



**Politecnico
di Torino**

Politecnico di Torino

Media and Cinema Engineering

A.Y. 2023/2024

Graduation session April 2024

A web-based platform for improving Erasmus experience management

Supervisor:

Luigi De Russis

Candidate:

Giovanni Mercorillo

Abstract

Task management systems and cloud drives play a key role in enhancing productivity and organization, benefiting both individuals and teams. These solutions provide methods for managing tasks, deadlines, priorities, and documents. However, actual available solutions often lack a mixed approach between them, especially for student's usage. During his Erasmus internship the candidate worked on the Costabex site, a project with the abroad students' mental health as the main theme and developed an interactive customized service for student's life-work management called the Costabex checklist to fill the early mentioned insufficiency. Following digital interaction design principles, the prototype combines a document drive and a task management tool to address the bureaucratic and ordinary challenges faced during an Erasmus exchange. Within the thesis, the Costabex project will be presented, along with the work carried out during the internship. This includes the development of the Costabex checklist web app, the listing and focus on the state of the art and lastly the main tools, future objectives and the principles used.

Acknowledgements

I want to express my sincere thanks to my supervisor: prof. Luigi De Russis for all the help and advice he gave me during my thesis work, from the beginning of my internship abroad to its very last making day.

To my family, I am not grateful enough for being with me all the time, your unstoppable support and love has always been a great source of strength at every stage of the journey.

To my friends, thank you for having been there with me in both the good and bad times of this ride through my years studying. Your encouragement has been the greatest gift to me. This thesis was a hard job, but with your support I was able to accomplish it.

I am thankful for everything and thrilled to go on my way with the knowledge and skills acquired. Thank you all for being part of this important chapter in my life.

Giovanni

Table of Contents

List of Figures	VII
1 Introduction	1
1.1 Context	1
1.2 Goal	2
1.3 Thesis structure	3
2 Background	5
2.1 Digital design	5
2.1.1 User-centered design	5
2.1.2 Interaction design	6
2.1.3 The 5 IxD dimensions	7
2.2 Additional features	9
2.2.1 Usability	9
2.2.2 Feedback, Affordances, and Signifiers	10
2.2.3 Accessibility	10
2.2.4 Navigation and Information Architecture	10
2.2.5 Prototyping, Testing, and Iteration	10
2.3 Design thinking fundamentals	11
2.3.1 Design thinking actions	11
2.4 What happens if the user is trascurated?	13
2.5 Future Trends and Technologies	13
2.6 Applications in Costabex project	13
2.6.1 Informative module - Costabex website	13
2.6.2 Interactive module - Costabex checklist	13
3 The Costabex project	15
3.1 Introduction	15
3.2 Site structure	16
3.2.1 Landing page	16
3.2.2 Training section	16

3.2.3	Counseling section	18
3.2.4	Checklist section	20
3.2.5	For institutions section	21
3.2.6	About section	22
4	The Costabex checklist	27
4.1	The state of the art	27
4.2	Presentation	28
4.3	Mock-ups	28
4.4	The prototype	31
4.4.1	Landing page	31
4.4.2	Registration and login page	31
4.4.3	Dashboard page	33
4.4.4	Add task page	35
4.4.5	Show task page	36
4.4.6	Edit task page	37
4.4.7	Account page	39
4.4.8	Error page	39
4.5	Future developments	40
5	Implementation	41
5.1	Introduction	41
5.2	Front-end development and tools	41
5.2.1	CSS and flexbox	41
5.2.2	Bootstrap	42
5.2.3	EJS and Partial:	42
5.2.4	Views Folder with Layouts and Public Folder	42
5.2.5	Flash package	43
5.3	Back-end development and tools	43
5.3.1	JavaScript	43
5.3.2	DOM, AJAX, AJAJ	44
5.3.3	Node.js	44
5.3.4	Express, Express Router and Session	45
5.3.5	MongoDB	45
5.3.6	Mongoose	46
5.3.7	Middlewares and error handling	46
5.3.8	Passport authentication	46
5.3.9	Authorization	46
5.3.10	Joi schema validation	47
5.3.11	Cloudinary	47
5.4	Code overview	48

5.4.1	App.js file	48
5.4.2	Models	52
5.4.3	Public folder	57
5.4.4	Routes	57
5.4.5	Controllers	63
5.4.6	Middlewares	67
5.4.7	Views	69
5.4.8	Joi validation and utils	76
6	Conclusions	78
	Bibliography	79

List of Figures

2.1	UX, UI and IxD	6
2.2	First IxD dimension	7
2.3	Second IxD dimension	7
2.4	Third IxD dimension	8
2.5	Fourth IxD dimension	8
2.6	Fifth IxD dimension	9
2.7	Design actions	11
2.8	Actions process	12
2.9	Convergence-divergence model	12
3.1	Costabex logo	15
3.2	Costabex landing page	16
3.3	Costabex training page	17
3.4	Workshop details	17
3.5	Counselling page	18
3.6	List of topics related to the mobility period	18
3.7	Specific topic article	19
3.8	Counseling contacts	19
3.9	Counseling checklist page	20
3.10	Checklist topic article	20
3.11	For institution - handbook	21
3.12	For institution - materials	22
3.13	Costabex about page	24
3.14	Costabex results page	25
3.15	Costabex partners	25
3.16	Costabex partners	26
3.17	Costabex news	26
4.1	Mockup landing page	28
4.2	Mockup register page	29
4.3	Mockup login page	29

4.4	Mockup dashboard	30
4.5	Landing page	31
4.6	Sign up page	32
4.7	Login page	32
4.8	Student dashboard	34
4.9	Tutor dashboard	34
4.10	Add task - student	35
4.11	Add task - tutor	36
4.12	Show task - student	37
4.13	Show task - tutor	37
4.14	Edit task - student - tutor assigned task	38
4.15	Edit task - tutor	38
4.16	Account page	39
4.17	Error page	40
5.1	Prototype's files	48
5.2	Public folder	57
5.3	Views folder	69

Chapter 1

Introduction

1.1 Context

In a world becoming increasingly connected, student exchange programs similar to Erasmus plus have become essential parts of higher education, giving the students a chance to develop academically, culturally, and personally. Issues like bureaucratic obstacles, balancing academic workload with personal well-being, however, could be overwhelming sometimes. For this reason, task management systems and remote drives are some necessary tools when it comes to deal with these challenges, as they provide great features for organizing tasks, deadlines, and collaboration among students and teams. Though they are beneficial, current solutions do not offer a complete tool with proper integration between task and file management. It is also important to remember how the pandemic drastically changed the perception of students' mental health, giving more importance to it, opening psychological support platforms and organizing meetings for them. To better understand and face this topic some programs like *Mental Health Unboxed* by ESN [1] were conducted with some volunteers trying to answer to some questions like how did the pandemic influence volunteer's education and work as well as their personal and social life; what were the challenges they have faced and factors that have contributed to their mental health issues; what type of support they have received and the support they think should and/or would like to receive. This synthetically results on trying to have as many social events, training and preparatory workshop as possible for all people participating, leading them to make a strong network and give them good preparation before, during and after the mobility without taking anything for granted. Additionally, huge progress was made on the digital tools, even if some more improvements can be done. During Erasmus plus internship for Academia Institute of Technology in Maribor (Slovenia), conducted within the context of the Costabex project, the candidate had direct experience facing

these issues and embarked on a mission to address the trials and tribulations by developing a first, tailored solution aimed at enhancing student well-being and productivity: the Costabex checklist, a combination of storage and task management functionalities, which has been specifically prototyped to facilitate the Erasmus students' documents and tasks' management. This thesis is a comprehensive representation of the Costabex project through which the Costabex checklist web application's evolution was traced, from the initial concept to the development and implementation. By doing hands-on experience, the complexities of students' life-work balance were explored, and the intricate challenges faced during international exchanges were tried to be solved, prototyping creative solutions for tackling them through technological tools.

1.2 Goal

The goal of the thesis is to showcase the development process and functionality of the Costabex website, as well as detailing the interaction between students and Erasmus tutors facilitated by the Costabex checklist. This project aims not only to provide a comprehensive digital platform for academic and logistical support, but also to enhance the whole abroad experience for both students and tutors by streamlining communication and documents' exchange processes. The work is divided into two complementary stages, each with distinct but linked objectives and expected outcomes:

1. **The creation of the layout, graphic aspects, and content design of the Costabex's informative module (Costabex website).**
 - **Objective:** to develop an intuitive, engaging, and informative website that serves as the first touchpoint between the program and its participants. This includes the integration of user-friendly navigation, aesthetically pleasing graphic elements, and comprehensive content that effectively communicates all necessary information about the Erasmus experience.
 - **Expected Outcome:** a fully functional, visually appealing, and content-rich website that enhances user engagement, improves access to information, and supports the educational objectives of the Erasmus program.

2. The full implementation of the interaction module (Costabex checklist)

- **Objective:** to facilitate a seamless interaction channel for efficient and effective communication and document sharing between students and their Erasmus tutors. This involves creating a user-friendly interface that encourages active participation and engagement.
- **Expected Outcome:** an operational interaction module that simplifies the document exchange process, ensuring students and tutors have timely access to important information, thereby improving the overall abroad experience and support system.

1.3 Thesis structure

The thesis is structured into several chapters, described as follows:

- **Chapter 2. Background:** the principles presented in the second chapter are the ones related with design thinking methodologies. User interaction and User experience are some of the main features to take into account when a service involving people usage is developed. It's how some of the rules and best practices indicated in these theories were followed in order to make the informative (Costabex website) and interactive modules (Costabex checklist) easier to deal with.
- **Chapter 3. The Costabex project:** in the third chapter, the complete analysis of the Costabex project is drawn up. The topics shown are the structure of the site, presented in a detailed way in order to properly explain every page contents and front-end design made during the candidate's internship, the mental health project presentation, the project's objectives and programs.
- **Chapter 4. The Costabex checklist:** after a brief introduction of the state of the art, The thesis' fourth chapter is a complete introduction to the interactive module of the Costabex project, the Costabex checklist. A solution entirely thought, designed, and implemented by the candidate to reduce the communication gap among students and tutors interaction during Erasmus experience. Mock-ups, tutor tasks and personal tasks management and workflow are some of the features presented. Lastly, its future developments.

- **Chapter 5. Implementation:** the fifth and core chapter of the thesis covers the technical aspects of the web-app development, from the concept's dawn to the final prototype of the application. The main sections are constituted by the front-end and back-end making process with particular emphasis on code and scripts evolution, packages, libraries and features applied.
- **Chapter 6. Conclusions:** the sixth and conclusive chapter regards the results obtained and future direction of the Costabex checklist prototype, highlighting opportunities for development, refinement, and integration within the bigger picture of student's mental health support Costabex project.

Chapter 2

Background

In this chapter, a few concepts regarding digital interaction design, which include both User Interface (UI) and User Experience (UX) design, are presented. These concepts concern about how to make digital products that are useful and, at the same time, enjoyable. This overview covers the principles, processes, and methodologies that designers use to produce user-friendly and intuitive digital spaces. Some of them constituted and were considered for the process, design and development of the Costabex services. Design is the conscious act of conception, planning and prototyping products, environments, systems and services to reach a goal. This approach must lead to useful, desirable and sustainable contents and behaviors for users. The requirements for designs are the comprehension of users needs and technology related constraints, for this reason, the analysis should be oriented for both people and technology.

2.1 Digital design

Digital design is the activity related to the realization of digital products, services and systems, automatic elaboration of contents, distance transmittable and replicable data and services being usable through different devices.

2.1.1 User-centered design

During last years the product development shifts from a *machine-centered* approach, where the users were supposed to adapt their usage to the machine behavior to a user-centered approach focused on the users' needs, behaviors, and satisfaction in the first place. This design re-orientation toward user needs demands a good grasp of target users, which can be achieved through techniques such as user personas, user interviews, and journey mapping. Through user-centered design, designers

are able to put the users at the center of the build out process, and thus ensure that the final product is relevant to the intended audience and meets their explicit needs.

2.1.2 Interaction design

All the products have a status and a behavior, digital ones tend to have multiple of them so they request proper process to support designer action, putting the user at the center of the project development. Using John Kolko definition, *Interaction design is the creation of a dialogue between a person and a product, system, or service. This dialogue is both physical and emotional in nature and is manifested in the interplay between form, function, and technology as experienced over time.* Considering the various aspects related to the interaction design, they mainly converge from UX and UI to the overall Interaction experience Design (IXD). The

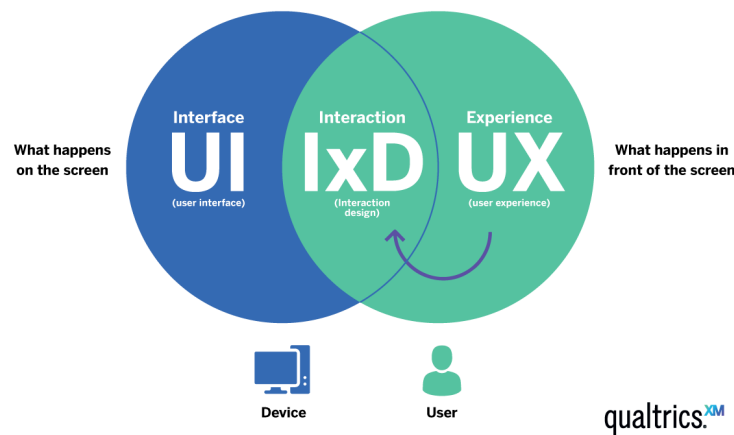


Figure 2.1: UX, UI and IxD

former consider the whole using experience of a products (reference target, user personas, branding, design, and usability tests) while the latter is based on the physical interaction and is a crucial part of the user experience because it is focused on the exact use moment. The reason for trying to develop these concepts is for creating products that make people reach a goal in the easiest and efficient possible way, considering and coordinating factors like aesthetics, sound, movement, and space. [2]

2.1.3 The 5 IxD dimensions

There are 5 dimensions related to the IxD which are a good reference to make proper products and pleasant user experience. They are:

Text: must have a significance and be easy to understand.

