As mentioned earlier in the overview, the platform contains two main paths: explore for studying the theory and challenges for improving skills by solving problems.

The theory includes basic modules for beginners to advanced levels modules for skilled programmers. In each module, not only theory is given as content and examples, but there are also several questions for learners to apply the lesson. The study path must be taken from easy to hard level respectively. Challenges are including the list of questions and problems from easy to hard within different categories. There are also solutions and discussion topics opened for each challenge where users can discuss and learn from each other for deeper understanding.

Similar to other platforms, before developers can start their learning path, they must get authenticated to the system. After that, they can freely choose an issue which they want to solve and submit their code with our editor provided. Our system will analyse their answers and return feedback on whether the answer is acceptable.

In the future development, users' progress will be saved into users' profiles for reviewing and editing. On the other hand, the study progress can be skipped by passing the test after each module, therefore skilled learners do not waste their time at an easy level.

2 THEORETICAL BACKGROUND

This section will introduce the study and explain technologies will be used in this application.

2.1 Design

2.1.1 Definition of UI and UX

As the target is aimed at developers in general and Python programmers in specific, Pymestari is designed and developed as a web application, which is mainly opened by widescreen browsers. The design of Pymestari was made first on paper then converted into prototypes, by Figma. In this project, as aiming for a website, the definition for design will be introduced for websites in specific.

This definition includes two main terms: user interface (UI) and user experience (UX). Firstly, what is the user interface? It is the part of computer software that users can communicate with (1). UI design includes designing any visual surface that users can interact with. For a website, a user interface includes the layout of the website, color and color theme, typography, and animations (2). Eventually, user experience is about the product's usability and how convenient it is when a user starts to use the application. Figure 1 below gives a simple comparison between the UI and UX.

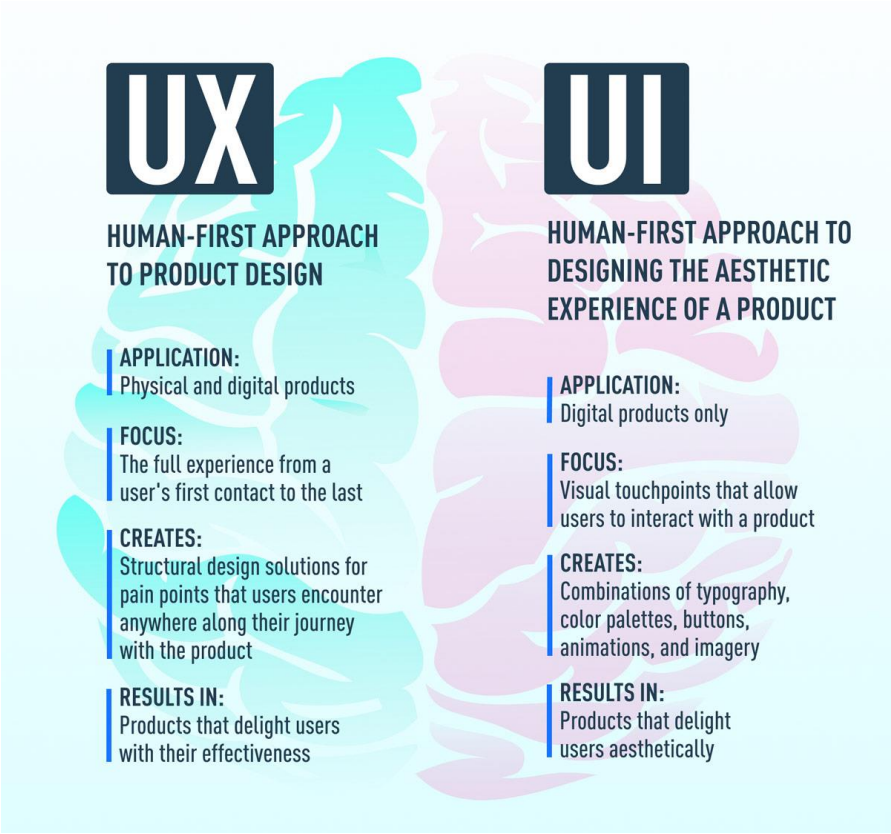


FIGURE 1. Comparison of UX design and UI design (1)

By Vitaly from Smashing Magazine, design for the user interface is not only about layout, buttons, and text but also includes the interaction of users to the product. It means that the position of each element also affects the way users use the product. In other terms, UX is said to be the way how users encounter while using products. (3)

In conclusion, user interface design is designing how a product looks, feel, and collection. Additionally, user experience design is about the way users interact and use the product – design how users will experience the product. (3)

2.1.2 Principles for good UI design

A good UI design product is evaluated on seven criteria as main principles. (4)

Grid – layout

The first criterion is about the layout – “line things up”. The method to keep all of the items in a design is to create a grid for it. Adding a grid layout into a site helps items align on a straight line and make our design clean. A grid is added to help designers easier in deciding where to put the element on a website, instead of choosing a random position on a full blank page. Line all elements make a feeling of unity. Figure 2 below shows a 12-column grid from Bootstrap, which is a commonly used styling library, very helpful and easy to adjust to most of the designs. (4)

FIGURE 2. The grid system from Bootstrap (5)

Colors

The next aspect is important as layout - colors. Colors are playing a key role in design. Different colors give different emotions and feelings in design and from users’ perspectives. Warm colors make people feel energized and active while cold colors give a calming and stable feeling. (3)

When starting to design a website interface, choosing a color theme is a must. A palette of colors should contain only from two to four colors because if the number of colors is more than four will make a website look too busy and messy, which implies too many things on a website. (4)

For making a strong emphasis on an important and attractive element, one pop-out color is chosen with different shades from the others within a palette. A clear pop color helps users easier to reach the website and its core content. (4)

Figure 3 and figure 4 reveal two different websites whereas one has a strong contrary and the other does not.

FIGURE 3. The Siminki website main page does not have a strong pop of color, so the page is hard to read (4)

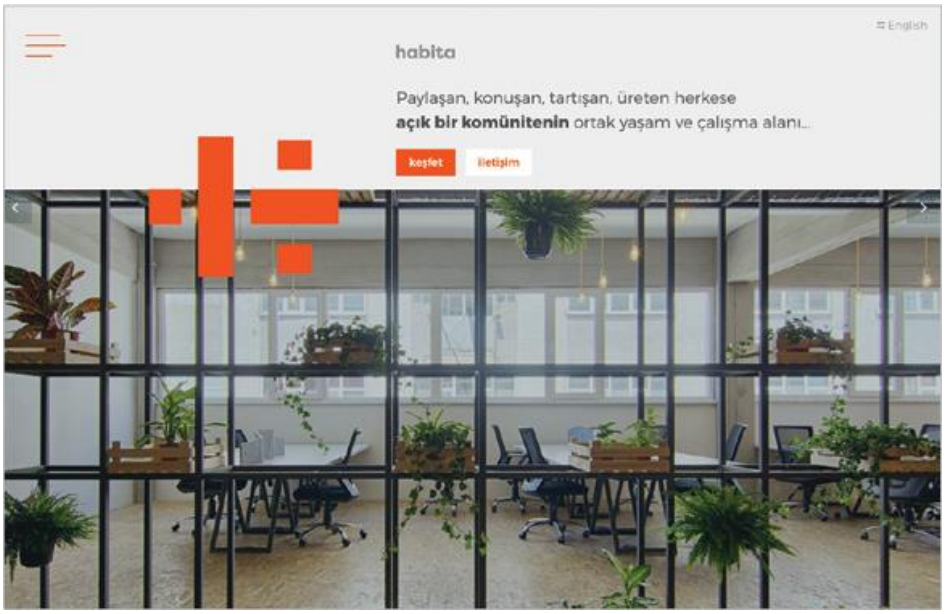


FIGURE 4. The Habita page has a strong pop of color that makes the page more readable. (4)

Colors used in a color palette are chosen by designers' perspective, experience, feeling, and by the website concept. However, there are many internet sources available for references and ideas. They provide different color palettes which are

ready to use or provide the idea of how colors are combined, such as Colormind, Material Design from Google or Adobe Color sites. (4)

Typography

Moreover, the title and content of a website are usually a focal point. Therefore, typography becomes one standard to decide how friendly a UI is. The strategy applying for paragraphs is to make the content easy for reading and send enjoyable pleasure to readers. (4)

There are two main font family types: serif and sans-serif. The difference between serif and sans-serif is the little bit on a letter of serif fonts. The sans-serif fonts are mostly used on electrical means, on-screen devices, while serif often belongs to printed text. (4) (See figure 5 below)

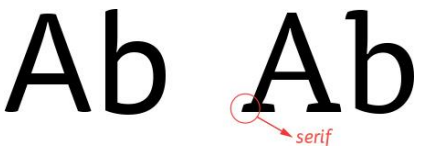


FIGURE 5. On the left is Tisa Sans Pro (sans-serif) font and on the right is Tisa Pro font (serif) (4)

Besides two commonly used types of Serif and Sans-Serif, there are available other font categories including Slab Serif, Monospace, Display, and Handwriting. Using a special font makes an element emphasized and attractive. (4)

The readability of website content is affected by the font family, leading, and kerning. The ‘leading’ term indicates the height of a line of text, while the ‘kerning’ term describes the space between words. To make the paragraph of text easily read and understand, leading and kerning must be applied in a suitable space. (4)

Although text should not be staying too close together as it makes no space illusion and becomes the messy for users’ eyes, the text should not have large space either as it creates sparse and infrequent feeling. In terms of kerning, most font families have a good scale by default, even though designers can make a

slight change for a better result. On the other hand, about leading, to avoid the bad effect given from the inappropriate space, the good ratio to get started from is 1.6. It means that the height of a line is 1.6 times the size of the font. This ratio can be slightly adjusted to get the better effect as application's demand. For example, text at size 12px will go with the height of 19.2px. (4)

By experts, the good amount of font families used on a website should be limited to two as more typefaces will make the site look disorderly. (3) Choosing two simple typefaces, one for headline and one for the content is enough. (4)

Additionally, justifying text should be avoided as it will give more space while reading and increases the distraction. So just simply using left-aligned makes the text easy to read. On the other perspective, center-aligned text can be applied to the header or title line. But using centered text for a paragraph will increase the reading difficulty level as each line will have different starting and ending points, which causes the loss of focus while reading. Users are stated to be more focused when all text lines start at the same position. (4)

To maximize the readability level, each line of text also needs to have a suitable length. The good length as recommends in the 'Hello Web Design' book is between 45 and 75. (4)

Likewise, the wide range of tools generates color palettes for designs as mentioned in chapter 3.2. There are many resources from the Internet available for selecting good font families for free or paid, from simple to complicated. A large library for wide selections of fonts, which is commonly used and recommended by designers, is Google Font. (4)

White space

It is stated that white space can be one of the tools that makes an impressive change to the design. White space can decrease the feeling of messy between elements if they are too near to each other. White space in design is not only

about the space in white, but it also stands for the space without an element – an empty space. (3)

From a design perspective, it does not mean if a page full of information will perform the best result. Too crowded information makes the page have too many things and it increases the hardness in reading and confuses for readers. The proper amount of space between elements makes the page more breathable and readable. (4)

For example, the Foursquare website in the figure 6 below gives users enough blank space, so they know where to focus, what to read, and how to use it. Users know that they can get authenticated from the top right corner buttons or that they can search from the middle of the site. (4)

FIGURE 6. Foursquare website front page (4)

In conclusion, white space plays an important role in raising awareness, improving readability, giving a tone of theme, and increasing the willingness to take action. Space between text helps improve the readability, meanwhile, space between elements makes them apart and helps readers' eyes easily pick the information. Therefore, objects are divided into groups to make the distinction by this given space. (4)

Layout and hierarchy

Then, the next rule is about how components are set, which describes the term layout. The layout is a more abstract concept in a design theory than the mentioned aspects above. (2)

Nowadays, most people in the world have known English as their second language, therefore most of our consumers know English. It is stated that the layout of our website should be in an “F” pattern, as English readers habit reading from left to right as shown in figure 7. Therefore, applying setting components up as in the “F” pattern will save time on the moving of eyes when reading. However, for a website with different languages which has a different direction of reading text, from right to left, the “F” pattern is useful as in the direction from left-to-right websites. (4)

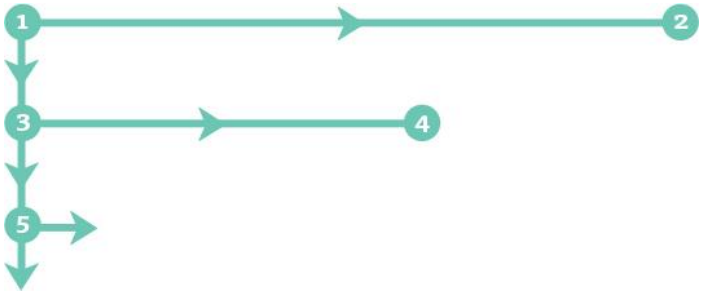


FIGURE 7. Eye tracking heatmap shows how English readers read in “F” pattern (4)

Content

Even though content does not have much relating to design, it takes an important effect on users’ eyes. A paragraph of text with enough space would let users easier to read and get the main idea. Most users skim the paragraph instead of reading all text on web applications in specific and on electrical screens in general. Lots of text will be skipped and the main content might be lost as well. It can be shown on the path how users’ eyes are going through in figure 8. (4)

As in design, the “less is more” term stands for writing less text and using common words to help people understand the core meaning. (2)

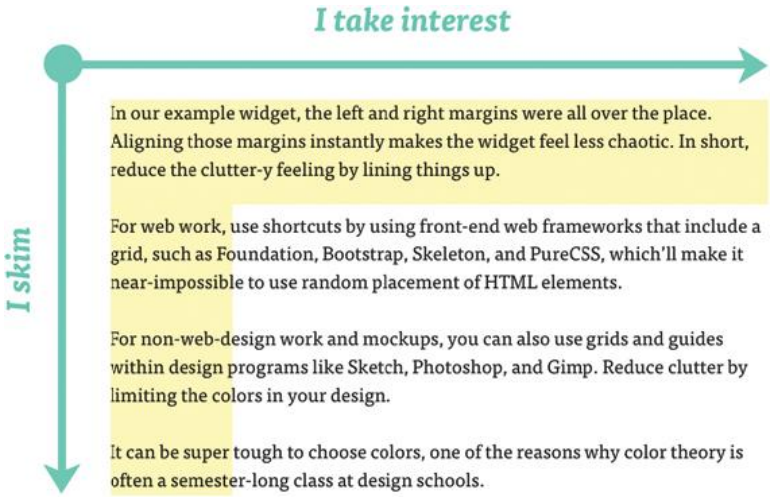


FIGURE 8. The way user read in website (4)

It is stated that users only read 80 to 200 words in the content despite the length of the content. Hence, the content is made to be extracted and simple. Methods that designers and content writers often use are to shorten the content, use bullets, and make text bold for the important information. Therefore, the attention of readers could be controlled by the bold elements. (4)

Images

Last but not least, images are elements which usually present on modern websites. However, whenever using images, if the images were not designed by designers of the team, their licenses are required. It follows that images used in an application can be taken from other sources with copyright or creative designs from teammates. (4)

There are many Internet sources for images that are free to use, such as Pexels and Allthefreestock.com. They provide images of good quality and use for commercial and non-commercial.

However, images and icons are not required in an application. Instead of using a logo, typography with a designed font or a simple font can be used as replacing. In addition, designing with text and adding images later saves time and avoids paying too much attention to an aspect that is not the most important. (4)

Images' size can be various so the step of checking and resizing their size is considered. The large-sized image will require more time to load and even worse on a device which has a slow Internet connection, the loading will not be efficient. Therefore, images should have the maximum size of the demand, for example, a 2000px image is unnecessary for the frame of 1200px wide. (4)

2.1.3 Figma

Figma is the design tool for multiple platforms. It has both a browser design interface and a desktop application. With Figma, an environment for a team to work together on a project is provided along with lots of free features and commonly used design packages. With Figma, all the basic tools are available for creating layouts and elements. Designers can emulate the products by making the interaction between pages with animations. At the moment, Figma can be shared and edited in a team without a limit of three people as previously. (6) (See in figure 9 below)

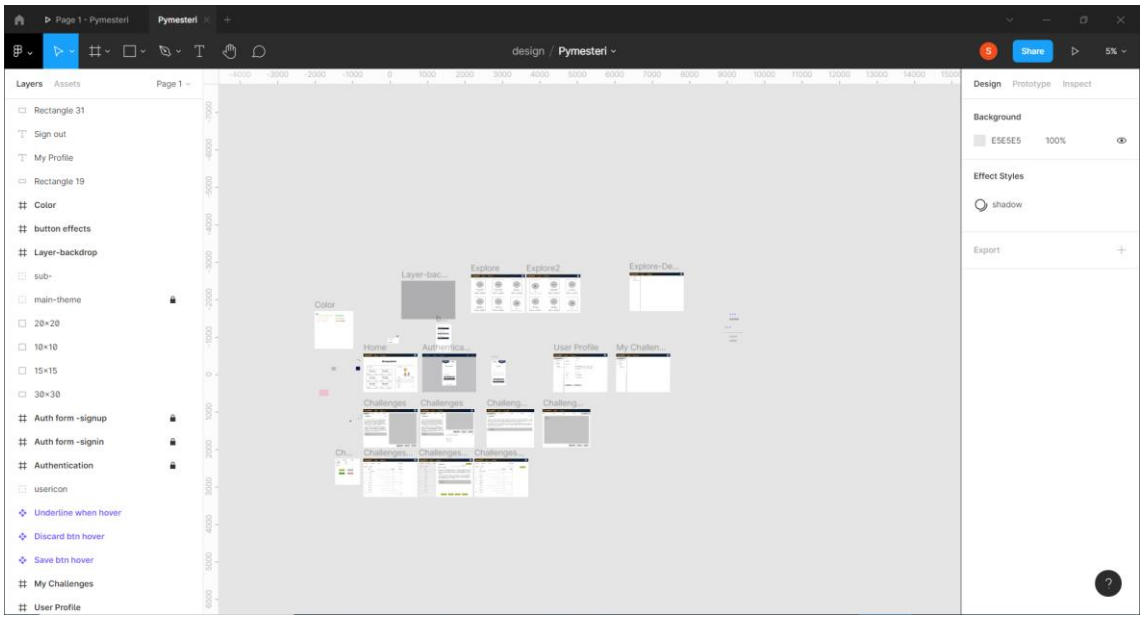


FIGURE 9. The interface of Figma on desktop application

2.2 Technology

2.2.1 React

Introduction

React is a JavaScript library, developed by Facebook, which is used for building client sides – the interface. React designs each element as a component that can be rendered and updated by its state and prop. This plays the role of V (view) in the MVC model, which stands for Model - View - Control. React uses components and scopes for reusable and easy code flow control. React components are used to show how to handle events (by user: e.g. click, scroll), state changes, and display data. It can be used with JavaScript or TypeScript. (7)