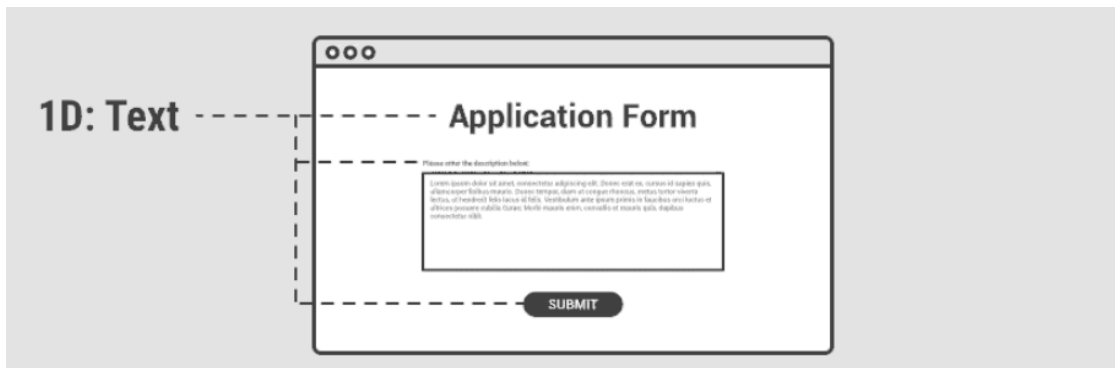


Figure 2.2: First IxD dimension

Visual representation: pictures, font, icons through which the user interacts. The better these things are optimized, the faster user interface comprehension will be.

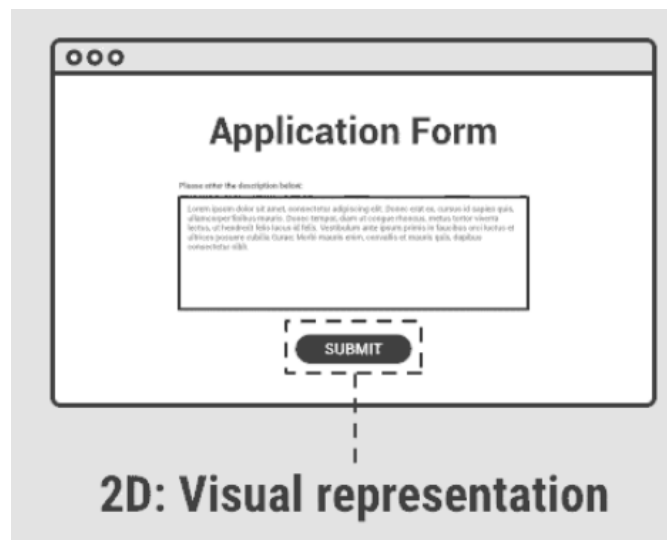


Figure 2.3: Second IxD dimension

Physical objects / space: related to the device through which the user interacts with the product. In addition, the physical space where it is used. Both elements influence the interaction.

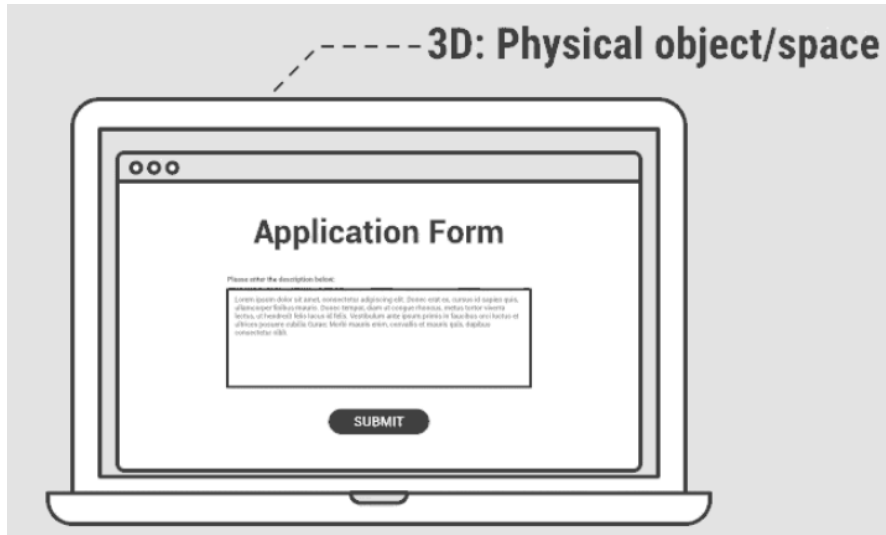


Figure 2.4: Third IxD dimension

Time: the related time to do an action and what influence the user like animations, feedback and more.

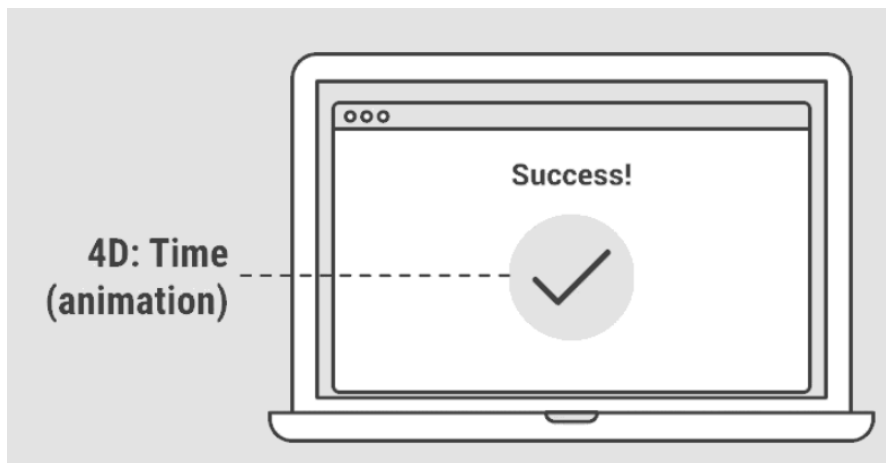


Figure 2.5: Fourth IxD dimension

Behaviour: how the user reacts to both interactions (how the actions are executed in the system) and reactions (how the user responds to feedback) [3]

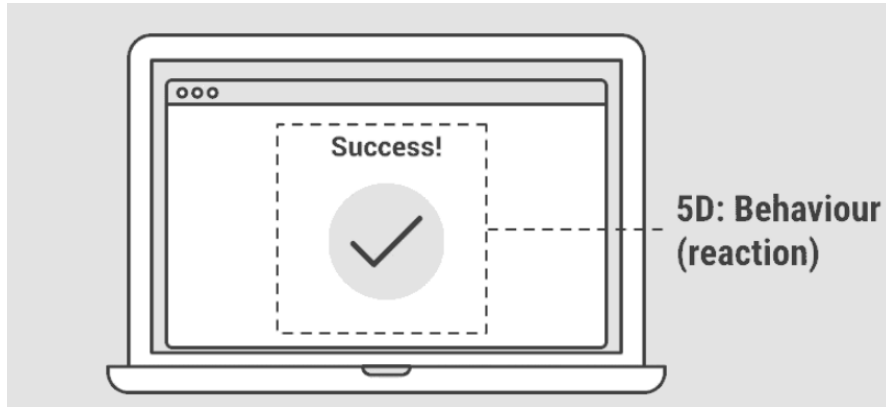


Figure 2.6: Fifth IxD dimension

2.2 Additional features

In interaction design theory there are other features to take into account during the development process and some of them are: *Usability, Feedback, Affordances and Signifiers, Accessibility, Navigation and Information Architecture, Prototyping, Testing, and Iteration.*

2.2.1 Usability

Usability is the fundamental principle of effective interaction design in digital environment. The ease of use is a concept that covers the simplicity with which users can operate a digital product and complete the tasks in a timely manner. The major elements are simplicity, ease of acquisition, and error prevention. Usability testing like A/B testing, usability labs, and remote testing, are an important part of the process. These tests help to show the obstacles and problems that a user faces during his journey, thus the designers will be able to make the interface better and more effective. [4]

2.2.2 Feedback, Affordances, and Signifiers

In practice, successful digital interaction design allows users to understand the system status through feedback and also uses affordances and signifiers to show how to use the interface. Feedback can be visual, auditory or haptic, and it is very immediate and intuitive, thus helping the user to guide his or her actions. In the Costabex checklist prototype, they are applied through Flash messages. The affordances encourage the user to interact with the buttons that appear to be clickable and the signifiers, such as icons and labels, provide information of their functions. [5]

2.2.3 Accessibility

Designing for accessibility means that any digital product is usable by people with different abilities, including those who are visually impaired, have hearing difficulties or with motor or cognitive disabilities. Accessibility considerations include creating keyboard-navigable interfaces, using suitable color contrast, and providing alternative texts for images. Through compliance with standards like the Web Content Accessibility Guidelines (WCAG), designers can design participatory experiences that are accessible to all. In Costabex website accessibility is supported by UserWay plugin. [2]

2.2.4 Navigation and Information Architecture

An effective navigation system and a good information architecture are the essential elements for users to find information and perform tasks. This will include organizing content logically, designing navigational menus that are user-friendly, and using clear labels. The well-designed information architecture and navigation systems decrease the cognitive load, giving the users the opportunity to understand how to use a digital product and to find the needed information. [6]

2.2.5 Prototyping, Testing, and Iteration

Prototyping and the iterative design are the dynamic processes that enable designers to explore concepts, test ideas with users and refine the solutions. From low-fidelity sketches to high-fidelity interactive prototypes, these tools assist in the experimentation and feedback's process. Iteration, based on the user testing and analysis, provides a process by which the design adapts to the actual user needs and behavior. [2]

2.3 Design thinking fundamentals

Design thinking is a human-centered approach to innovation that focuses on understanding the needs, desires, and behaviors of users. It consists of a sequence of cyclic stages that are used to identify and solve complex problems in a creative and caring manner. The process is made up of several stages such as empathizing, defining, ideating, prototyping, and testing. [7]



Figure 2.7: Design actions

2.3.1 Design actions

Empathize: the first design thinking stage is about getting to know users' needs, motivations and challenges. This usually includes conducting interviews, observations, and other types of research to acquire the user's experiences and perceptions.

Define: after the users' needs have been identified, the next step is to formulate the problem statement or challenge that the design team will tackle. This implies the integration of the research outcomes to arrive at a clear and implementable problem statement that will help in the design process.

Ideate: during this stage, design teams come up with a variety of possible solutions for the problem settled in the define stage. Quality is not the focus, but the number of ideas and diversity is what counts in order to stimulate creativity and innovation. Techniques, like brainstorming, mind mapping and rapid prototyping are usually employed to come up with, and explore, ideas.

Prototype: after brainstorming, the design team makes models or drafts of the possible solutions. Prototypes can be represented in many ways, ranging from sketches and wire frames to physical models or digital mock-ups. The main purpose of prototyping is to find the best ideas fast and cheap, and to get users' feedback.

Test: finally, the design thinking process concludes with a testing phase of prototypes with users to collect feedback and refine the design. This often involves usability testing, interviews, and observations to find out how good the prototypes meet the needs of users. The design team takes the feedback and makes the necessary adjustments and improvements to the prototypes, repeating the design process as needed.

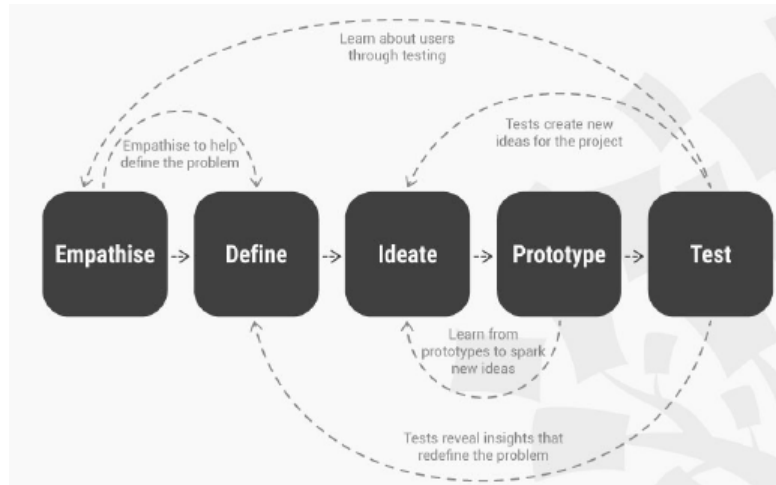


Figure 2.8: Actions process

The best feature of this approach is that all these processes are not sequential, and there's complete freedom for going ahead or back. This leads to divergent thinking and convergent thinking based on the fact that the process start from the analysis and end to a synthesis or the vice versa.

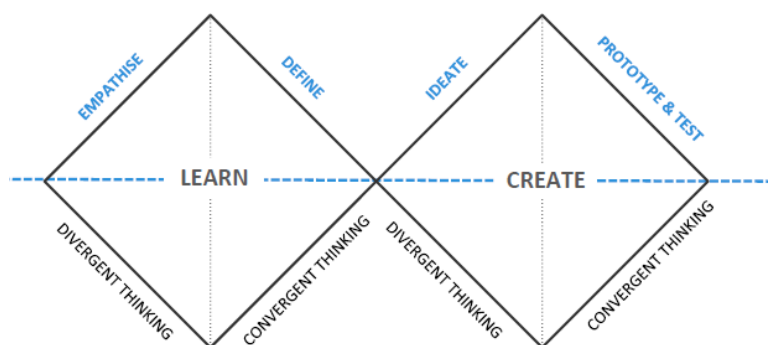


Figure 2.9: Convergence-divergence model

2.4 What happens if the user is trascurated?

Considering designing and developing application without consider the user needs and behavior would result in a raw product/service, with no easy comprehension for the user. That requires a big effort and makes easy operation complex. The good thing, as a developer, is to always think about the users without comparing them to himself, trying to reach the user desired expectations. [3]

2.5 Future Trends and Technologies

Emerging techs like voice interfaces, AR, VR and AI are broadening the horizon of interaction design in digital realm. Designers need to be always aware of such developments in order to exploit the new technologies as a way to improve user experience and taking into account ethical issues and the risks for the society too. [8]

2.6 Applications in Costabex project

2.6.1 Informative module - Costabex website

The project team incorporated some principles of user-center design and design thinking throughout the site development process. Initially, student research and feedback were utilized to empathize with the needs and preferences of the target audience. After internship beginning, this involved direct engagement of the candidate to gather insights on potential improvements and desired features for the main project. Although his involvement in the main project was initially indirect, having joined after the initial stages were already begun, he actively contributed by providing continuous feedback and collaborating with partners on further enhancements, designing the site pages' layout and identity.

2.6.2 Interactive module - Costabex checklist

In contrast, for the interactive module, entirely managed by the author, these principles were directly apply from the outset. Engaging in feedback research and gathering, directly asking colleagues for exchange programs needs and missing tools, doing brainstorming sessions and subsequently defining potential improvements and desired features. With this information, initial design mock-ups were created and iteratively refined the project's logic and functionality through prototyping was done. Subsequently, these insights were translated into actionable design decisions

by optimizing the application. Ideas were refined through feedbacks, ensuring that the product met the identified user needs and expectations.

Chapter 3

The Costabex project

3.1 Introduction

This chapter describes the informative module of the Costabex project, the Costabex website, with all pages' main features description and how they were implemented and thought, applying some of the previously seen interaction design principles as well as following IxD dimensions or design thinking actions like definition, ideation and prototyping. Every page was implemented with the help of the candidate during his internship in one of the program's institutional partner: Academia Institute of Technology in Maribor, Slovenia. Features like colors' palette, pictures and layout are some of his work part. The project website responds to the necessity of providing project-related materials accessible to all students. The decision to establish this platform stems from recognizing the significance of prioritizing the digitization of educational resources, with the objective of ensuring all project-related materials being readily available online. The website will primarily feature the online component of the educational program for students along with supplementary materials associated with it.[9]



Figure 3.1: Costabex logo

3.2 Site structure

3.2.1 Landing page

This page briefly introduce what actually Costabex is, a mental health support for Erasmus plus students and an educational program focused on their mental health, aimed to prepare to potential risks and problems that may occur during their mobility. It shows the program main activities: training and counselling, giving to the user a direct link to better understand in which they consist.

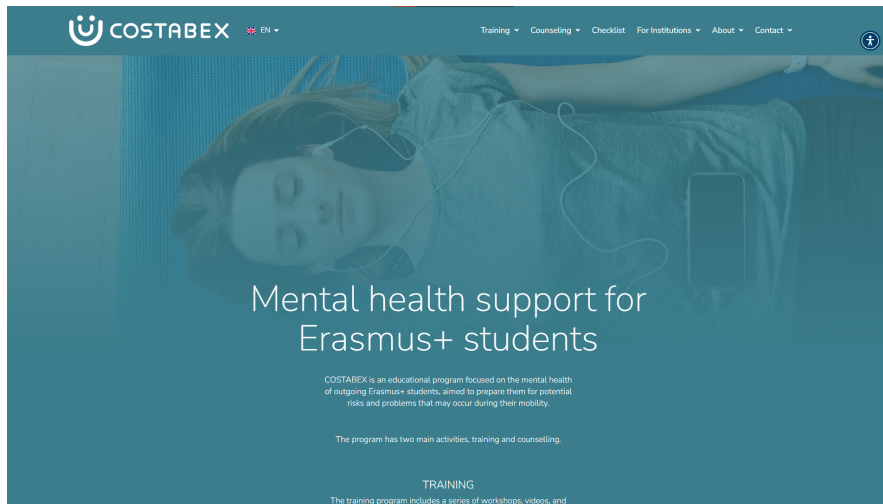


Figure 3.2: Costabex landing page

3.2.2 Training section

This page is dedicated to the training section of the site, seven workshops are included here regarding different topics: how to deal with stress, culture shock, mental health, crisis situations, intercultural communication, conflicts and finance management. Clicking on every workshop, a dedicated page appears. In this specific pages students find advices, video explanations and audio related to that topic. An additional page where students can listen and download meditation audio is present too.

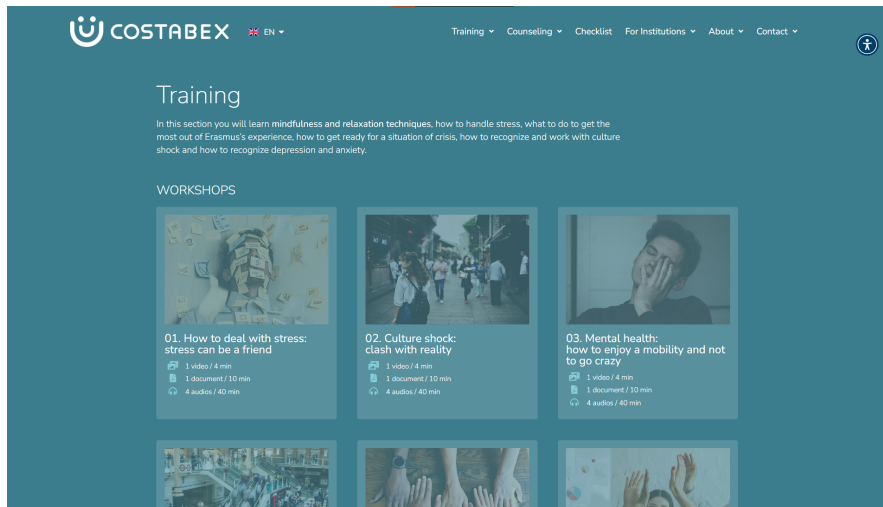


Figure 3.3: Costabex training page

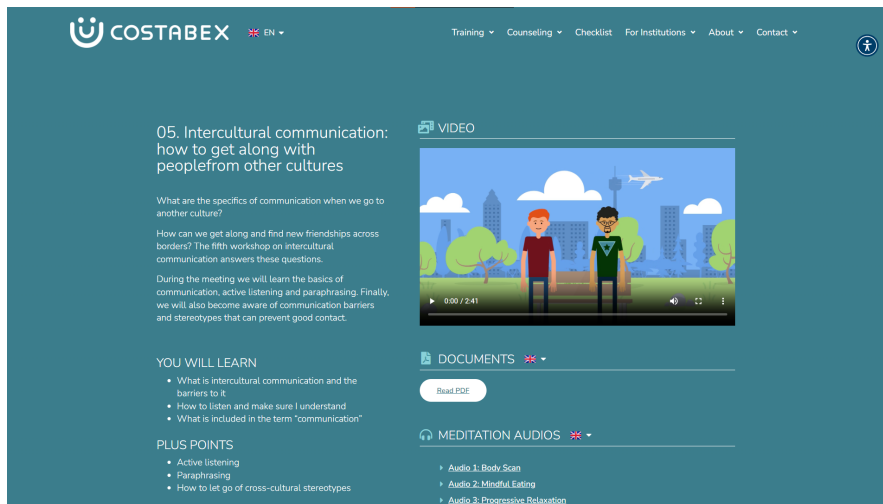


Figure 3.4: Workshop details

3.2.3 Counseling section

These pages present the counseling section of the site. It is constituted by several elements. Firstly, the *tips for student* page which shows different topics, based on the abroad experience's periods: before, during and after the mobility. Clicking on them, the mentioned specific topics are listed with a deep explanation for each one. This part of the site is the one from the Costabex checklist prototype takes inspiration for tasks management, basing them on periods and topics shown in this page.

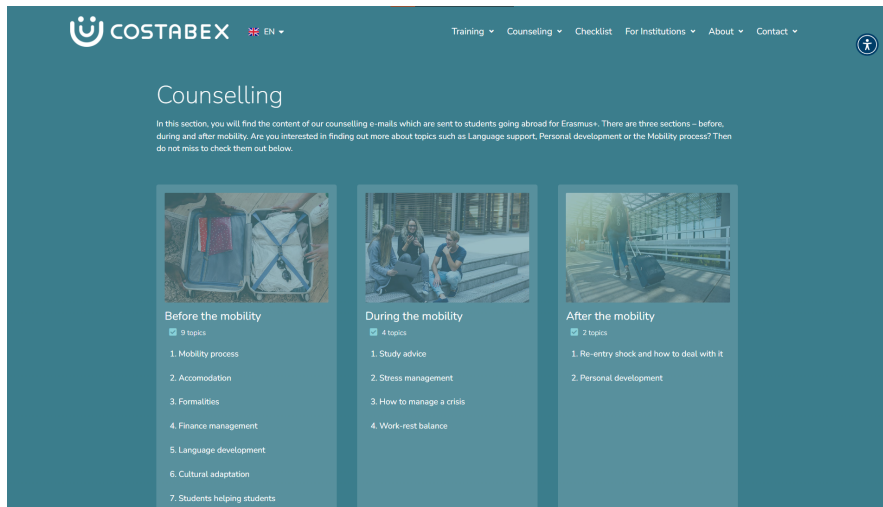


Figure 3.5: Counselling page

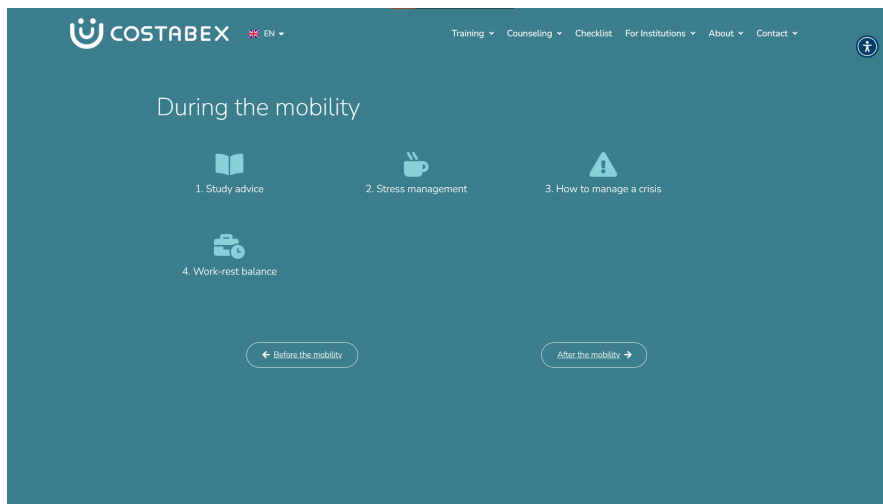


Figure 3.6: List of topics related to the mobility period

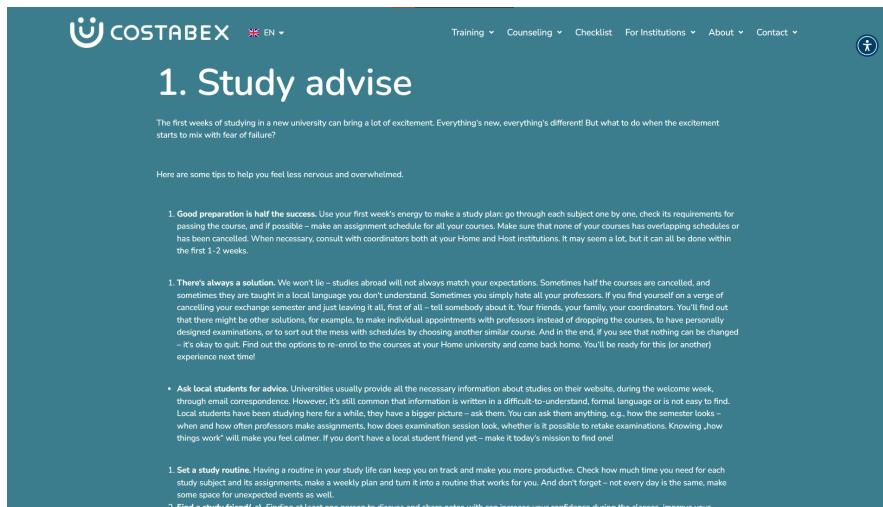


Figure 3.7: Specific topic article

The page related with counseling contacts is the last element of this section where students can find contacts from the partners' institutions in order to receive help in different fields: psychological, career, special needs, international office, study and law support.

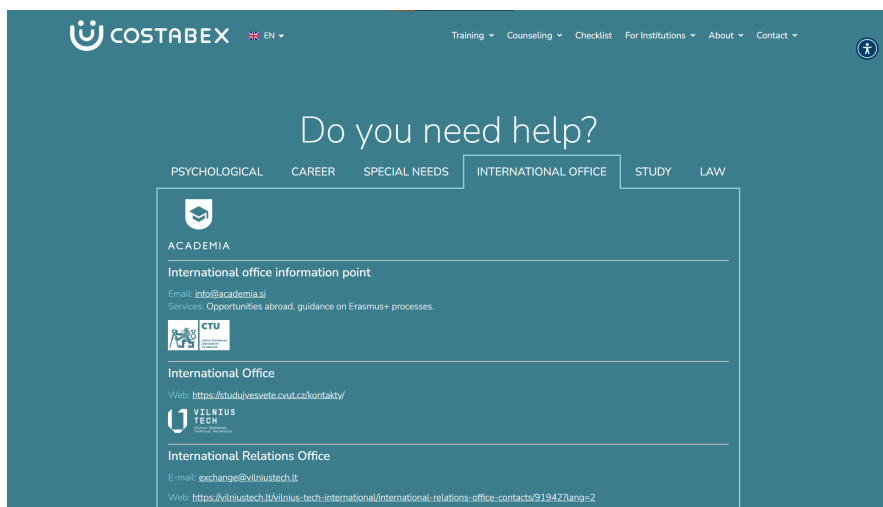


Figure 3.8: Counseling contacts

3.2.4 Checklist section

This page will be the one where the costabex checklist web app will be placed. Right now, it includes an extra page where additional info are displayed in order to give to students a comprehensive overview of what can't be missed during the mobility and what instead could be considered irrelevant for the experience scopes.