By the survey of which framework is loved from StackOverflow 2020, React is on the second on the ranking of The Most Loved Web Framework (Figure 10). It is stated that developers have used and shown that they still want to use it again in future projects. (8)

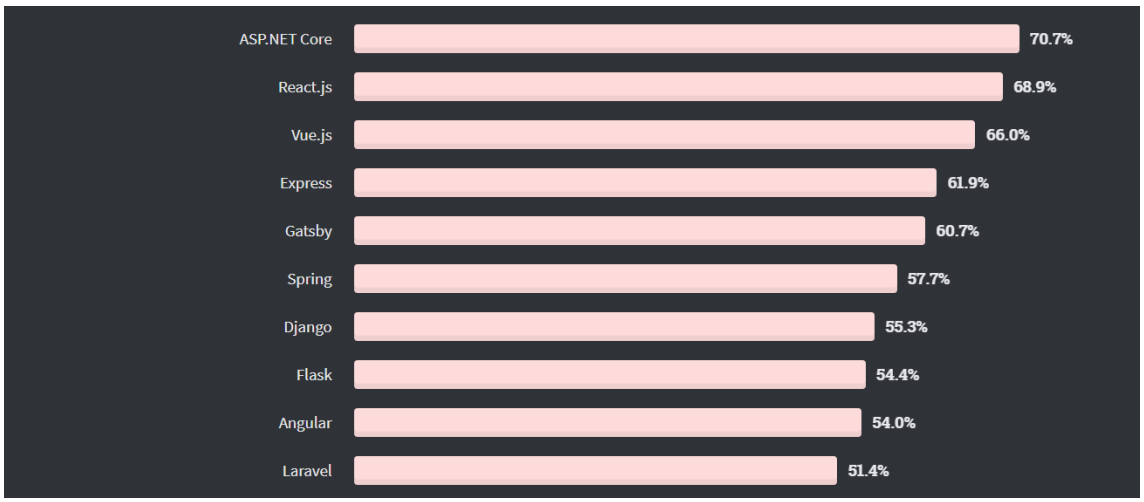


FIGURE 10. Web framework that developers love to use in percent (8)

In Pymestary, React was chosen for frontend tools for three reasons. First of all, it is a famous framework that is used by many developers in community, therefore, it has a large support from community when it goes to bugs and issues. Secondly, React allows us to make reusable components, which helps to save

time for developing and maintenance later on. Lastly, React uses JSX or TSX to write code that allows to use scripts and logic within the HTML syntax, which helps to render the UI faster. (23)

JSX and TSX

React uses JSX and Babel to compile JSX into JavaScript. Same as XML is an extension for HTML, JSX is an extension of XML to JavaScript, which comes to use with ES6. JSX is not simply JavaScript code nor HTML code. It can also have a tag similar to HTML and can contain JavaScript code, such as expression or variable in curly brackets. (7)

In the JSX tag, a string wrapped by quote marks or an expression wrapped by curly brackets can be used as its attribute. As in the figure 11 below, the div tag has the “className” attribute and it receives “styles.title” as the value covered by the curly bracket. The next one is the Link component, and it has a property “to” to receive a string “/challenges” as a value. (7)

```
<div className={styles.title}>Welcome back!</div>

<Link to="/challenges">Challenges</Link>
```

FIGURE 11. Example of an attribute in JSX tags

In React with JavaScript, a JSX extension is used to express the file that contains a JSX structure while with TypeScript, the extension becomes TSX. TypeScript will be introduced later in the following part.

Elements and components

React has two concepts that might make developers confused: elements and components. Elements are the block for building a React application and components are made from elements. By default, when creating a React app with command create-react-app from npm or yarn (the package management), the “index.html” file in the public folder will be a structure for the body tag. (7) (Figure 12)

```
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
  <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
    The build step will place the bundled scripts into the <body> tag.

    To begin the development, run `npm start` or `yarn start`.
    To create a production bundle, use `npm run build` or `yarn build`.
  -->
</body>
```

FIGURE 12. Default body tag, which contains the div with ID "root"

The React application takes this root div and manages the whole application within this node. This node is called a DOM node – Document Object Management. This is the root node as React will manage all components inside this node and render an application. In figure 13, the method ReactDOM.render() takes two parameters. The first is the component and the second is the destination – root node. (7)

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

FIGURE 13. The index.js file

An element is not changeable. So whenever React updates an element, it means that React creates a new element and replaces the old with a new one. This logic is done by methods within a component. (7)

React components are similar to JavaScript functions. It receives inputs as props and returns an element for render in DOM. Props is the term used in React and it is the short form of properties. In React there are two types of components: function components and class components. (Figure 14 and 15)

Function components are simply a JavaScript function which returns an element while class components are created by using ES6 classes. From the view of React, these two components are doing the same work and creating the same element. (7)

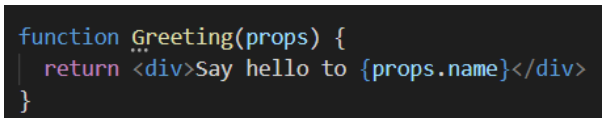


FIGURE 14. Function component

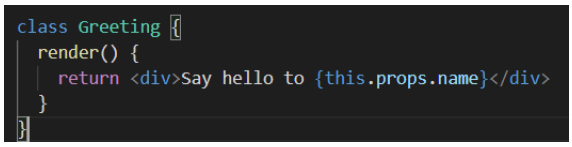


FIGURE 15. Class component

A React component can use other React components in its output as many times as desired. This feature makes React components become reusable. It also means that developers can split large components into smaller ones in order to make the management work easier. (7)

Whenever a prop received a change from a component, the component will render again. For example, in this project, the TabPanel component is created and reused in other components. By providing different props and call multiple time, the panels are created differently and individually. (7) (Figure 16 and 17)

```tsx
export default function TabPanel(props: TabPanelProps): JSX.Element {
  const { children, value, index, styles, ...other } = props;
  return (
    <div
      role="tabpanel"
      hidden={value !== index}
      id={`wrapped-tabpanel-${index}`}
      aria-labelledby={`wrapped-tab-${index}`}
      className={styles.formRoot}
      {...other}
    >
      {value === index && (
        <Box component="div" display="block" style={{ height: '100%' }}>
          {children}
        </Box>
      )}
    </div>
  );
}
```

FIGURE 16. TabPanel component structure

```tsx
class Authentication extends React.Component<ComponentPropsType> {
  render(): JSX.Element {
    const { display, handleClose, tab, setTab, resetError } = this.props;

    const handleChange = (event: React.ChangeEvent<{}>, newValue: string) => { ···
    };

    return (
      <div className={styles.root}>
        <Dialog
          open={display && tab !== null} ···
        >
          <DialogContent className={styles.dialogContent}>
            <StyledTabs ···
            <TabPanel value={tab} index="login" styles={styles}>
              <Login setTab={this.props.setTab} />
            </TabPanel>
            <TabPanel value={tab} index="signup" styles={styles}>
              <Signup />
            </TabPanel>
            <TabPanel value={tab} index="resetpassword" styles={styles}>
              <ResetPassword setTab={this.props.setTab} />
            </TabPanel>
          </DialogContent>
        </Dialog>
      </div>
    );
  }
}
```

FIGURE 17. Authentication component which uses TabPanel components as a child in its output

Properties given to all components are immutable. They can only be read but cannot be changed. For making a change in React, a different concept is used: states. A strict rule declared on their official page when using React: even React is flexible but all React components are pure functions with their corresponding props. (7)

Life cycles in component

There are four main stages for a life of a component: mounting, updating, unmounting, and error handling. Each stage has different methods. The below diagram in figure 18 shows all methods that a component calls in all stages. For getting a full control of the component, developers can use all methods in a class component. However, not all methods will be used and commonly used. The next part contains short descriptions of commonly used methods. (7)

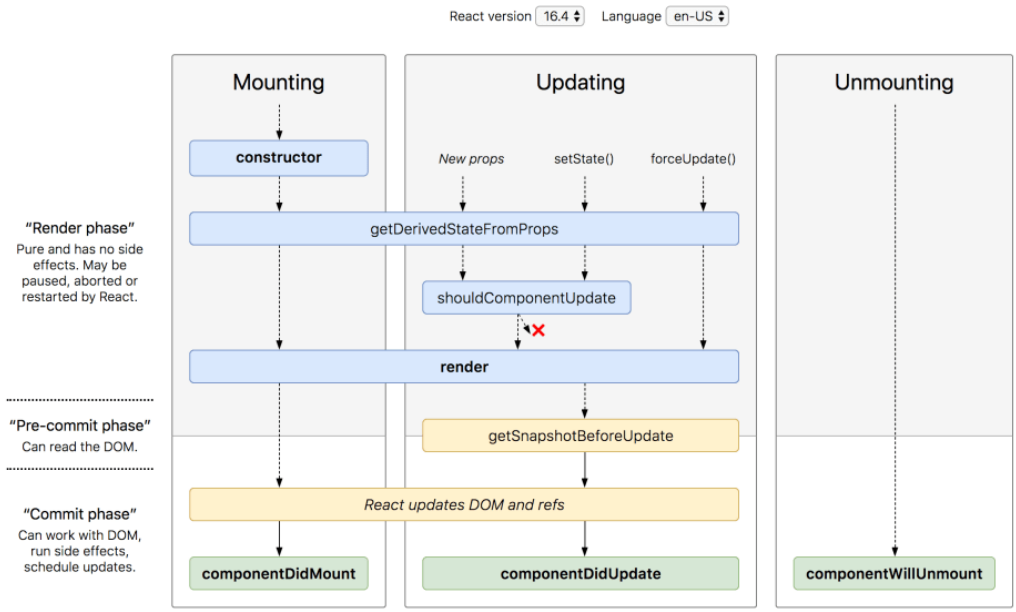


FIGURE 18. React component lifecycle (7)

The first one is the render method. This method is compulsorily used in all components because an element is returned from here. This method must be pure, which means there should not have any method or action to change the state, which causes side effects and force the component to render again. The 'render()' method can either return an element or a null value if there is nothing to be displayed. In short, this method has only one job to do: return JSX that will be displayed. This makes our React application maintainable. (7)

The second most common and compulsory method is constructor(). This method is called when a class component goes to the DOM tree. It is the place where all initialization is called, such as states or binding methods. This is the only place

where a state can be set directly by expression this.state, else the this.setState() is the only way to modify states. The other thing to remember is that if developers want to use props in the constructor(), a call of “super(props)” is required to avoid bugs and other issues. (7) (Figure 19)

```
constructor(props) {
  super(props);
  // Don't call this.setState() here!
  this.state = { counter: 0 };
  this.handleClick = this.handleClick.bind(this);
}
```

FIGURE 19. An example of constructor() (7)

The next method is componentDidMount(). This method is automatically called right after the component goes into the tree DOM (mounting). In this method, it is possible to make any network requests, and changing component local states. A subscription can be made here. Any changes in states here will make React call method render() again. Same as constructor(), this method is invoked only one time. (7)

componentDidUpdate() is the next method on the list of commonly used methods. This method runs after the component is updated, which means that it runs only from the second render. This place can be used to call network requests and set local component’s states, but it must come with conditions, e.g. by comparison, in order not to make an infinity loop. (7)

Before the component goes out of the DOM tree, the componentWillUnmount() will be invoked. This is the method that calls immediately before the component goes out, so it is the best practice for cleaning up the component here. Any changing of the local state of component making from this method will be useless as no re-render will occur after this method. (7)

The five methods above are commonly used. React still has other methods that developers rarely use. By default, there is no need to make any changes if necessary.

However, in functional components, hooks are used to perform features of React creating classes. There are most commonly used built-in hooks to manage states: useState and to perform side effects: useEffect. useState hook is equivalent to setState method and useEffect is similar to componentDidMount and componentDidUpdate methods. Hooks are only used in functional components and are called for definition on the top level of the component. (7)

The useState hook takes an argument of the initialized state and gives a pair of values: the value of the current state and the function to update the state in square brackets. useState can be called many times as necessary, likely to define many states in this.state in the concept of class components. (7) (Figure 20)

```
const [code, setCodeEditor] = React.useState(''); // input code

// State of code editor
const [canTestCode, setCanTestCode] = React.useState(false);
const [editedCode, setEditedCode] = React.useState(false);
```

FIGURE 20. useState hook usage in a functional component in Pymestari

Likewise, useEffect hook is able to use multiple times to perform different effects with unrelated logic within a component. useEffect is stated to perform side effects, such as fetching data, making subscription and changing a DOM node, in replace for componentDidMount, componentDidUpdate, and componentWillUnmount in the class components concept. (7) In figure 21, there is an example of how useEffect is used in Pymestari.

```
React.useEffect(() => {
  dispatch(getModulesListByRoadTypeAction('challenges'));
}, []);
```

FIGURE 21. useEffect hook usage in a functional component in Pymestari

Updating a component

In React, there is a different way to make the component re-render by using the local states within a component. A state can be set to update from the life cycle methods of the component. This is the value that can be defined by users. Same as props, “this.state” should be seen as unchangeable value. The correct way to

change a component's state is to call "this.setState()" or the function to update state declared in useState hook. (7)

However, components can be forced to re-render by using a method called forceUpdate(). This way will make the parent component (which is using the method forceUpdate()) update, but children components are run as it is without any effects from the parent component. (7)

Styled components with CSS modules and library

CSS modules

In this part, there is a short description of how to style components in React and which way was used in Pymestari development. There are many ways to implement styles for React components but in Pymestari, the approach for styling is using CSS module and inline CSS styling.

The approach using CSS module was chosen as styles are written with CSS way and belong to a separate file. Styling elements with CSS modules is said to create a familiar feeling in comparison to styling plain JS and pure HTML files. (9)

Inline CSS styling was chosen as a recommend from React official page. However, for small changes and only changes in one component, implementing inline CSS styles will be faster in development. (7)

The naming convention for CSS module file must include the "module.css" with a customized name (9). The four figures below (figure 22-25) are describing the structure of the file which includes CSS modules, and this concept is used in a component. By importing the module as "styles" instance, in a component, developers can put it under the "className" property with a suffix of a class name, which is defined in CSS module file. (Figure 25)

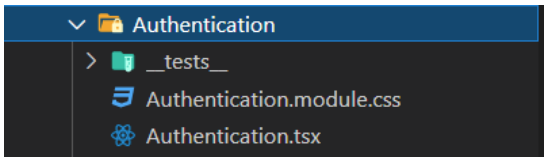


FIGURE 22. The structure of a file with module CSS and TSX file

```
import styles from './Authentication.module.css';
```

FIGURE 23. Importing module CSS to a component in .tsx file

```
<div className={styles.errorText}>
  {this.props.errorLogin} <br /> please try again later!
</div>
```

FIGURE 24. Using styles from CSS module as a class in div tag

```
.errorText {
  color: var(--red);
  font-weight: 400;
  text-align: center;
  text-transform: capitalize;
}
```

FIGURE 25. An example of styling in the CSS module

Styling with Material-UI

In Pymestari, Material UI package was used for UI components. This library is stated as the most popular UI framework for React. This framework contains many styled components with the consistence style. It also allows developers to modify styles for customized styles for components. Using this library helps to reduce the timing in writing the components from scratch. (20)

Type checking and TypeScript

When using JavaScript for development, React developers can use the “prop-types” package to check the type of components and properties. Checking types is stated as a good practice for reducing the bugs and issues when implementing a large project. “prop-types” provides multiple validators for multiple types: e.g. array, Boolean, string, number, node, element. (7)

React also supports using other extensions of JavaScript such as TypeScript and Flow for more accurate and strict types. (7)

In Pymestari, for strict and consistent types using along within the project, TypeScript was used for achieving this goal. TypeScript is an extension of JavaScript, a subset of JavaScript. Figure 26 below shows that TypeScript is one of the most popular loved languages in 2020, stated by StackOverflow developers. (8)

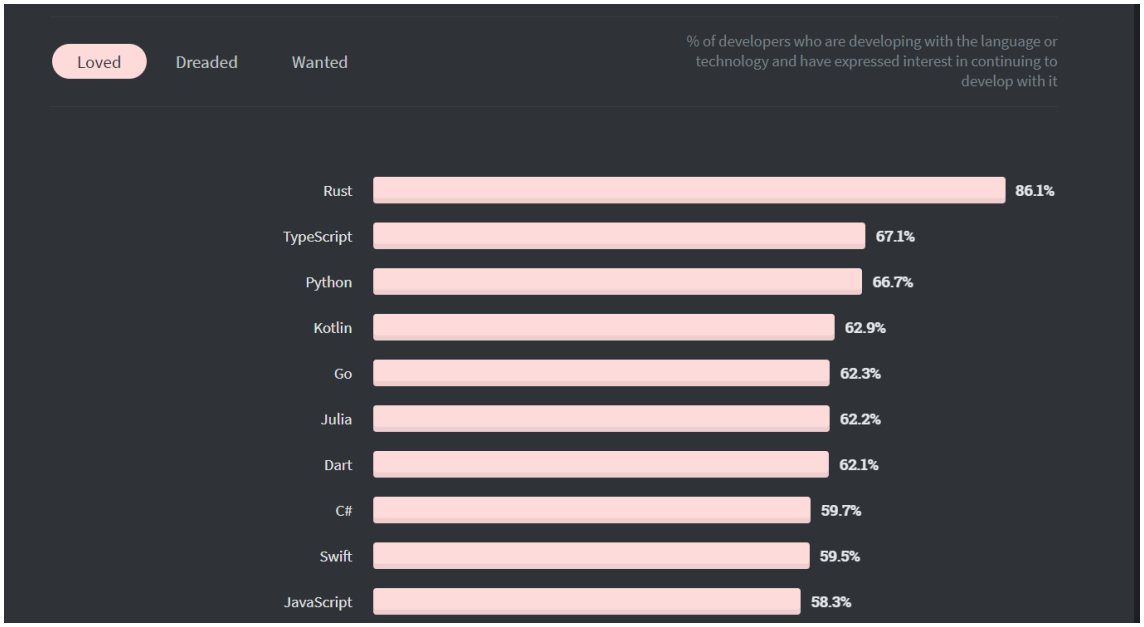
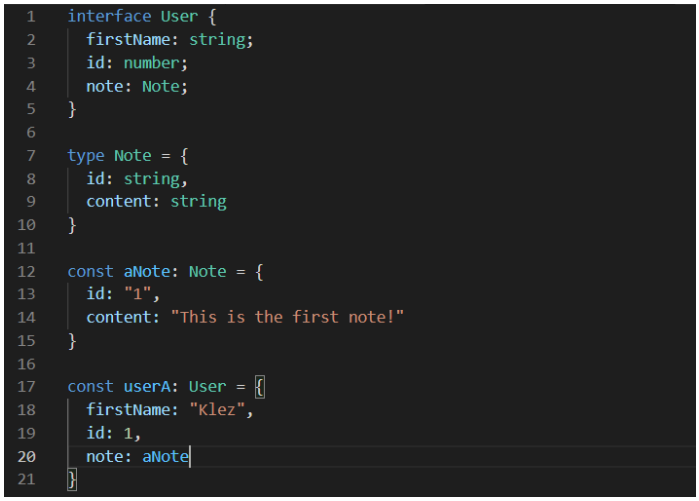