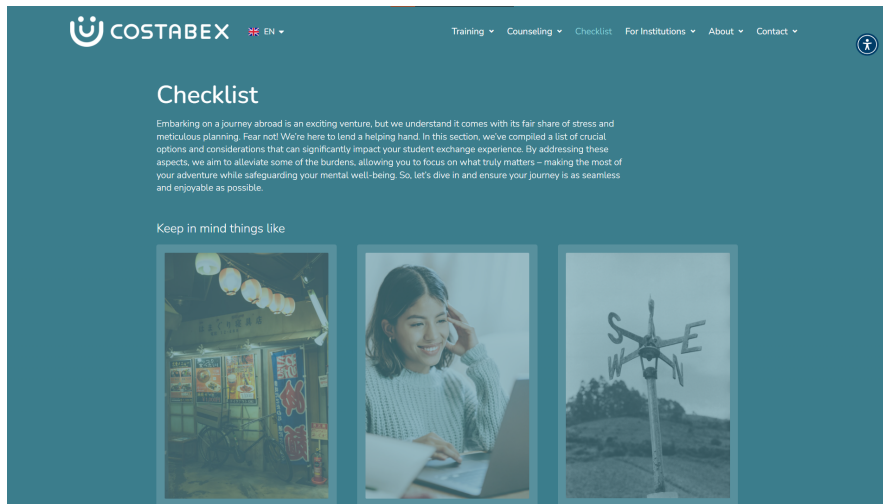


Figure 3.9: Counseling checklist page

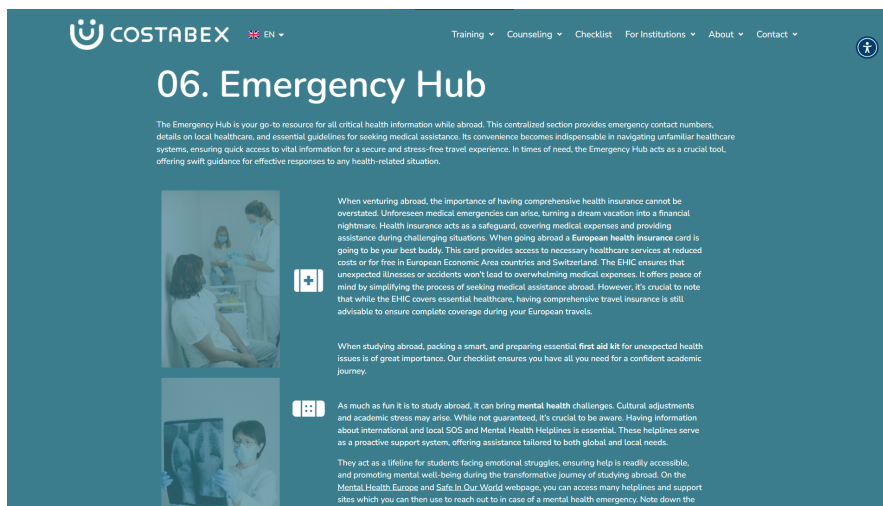


Figure 3.10: Checklist topic article

3.2.5 For institutions section

These pages are the ones where it is possible to view and download all the detailed and helpful materials sent by email to the students and tutors in order to get the best preparation for the experience. They are composed by two subsections: the handbook and the materials.

Handbook

The handbook is a project result still under development, specifically prepared for the employees of the university's international department. It is a booklet containing information about all the outputs together with the information on how to employ them at any European university available. The goal of the handbook is to serve as a manual on how to approach the online counselling service, how to train employees and students so that they are able to provide advice and support to outgoing students in need and how to use the materials which will be freely available on the project's web page. The handbook will include a manual on how to implement the program using the preparatory activities for the outgoing students at their respective universities. There will also be a manual on how to administer the project's web page so that the employees might update the posted information based on the current development of risks associated with the mobility of their students.

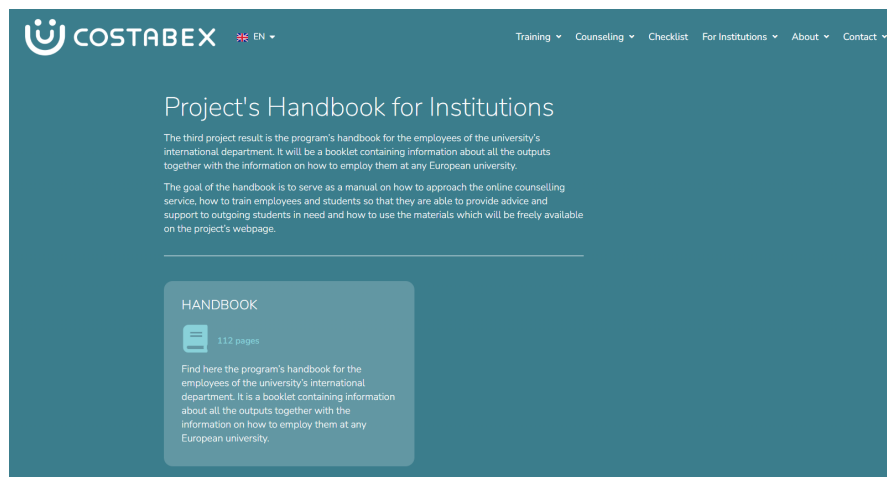


Figure 3.11: For institution - handbook

Materials

The other page present in the *For institution* section is the one where all the materials needed for both students and tutors are allocated. The purpose of these materials is to enable Erasmus plus coordinators at the institution to offer advice and support to outgoing students in need. They can freely utilize these resources for effective assistance.

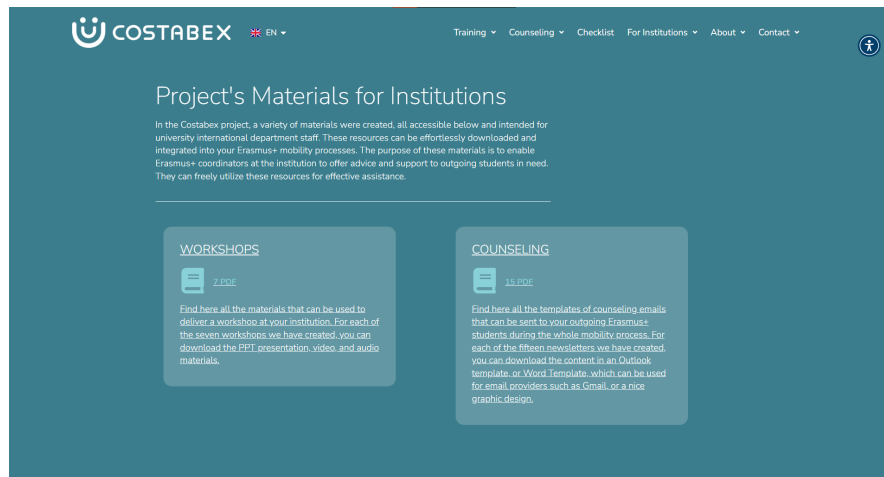


Figure 3.12: For institution - materials

3.2.6 About section

The about pages are the ones where the entire project is described: It's a summary of its main purpose, for what it's meant to. They are related with:

Project's objective description

The main target of the project is to increase the inclusiveness of the Erasmus plus program. By introducing a preparatory pre-departure educational course, those going on mobility will be informed of possible challenges and risks that can come along the way and will be equipped with the right strategies to cope with them. Subsequent assistance, provided through an online counseling service and all educational materials placed on the project's website, will aim to strengthen and make students feel more secure. These project outcomes are targeted to help mitigate the fears and concerns that often constitute the main barriers preventing students from going abroad. To be more specific, there is a will to heighten the enthusiasm of students dealing with mental health problems and with insufficient support who are looking for possibilities to study abroad.

Mental health program description

The Mental Health program, which is an integral part of the project, aims to help students cope with the stress of moving and make their mobility experience a pleasant one. The aim is achieved by delivering the training course before the departure, offer help during mobility and continuing to provide assistance after the return. This will make students feel more involved in Erasmus plus program and assure that the university is committed to facilitate the enjoyable and rewarding mobility experience. With the continuing decrease in the number of students who are outgoing, measures are focused on making the numbers go up. It is considered vital to constantly work on providing a complete package of convenience and good experience.

Online counseling service description

The online counseling service and the materials available at the website are used to maintain contact with the students in the exchange programs to ensure they have a favorable and fulfilling experience. The objective of the partners is to create the situation when students come back from the time they spent abroad satisfied and with unique experiences which will be helpful for their academic, professional and personal development. The educational program will be the main means for students to develop the competencies to manage the multiple hazards that may be faced. In case the situation calls for, the student may visit the university's counseling center for help. By engaging the returned students as mentors of the students that will be going out, the chances of their experiencing reverse culture shock is reduced and their reintegration into their home environment is easy. Another goal that the partners are pursuing is to build an intercultural student community that would help students to help each other and share information. Ultimately, the project aims at raising the awareness on the fact that mental health should be a topic of discussion and that outgoing students should feel free to disclose any psychological problems they might have, in order to eliminate the stigma that surrounds them. The dissemination of the project's outcomes will be necessary for the implementation of the standardized approach to supporting the departing students across the university of Europe.

Educational program and webinars description

The training program is the main output of the project. The aim of the program is to develop a series of educational activities (presentation workshops, educational and motivational videos, online webinars, articles etc.) that will help the staff of the international departments of European universities to prepare students for going abroad.

Online counselling for students description

Another key output of the project is an online counselling program. The main goal of it is to support students throughout the entire duration of Erasmus+ mobility, from the preparation for departure to reintegration after returning home. The counseling program consists of the series of informational e-mails which are divided into the three parts: before, during and after mobility, and sent to students on the appropriate timeline. E-mails cover useful information about studying abroad, e.g., mobility process, finding accommodation and – most importantly – focus on student psychological preparation and well-being with topics on stress management, cultural adaptation, work-rest balance and many others (that will be the costabex checklist categorizing base). A separate part is dedicated for resources and useful contacts to ensure that students are assisted when further help or information is necessary. The content of each e-mail includes general information and insights collected from international officers, students with mobility experience and psychologists, therefore, can be applied by all higher education institutions.

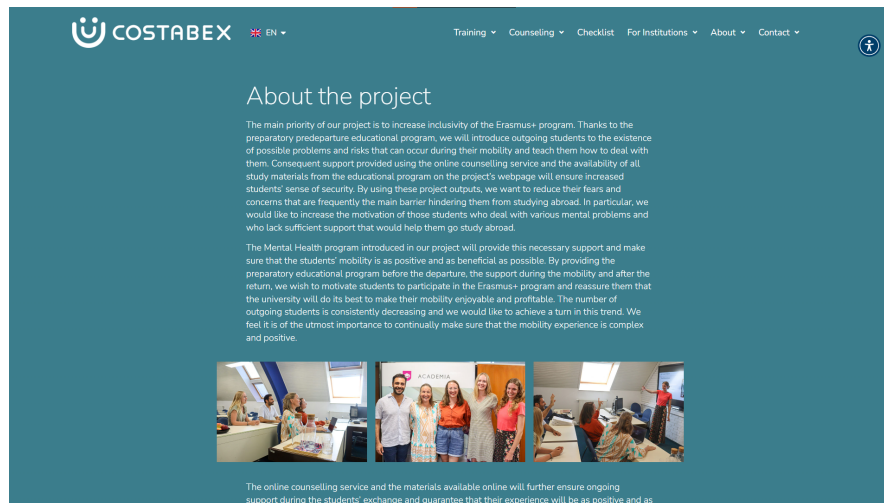


Figure 3.13: Costabex about page

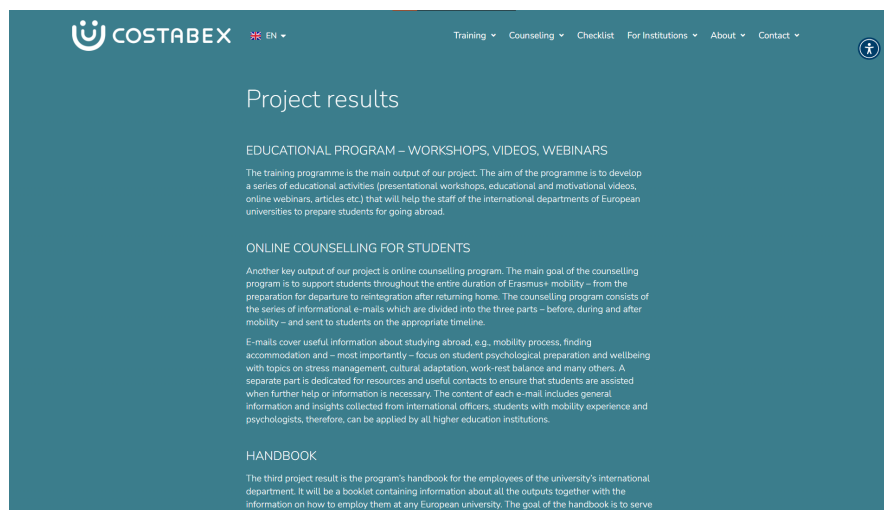


Figure 3.14: Costabex results page

The partners

This page shows the different partners participating in the Costabex project development that are the *Czech Technical University of Prague*, the *Vilnius Gedeminas Technical University*, the *Palacky University Olomouc* and finally *Academia Institute Of Technology*.

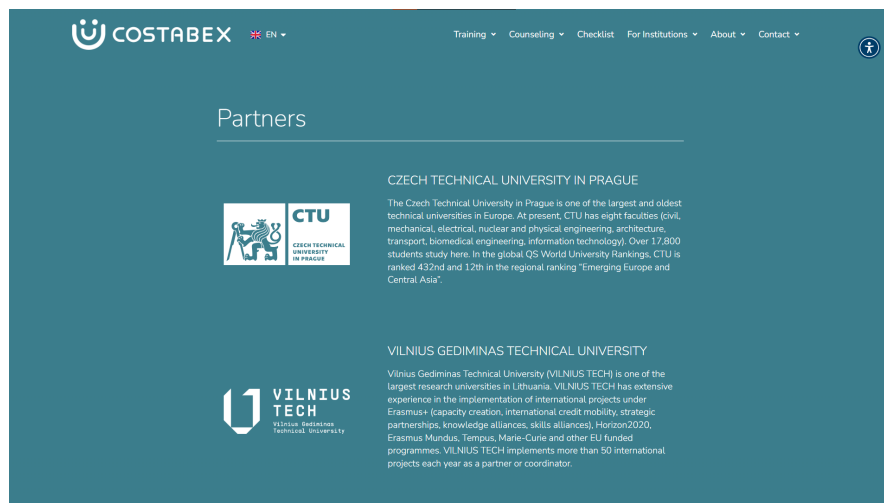


Figure 3.15: Costabex partners



Figure 3.16: Costabex partners

News section

The last page present in the about section is the one related with news about real experiences from other students. Here is possible to read articles regarding students' direct experiences that could be helpful to encourage future ones to take part in the outgoing experience.

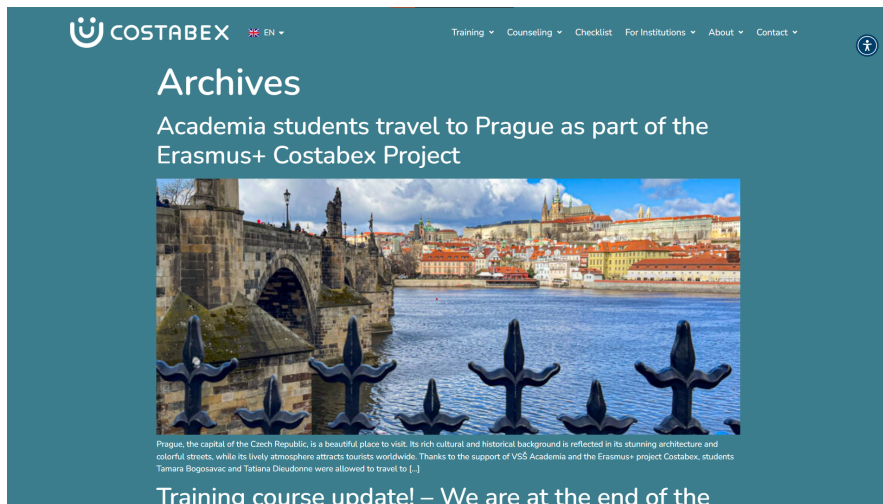


Figure 3.17: Costabex news

Chapter 4

The Costabex checklist

4.1 The state of the art

Nowadays, there are many tools for managing tasks and files, few combine these functions into one platform but no one of them integrate functionalities for managing mobility for Erasmus students or abroad students in general. This lack of integration could be attributed to several reasons:

1. **Fragmented workflow:** most task management and file management tools work separately, which creates a disjointed workflow for users. People have to switch between different apps to manage tasks and access related files, which can be frustrating.
2. **Limited Integration:** although some task management tools can integrate with file storage services like Google Drive or Dropbox, this integration is often basic. They don't provide advanced features or a seamless workflow. In addition to this, no students' tailor solution is present on the market
3. **Lack of Standardization:** there is no standard way for task management and file management tools to communicate with each other. This makes it challenging to create cohesive platforms that integrate both functions effectively.
4. **Different Development Focus:** task management and file management tools are often developed by different teams with different goals. This can lead to a lack of collaboration and understanding between the teams, making it difficult to create integrated solutions.

in the following rows, an initial prototype will be presented, and it could be seen as a first approach for solving these issues.

4.2 Presentation

Considering some of the concepts presented in the previous chapters and keeping in mind the previously listed state-of-the-art assumptions, the Costabex checklist application was designed to seamlessly bridge the communication and workflow gap between students and tutors in the context of an Erasmus exchange program. The app, with its dual interface, caters to the relative needs of both students and tutors, facilitating a streamlined, efficient, and engaging educational experience abroad. Every section of this chapter shows the service's views and features and how it is supposed to work. In the next chapter, a deeper technical explanation will be done for a better comprehension.

4.3 Mock-ups

The first developing stage was characterized by the creation of first mock-ups of the application. Canva was the software used in order to make them. The pages designed were the landing page, register page, login page and the principal dashboard of the app. In this phase the main thought was to create a coherent look and feel, being similar, in color palette and style, to the Costabex website.

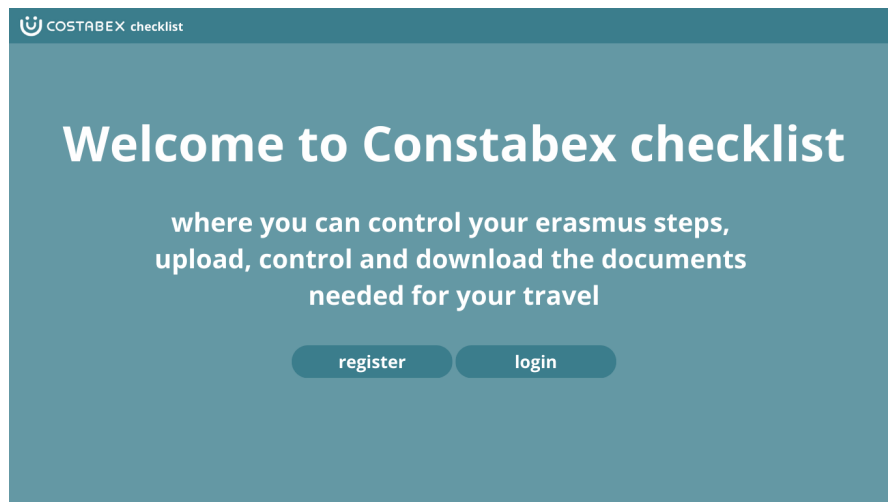
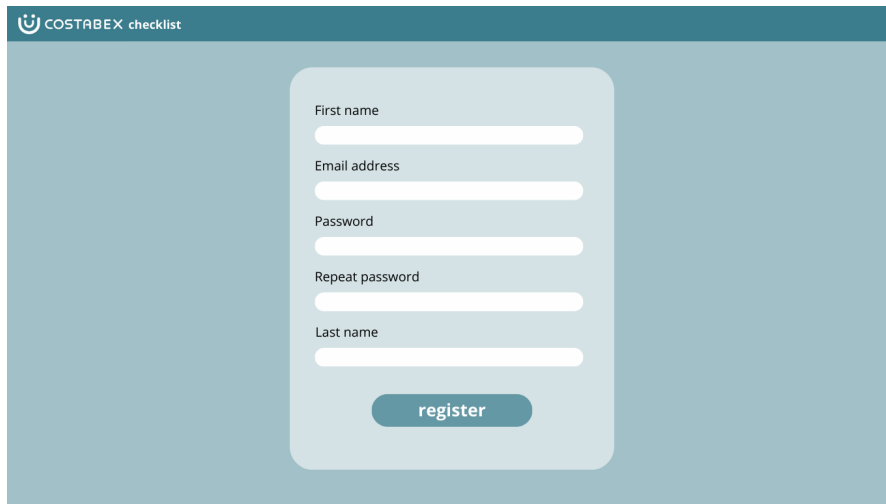
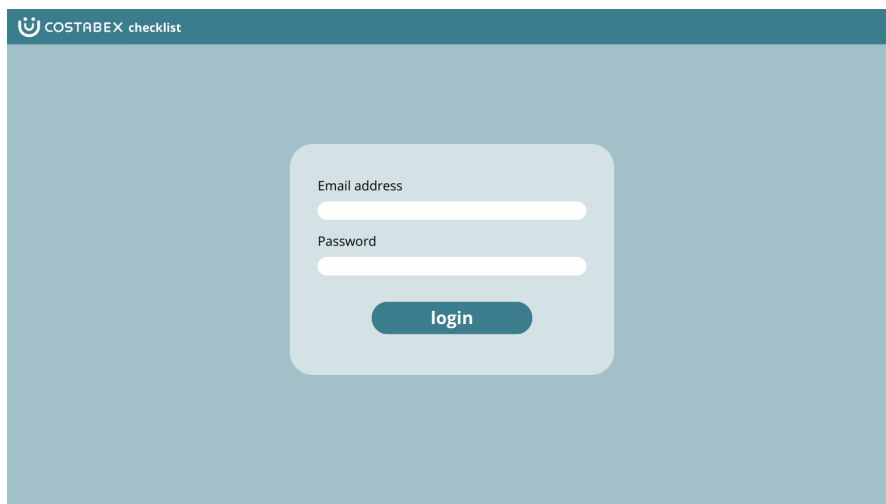


Figure 4.1: Mockup landing page



The mockup register page features a dark teal header with the 'COSTABEX checklist' logo. The main content area is a light teal background with a central white rounded rectangle containing the registration form. The form includes input fields for 'First name', 'Email address', 'Password', 'Repeat password', and 'Last name', each with a white underline. A dark teal 'register' button is positioned at the bottom of the form.

Figure 4.2: Mockup register page



The mockup login page features a dark teal header with the 'COSTABEX checklist' logo. The main content area is a light teal background with a central white rounded rectangle containing the login form. The form includes input fields for 'Email address' and 'Password', each with a white underline. A dark teal 'login' button is positioned at the bottom of the form.

Figure 4.3: Mockup login page

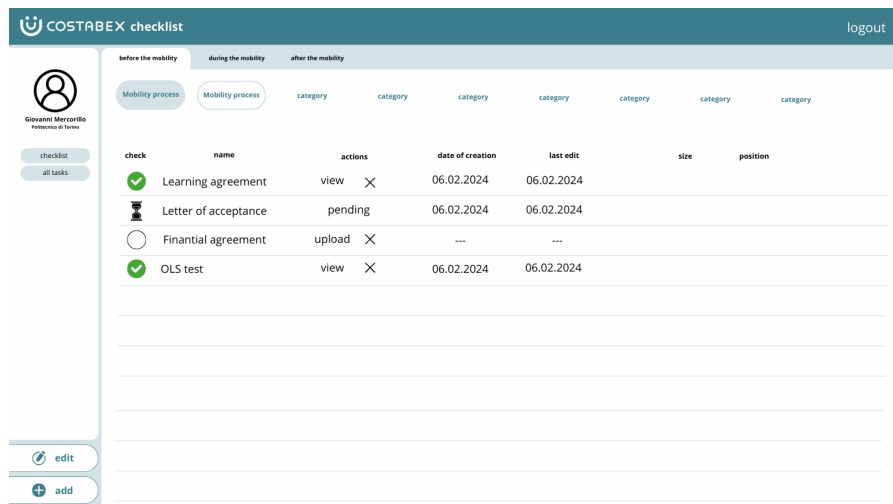


Figure 4.4: Mockup dashboard

4.4 The prototype

4.4.1 Landing page

The landing page is the one used to give to the user a first approach to the app. It is characterized by the nav-bar, present in every page of the app, that, in this page, has just the Costabex logo on the left side. Additional features in this page are the app presentation with *Welcome to the Costabex checklist* text and a brief description of the main scope of it. Finally, the login and register link buttons.

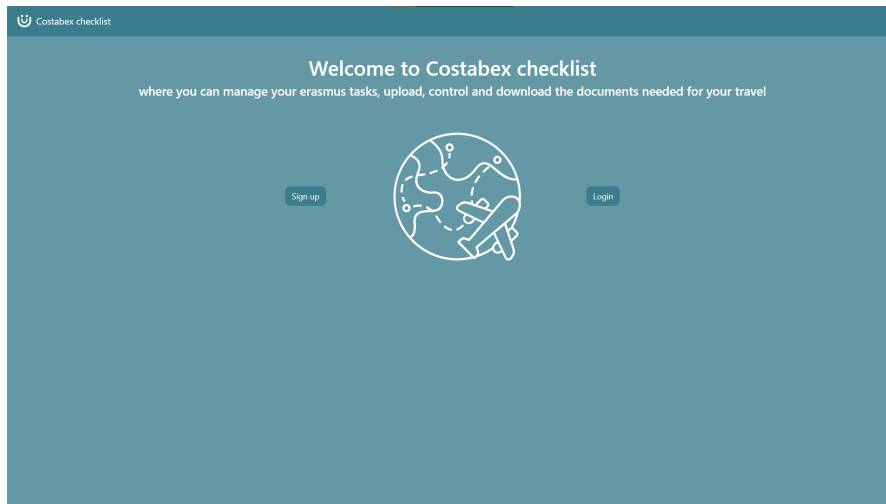


Figure 4.5: Landing page

4.4.2 Registration and login page

Registration page

The register page is the one used to add a new user to the service database. The information requested here are the user first name, last name, institution, username, mail, password and the user role: student or tutor. In case of tutor role, an additional input field appears to insert the tutor key word to verify if the user is actually a tutor.

The screenshot shows a registration form titled "Register" on a light blue background. The form includes the following fields: "First name", "Last name", "University", "Username", "Email", "Password" (with a strength indicator), "Role" (a dropdown menu with "Tutor" selected), and "Tutor Key" (with a strength indicator). A "Register" button is located at the bottom of the form. The top navigation bar contains the "Costabex checklist" logo on the left and a "login" link on the right.

Figure 4.6: Sign up page

Login page

The login page it's obviously the one through which the user can access to its own dashboard with username and password input fields to fill in order to be done.

The screenshot shows a login form titled "Login" on a light blue background. The form includes two fields: "Username" and "Password" (with a strength indicator). A "Login" button is located at the bottom of the form. The top navigation bar contains the "Costabex checklist" logo on the left and a "Register" link on the right.

Figure 4.7: Login page

4.4.3 Dashboard page

The dashboard is the app's core. it is formed by 2 main sections: the current user information on the left side of the page and the task section on the right one. In all pages related to an active login session the nav-bar has, in addition to the Costabex logo, three buttons: a list shaped button that redirects to the dashboard, an account button that redirect to the account info page and a logout button that close the session. The overall dashboard page is similar for both kind of users role with little differences presented below.

For the student

Account section The account information left side of the dashboard presents user principal information related to account picture (which click redirects to the account info page), first name, last name and institution. Under these infos there are three buttons: the *checklist* button that allow the categorized listing of the tasks basing it to the mobility periods and different topics related (equal to periods and topics subdivision of the counseling page in the Costabex counseling section), the *all tasks* button that allows to see all the tasks related to the user and finally the *add task* button that redirect to the add task page.

Tasks section the dashboard's tasks section is where the tasks are displayed. If the checklist button is clicked, the top part of it is constituted by three buttons indicating the mobility periods and the specific categories related to each period below. In this case, just tasks linked with that specific period and category will be displayed. As mentioned before, in case of *all tasks* button pressed, the whole tasks related to that user will be displayed. The last part of this section presents, of course, the task listing table with different information related to it: task status that could be pending or completed, task name, period, category, creation date, deadline and additional button for editing or deleting the task, enabled in case of own tasks, disabled in case of tutor assigned tasks (that will be distinguished by personal ones with a different background-color. Furthermore, every table's column allow the user to sort the tasks in an ascending or descending manner basing the order on every t-head present (status, name, category, period, creation date and deadline).