FIGURE 26. Survey of loved languages by developers (in percent) (8)

If in JavaScript, the program is easily run with a lack of type checking, yet, all types are checked strictly before executing in TypeScript. The type checker in TypeScript is made to catch errors caused by using wrong types. Even though errors are checked and caught from the code when pasting from a JavaScript file to a TypeScript file, the code can run as it is in a JavaScript file. The behavior in runtime does not change even the error of type were found. However, after compiling, the result of a TypeScript will be a JavaScript file without any types. (11)

Developers can define a type before declaring any value to a variable. They can define it by making an interface or declaring a type. Then, they can assign a variable based on the type defined. Below is an example of how to define a type

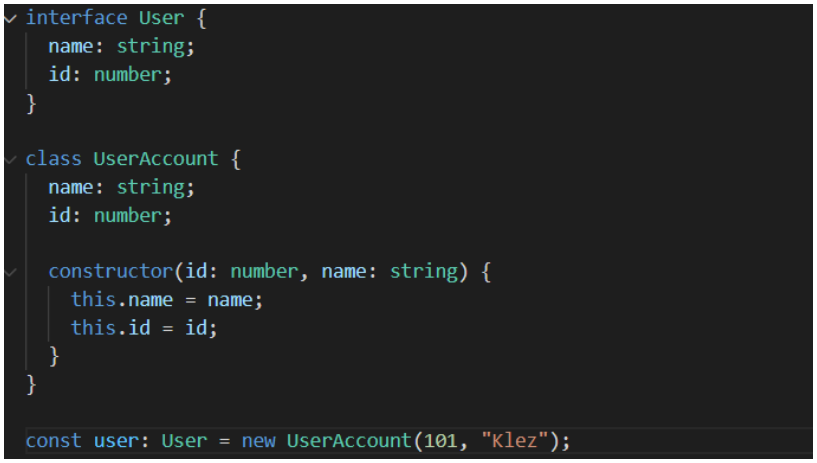
by both using an interface and a type and declare variable corresponding to types. (Figure 27)



```
interface User {
  firstName: string;
  id: number;
  note: Note;
}

type Note = {
  id: string,
  content: string
}

const aNote: Note = {
  id: "1",
  content: "This is the first note!"
}

const userA: User = {
  firstName: "Klez",
  id: 1,
  note: aNote
}
```

FIGURE 27. Define the type and declare a variable.

TypeScript is similar to JavaScript, the interface can be declared by classes. (11) (Figure 28)



```
interface User {
  name: string;
  id: number;
}

class UserAccount {
  name: string;
  id: number;

  constructor(id: number, name: string) {
    this.name = name;
    this.id = id;
  }
}

const user: User = new UserAccount(101, "Klez");
```

FIGURE 28. Declare interface by class.

When declaring a function, the return value type should be given, and it can be given after the parameters. The below code in figure 29 shows how to define the type of return value from a function and an arrow function.

```
function getUser(): User {
  return {name: "Joe", id: 102}
}

const getSubUser = (): User => {
  return {name: "Joe", id: 102}
}
```

FIGURE 29. Define type of return value from functions

If the function has any parameters, the parameters can declare a type inside annotate parameters as in figure 30. (11)

```
function addUser(user: User) {
  // ...
}
```

FIGURE 30. Declare type of parameter in a function

Union can show the list of values the variable can take or the list of types the variable can be. Figure 31 shows a type of variable declared by union. The constant “FName” can take one of these strings to be its value: “Bob”, “Alice”, “Joe”, and “Jenny”. If it takes “Elena”, the error in figure 32 will occur. Same as “EText” in figure 33, it can receive a number or string as its value but it will issue an error if another value type is given.

```
1
2   // Union by values
3   type IValues = "Bob" | "Alice" | "Joe" | "Jenny";
4
5   // Union types
6   type IText = string | number;
7
8   const FName:IValues = "Bob";
9
10  const EText:IText = 12;
11
12  const EText2: IText = "Ring Ring";
13
```

FIGURE 31. Declare type by union

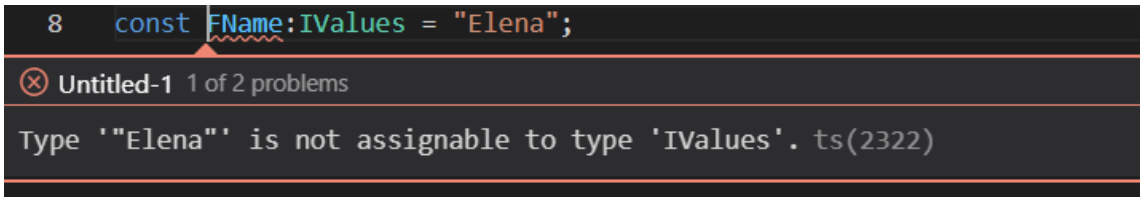


FIGURE 32. Error when "FName" takes value other than "Bob", "Alice", "Joe" and "Jenny"

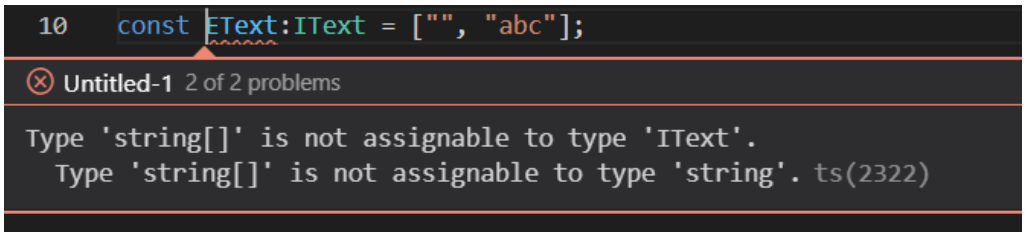


FIGURE 33. Error when "EText" takes an array of strings as value

Equally, generic brings values to declare types. (11) For example, an array without generic can store anything in an array, in contrast, an array with a particular type must store the value of its type. (Figure 34)

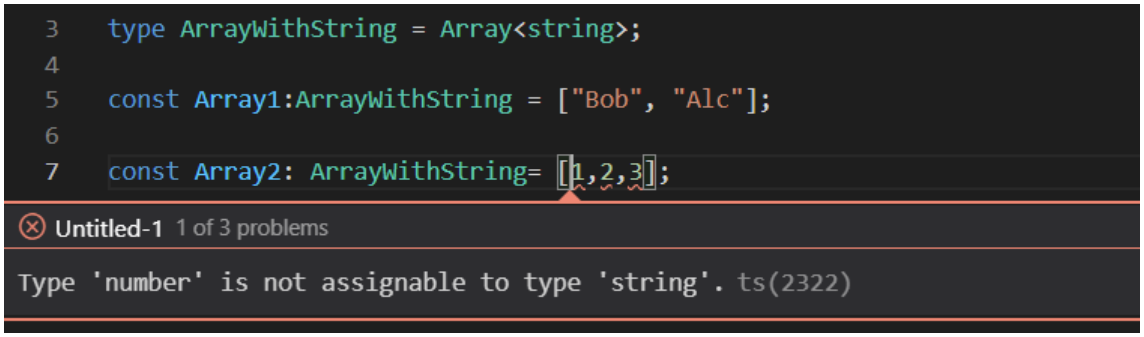


FIGURE 34. Declare type in a generic way

2.2.2 Libraries and packages

Redux

Redux is a state management library that can be used in any UI layer. This library is light and supports multiple addons for developers' need of a predictable, acting steadily, debugging easily and centralized application. "action" and "reducer" terms are used in order to manage and update the state, then return a new state. States in Redux can be globally used, which means that it can be accessed from anywhere with an application by its patterns. (12)

The figure 35 below shows an example about the advantage of using Redux: the central store in an application using Redux can be requested to get access from any components, in contrast, without the help of Redux, the value must be passed to every connected component to be accessible when it changes.

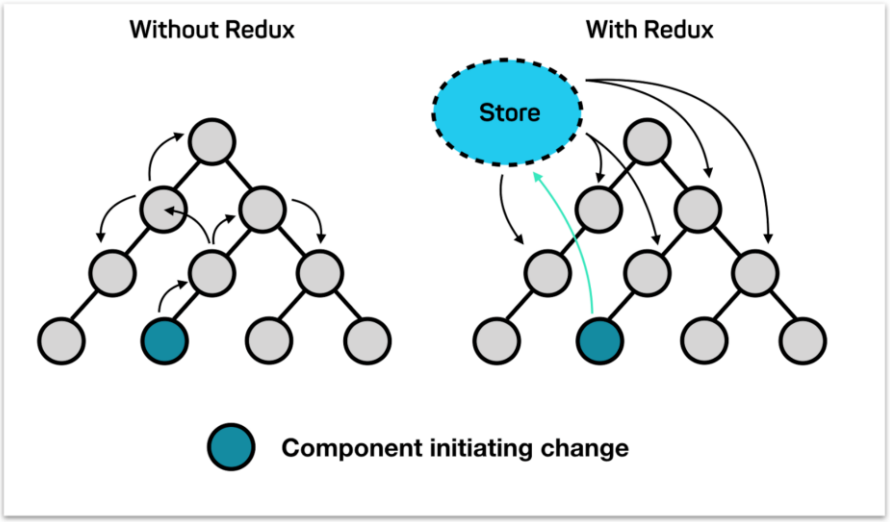


FIGURE 35. Comparison between applications with and without Redux (12)

Not all applications will need Redux but it is recommended to use for building a medium and large size codebase application, or an application that has states used frequently or needs to be updated with complex logic. For starting using Redux in development, developers can choose to use “redux-toolkit” from the official release or use the “react-redux” package in a React application. (12)

In Pymestari, Redux is used as a dependency and is connected by “react-redux”. This library supports methods and hooks for using Redux in both class components and functional components. (13)

There are four concepts about Redux that need to be understood before using it. (12)

The first one is “state management”. When it comes to the state definition, it means to change the state and update the view to display to user. In figure 36, the component’s workflow is shown as at a specific time, the state has a value

and the UI shows the value of states; then, users make an action and the state is updated based on the action; finally, the UI will be rendered again with a new state. (12)

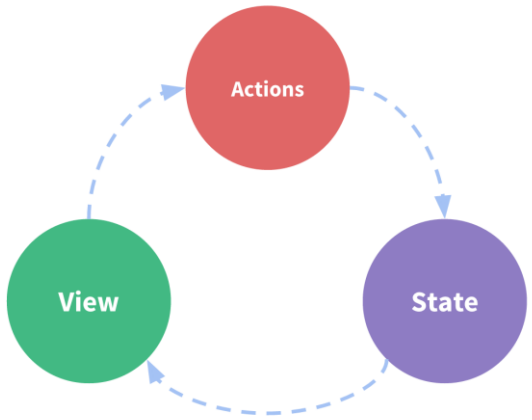


FIGURE 36. Workflow with state (12)

Therefore, a problem of the approach to access and modify states in multiple components system is popped out. The Redux developer team recommended a solution for this problem: put all shared states from all components into a centralized store, which is outside of the components tree, therefore, the store can be accessed from any component without a complicated transition. Accordingly, the “view” and “state” parts are split separately and more organized. It follows that the job of structuring and preserving is more efficient. (12)

The second concept is “immutability”, which literally means that states are unchangeable. Redux updates all state in an immutable way: as state in Redux is basically a JavaScript object, when updating a state, Redux makes a new object which is a clone from the old state, then it modifies properties from the new object. (12)

The third concept is “terminology”. This concept includes six terms: “actions”, “action creators”, “reducers”, “store”, “dispatch” and “selectors”. (12)

First, “actions” are plain JavaScript objects which must have a property “type”. “type” is the value that describes what occurs in the application, and usually a

string value. In addition, developers may need to add additional information in a property called “payload”. “payload” can be any type of value, string, array, or object, depending on demand. (Figure 37)

```
const addingAction = {
  type: "ADDING",
  payload: {data: "example data"}
}
```

FIGURE 37. An example of action object (12)

The “action creators” term indicates all functions which create action objects. These functions are often used and created in a separate file, instead of writing plain “actions” objects. (12)

After an action is called, “reducers” is the place that listens to actions. “reducers” is a function that receives two values, the current state, and a receiving action, and returns a new state. “reducers” is a listener that listens to the action and handles event based on the received action. Figure 38 shows the logic of “reducers” is displayed by following steps: checking the type of action if this reducer works with it, if it does, “reducers” makes a copy of the old state, updates a new value, and returns the new state. On the contrary, it will return the existing state without making any changes. (12)

```
const initialState = { value: 0 }

function counterReducer(state = initialState, action) {
  // Check to see if the reducer cares about this action
  if (action.type === 'counter/increment') {
    // If so, make a copy of `state`
    return {
      ...state,
      // and update the copy with the new value
      value: state.value + 1
    }
  }
  // otherwise return the existing state unchanged
  return state
}
```

FIGURE 38. An example of “reducers” (12)

“reducers” must follow its rule. A new state is only proceeded based on two arguments, “state” and “action”, that it receives. The existing state cannot be changed but is able to make a new change immutably. Any synchronous requests or commands are prohibited as the cause of side effects is followed. (12)

A follow-up term is “store”, which is an object that contains all available Redux states. It was also mentioned as a centralized store. The store is created by using “reducers” and it returns the current value of the state. A state can be extracted by using the “getState()” method. (12)

In Redux, an action is triggered by dispatching an action creator. “dispatch” is set in the “store.dispatch()” method, which receives an action as its parameter. When an event is triggered, the store will call “reducers” to receive the “action” object and the reducer will do its job. (12)

The last term is “selectors”. “selectors” are functions to indicate the exact piece of value from a store. In a larger code base project, there will be many reducers with different names combined in “store”. Therefore, using “selectors” avoids the same logic and data used repeatedly. (12) (Figure 39)

```
const selectCounterValue = state => state.value

const currentValue = selectCounterValue(store.getState())
console.log(currentValue)
// 2
```

FIGURE 39. An example of “selectors” (12)

“Dataflow in Redux application” is the last concept to be mentioned. In figure 40, there are three main parts including “store”, “UI” and “event handler”. The “store” is the place where all states and reducers are processed. Users can interact with the application in the “UI” part, by giving an input – an event trigger and the UI is the place where the state is rendered to users. The “event handler” is where actions are dispatched and sent to reducers. (12)

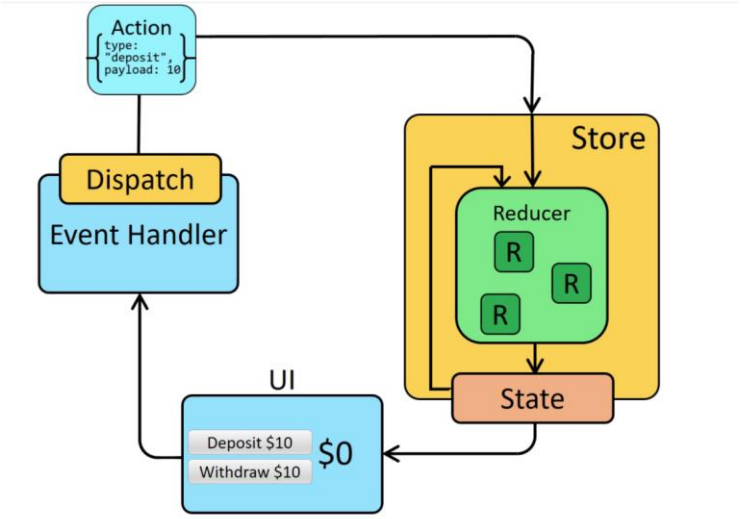


FIGURE 40. Redux application dataflow. (12)

The workflow for the initial run starts from the “store”, where it setups a root reducer and calls the root reducer once to return the initial state. Then the “UI” renders the current state from the reducer and subscribes to the store for the later listener. When a user first interacts with the UI and makes an event, such as a click or an input, action is made and called by a dispatch method. The dispatch sends that action to the store, and it calls reducers again to process a new state based on the action. Finally, the store sends the updating notification to all components in the UI. Therefore, the UI can use the new state to display to users by forcing an update. (12)

In Pymestari, Redux played the role of central store where contains all state from the authentication, user, form and editor. As there are lots of state which can be divided into groups, therefore, state are grouped into five groups and one comes from the form redux package, which is controlling the form value. All these groups are combined by reducers in the root file and the store is created based on this root reducer. (Figure 41 and 42)

```
export const rootReducer = combineReducers({
  auth: authReducer,
  editor: editorReducer,
  form: FormReducer,
  general: generalReducer,
  learningmodule: learningmoduleReducer,
  user: userReducer
});

export type RootState = ReturnType<typeof rootReducer>;
```

FIGURE 41. Combine reducers

```
export default createStore(rootReducer, enhancer);
```

FIGURE 42. Create a store with reducer

Material-UI

As mentioned earlier, the Material-UI library is the most popular UI library compatible with React. This library is designed to implement Google Material Design into components and supports multiple common and easy-to-adapt components. (20)

There are two ways to import Material-UI components into developing components: first is declaring one by one and the second is putting all components into one (Figure 41). The first approach will get exact addressed components; therefore, it will save time and the size of bundles in runtime. On the other hand, the second method will first import the whole package, then extract the asked components.

```
import Dialog from '@material-ui/core/Dialog';
import DialogContent from '@material-ui/core/DialogContent';

import { Dialog, DialogContent } from '@material-ui/core';
```

FIGURE 43. Example of import Material-UI components statement one by one (up) and all in one from the core package(down)

Material UI provides numerous components that are ready to use, ranging from the toolbar, containers, to text fields and buttons. These components are isolated

and can be used separately or combined by demand. The styling of each component is only imported when it is called to display, so therefore, the whole package is not imported all in one time, which creates the size of bundles. (20)

Yet, components can also be customized by overriding or customizing the whole theme. The first approach is overriding styles on the components. In the document, they stated that developers can override directly on the components, making a styled component reusable or customizing by theme. The second approach is to create a theme and wrap the application or part of the application under the API ThemeProvider. (20)

In Pymestari, the approach overriding components was used: wrapped the component with the hook withStyle and modified based on theme object provided by Material UI to return a new styled component. (Figure 44)

```
const Input = withStyles((theme: Theme) =>
  createStyles({
    root: {
      'label + &': {
        marginTop: theme.spacing(3)
      }
    },
    input: {
      borderRadius: 4,
      position: 'relative',
      backgroundColor: theme.palette.background.paper,
      border: '1px solid #ced4da',
      fontSize: 16,
      padding: '10px 26px 10px 12px',
      transition: theme.transitions.create(['border-color', 'box-shadow']),

      '&:focus': {
        borderRadius: 4,
        borderColor: '#80bdff',
        boxShadow: '0 0 0 0.2rem rgba(0,123,255,.25)'
      }
    }
  })
)(InputBase);
```

FIGURE 44. Overriding component

Monaco Editor React

While choosing the editor package for Pymestari, “monaco-editor” was the chosen approach as it provides features powered from Visual Studio Code IDE. (24) Visual Studio Code is the IDE that provides many features that help developers

easier to program such as broad features code editing and navigation. It is a powerful source but also a lightweight IDE. (26)

“monaco-editor” is the package developed by the Microsoft team, which supports creating the code editor for web applications on the desktop. It contains many features similar to Visual Studio Code IDE with the code editor supports for all languages the IDE has. However, some of them have rich features and the others only provide the basic color highlight. Python is supported in “monaco-editor” but only for highlighting and syntax. (24)

“@monaco-editor/react” is the package that is written based on “monaco-editor” released from Microsoft and integrated with React. This package was chosen as it makes the process of importing the IDE to components simple and fast with only one line of declaring and modification. It provides a range of selection for customizing the editor similar to the IDE, besides from language of the code inside the editor, such as theme, height, line number, etc. (25)

The component of the editor from this package also has the properties of React: value, handle changing, handle mounting. Therefore, the value of user input can be controlled by React state or by refs. (25)

2.2.3 Code integration

ESLint and Prettier

Clean code makes programmers feel at ease and helps their teammates easier to understand code and save time. The code quality has a strong connection with understandability (14). There are tools, which help increase the effect in coding and making the code clean, which can be installed from package management. In Pymestari, ESLint and Prettier are used.

ESLint has been the most popular JavaScript linter. This linter's main goal is to find and fix problems while coding. ESLint has been supported by most of the code editors. ESLint provides a tool that can be used to fix the code format and

syntax automatically. This helps programmers save time in finding errors and have a better performance at work. ESLint provides a library of rules for the JavaScript syntax. (16)

Prettier is the code formatting tool. The tool is aimed at automatically formatting code by simply saving it. It supports a wide range of languages and frameworks: e.g. JavaScript, HTML, CSS, JSON, Angular, Markdown. Prettier takes all the code and process in a consistent way, which is provided and modified by developers in the config file. It removes the current code formatting and prints from scratch with the style consistently in all files with the same extension. (15)

Prettier formats all code automatically, and it is the only formatter until now which has this support. Pretty code makes code understood easily in the least time. It also saves time when writing code, developers do not need to worry if the code is written in a neat, clean, and pretty way. For a team member as a reader, code in uniform is saving much time and pain to get used to. (15)

The difference between Prettier and linters, for example, ESLint, is that Linter has two categories of rules, formatting and code-quality, and Prettier only focuses on formatting. Prettier has a standard way and developers can modify styling to fit with the team, while in ESLint, these rules should be specific. In the code quality aspect, Prettier does not have any effect on it. (16)

In general, Prettier takes care of code formatting and ESLint catching bugs. Both Prettier and ESLint can be added to the project as a good combination practice to make the code look cleaner. (16)

Plop

While writing components, the initial format for all components is similar with these parts: imports, declaring properties, and exports. For saving time and avoiding copying old components to a new one and re-write, plop is a solution. Plop provides the tool to automatically generate new components based on the template with a simple command. As stated on the Plop main document page:

“consistency makes simple”, plop generates code files or text files consistently based on the created generators. It is simple to create and easy to learn as it uses “handlebar” for templates and “inquier” for prompting. (17)

```
module.exports = function (plop) {
    // controller generator
    plop.setGenerator('controller', {
        description: 'application controller logic',
        prompts: [{
            type: 'input',
            name: 'name',
            message: 'controller name please'
        }],
        actions: [{
            type: 'add',
            path: 'src/{{name}}.js',
            templateFile: 'plop-templates/controller.hbs'
        }]
    });
};
```

FIGURE 45. An example of plop file (17)

Plop structures with a “setGenerator()” method and takes two required parameters: generator’s name and its setup, which includes a description and a list of actions. In the above example in figure 45, there is also a field called “prompts”, which is the question for users to modify the generator while in use. This is not a required field. (17)

Figure 46 shows a .hbs file, which is a template for a functional React component in the Pymestari application.

```
import React from 'react';
import PropTypes from 'prop-types';
import styles from './styles.module.css';

const {{ pascalCase name }}: React.FC = (props: any) => {
  return (
    <div className={styles.root}></div>
  );
};

{{ pascalCase name }}.defaultProps = {};

{{ pascalCase name }}.propTypes = {};

export default {{ pascalCase name }};
```

FIGURE 46. An example of .hbs template for a React component in Pymestari

Husky

To make ESLint and Prettier work better in a project, developers not only run the script for ESLint and Prettier, but they can also use the hook for making automation. (19)

Nowadays, most team projects are using Git for controlling versions. Git provides hooks that trigger automatically when working with it, for example, “pre-push” and “pre-commit”. “pre-push” is the hook that is called just before pushing to Git by a command “git push”, and similarly, “pre-commit” is called before “git-commit”. Developers can manually set up these hooks but they can use the Husky tool as well. Husky is the tool that makes Git hooks easier and friendlier. (19)

For using Husky, developers first need to install it into the project by using package management and putting it under “package.json”. (Figure 47) Then, they put any hooks under “husky” in “package.json” and when the hooks are called, scripts are called as well. (Figure 48) (18)

```
// package.json
{
  "scripts": {
    "prepare": "husky install"
  }
}
```

FIGURE 47. Put the script to initialize husky to project. (18)

```
{
  "scripts": {
    "prettier-format": "prettier --config .prettierrc 'src/**/*.ts' --write",
    "lint": "eslint . --ext .ts",
    ...
  },
  "husky": {
    "hooks": {
      "pre-commit": "npm run prettier-format && npm run lint"
    }
  }
}
```

FIGURE 48. Example of hooks setup for husky in package.json (19)

3 IMPLEMENTATION

3.1 Designing the interface with Figma

3.1.1 Study cases – from other online study platforms

Before deep-diving into design the user interface for the Pymestari application, Leetcode and Edabit are the pages taken into study for styling and structuring. Both of them are challenging playground for programmers and they support for learning different programming languages.

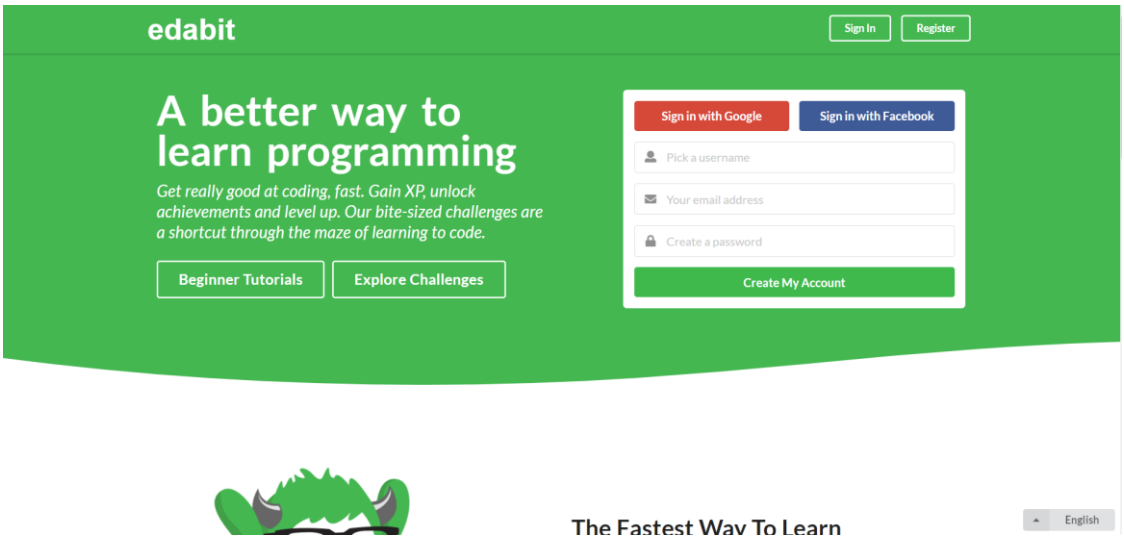


FIGURE 49. Edabit.com homepage

In figure 49, it can be easily seen that green is the theme color of Edabit. As mentioned in the theory part, green shows success and moving forward. Therefore, this page obviously makes developers feel fresh and motivated to finish challenges.

The page is simple with the authentication form on the right side. On the left side, there are two buttons that help new users know what to do at first sight. When users navigate to the tutorial page in figure 50, there are two modules to be

chosen placed in the middle of the page. Thus, users can easily choose which module they want to study.

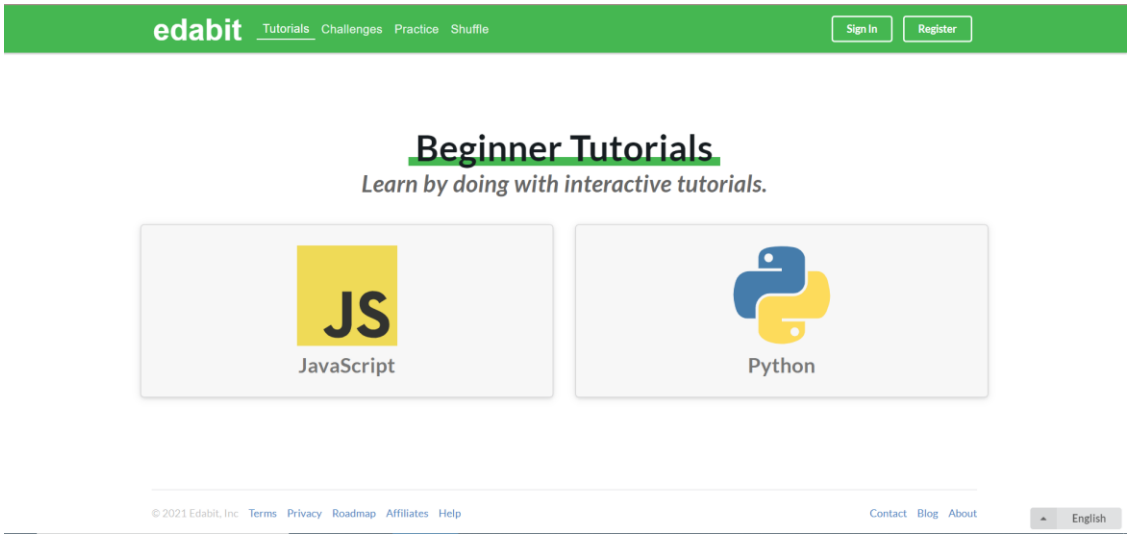


FIGURE 50. Edabit tutorials page

There are many pages: Tutorials, Challenges, Practice, and Shuffle. The current page is indicated by a different effect: A white underline.

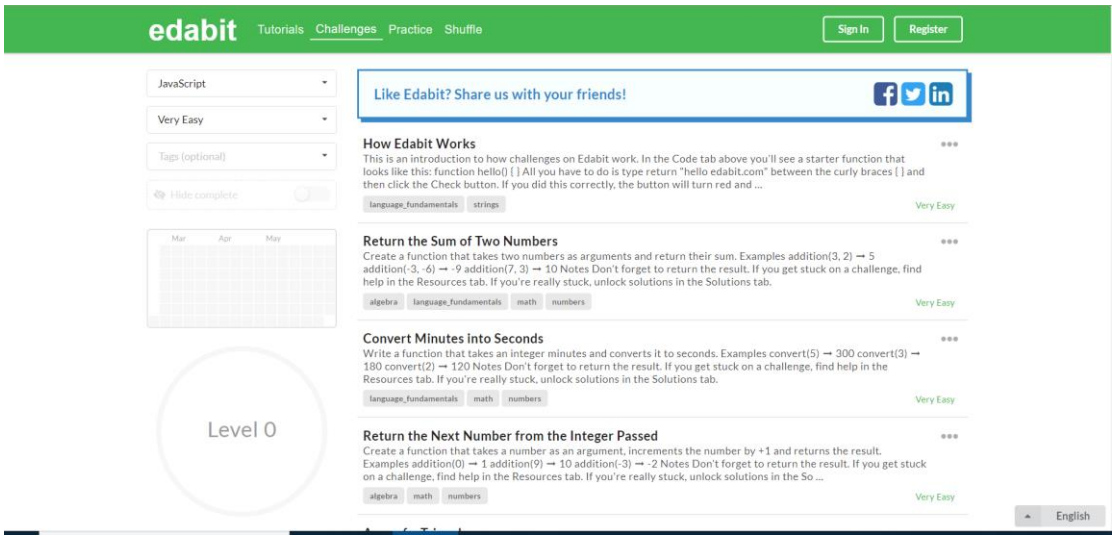


FIGURE 51. Edabit challenges page

On the challenge page, there is the selection of a filter on the left column, and on the right column, there is the list of challenges. The white space outside two columns makes the page straight and lets users focus on the middle part. However, in my opinion, on the right column, the text of description for each

challenge is too much and tags for each challenge are not visible at first sight. (Figure 51)

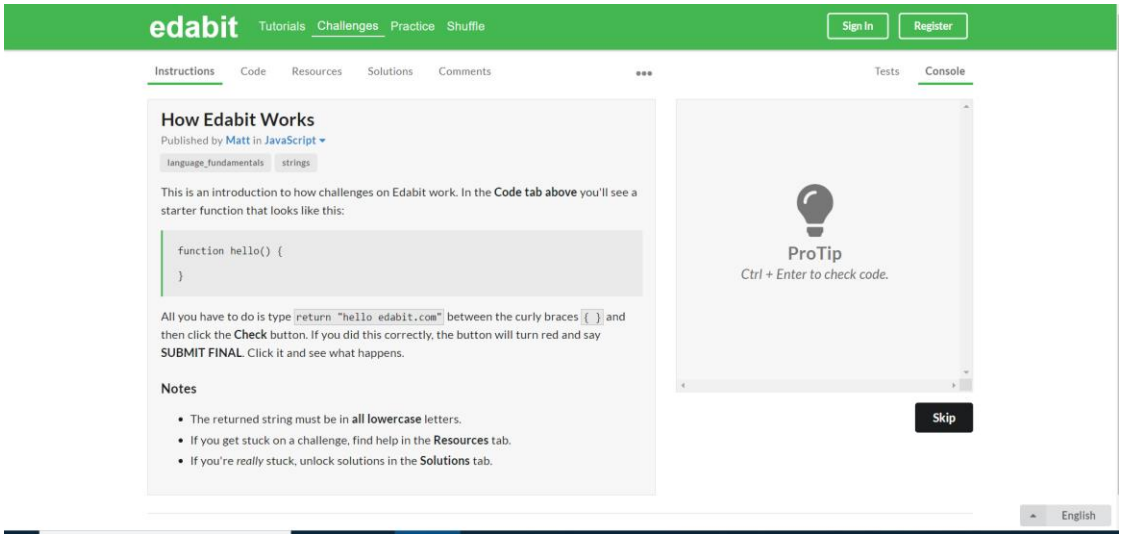


FIGURE 52. Edabit Challenge editor page.

Opening a challenge will navigate to the page in figure 52. On this page, Edabit makes two columns and preserves two blocks of white space outside, the same concept as the pages.

Edabit makes the webpage simple and uses only two colors shade: green for the title, navigation bar, and grey for the content. It also keeps the principle of design: using only two font families, Helvetica for the title in the navigation bar and other using Lato. Both of them are sans-serif font.