The Costabex checklist

Checked Status	Name	Period	Category	Author role	Creation Date	Deadline	Action	Delete
✓ completed	Team collaboration tasks	during	what could go wrong	student	17/02/2024	13/05/2024		
✓ completed	Research project	during	stress management	student	21/03/2024	31/03/2024		
✓ completed	Research project	during	stress management	student	26/02/2024	09/06/2024		
⏸ pending	Professional development workshops	after	stress management	student	28/12/2023	08/08/2025		
⏸ pending	Professional development workshops	after	stress management	student	24/12/2023	09/06/2024		
✓ completed	Presentations or workshops	during	it's packing time	student	25/09/2023	24/11/2025		
✓ completed	Presentations or workshops	during	it's packing time	student	18/11/2023	01/07/2024		
✓ completed	Orientation session	before	formalities	student	01/01/2024	17/04/2025		
✓ completed	Orientation session	before	formalities	student	04/01/2024	08/06/2024		
⏸ pending	NEW TUTOR TASK	before	mobility process	tutor	18/03/2024	26/04/2024		
✓ completed	new task test	before	mobility process	student	24/03/2024	Not specified		
⏸ pending	Networking with professionals	during	students helping students	student	13/03/2024	18/11/2025		
⏸ pending	Networking with professionals	during	students helping students	student	31/10/2023	14/07/2024		
✓ completed	Networking events	during	students helping students	student	08/09/2023	18/08/2024		
✓ completed	Networking events	during	students helping students	student	22/10/2023	12/04/2024		
✓ completed	Learning agreement	before	mobility process	student	15/09/2023	05/03/2025		
✓ completed	Learning agreement	before	mobility process	student	15/10/2023	25/04/2024		
🔄 refresh	Lancuase courses	durina	study advice	student	24/03/2024	>8/10/2025		

Figure 4.8: Student dashboard

For the tutor

When the current logged user is a tutor, the dashboard is essentially the same with the only difference that he can't see students' personal tasks but just his own ones and the ones assigned to students, which he obviously can edit or delete.

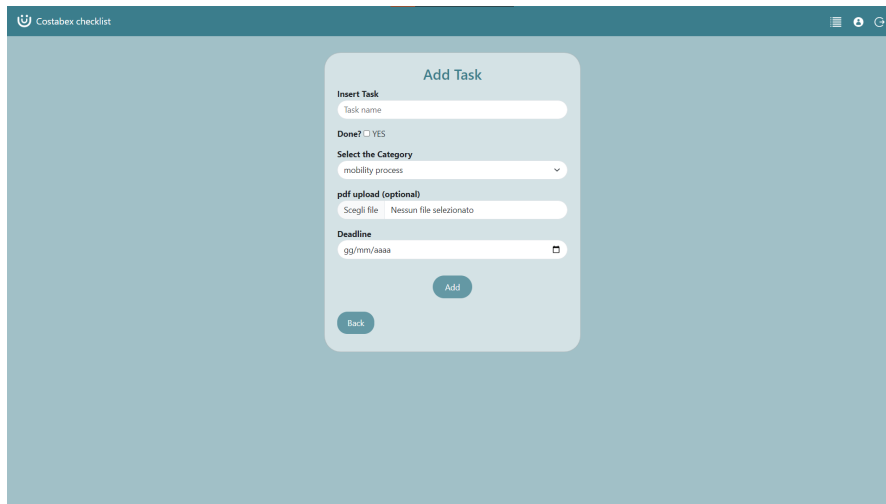
Checked Status	Name	Period	Category	Author role	Creation Date	Deadline	Action	Delete
⏸ pending	Thesis pdf	during	study advice	tutor	24/03/2024	28/03/2024		

Figure 4.9: Tutor dashboard

4.4.4 Add task page

For the student

When the user click the add task button, present in the dashboard' account section, it will redirect to the add task page. In this page, a task related form will be shown that allows the student to set different data: task name, task status, category related to it (that automatically set the mobility period too), a PDF file input and a deadline. The creation date will be automatically registered once the user will click the add task submit button.



The screenshot shows a web browser window with the title 'Costabex checklist'. The main content is a light blue modal form titled 'Add Task'. The form contains the following fields and controls:

- Insert Task**: A text input field labeled 'Task name'.
- Done?**: A checkbox labeled 'YES'.
- Select the Category**: A dropdown menu with 'mobility process' selected.
- pdf upload (optional)**: A file selection input field with the text 'Scegli file' and 'Nessun file selezionato'.
- Deadline**: A date input field with the format 'gg/mm/aaaa' and a calendar icon.
- Buttons**: A blue 'Add' button at the bottom right and a blue 'Back' button at the bottom left.

Figure 4.10: Add task - student

For the tutor

In case of a user with tutor as a role, the page it's mainly the same with the possibility to assign the task to a specific student or keep it as a personal one selecting the *Not Assigned* dropdown's option.

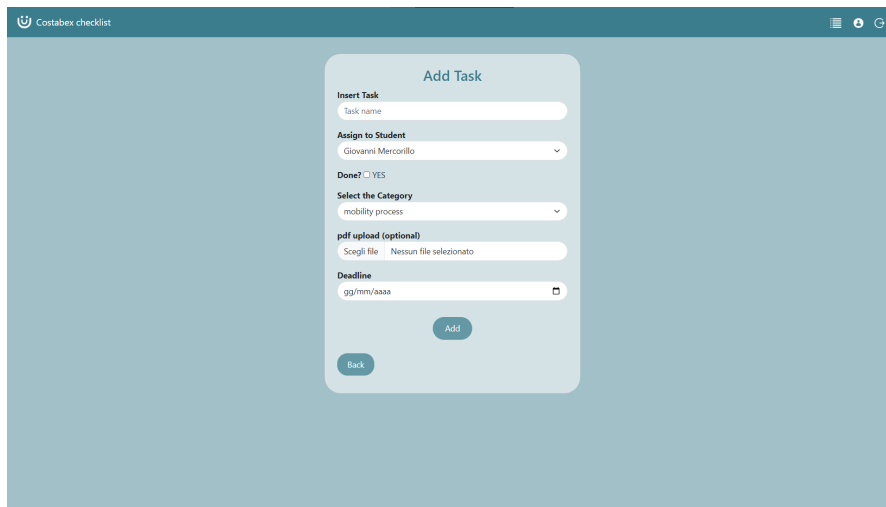


Figure 4.11: Add task - tutor

4.4.5 Show task page

For the student

The show task page is the one used to read the specific task data. Here there is a *preview file* button enabled or disabled depending on the file's presence. In case of own made task, the student will see two delete buttons: for the entire task and to remove the file from it. Another button is the *edit task* one that redirects to the edit task page. In case of tutor assigned task, the student can't delete or edit the task but just upload a file related to it and only if the task file is *not approved* or countersigned as *done* by the tutor.

For the tutor

In case of tutor user, the show page will be exactly the same but without the students' limitations related to task assigned tasks previously described.

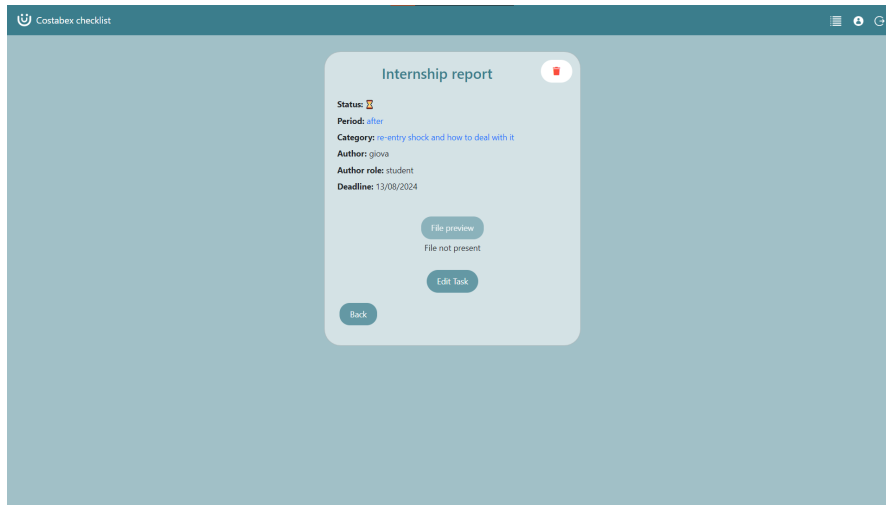


Figure 4.12: Show task - student

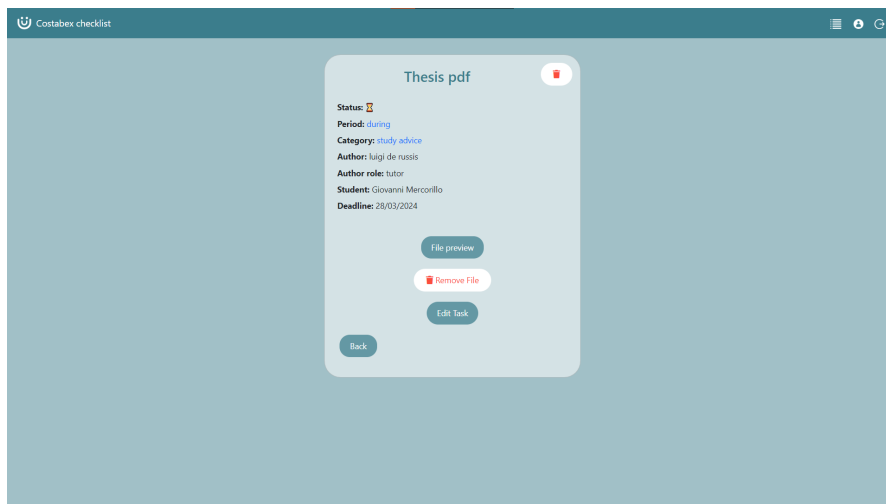


Figure 4.13: Show task - tutor

4.4.6 Edit task page

For the student

The edit task page is the one where it is possible to edit task data (except for the task name). The user can change data related to the task and upload a new file. In case of tutor assigned task, only this last data could be changed, and a notification will consequently appear in tutor's dashboard.

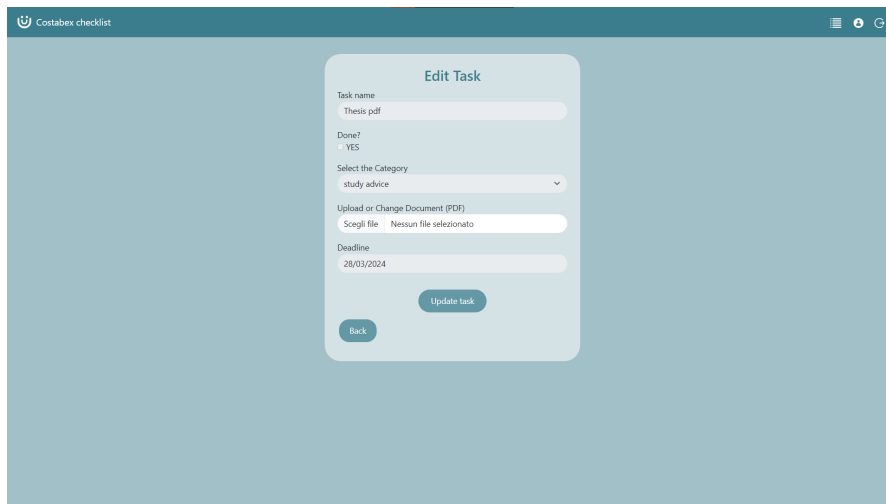


Figure 4.14: Edit task - student - tutor assigned task

For the tutor

In case of tutor user, the edit page will be exactly the same but without the students' limitations related to task assigned tasks previously described.

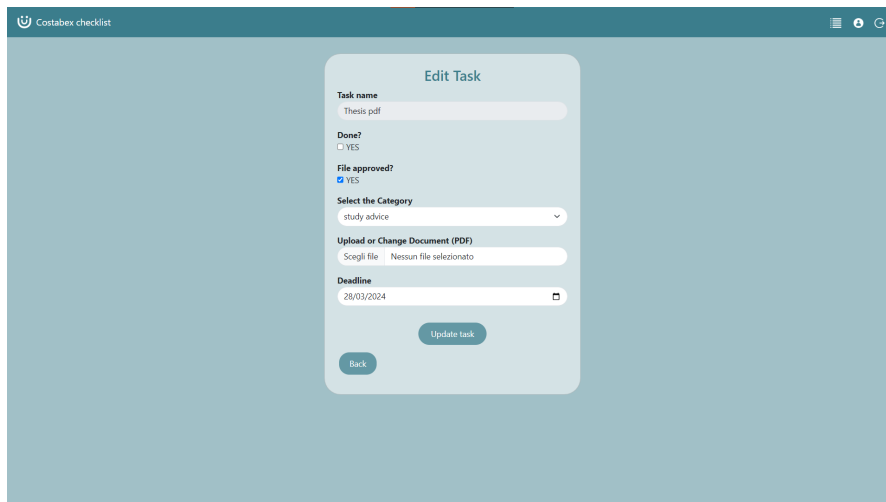


Figure 4.15: Edit task - tutor

4.4.7 Account page

The account page is the one where the user can view its own info and update his profile picture. It is accessible from the nav-bar button or clicking on the profile picture in the dashboard page. This page is exactly the same for both student and tutor.

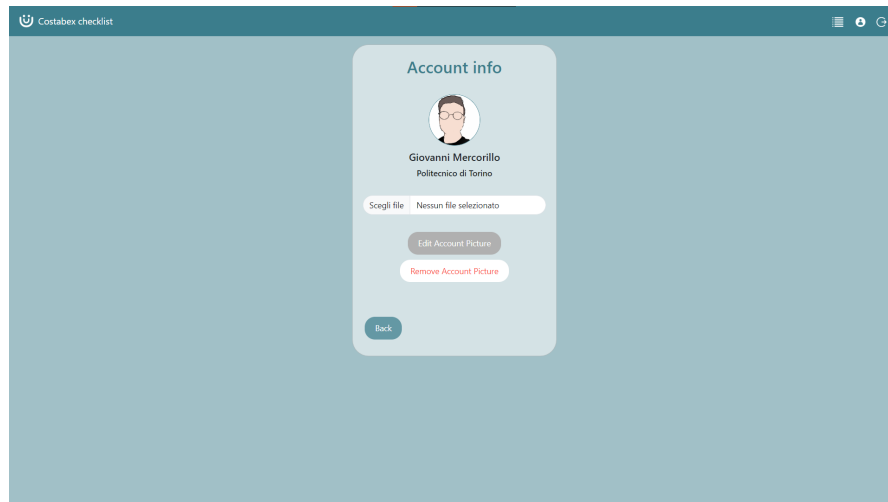


Figure 4.16: Account page

4.4.8 Error page

In the entire web-app, Error handling is properly settled in both client-side and server-side way. In the next chapter will be explained in which way it was done.