In opposed to Edabit, Leetcode has its own front page with more details and features of its page. (Figure 53)

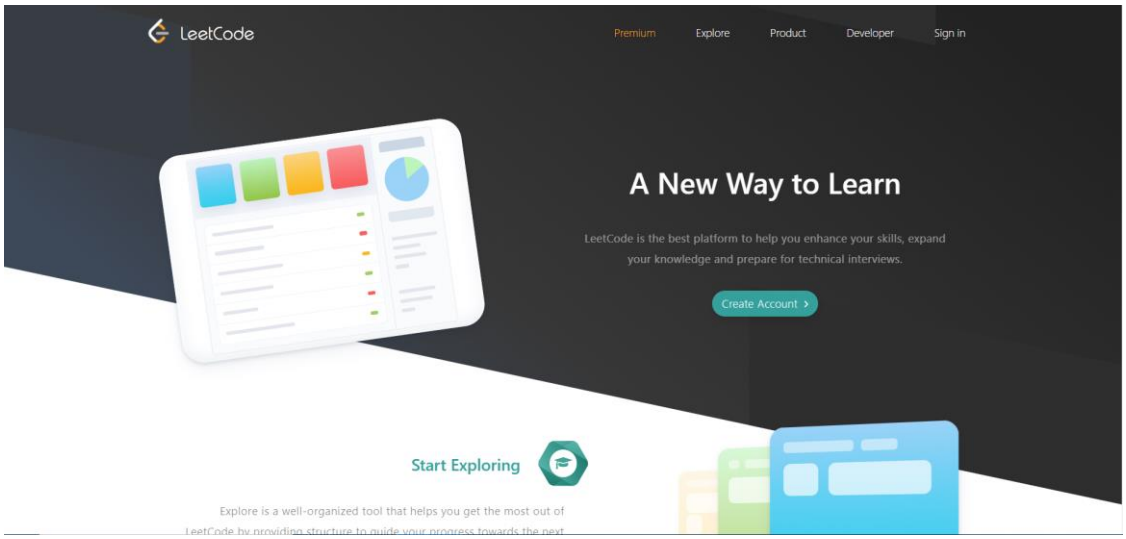


FIGURE 53. Leetcode.com home page

It has a separate home page compared to Edabit and its homepage is a landing page: showing features on one page. All Leetcode action is starting from the Explore page.

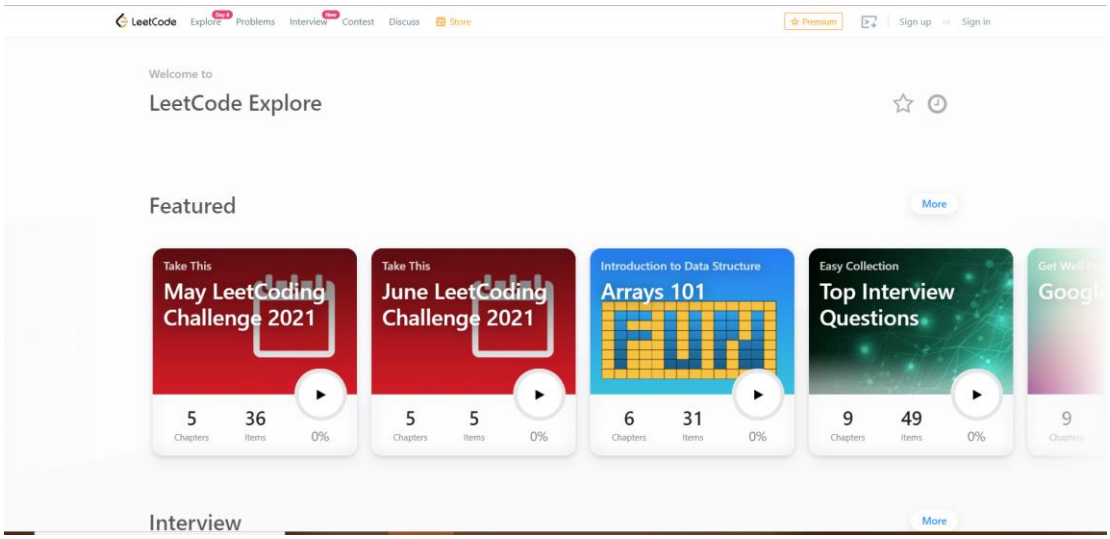


FIGURE 54. Leetcode.com explore page

If in Edabit, the navigation bar is made in green and large, more visible to users, then in Leetcode, the navigation bar is narrowed down and the space is designed only enough for text. In this way, developers do not pay attention to the navigation bar but to the content. On the Explore page, Leetcode lists their most focused features. Leetcode divides challenges into interview questions, challenges by month, and provides learning modules by categories. (Figure 54)

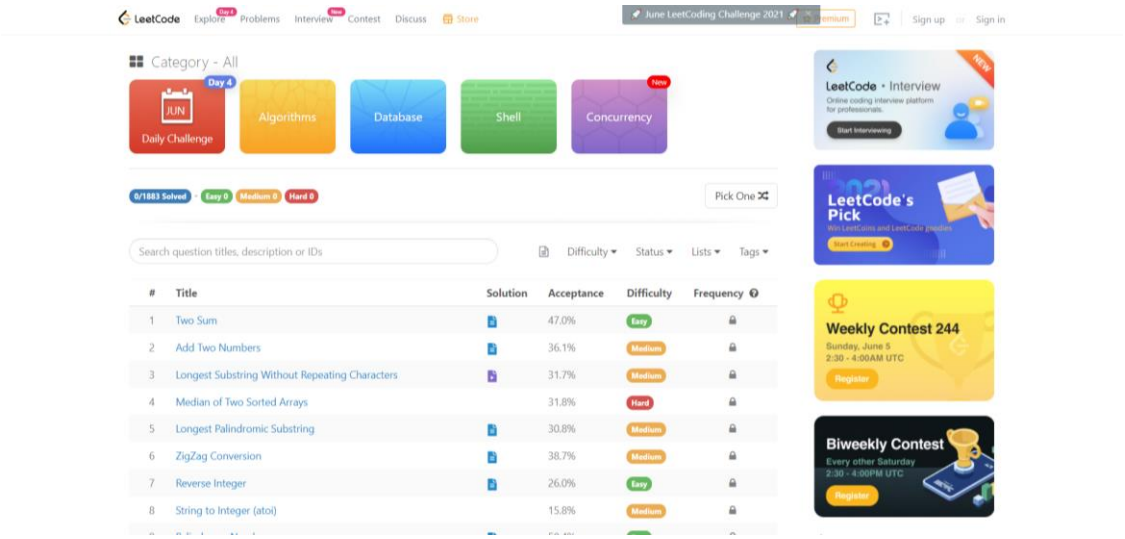


FIGURE 55 Leetcode problems page

On the problems page, there are two columns in which the left one takes most of the page. In the right column, there are lists of contests organized by Leetcode ongoing. On the left side, there is the filtering section for filter problems. Each problem line shows the abstract detail which can be viewed easily. Similar to Edabit, Leetcode also has a separate page for solving issues. (Figure 55)

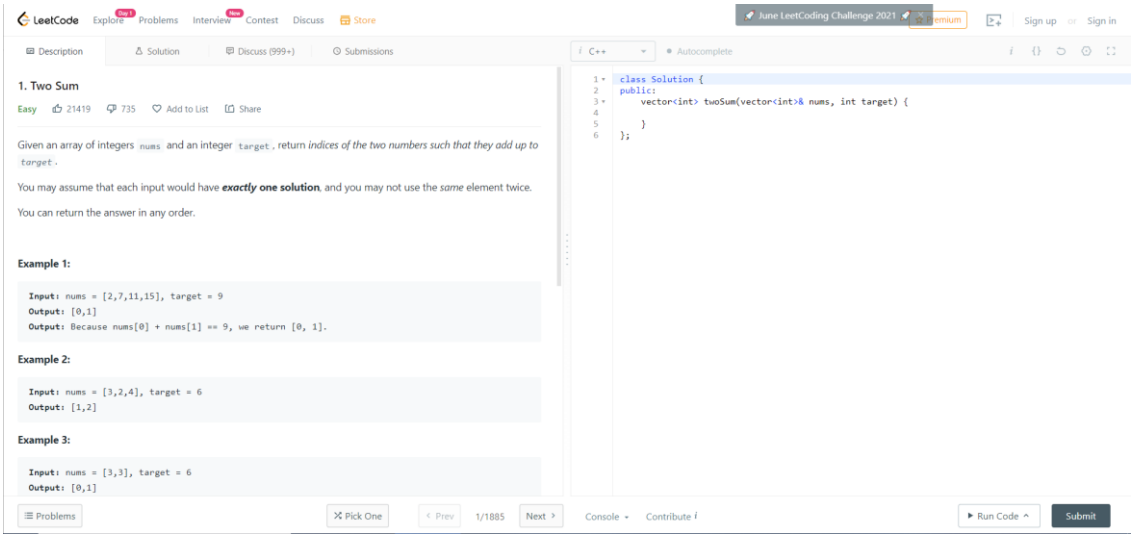


FIGURE 56. Leetcode problem detail page

In figure 56, all the content is stretched to be full width. The container is divided into two equal sections: the left one contains all information from problem

description to solution and discussion, and the right one contains only the code editor with an action. It can be seen easily that the whole application takes only white background without any main colors and only one font used for all text is Segoe UI.

In conclusion, both Leetcode and Edabit page implements well in design principle. Both of them are making a website with a simple theme and have done the job of focusing on the main content well.

3.1.2 Pymestari's design and workflow explanation

In this design, the contrast theme was chosen for high visibility: toolbar with dark colors and the page in white. There were three chosen colors, dark orange, dark blue, and white. As in the design theory above, the theme should contain only two to four colors for showing the consistency and ability to follow.

The logo was designed by using a handwriting type, Gochi Hand, to emphasize and make it obvious to the page content. And the choice of sans-serif font in Pymestari was Poppins, which is suitable for showing text on web applications and it is also a popular font for designing a website. This font was chosen for the whole content, including the title and normal text in paragraphs. To make the title more obvious, the thickness and the size of the text were added. In this application, only two font families were used to maintain consistency.

The toolbar was designed with a gradient range from dark orange to dark blue while the whole page content was in white. The orange was chosen to make the high energized and warm, while blue was coming with the meaning of consistency. Both in the dark shade were meant to raise the energy consistently and professionally.

As mentioned earlier in the description of this application, the Pymestari platform has two main parts: challenges path and explore path. These two parts are separately on the navigation bar and can be accessed through buttons.

For the main page, top problems taken in a month and a programmer who has worked the most and gained the best achievement are shown as for courage and interests to other users. (Figure 57)

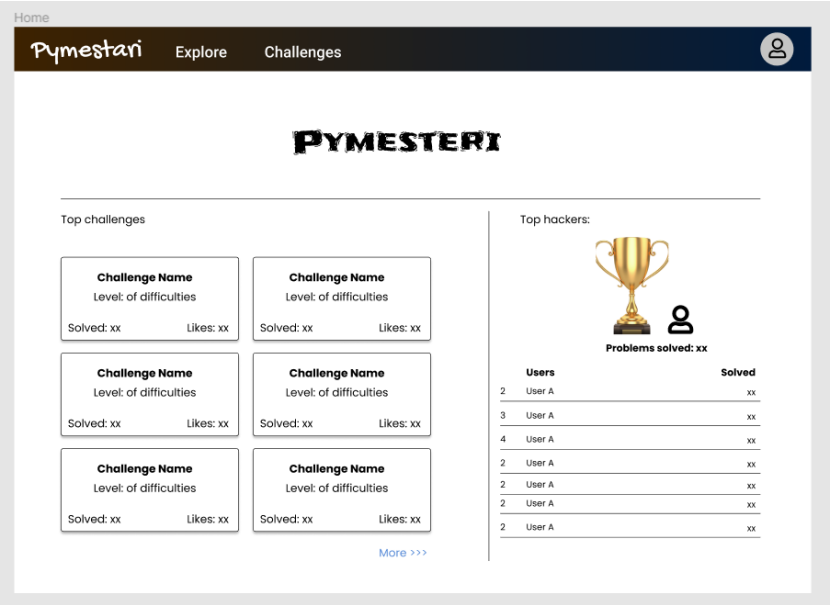


FIGURE 57. Home page interface

The main page has the layout of two parts, the banner in the holder of Pymestari text, below the navigation bar, and the container contains two panels: the left panel with top challenges and the right one with programmers who made the best achievement in the month. These two parts are sharing the width of screen with ratio of 7:5. The top challenges are shown as cards, with a summary of the challenge: title, level of difficulty, the number of people who have solved the challenge, and the number of people who have liked as their favorite. On this page, only the top six famous challenges are shown. On the other side of the content container is a list of honor participants of the month, which are designed in order to help courage other hackers and programmers on the platform. (Figure 57)

If users desire to get closer to the platform, they will be asked to get authenticated by login into an account or signing up for a new account. The popup below will be shown in figure 58. A decision to use a popup instead of a new blank page

was because of less routing, and popups are familiar with users as most websites nowadays use popups for authentication.

The dark blue color, which was used in the navigation bar, was used for the indicator of the popup, to address users to know which form they are opening. Buttons were designed with the same gradient to the navigation bar for the active state. In the disabled state, when the form cannot be submitted, buttons were in the lighter shade to the indicators' color.

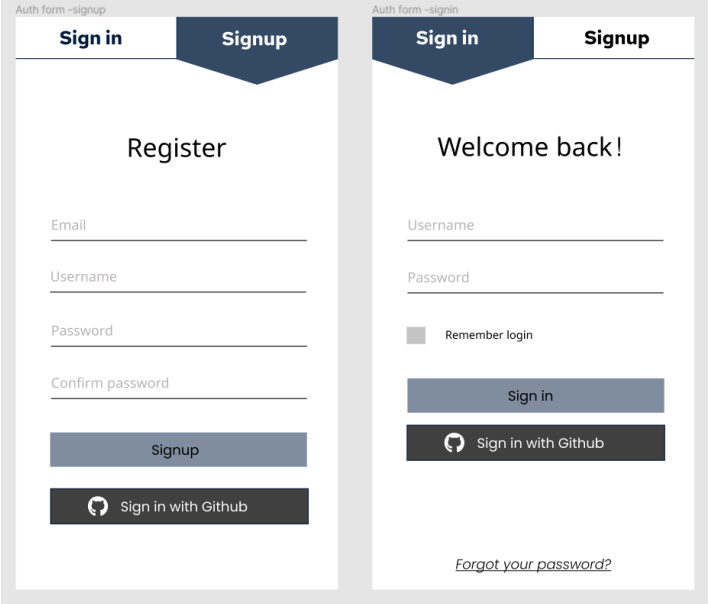


FIGURE 58. Sign in and signup form as popup

The next page will show up the list of challenges of the platform, which contains the list of challenges, with a search bar and filters, and it opens the preview abstract description before users decide to take the challenge without moving to a new page. (Figure 59)

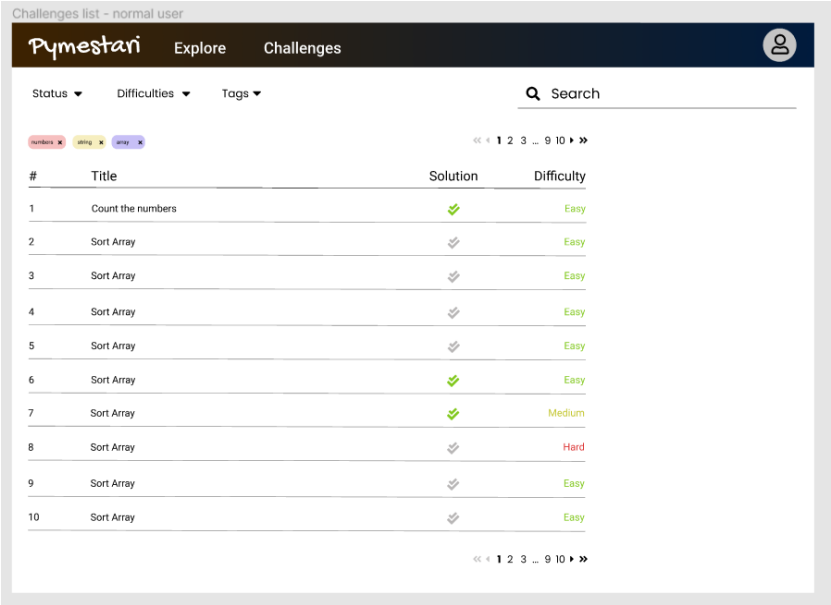


FIGURE 59. Challenges list page

The solution ticks have two states: available in green and not available in grey. And similar to the difficulty levels have three states: red for hard, yellow for medium and green for easy. (Figure 60)

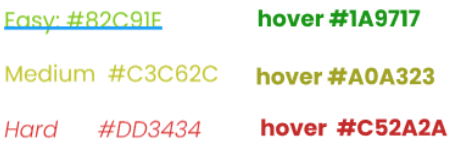


FIGURE 60. Colors table for difficulty text

On the page in figure 59, users can go around the page to find the problem that they want to or are able to solve. The list of challenges is shown 10 challenges on each page and can be navigated without changing the route. There are filters for the challenges, categorized by a tag of type of challenge, for example, e.g. String, Array, Boolean, or by the level of problems' complexity. Each category is designed with a random color tag component.

Programmers can preview the challenge by clicking or tapping into the line of challenge, the drawer of the challenge's detail will appear on the right side on the wide screen or as a dialog (full page) on the narrow screen. On this panel, the challenge is described in more detail, and for taking an action, users can go

directly to solve a challenge (Open), save to the favorite list (Save), and open the solution of author and community (Solution). (Figure 61). After deciding to take a challenge, users will move to the challenge page in figure 62.

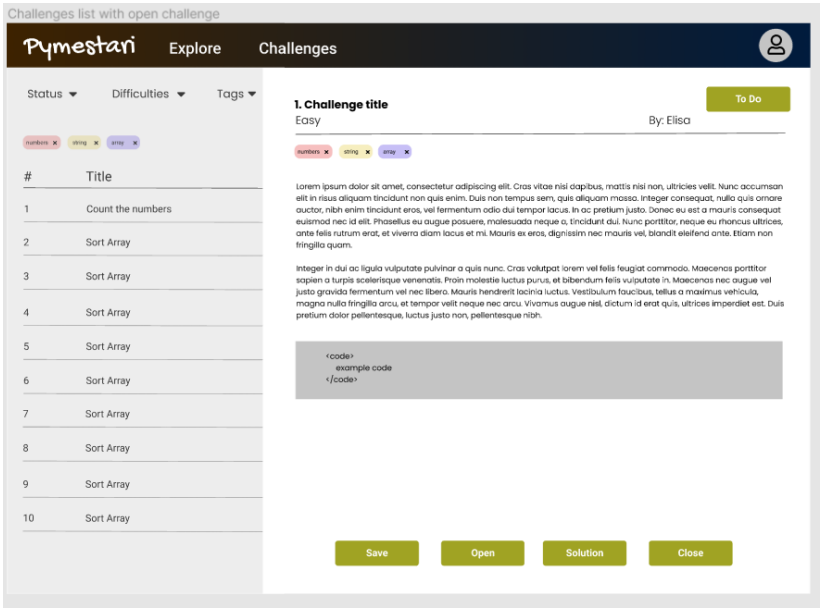


FIGURE 61. Challenges list page with an opened preview panel

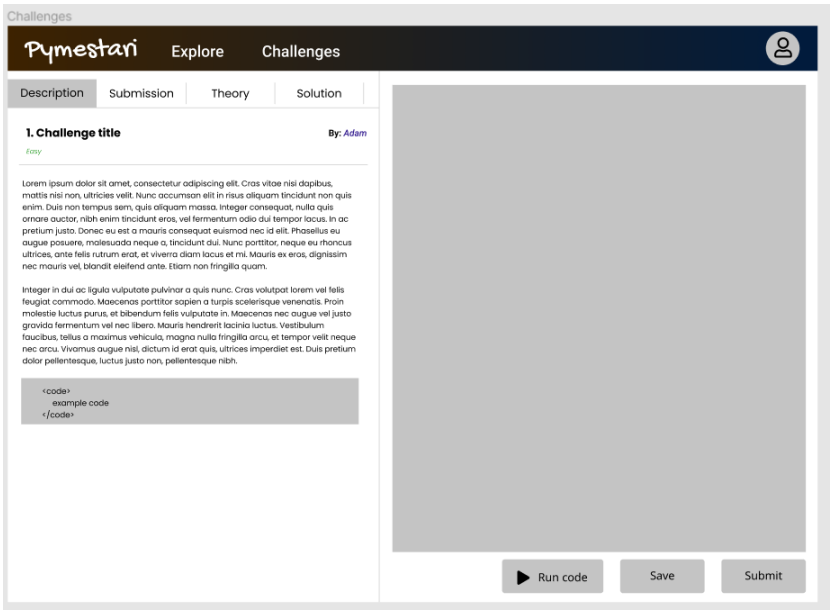


FIGURE 62. Challenge page with responsive

On the left side, there are 4 tabs: Description (description and the question of challenge), Submission (the submission history of user), Theory (a theory that

can be used to solve the current open challenge), and Solution (a solution which is provided by author or community). On the right side is the code playground which contains the code editor and action buttons: Run code (Run the code inside the editor with the test of the challenge), Save (save the current editor for future editing, without submitting the answer) and Submit (to submit the code inside the editor). The result of the code inside the editor will be run with the test provided with the challenge by the author and shown below the editor, with the collapsible panel. (Figure 63)

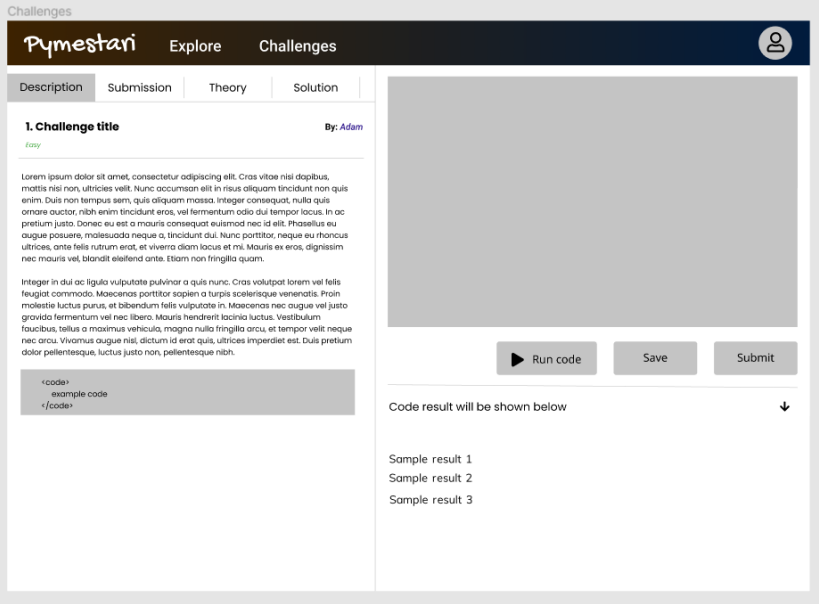


FIGURE 63. Code playground with opened test code result panel

This page is also responsible for smaller device screens. As this platform is the code challenging playground so the device users would use no smaller than 1200px, which is the size of a tablet. The breakpoint of 1440px width was used in order not to have a too narrow code editor. Which results the editor would have the smallest width at 1200px. (Figure 64 and 65)

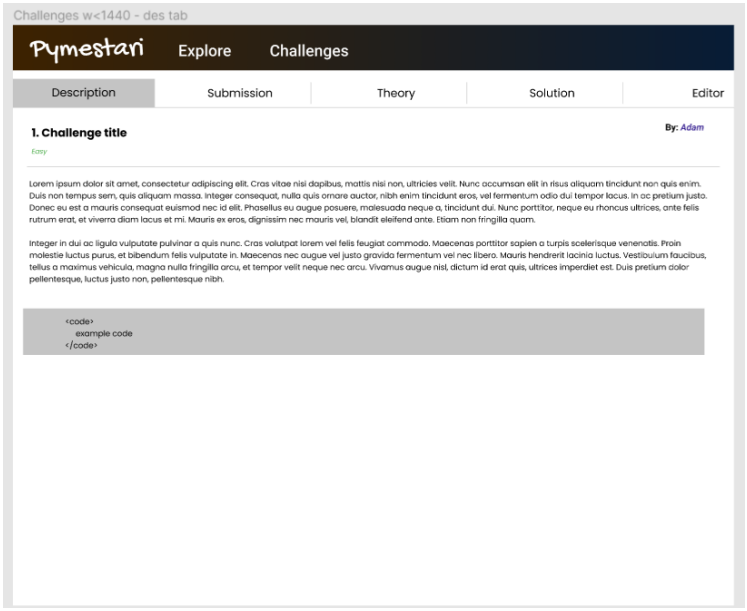


FIGURE 64. Challenge page opened with the device has a width smaller than 1440px – Description tab

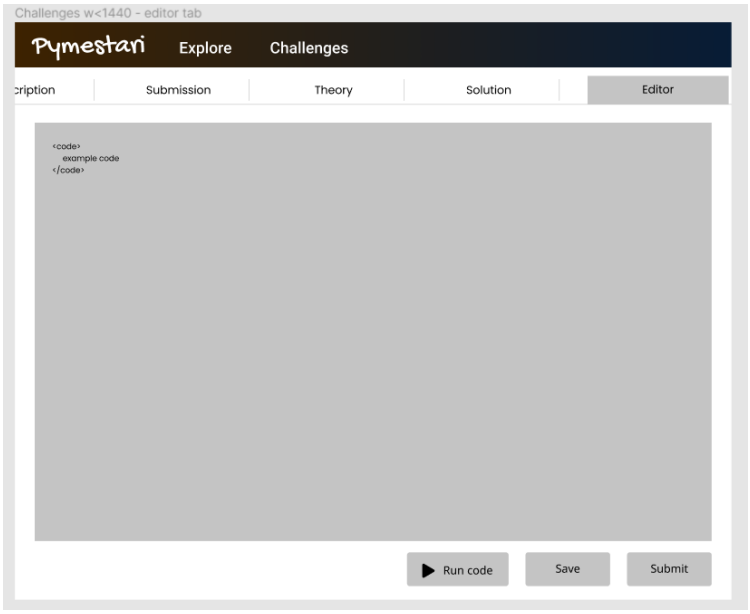


FIGURE 65. Challenge page opened with the device has a width smaller than 1440px – Editor tab

The next part is about the study path. It is inside the route of exploring “/Explore”. It contains all the study modules and will be shown as cards for each module. Each card has a simple detail about the module: Module name, level of difficulty, number of lessons, and number of exercises it includes. (Figure 66)

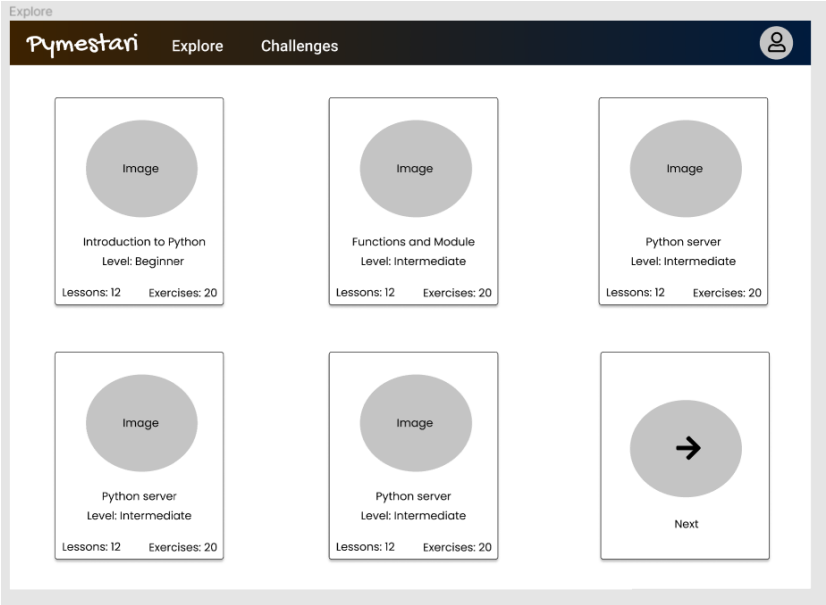


FIGURE 66. Study path (Explore)

3.2 Structuring and setting up the environment for Frontend coding

3.2.1 Structuring folders and files

To get started developing project, the installation of required packages were made through npm. In this project, the script for installing React added "--template typescript" for having the structure of React application in TypeScript. By default, the React application is opening on port :3000 by starting with the script "npm start". There are also other scripts available for development progress. (Figure 67)

```
"start": "react-scripts start",
"build": "react-scripts build",
"test": "react-scripts test",
"eject": "react-scripts eject",
```

FIGURE 67. Default scripts by create-react-app.

As using create-react-app, the React application is created with the default setup by Babel, webpack installed. Therefore, developers do not need to set up webpack or Babel themselves. Any further installation of webpack, ESLint, Babel will cause the conflict while run time. (7)

Figure 68 below shows the current structure of the Pymestari application on the client side.

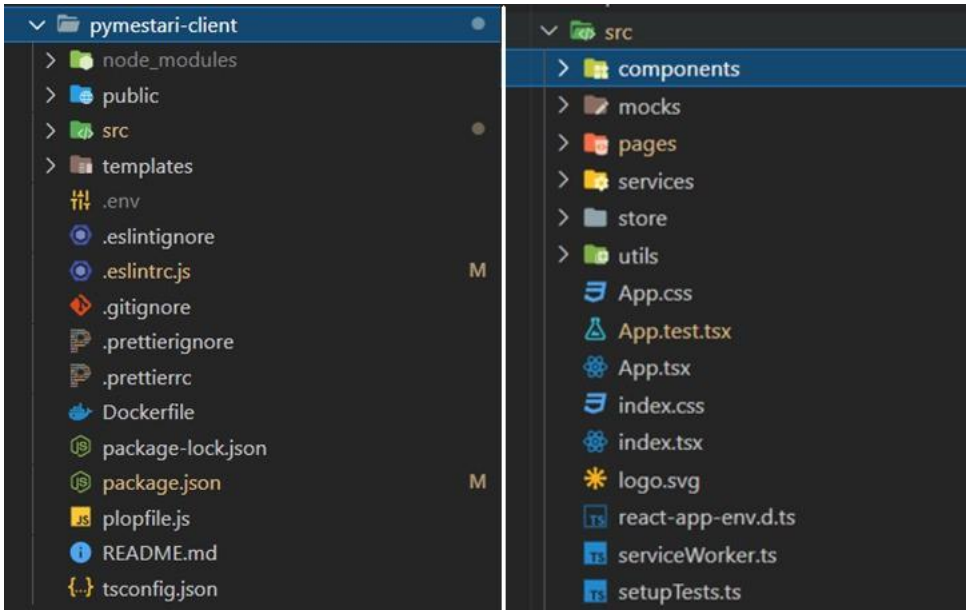


FIGURE 68. Files structuring in frontend (pymestari_client) folder (left) and src folder (right).

The left side of figure 68 shows three main folders created by create-react-app: node_modules, public, and src. “node_modules” is the folder where all packages’ dependencies are stored. “public” stores the main HTML document files and all built scripts compiled from the webpack. When the script “npm start” is called, the HTML file in this folder is taken to use. And “src” is the main folder storing all React components, hooks, routes, store, and tests. “.prettierrc” and “.eslintrc.js” are the config files for Prettier and ESLint respectively. Any ignore checking patterns would be added to files “.prettierignore” and “.eslintignore”.

Inside “src” folders, all components are sorted into components and pages. “components” is the folder which includes all small components, which are used in other components, whereas, “pages” folder stores larger components which are displayed as pages in routing. (Figure 68, right)

3.2.2 Setup environment

“package.json” is the file where all project dependencies, scripts, and other information about the project are stored. Here is the package file for the client part. As mentioned above, ESLint and Prettier were used for setting up clean code as these tools have wide support for React and TypeScript. Before figuring the config file (.eslintrc.js and .prettierrc), all required packages were installed from npm and listed under the devDependencies property in “package.json” file.

FIGURE 69. Required packages for using ESLint and Prettier

After installing all above packages in figure 69, the ESLint config file was set up by the command “eslint --init”. This command provided prompts for developers to modify the lint file by habits and favorites. The next thing was to modify the prettier config file by creating a file named “.prettierrc” (begins with dot) and add rules in the JSON format in figure 70.

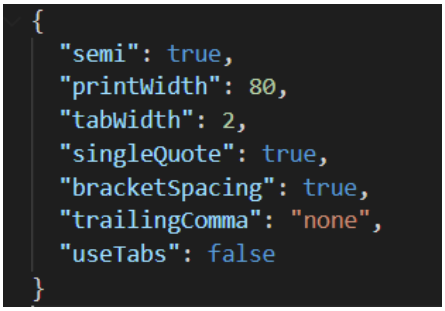


FIGURE 70. Define code format with Prettier

By default, both ESLint and Prettier are used for formatting code, the plugin eslint-plugin-prettier was installed to certain that ESLint will use Prettier as its plugin. (16) In the previous section, Prettier is used for formatting and ESLint is used

when debugging has been stated. There were also additional rules that were added into debugging, as by using plugins, there were set rules which were not suitable in this project. (Figure 71)

The extension plugins are added to the extends array, which are the libraries will be used along with ESLint: ESLint for React, TypeScript and the combination package of Prettier and ESLint. Each of them includes different rules other than the default in formatting code.

```
module.exports = {
  extends: [
    'plugin:@typescript-eslint/recommended',
    'plugin:react/recommended',
    'plugin:prettier/recommended',
    'prettier/@typescript-eslint'
  ],
  parser: '@typescript-eslint/parser', // Specifies the ESLint parser
  parserOptions: {
    ecmaVersion: 2018,
    sourceType: 'module',
    ecmaFeatures: {
      jsx: true,
      arrowFunctions: true
    }
  },
  plugins: ['react', '@typescript-eslint', 'prettier'],
  settings: {
    react: {···
    },
    'import/resolver': {···
    }
  },
  ignorePatterns: [···
  ],
  rules: {
    '@typescript-eslint/no-empty-function': 'warn',
    '@typescript-eslint/no-empty-interface': 'warn',
    'prettier/prettier': [
      'error',
      {
        endOfLine: 'auto'
      }
    ],
    '@typescript-eslint/no-explicit-any': 'off'
  }
};
```

FIGURE 71. Config file for ESLint

For using ESLint and Prettier, “lint”, “lint-fix”, “pretty-code” and “format-all” added scripts into the “package.json” file. (Figure 72)

```
"lint": "eslint **/*.{js,ts,tsx} --quiet",
"lint-fix": "eslint **/*.{js,ts,tsx} --quiet --fix",
"pretty-code": "git status -s -u | xargs npx prettier --write",
"format-all": "prettier --write src/**/*.ts{,x}",
```

FIGURE 72. Scripts for using ESLint and Prettier in project

The first script “lint” is to check all code if there are any issues found by ESLint. If there are any issues, they will be shown in the bash. “lint-fix” is the command that will fix all issues above. With the choice “--quiet”, all warnings and errors will be fixed silently without any line of notification. “pretty-code” will first take the current Git status and fix the styling of all files that are modified. And “format-all” will fix styling with Prettier of all files with the extension “.tsx” or “.ts” (the TypeScript React and TypeScript files’ extension respectively).

The next task was setting the automatically fixed styling when pressing to save code. As the Visual Studio Code editor was used for development, the below settings were added into “settings.json” under the “.vscode” folder at the root level. These settings help to automatically format code when pressing save the works changed. (Figure 73)

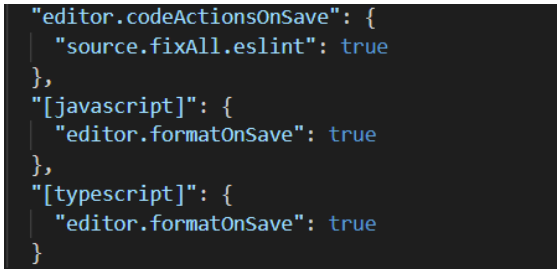


FIGURE 73. Settings for autosave with Prettier in VSCode editor

To save time and not run the script manually, the settings of fix styling with Prettier on saving are recommended to set. Therefore, for certainly making these scripts work, the setup for Git hooks “pre-commit” and “pre-push” were made to run above scripts under “husky” property settings in “package.json”. (Figure 74)

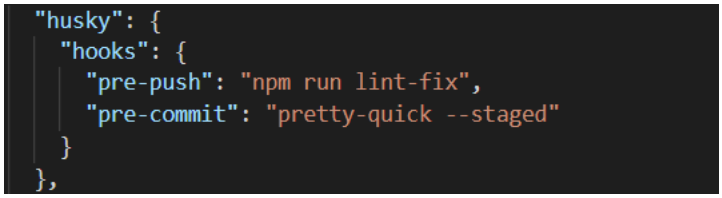


FIGURE 74. Setup husky for Git hooks

For creating templates, plopjs was chosen to be used. As mentioned above, boilerplate for common use types of elements were used for saving time and

preventing copying an old component structure to create a new one. Getting started with installing plop into a project with an install package from npm.