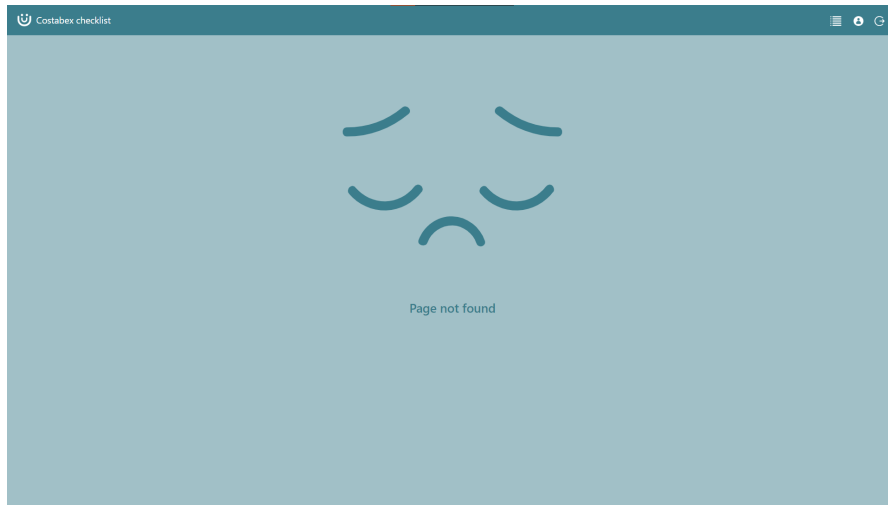


Figure 4.17: Error page

4.5 Future developments

The presented prototype is just the basic version of what will be the complete web-app service. Future implementations to add will be mainly related with the notification system and mail management. Right now, the notifications are just dropped and displayed as flash messages to the students and tutors. When a file is uploaded by a student to the tutor, to check the task and approve the uploaded file or not. In the opposite way, a student is notified when a task is assigned by a tutor to him. Future development will be related with the optimization of this notification system with mail usage integration. A complete app overview guide will be even added to the landing page to give the user thorough details on how to use the service. Another step will regard the implementation of a mobile app version, even if the entire application is already completely responsive to accomplish the actual browser standard.

Chapter 5

Implementation

5.1 Introduction

In this chapter, the Costabex checklist implementation is presented. Before doing the web-app code overview, all the features, tools and packages used are listed to give a global introduction and facilitate the subsequent explanation when they will be cited.

5.2 Front-end development and tools

Creating a compelling and user-friendly front-end for a web application, such as a platform facilitating interactions between students and tutors in an Erasmus exchange program, requires a deep dive into the synergies between styling frameworks, templating engines, and the organization of the application's structure. The main tools used are now explained:

5.2.1 CSS and flexbox

The abbreviations for CSS are Cascading Style Sheets. It is a markup language, which in most cases is used with HTML and other markup languages (such as EJS for the app) for the purpose of defining the web pages' visual appearance, as well as the style and format. CSS serves as a tool used to manipulate elements and to have control over their fonts, colors, spacing, layout, and any other thing that could affect the design's look. Its aim is to have the same effect across various browsers and screens. It basically mandates website selection of HTML elements and apply styles to them. This can be done within an HTML document or through an external CSS file so that the content remains separated from its stylesheet.

Flexbox or Flexible Box Layout is a CSS designed feature for exhibiting well-built and more flexible web pages layouts. It lets to apply containers accordingly to any elements that'll be arranged inside, even if these might be different in terms of size or order. With flexbox, it's easier to drive the draping, ordering and partition elements along one axis (either horizontal or vertical) or along both the axes in a single go. In summary, Flexbox is basically an awesome helper for creating responsive designs and complex layouts with less code, saving time and efforts.

5.2.2 Bootstrap

Bootstrap is a prevalent front-end framework that can be used to make websites or apps that are responsive and mobile adaptive. Created by Twitter's developers, it is an open source framework that provides a library consisting of predefined HTML, CSS, and JavaScript components such as buttons, forms, navigation bars, and grids, to make fast prototyping and integration possible. Bootstrap's grid system empowers developers to make flexible and responsive designs which are adapted to diverse screen sizes and various devices. Furthermore, Bootstrap features various template themes, abundant documentation, and a big community which makes it user-friendly and a commonly preferred tool for web developers.

5.2.3 EJS and Partial:

EJS is another templating language for JavaScript that helps to easily generate dynamic contents within web applications. It features a code to which the JavaScript scripts can be directly embed inside an HTML-like template, which, in turn, allows the creation of reusable components and dynamic views. Thus, it is possible to directly inject values from JS files by only writing simple codes inside it and print out the data on the web pages. Partial are the reusable components or sections of a web page that can be embedded into another page or view. Partial allow diminishing repetition and work more effectively, which is made possible by the feature of defining a component once and then apply it as many times as needed. In the EJS, partials are usually implemented by using the '`<%- include('partial-name') %>`' syntax; 'partial-name' specifies the file that contains the code for the partial part. Developers, thus, are able to break the code into smaller parts and then combines them together to create reusable components. This helps in creating a more structured and manageable web applications.

5.2.4 Views Folder with Layouts and Public Folder

The organization of a web application's front-end architecture often includes a 'views' folder and a 'public' folder. The 'views' folder typically contains the application's

EJS templates, including layouts and partials. Layouts define a base template for the application (e.g., a template that includes the header and footer partials), which can be extended or overridden by specific views to alter the content displayed within the common layout structure. This setup enables a consistent look across different pages while allowing content variation. The 'public' folder is used to store static files that need to be directly accessible by clients, such as CSS files, JavaScript files, and images. These files are referenced in the EJS templates and are crucial for the styling and interactivity of the client-side interface. The separation of static files into the 'public' folder helps in organizing the application's structure, making it easier to manage the resources needed for the front-end.

5.2.5 Flash package

The Flash package provides a way to send temporary messages between requests. These messages, often called flash messages, are typically used to display feedback or notifications to users after certain actions, such as form submissions or authentication events. Flash messages are stored in the session and are available only for the next request, making them ideal for displaying one-time messages. The Flash package, thus, simplifies the process of managing applications' feedback by providing middleware (lately explained) for setting, retrieving, and clearing messages.

5.3 Back-end development and tools

5.3.1 JavaScript

JavaScript is a high level programming language that is mostly used for designing dynamic and interactive logic into web pages. Originally created for browser-based scripts, JavaScript has recently expanded to the multipurpose client and server-side development, mobile apps, game, and so on. Previously, most use of JavaScript was to modify the DOM of the web page, in order to change HTML content, style elements, react to users' interactions, and do a lot of other things right when an event occurred. JavaScript is also frequently used for filling out online forms, animation, server data retrieval (AJAX and AJAJ), and client-side code within web applications. The multi-paradigms feature of the JavaScript code is well-known because it supports functional as well as object-oriented programming paradigms. It is an ecosystem of libraries and frameworks, which include React, Angular, and Vue.js to ensure that it covers the needs of a developer in making web development simpler and easier. Generally, JavaScript is a key circle in modern web application development, which provides an opportunity for programmers to make websites with a large variety of functions that run smoothly on any computers and devices.

5.3.2 DOM, AJAX, AJAJ

The DOM (Document Object Model) is a structure that enables interactions with web documents via programming. It represents the parts of the document which include elements, attributes, as well as text. It may be arranged in a tree-like structure resembling HTML and XML documents. The DOM enables how the programs can virtually supply, access to and change the content, structure, and style of webpages. Through JavaScript, developers can access the user's DOM, which gives them the ability to make websites more dynamic and interactive by adding/removing or changing elements, styles, as well as react to a user's action. AJAX (Asynchronous JS and XML) is a website development method employed for the purpose of non-blocking requests to a web server to get and send data. It permits more responsive and interactive web apps and information could be exchanged via the browser and the server in a hidden manner. It is a process where JavaScript is used without obstructing the user's interaction with the webpage sending asynchronous requests to the server, receive the server's response and update the webpage dynamically. AJAJ is similar to AJAX but use JSON (JavaScript Object Notation) files to redefine data communication between the client and the server instead of simple XML, this upgrade was made because it is lighter and easier to parse. JSON is a compact and simple data format that can easily be utilized, and this is one of the key reasons why it is preferred for web browser and server communication with structured data.

5.3.3 Node.js

Node.js is a run time environment written in JavaScript which enables the back-end developers to run the code independently of the browser. It is commonly used for its event handling capability, in which process is asynchronous and makes it the perfect choice for implementing various types of web applications such as real-time ones, APIs, microservices, and server-side applications in general. Among the great features of Node.js, there's a package system called NPM (Node Package Manager) which makes all open-source packages (including plugins) to be easily integrate into Node.js based applications. This comprehensive ecosystem combined with Node.js is based on strong features: flexibility, high performance and option to build a wide variety of web and server applications that made it highly popular among developers. Thus, Node.js gives developers an opportunity to use JavaScript everywhere for developing very fast, scalable and efficient back-end code thanks to the language's familiarity and flexibility. Every package present in the Costabex checklist was installed through NPM package manager.

5.3.4 Express, Express Router and Session

Express is a minimal and flexible node.js framework, responsible for improving developer's web application development experience and faster production time. It is a robust API offering different features including handling requests and responses, routing, and middlewares integration and, at the same time, is pretty simple to use. Among the members of the Node.js community, Express is one of the most used package, and is recognized for its high speed, flexibility, and scalability. Express Router is the inbuilt routes service of the Express framework, through which developers can split their program's routes and endpoints into separate modules. It solves the clustering issue where the routes for an old application would otherwise be unorganized and scattered across multiple big files or modules, eventually making the codebase unwieldy and not easy to maintain. Developers can use Express Router to define route handlers, responding to different HTTP methods, such as GET, POST, PUT, DELETE. It is important to mount them to certain URL paths, which will enable better organization and separation of concerns implementation in Express applications. Express Session is a service that helps with user session handling for applications that use Express as the basis for web server development. Sessions allow such users to save data specifically related to them for longer time than what the browser can handle and across multiple interactions with a server too. Express Session enables formation and control of them, ensuring session data's persistence on the server in association with the session identifier (usually client-side cookie). This pairs with capabilities such as user authentication, authorization, and personalized user experiences for web applications. In addition to this, it enables different session storage options, including internal memory sessions, database back-end sessions, and even external session stores, like Redis or MongoDB, making Express applications more flexible and scalable when managing several users.

5.3.5 MongoDB

MongoDB is an open-source NoSQL database with a focus on efficiency, scalability, and high performance. It stores data in BSON format that resembles JSON. In contrast to the relational databases that use tables and rows, the MongoDB is built on collections and documents. This document-oriented approach allows diverse data structures and nested data models that can represent different data types and structures. MongoDB has been designed to be highly available and scalable, with replication and sharding being inherent in its architecture. It is commonly found in the present-day web applications, big data solutions and in cases where the development process is rapid, and the data schema is flexible.

5.3.6 Mongoose

Mongoose is an ODM (Object Data Modeling) library for MongoDB and Node.js. It is a solution based on a schema that used to model application data. It comes with inbuilt type casting, validation, query building, business logic hooks, and so on. Mongoose enables the whole process of defining schemas which correspond to specific MongoDB documents. These schemas are the blueprints upon which models are built and represent documents in a MongoDB collection. Mongoose models offer the chance of writing functions that perform actions like creating, editing, retrieving, and deleting records.

5.3.7 Middlewares and error handling

The middleware function is a function that access the request object (req), execute and then send the response object (res), then call the next function that is involved in the application. Tasks such as logging, authentication, and parsing of data can be performed by these functions. They behold the ability to call the next middleware function in the stack or end the request-response cycle. Consequently, they constitute a strong principle to provide more functionality to the Express framework in a way that can be logically arranged and reusable. Express Error Handling support is an implemented mechanism for handling errors during requests handling. The function syntax for errors is a middleware function with the adding error parameter within the syntax (err, req, res, next). This kind of middleware play the role of handling the error and sending the right response to the client.

5.3.8 Passport authentication

Passport is a package suite for authentication in Node.js apps, particularly effective with Express integration. Passport allows the state of user authentication to be included and restored during web requests, which has the effect of enabling a smooth user experience. It uses its middleware functions for authentication purposes where, for instance, the only authenticated users pass the routes whereas the unauthenticated ones can be redirected to the login pages or a response with an error can be returned. Passport is immensely valuable because it has modularity and extensibility, which gives programmers an opportunity to have a simple and seamless authentication without requiring any huge efforts. This makes Passport a good choice as well as its documentation and community.

5.3.9 Authorization

Permission middlewares act as services that monitor incoming request, made to the protected parts of the website, checking if the user is permitted to get a

specific page. The middleware functions, at the same time, tailor solutions for database management or permissions into account for giving or banning access. Authorization middleware are generally embedded into express routes and authorize users to access the sensitive information according to predefined rules defined in the application.

5.3.10 Joi schema validation

Joi is a powerful validation JavaScript library. It is primarily used for the validation and sanitation of user inputs in Node.js applications. This removes the need for writing fancy and complex validation rules, since the field types are programmatically declared. Developers can define their data structure, constraints, and verification rules which include required fields, data types, existing values, strings' length, regular expressions, and many more. With Joi, developers can simply do validation to various data types, providing a bundle of built-in methods and functions basing on different need. These can be 'string()', 'number()', 'required()', 'min()', 'max()', 'valid()', 'email()', 'regex()', and others. Besides using complex data structures, nested schemas, custom validation functions and error handling, it is a one solution that can cover all data validation in Node.js applications. In short, Joi ensure that all data coming in satisfies given rules without risks of security loopholes, data accuracy problems, and programming errors.

5.3.11 Cloudinary

Cloudinary is a cloud-based media platform which allows users to store, manage, and serve images, visuals, and videos for their app and websites. It is feature-rich and allows uploading files smoothly by offering image and video pulling, storage, transformation, optimization, and delivery through content delivery network (CDN). With APIs and SDKs, Cloudinary facilitates developers in adding media management functions to their applications, performing dynamic growing, cropping, compression, and transformation of data that have to be modified instantly. Moreover, it grants many advanced options such as image format converting and compression for responsive generation, being reliable and persistent in multimedia data management.

5.4 Code overview

This part of the chapter explains how previous tools were involved and used and gives a complete overview of the final product.