As plop is only used for the development stage, this dependency was set to put under “devDependencies”. After installing the package, “plopfile.js” was created under the root folder. This was the file that was read when using plop. As plop is saved under the project, “npm run generate” script was added into the script part in “package.json” (Figure 75).

```
"generate": "plop"
```

FIGURE 75. script for plop in "package.json"

In plopfile.js, scripts for creating hooks, components, and Redux actions, reducers were created because of their frequent usage. However, some components that needed complicated logic led to the decision of creating boilerplates for both functional and class components.

3.2.3 Setting up routing

As this project has multiple pages, therefore a routing system was needed. For routing, there is a compatible package designed for a React application: react-router-dom. “react-router-dom” is an alternative package from react-router and is suitable for building routing in React web applications. (22)

This package allows developers to set up routing, set parameters and achieve them, and pass data as a state to other components. It also supports both class components and functional components. There are three properties linked to a route and they can be accessed from any route to define the location and data: history, params, and location. They can be accessed by using hooks, such as useHistory(), useParams() and useLocation(), in functional components or by accessing “this.props” in class components. (21) An example is shown in figure 76, taken from the Pymestari application.

```
const data = this.props.location.state.data;
const { id } = useParams<RouterProps>();
const history = useHistory();
```

FIGURE 76. Get router props by this.props (up) and hooks (down)

All routes in Pymestari were set up in the MainPage component. As the application is a single page application, it shared the header and footer components between pages and the routing is only working within the large main container. (Figure 77)

```
<div className={styles.mainContainer}>
  <Switch>
    <Route exact path="/" component={DashboardPage} />
    <Route exact path="/user-info" component={UserPage} />
    <Route exact path="/challenges" component={ChallengePage} />
    <Route exact path="/explore" component={ExplorePage} />
    <Route
      exact
      path="/explore/module/:id"
      component={ExploreDetailPage}
    />
    <Route
      exact
      path="/challenges/:moduleId/:challengeName"
      component={ChallengeEditorPage}
    />
    <Route component={ErrorPage} />
  </Switch>
</div>
```

FIGURE 77. Routing logic was set in the MainPage component

3.3 Implementation by features

3.3.1 User Authentication

For starting using the Pymestari platform, users are asked to be authenticated with simple steps. They can log in by using the basic authentication method: email and password or connect and register by GitHub authentication. In figure 78, there are two forms for users, one is for newcomers to the system and the other belongs to old users.

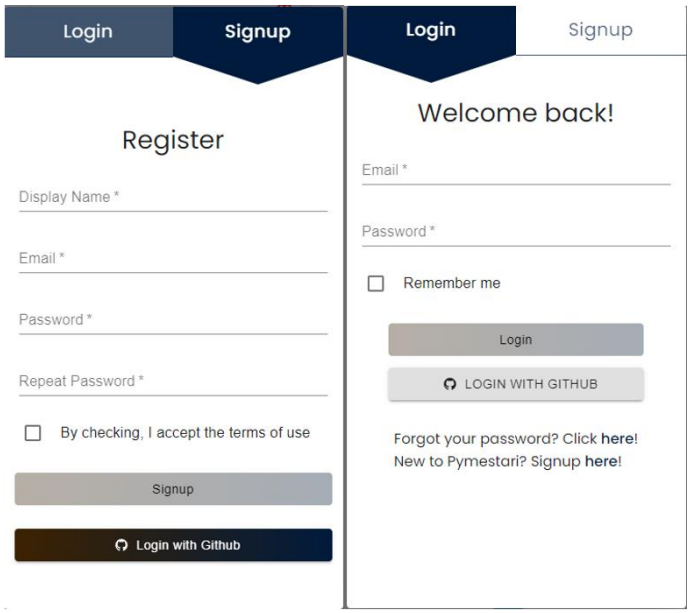


FIGURE 78. Signup (left) and Login (right) forms

On the programming side, authentication forms are put into a container containing tabs for different purposes: signup, login, and reset password. The container used Dialog and Tabs components from the Material-UI library. These two forms are similar and put under a simple routing tab. After hitting the Signup or Login button, the event from the client is called within the React component (Figure 79), then, an action in figure is dispatched to Redux.

```
submit = (e: React.SyntheticEvent): void => {
  e.preventDefault();
  const { email, password } = this.state;
  this.props.login(email, password);
};
```

```
submit = (e: React.SyntheticEvent): void => {
  e.preventDefault();
  const { email, name, password } = this.state;
  this.props.signup(email, password, name);
};
```

FIGURE 79. React events to submit login (up) and signup (down) forms

The form used component's state to control fields it wrapped and when to submit action is fired, fields' values are taken by dispatching a redux action. In figure 80, on the left side, the login action from redux took the data and called an API to send to the server, where all data would be handled. Then the server responded with authentication info such as email and display name for client-side to display.

The signup action is written similar to the login action but has a minor change for data fields, which can be seen on the right side of figure 80.

```
export const login = (email: string, password: string): any => (
  dispatch: Dispatch<AuthenticationActionType>
): any => {
  dispatch({
    type: `${LOGIN}_${ActionType.Pending}`,
    errorMessage: ''
  });

  AuthApi.login(email, password)
    .then((response) => {
      if (!response.data.success) throw 'Incorrect Username or Password';

      dispatch({
        type: `${LOGIN}_${ActionType.Fulfilled}`,
        payload: { ...response.data, errorMessage: '' }
      });
      localStorage.setItem('user', JSON.stringify(response.data));
    })
    .catch((error) => {
      dispatch({
        type: `${LOGIN}_${ActionType.Rejected}`,
        payload: { errorMessage: error }
      });
    });
};
```

```
export const register = (
  email: string,
  password: string,
  displayName: string
): any => (dispatch: Dispatch<AuthenticationActionType>): any => {
  dispatch({
    type: `${SIGNUP}_${ActionType.Pending}`
  });
  AuthApi.register(displayName, email, password)
    .then((response) => {
      console.log('registerAction response:', response);
      response.data.success &&
        dispatch({
          type: `${SIGNUP}_${ActionType.Fulfilled}`,
          payload: { ...response.data, errorMessage: '' }
        });
      // handleSetFormName(FormTypes.message_form);
    })
    .catch((error) => {
      dispatch({
        type: `${SIGNUP}_${ActionType.Rejected}`,
        payload: { errorMessage: error }
      });
    });
};
```

FIGURE 80. Redux login (left) and signup (register, right) actions are dispatched when React event is fired respectively

3.3.2 The Challenge Module

Challenge mode is one of two main modules, and it contains two pages in client structure: a list of challenges and detail of a challenge.

This module can be accessed by navigation on the header of the main page. This mode is meant to show the list of challenges that can be accessed by developers. The challenging detail must be cleared and show categories, as well as the difficulty before they decide to take the challenge. On the challenge page, the description for the challenge must be stated, and also place for comments and discussion about the question, as it is the help from the community when it goes to trouble.

On the list page, challenges can be filtered by one to three models: difficulty, challenges' status by users, and questions' category. In figure 81 below, challenges are divided into pages and show 10 lines per page, which can be navigated obviously with the indicators at the bottom of the table. Three filters and a search bar are put on top, above the list, and aligned with flex style. (Figure 81)

After filtering and searching, a state setting is performed, and the list of challenges will be rendered again by a new state.

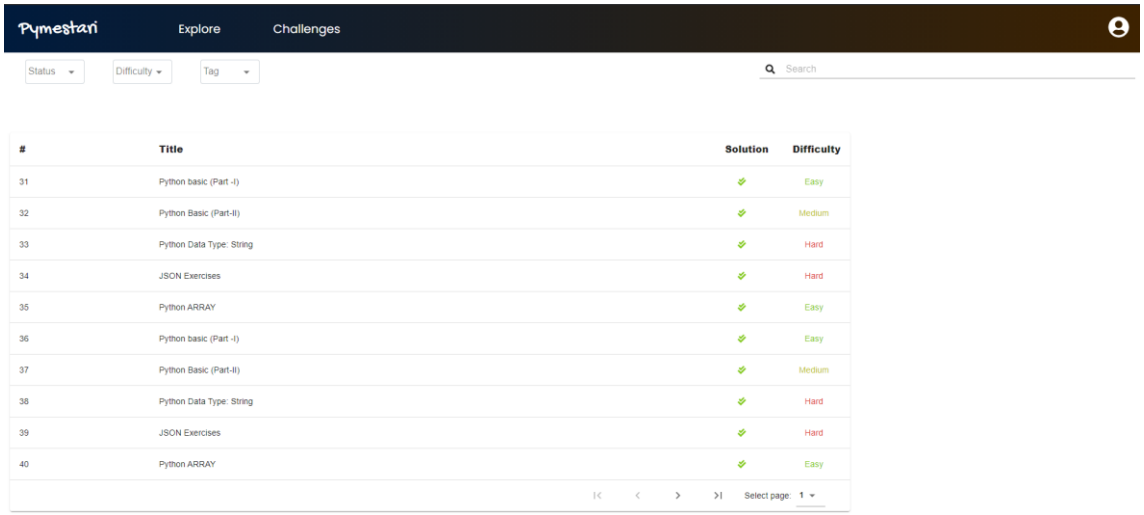


FIGURE 81. List of challenges page

Each line shows to extract data about a challenge: the title to describe what is the issue about, the level of difficulty, and the tick icon to show the availability of the solution. When users click to open an issue, a panel of the drawer will be slightly slid from the right side to show a short description of the challenge in figure 82.

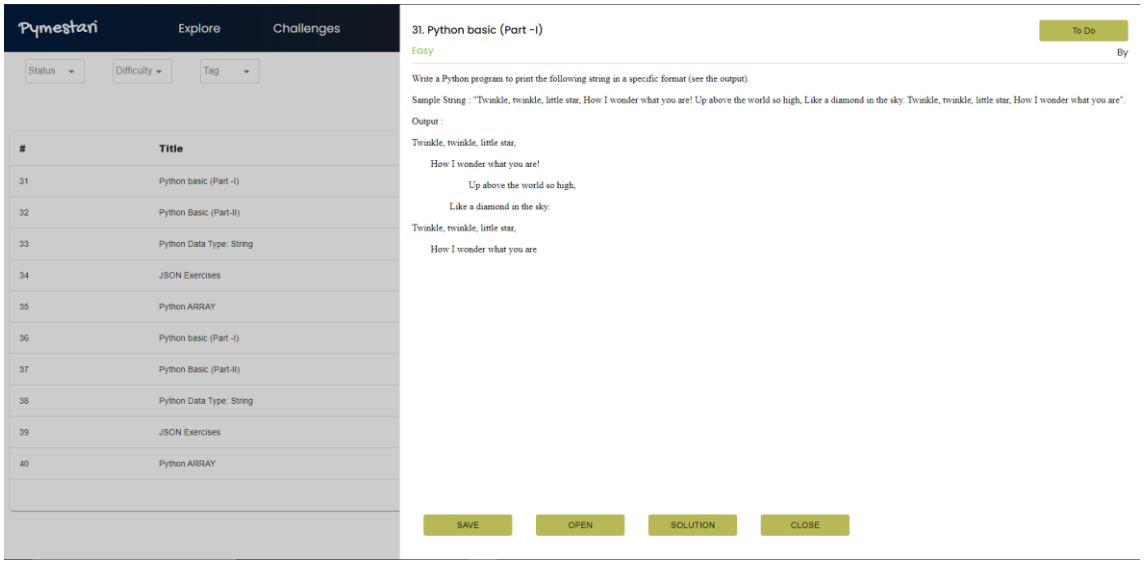


FIGURE 82. Challenge list with opened short description panel for a challenge

This page is an individual large component and contains smaller reusable components for displaying the interface. The page contains the heading and container components. The container component took all methods and data passed from the redux store (Figure 83). All actions in the table component were performed by the function from its parent and redux action by using passing properties to the component. The purpose of dividing the page into smaller components is to organize the structure and to save time for debugging and rechecking.

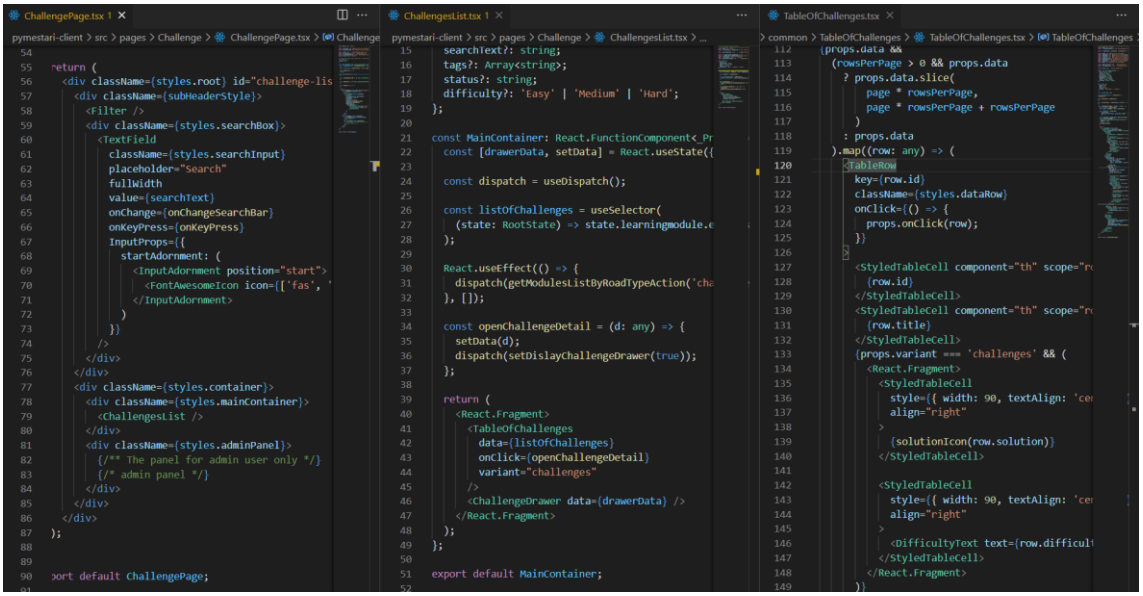


FIGURE 83. Challenges list page and smaller components

When users first access the challenge page, an action to get data is dispatched from the useEffect method. This method here is called with an empty array at the second params stands for calling only once when users access. The data is taken from the useSelector method when the component is rendered the second time and displayed by the TableOfChallenge component. (Figure 83)

3.3.3 The Explore Module

The next main feature is the explore mode. This mode contains learning paths with theories and exercises. This mode has several modules with different knowledge. Each module includes two parts: lessons and exercises. Lessons are theories divided into smaller sets and exercises are similar to challenges in challenge mode, but questions are relating to the module content.

Similar to challenges in challenge mode, the list of modules will be loaded from the server with redux actions and returned by state, which is retrieved from redux hooks, within the component.

The action is called to send a request to the server via the Redux store inside the scope of the useEffect hook. After receiving the response from the server, the page will be rendered again and now the result is stored to the reducer and taken by selector hook (Figure 85). The data in the listOfModules variable is displayed by cards that are aligned flexible to the width of the device's screen in figure 84. Currently, modules are using fake data during the development phase. Each card represents a module that contains simple information: title, level of learning, and in the saved list. Users can later add the module to their favorite list or learning list.

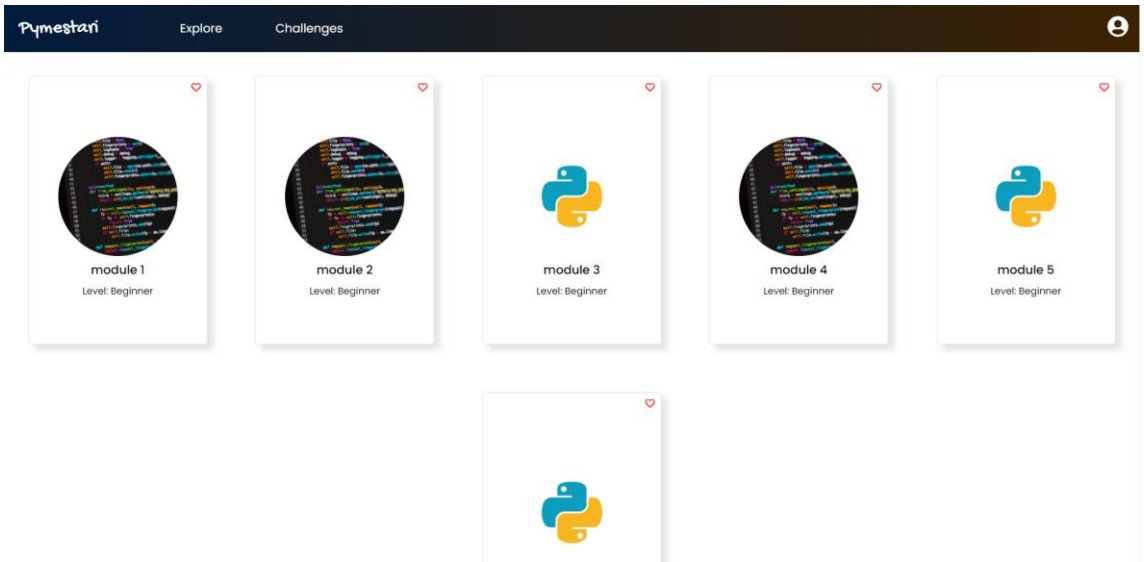


FIGURE 84. Explore module page

```
React.useEffect(() => {
  dispatch(getModulesListByRoadTypeAction('explore'));
}, []);

const listOfModules = useSelector(
  (s: RootState) => s.learningmodule.modulesList
);
```

FIGURE 85. Explore page - perform action

Opening a module will display a new page in the following figures 86 and 87. These two figures show the result of the lesson part and the exercise part from a module.

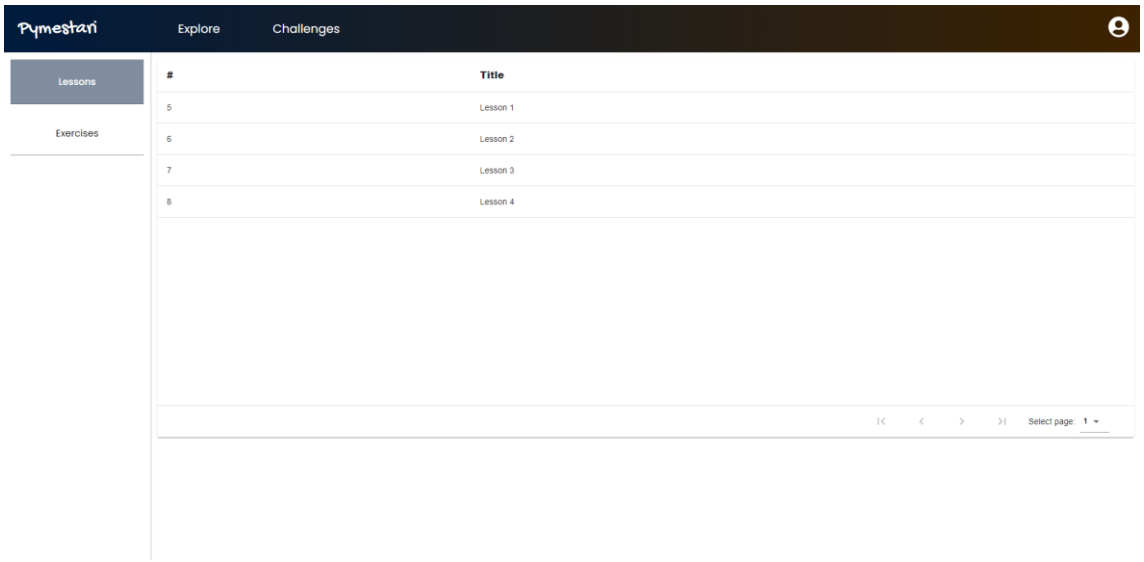


FIGURE 86. Explore page - Lessons tab

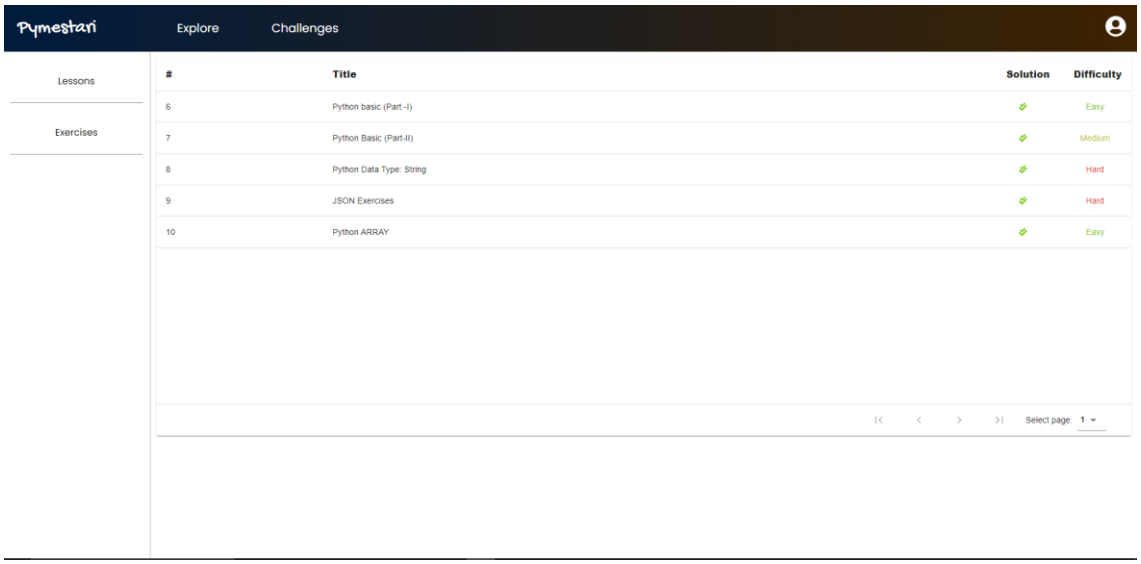


FIGURE 87. Explore page - Exercises tab

Figure 88 shows the works behind display data in figure 86 and 87. Before showing the data, the request for fetching the list is called within the useEffect hook with the ID read from the route parameter. Lessons and exercises are states that belong to the learning module reducer. Each state is the array of items respectively fetched from the database and the array is called from the component by useSelector hook, which is used for retrieving data from the Redux store. Arrays of items are looped to show on the interface by using the component tableOfChallenges (in figure 85).

```
const { id } = useParams<RouterProps>();
const history = useHistory();
const dispatch = useDispatch();
const [tab, setTab] = React.useState<tabType>('lessons');

const lessons = useSelector(
  (s: RootState) => s.learningmodule.currentModuleLessons
);

const exercises = useSelector(
  (s: RootState) => s.learningmodule.currentModuleExercises
);

useEffect(() => {
  dispatch(getModuleById(id));
}, []);
```

FIGURE 88. ExploreDetail component

3.3.4 Code editor

When users open a challenge or an exercise, the view of the detail of the problem is opened as shown in figure 89. On this page, there are tabs including description, submission, the theory about the issue, and the code editor.

The page is designed in two columns: one is for tabs about information of issue and one for the view of editor and validation of user edit. The page is aligned with a flex view with a ratio of 1:2. For the screen with a smaller size, below 1440px, all parts of the page become tabs, in order to get more space for the editor and display the other information, shown in figure 90.

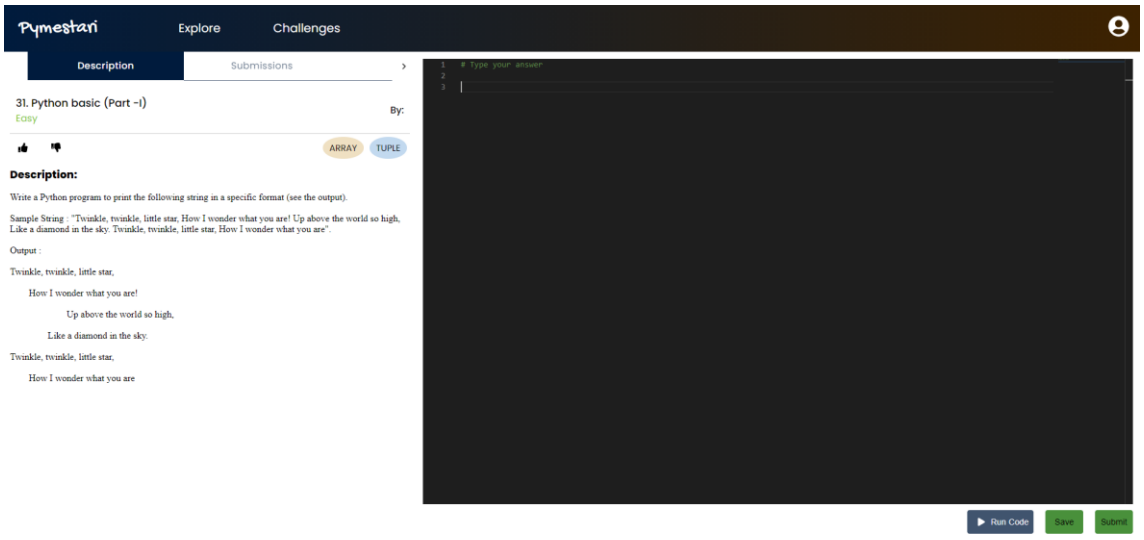


FIGURE 89. Challenge full view

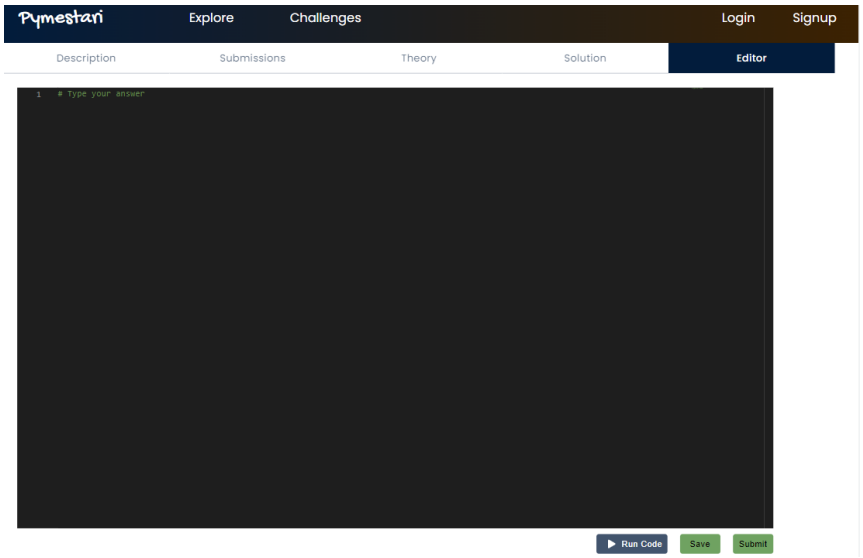


FIGURE 90. Challenge editor page in smaller screen

The editor is implemented by embedding the Monaco editor and adding a custom option object with the Python language (Figure 91). In this way, a code editor is created and highlights all keywords from Python when user editing. Monaco editor also provides listeners when the code editor appears and when it is changed by the user input. The value changes while the user input is saved to the component state in order to control what users have given to the interface.

```
const options: typeof monacoEditorProps.options = {
  selectOnLineNumbers: true,
  ariaLabel: 'Pymestari - Code Editor',
  folding: true,
  formatOnPaste: true,
  wordWrapColumn: 80,
  wordWrap: 'on',
  tabCompletion: 'on'
};
```

```
121      <Editor
122        value={code}
123        defaultValue={defaultValue}
124        language="python"
125        theme="vs-dark"
126        height={'100%'}
127        options={options}
128        onChange={(val) => onChange(val)}
129        onMount={(editor) => onMount(editor)}
130      />
```

FIGURE 91. Custom option and property on the Monaco editor.

When users press the submit button, the change users have made is saved into the database and the code is sent to the server for validation of the answer whether it is valid and correct to the question. In the container of the editor, when the submission is made, the collapsed part of the result is displayed, and the editor is resized to get space for showing the result responded from the server (Figure 92)

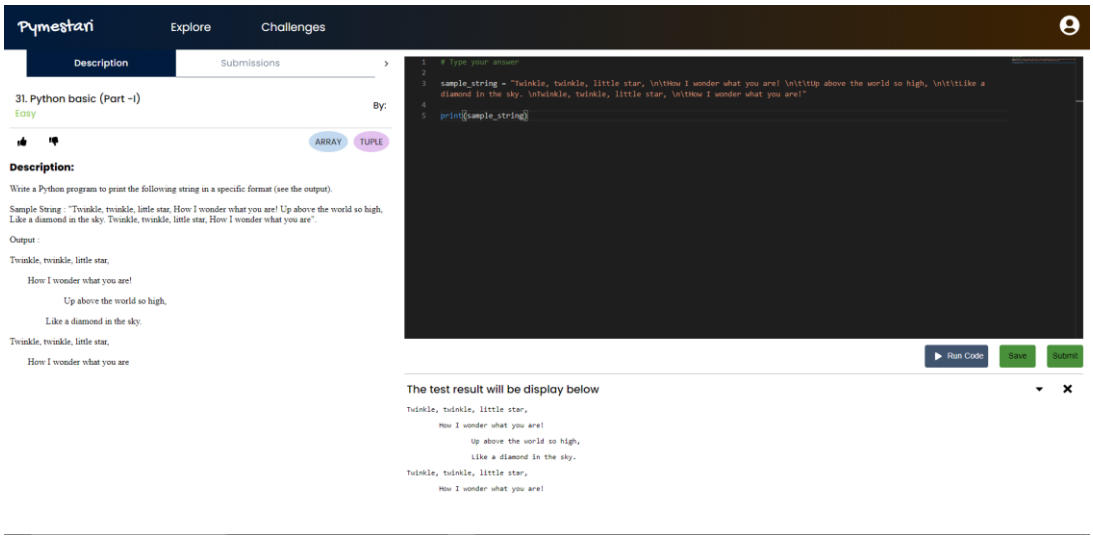


FIGURE 92. The code editor with the result

On the request side, the submit action is dispatched and the submitContent method from editorAPI is called. The dispatch call first sends the pending action to the reducer, then makes the post request to the API with the content of the users' input code. Then, whether the code is compiled succeed or fail, the result from the API is sent back to the client and displayed in the collapse bottom panel in figure 93.

```
export const submitAction = (code_content: string | undefined) => (
  dispatch: Dispatch<EditorActionType>
) => {
  dispatch({
    type: `${EDITOR}_${ActionType.Pending}`
  });
  editorApi
    .submitContent(code_content)
    .then((response) => {
      console.log('res: ', response);
      dispatch({
        type: `${EDITOR}_${ActionType.Fulfilled}`,
        payload: response.data
      });
    })
    .catch((error) => {
      dispatch({
        type: `${EDITOR}_${ActionType.Rejected}`,
        errorMessage: error
      });
    });
};
```

```
orApi.tsx ×
ari-client > src > services > editorApi.tsx > ...
const submitContent = async (code_content: string | undefined) => {
  return await axios.post(`/api/editor`, { code: code_content });
};
```

FIGURE 93. submitAction and submitContent methods

4 CONCLUSION

In summary, this thesis aimed to develop the user interface design and the client part with applied good design principles of a platform for Python learners, inspired by the demand of Interjektio. The application with fully implemented features will be used for their training session and period for new employees.

The main scope was to implement the early stage of the application including main features: lessons modules, challenges list and the editor with the editable and submittable availabilities. This thesis work also included the stage of researching and designing the interface for the application before the development stage.

The python learning platform has become the need of internal for Interjektio, therefore, the programming code was readable, maintainable, with consistent styles. This thesis had successfully created all the main features based on the design: the challenges path, the exploring path and the submittable code editor. Modern technologies, React framework and its relating packages, were applied for developing this project. Accordingly, the modern stack, React combined with Python was applied to create an internally run, simple, and easy-to-develop environment.

Moreover, this project was fully written in TypeScript, commented in English, and used the common and popular stack. Therefore, the future development can be implemented and followed by this early staged application easily, as the stack is commonly used by the community and the document was written.

For a further development discussion, the application will be built on this early stage. This application is still lacking the landing page (in the design in figure 57), which will be the next feature to implement. Next, the system will need to have a profile for users, which stores and displays their history works and the favorite list. Then, a new user guide will be added to help new users learn more quickly how to use Pymestari platform. Finally, when the application is widely used, the

test system will be developed for testing the level of developers. The feature of making the learning route from beginner level to advanced level would be designed, so they can only access to the next level. For example, a user might be at the intermediate level, after completing the previous path or passing the knowledge test.

REFERENCES

1. Robert J. K. Jacob. 2003. User interface. Encyclopedia of Computer Science. John Wiley and Sons Ltd., GBR, 1821–1826.
2. Lamprecht, E. 2021. The Difference Between UX And UI Design - A Beginner's Guide. Date of retrieval 12.05.2021. https://careerfoundry.com/en/blog/ux-design/the-difference-between-ux-and-ui-design-a-laymans-guide/
3. Friedman, V. 2011. User Interface Design in Modern Web Applications. Date of retrieval 12.05.2021. https://www.smashingmagazine.com/user-interface-design-in-modern-web-applications/
4. Osborn, T. 2021. Theory and Design Principles, Hello Web Design. No Starch Press. Date of retrieval 10.05.2021. https://learning.oreilly.com/library/view/hello-web-design/9781098128951/
5. Bootstrap. 2021. Bootstrap Document. Date of retrieval 1.10.2021. https://getbootstrap.com/docs/5.1/
6. Figma. 2021. Date of retrieval 14.05.2021. https://www.figma.com/
7. React. 2021. React Document. Date of retrieval 18.05.2021. https://reactjs.org/
8. 2020 Developer Survey. 2020. StackOverflow. Date of retrieval 18.05.2021. https://insights.stackoverflow.com/survey/2020
9. Adding a CSS Modules Stylesheet. 2019. Create React App Document. Date of retrieval 19.05.2021. https://create-react-app.dev/docs/adding-a-css-modules-stylesheet/
10. 5 ways to Style React Components in 2020. 2019. Bits and Pieces. Date of retrieval 19.05.2021. https://blog.bitsrc.io/5-ways-to-style-react-components-in-2019-30f1ccc2b5b/
11. TypeScript. 2021. TypeScript Document. Date of retrieval 28.05.2021. https://www.typescriptlang.org/
12. Redux. 2021. Redux Document. Date of retrieval 29.05.2021. https://redux.js.org/
13. React-Redux. 2021. React-Redux Document. Date of retrieval 30.05.2021. https://react-redux.js.org/

14. Dietz L. W., Manner J., Harrer S. and Lenhard J. 2018. Teaching Clean Code. ResearchGate. Date of retrieval 01.06.2021. https://mediatum.ub.tum.de/doc/1428241/file.pdf
15. Prettier. 2020. Prettier Document. Date of retrieval 01.06.2021. https://prettier.io/docs/en/index.html
16. ESLint. 2021. ESLint Document. Date of retrieval 01.06.2021. https://eslint.org/
17. Plop. 2020. Plop Document. Date of retrieval 02.05.2021. https://plopjs.com/documentation
18. Husky. 2021. Husky Document. Date of retrieval 02.05.2021. https://typicode.github.io/husky/
19. Stemmler K. 2019. Enforcing Coding Conventions with Husky Pre-commit Hooks. Date of retrieval 06.06.2021. https://khalilstemmler.com/blogs/tooling/enforcing-husky-precommit-hooks/
20. Material UI. 2021. Material-UI Document. Date of retrieval 12.07.2021. https://material-ui.com/getting-started/installation/
21. React router. 2021. React router Document. Date of retrieval 12.07.2021. https://reactrouter.com/web/guides/quick-start
22. Atto, E. 2019. Understanding The Fundamentals of Routing in React. Date of retrieval 17.07.2021. https://medium.com/the-andela-way/understanding-the-fundamentals-of-routing-in-react-b29f806b157e
23. Jadhav, D. 2021. Why choose React for Frontend. Date of retrieval 08.12.2021. https://dev.to/digvijayjadhav98/why-choose-react-for-frontend-4m23
24. Monaco Editor. 2021. Monaco Editor Document. Date of retrieval 01.12.2021. https://microsoft.github.io/monaco-editor/
25. Geek By Nature. 2019. Web based IDE with React & Microsoft Monaco Editor. Date of retrieval 02.12.2021. https://medium.com/@geekbynature/web-based-ide-with-react-microsoft-monaco-editor-5ad5eaebaf92
26. Visual Studio Code. 2021. Visual Studio Code Document. Date of retrieval 02.12.2021. https://code.visualstudio.com/docs