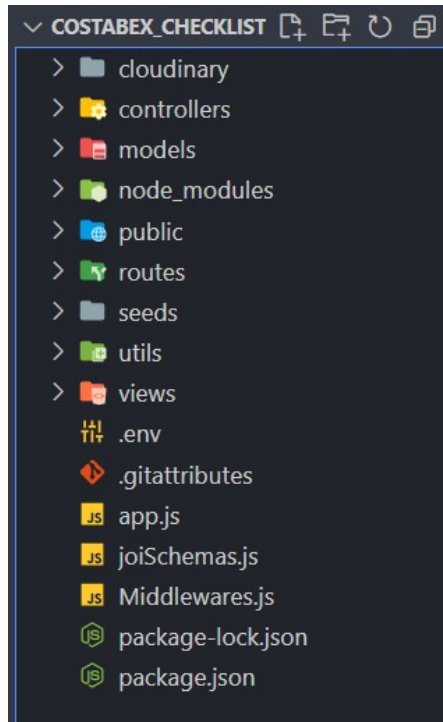


Figure 5.1: Prototype's files

5.4.1 App.js file

This file it's the prototype core, it sets the web application using Node.js, Express, and other libraries to provide functionalities such as environment configuring, database connection, session management, authentication, and error handling. Its main elements are:

- **environment configuration:** adds a '.env' file that is loaded by the application if it is not in the production mode, and thus the development and production settings are separated.
- **express application setup:** instantiates an Express app, configuring various middleware such as handling HTTP requests (body parsing, static files serving), bypassing method overrides, and setting up template engines EJS and ejs-mate, used to extend the templating features.

- **database connection:** defines the connection to a MongoDB database through Mongoose, which is responsible for modeling and managing application data.
- **session and Flash messages:** sets ‘express-session’, used for session management, flash messages are implemented by ‘connect-flash’ and they are used for storing data (for example, error messages or form inputs) across requests.
- **authentication:** passport is the module requested for passport authentication and configuring session with local strategy and session support. Incorporates users’ serialization/deserialization to keep a track of sessions.
- **routes:** provides the routes to the tasks and user operations, which keeps the concerns separate and makes the codebase more modular and manageable. It provides a landing page route and applies middleware for the given settings (for example, current user and flash messages) to all the requests.
- **error handling:** it sets a catch-all route for unmatched paths and if no match is found, it returns an HTTP status code of 404 and has an error-handling middleware to format and display the errors using a dedicated template.
- **server initialization:** through the listen port (app.listen), the application server runs and listens for incoming requests.

app.js

```
1 | if (process.env.NODE_ENV !== "production") {
2 |   require("dotenv").config();
3 | }
4 | const express = require("express");
5 | const app = express();
6 | const path = require("path");
7 | const methodOverride = require("method-override");
8 | const ejsMate = require("ejs-mate");
9 | const catchAsync = require("./utils/catchAsync");
10 | const ExpressError = require("./utils/ExpressError");
11 | // mongoose
12 | const mongoose = require("mongoose");
13 | main().catch((err) => console.log(err));
14 | async function main() {
15 |   await mongoose.connect("mongodb://127.0.0.1:27017/costabex-checklist");
16 | }
17 | //add session
18 | const session = require("express-session");
19 | const sessionConfig = {
20 |   secret: "segreto",
21 |   resave: false,
22 |   saveUninitialized: true,
23 |   cookie: {
24 |     httpOnly: true, //basic security
25 |     expires: Date.now() + 1000 * 60 * 60 * 24 * 7,
26 |     maxAge: 1000 * 60 * 60 * 24 * 7,
27 |   },
28 | };
29 | //add flash
30 | const flash = require("connect-flash");
31 | //add passport
32 | const passport = require("passport");
33 | const LocalStrategy = require("passport-local");
34 | // user model
35 | const User = require("./models/user");
36 | // Task routes
37 | const tasksRoutes = require("./routes/task.js");
38 | // User routes
39 | const userRoutes = require("./routes/users");
40 |
41 | // session and flash
42 | app.use(session(sessionConfig));
43 | app.use(flash());
44 |
45 | // flash + path middleware
46 | app.use((req, res, next) => {
47 |   res.locals.currentRoute = req.path;
48 |   res.locals.success = req.flash("success");
49 |   res.locals.error = req.flash("error");
50 |   next();
51 | });
52 |
53 | // Set and use
54 | app.set("view engine", "ejs"); //ejs-mate call for express
55 | app.set("views", path.join(__dirname, "views"));
56 | app.use(express.urlencoded({ extended: true }));
57 | app.use(express.static(path.join(__dirname, "public")));
```



```

58 | app.use(methodOverride("_method"));
59 | app.engine("ejs", ejsMate);
60 |
61 | // passport config
62 | app.use(passport.initialize());
63 | app.use(passport.session());
64 | passport.use(new LocalStrategy(User.authenticate()));
65 | passport.serializeUser(User.serializeUser());
66 | passport.deserializeUser(User.deserializeUser());
67 | // passport middleware to pass the user
68 | app.use((req, res, next) => {
69 |     res.locals.currentUser = req.user;
70 |     next();
71 | });
72 |
73 | // TASKS ROUTES
74 | app.use("/tasks", tasksRoutes);
75 |
76 | // USER ROUTES
77 | app.use("/", userRoutes);
78 |
79 | // Landing page
80 | app.get("/", (req, res) => {
81 |     console.log("Main page");
82 |     // Log out the user if there's an active session
83 |     if (req.isAuthenticated()) {
84 |         req.logout(() => {});
85 |     }
86 |     res.render("home");
87 | });
88 |
89 | //-----
90 |
91 | app.all("*", (req, res, next) => {
92 |     next(new ExpressError("Page not found", 404));
93 |     // res.redirect("/error") //send error to the console
94 | });
95 |
96 | // error middleware
97 | app.use((err, req, res, next) => {
98 |     const { statusCode = 500 } = err;
99 |     if (!err.message) err.message = "Something went wrong";
100 |     res.status(statusCode).render("errors", { err }); //error send to the errors.ejs
template
101 | });
102 |
103 | // Route: listening
104 | //-----
105 | app.listen(3000, () => {
106 |     console.log("listening on Costabex-checklist port");
107 | });
108 |

```

5.4.2 Models

Schemas and models in a web application are used for defining the structure and behavior of data. Schemas establish the blueprint for data entities by specifying their properties and validation rules, ensuring data integrity. Models, on the other hand, represent these schemas in code and provide methods for interacting with the database. They handle operations like querying, inserting, updating, and deleting data, serving as the bridge between the application's logic and the database. Overall, schemas and models are essential components for organizing and managing data effectively.

Task model

the task model is the one used for task management, and it is formed by different features like name, period, category and others. This file defines a schema and model for a *Task* entity using Mongoose. The purpose of this model is to structure and manipulate tasks within the database. Its components are:

- **dependencies:** for importing the necessary libraries (Mongoose, Joi, ...).
- **categories array:** defines a list of categories ('categories') under which tasks can be categorized. These categories cover a range of topics relevant to students, such as mobility, formalities, finance, and personal development, among others, the same as the Costabex website counselling part.
- **task Schema:** a TaskSchema is defined using mongoose's 'Schema' constructor. This schema outlines the structure of a task document in the MongoDB database, including fields for the task's name, status (checked or not), period (before, during, or mobility), category (which must be one of the predefined categories), author, student, document details (URL, filename, approval status), creation date, and deadline.
- **name:** a string that is required for each task.
- **checked:** a boolean indicating whether the task has been completed.
- **period:** a string that must be one of three values: *before*, *during*, or *after*, indicating when the task should be performed.
- **category:** a string that must match one of the predefined categories listed in the 'categories' array.
- **author and student:** both are ObjectIds linking to a *User* model, indicating the creator of the task and the student assigned to it, respectively.

- **document:** an object containing details about an associated document (PDF format in the app), including its URL, filename, and whether it has been approved.
- **createdAt:** a date indicating when the task was created, defaulting to the current time.
- **deadline:** a date indicating the task's deadline, with a far future default value as a placeholder of "Not specified" for tasks without a specific deadline
- **model export:** the schema is compiled into a model with the name *Task*, which is then exported. This model is used throughout the application to interact with the 'tasks' collection in the database, allowing for the creation, querying, updating, and deletion of task according to the defined schema. This file essentially sets up a structured way to manage tasks in MongoDB, with a focus on enforcing consistency and providing a convenient interface for data manipulation.

models\task.js

```
1  const { bool } = require("joi");
2  const mongoose = require("mongoose");
3  const Schema = mongoose.Schema;
4
5  const TaskSchema = new Schema({
6    name: {
7      type: String,
8      required: true,
9    },
10   checked: {
11     type: Boolean,
12     required: true,
13   },
14   period: {
15     type: String,
16     enum: ["before", "during", "after"],
17   },
18   category: {
19     type: String,
20     enum: categories,
21   },
22   author: {
23     type: Schema.Types.ObjectId,
24     ref: "User", // reference to the model
25   },
26   student: {
27     type: Schema.Types.ObjectId,
28     ref: "User", // reference to the model
29   },
30   document: {
31     url: {
32       type: String,
33       default: "none",
34     },
35     filename: {
36       type: String,
37       default: "none",
38     },
39     approved: {
40       type: Boolean,
41       default: false
42     }
43   },
44   createdAt: {
45     type: Date,
46     default: Date.now, // Set default value to current date
47   },
48   deadline: {
49     type: Date,
50     default: new Date("9999-12-31T23:59:59Z"), // Setting the default value to the maximum
51   }
52   date
53 });
54
55 module.exports = mongoose.model("Task", TaskSchema);
56
```

User model

the user model is the one used for user management, and it is formed by different features like first name, last name, mail, password and others. This file defines a schema and model for a *User* entity using mongoose. It also utilizes ‘passport-local-mongoose’ for adding username and password fields to the schema and facilitating user authentication. Its components and functionalities are:

- **dependencies:** imports necessary libraries such as mongoose for database interaction, ‘passport-local-mongoose’ for simplifying user authentication with Passport.js, and ‘Schema’ from mongoose.
- **user schema:** the ‘UserSchema’ is defined using Mongoose’s ‘Schema’ constructor. It outlines the structure of a user document in the MongoDB database, including fields for email, first name, last name, institution, role, and account picture details.
- **email:** a string representing the user’s email address, which is required and must be unique.
- **firstname and lastname:** strings representing the user’s first and last names, respectively, both of which are required.
- **institution:** a string representing the user’s institution, also required.
- **role:** a string indicating the user’s role, which must be one of two values: *student* or *tutor*. This field uses an enum to enforce a specific set of values.
- **accountPicture:** an object containing details about the user’s account picture, including its URL and filename. Both fields have default values, providing fallback options if a user doesn’t upload a custom picture.
- **passport-local-mongoose package:** the ‘passport-local-mongoose’ plugin is applied to the schema using ‘UserSchema.plugin(passportLocalMongoose)’. This automatically adds username and password fields to the schema, along with additional methods and functionality for user authentication.
- **model export:** the schema is compiled into a model with the name *User*, which is then exported. This model can be used throughout the application to interact with the ‘users’ collection in the database, allowing for the creation, querying, updating, and deletion of user documents according to the defined schema.

models\user.js

```
1  const mongoose = require("mongoose");
2  const Schema = mongoose.Schema;
3  const passportLocalMongoose = require("passport-local-mongoose");
4
5  const UserSchema = new Schema({
6    email: {
7      type: String,
8      required: true,
9      unique: true,
10   },
11   firstname: {
12     type: String,
13     required: true,
14   },
15   lastname: {
16     type: String,
17     required: true,
18   },
19   institution: {
20     type: String,
21     required: true,
22   },
23   role: {
24     type: String,
25     enum: ["student", "tutor"],
26   },
27   accountPicture: {
28     url: {
29       type: String,
30       default:
31         "https://t4.ftcdn.net/jpg/02/29/75/83/240_F_229758328_7x8jwCwjtbMmC6rgFzLFhZoE
32         pLobB6L8.jpg",
33     },
34     filename: {
35       type: String,
36       default: "default-account-picture.jpg",
37     },
38   },
39 });
40 UserSchema.plugin(passportLocalMongoose);
41
42 module.exports = mongoose.model("User", UserSchema);
43
```

5.4.3 Public folder

The public folder is the one containing all the static files served to others. This includes icons and pictures, JS scripts and stylesheets linked to EJS views.

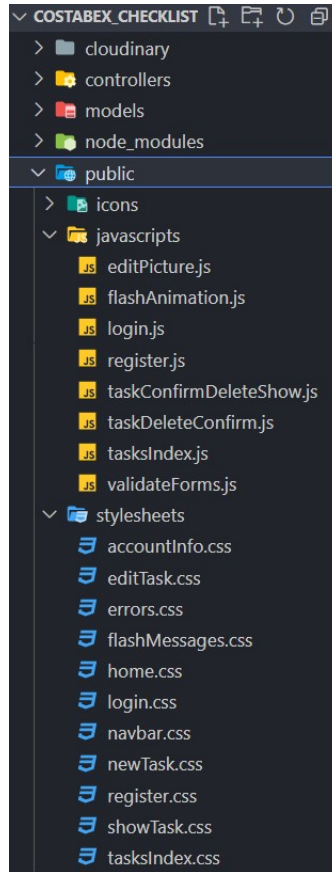


Figure 5.2: Public folder

5.4.4 Routes

The routes involved in costabex checklist are the one related with tasks and user management, through these routes is possible to catch HTTP requests and react to them via responses sent through the specific routes.

Tasks routes

This file defines routes for handling CRUD (Create, Read, Update, Delete) operations related to the tasks. These are the relative sections:

dependencies: imports necessary dependencies such as Express, the Express Router, controller functions for handling tasks, middleware functions for error handling and authentication, and Multer for handling file uploads.

router initialization: initializes an Express Router by calling `express.Router()`. The `mergeParams: true` option is passed to allow access to parameters from the parent router, which is useful when using nested routers.

file upload setup: Multer package is configured to handle file uploads. It utilizes a storage configuration provided by a function named `storage` imported from `../cloudinary`. The `upload` variable is then initialized with this storage configuration.

routes definition:

- *GET* `"/new"` renders a form for creating a new task. Requires the user to be logged in (`isLoggedIn` middleware) and is handled by the `newTaskForm` controller function.
- *POST* `"/"` handles the creation of a new task. Requires the user to be logged in (`isLoggedIn` middleware), validates the task data (`validateTask` middleware), and handles file upload (if any). It is processed by the `addTask` controller function.
- *GET* `"/"` renders a page displaying all tasks. Requires the user to be logged in and is handled by the `showTasks` controller function.
- *GET* `"/:id"` renders a page displaying details of a specific task identified by its ID. Requires the user to be logged in and is handled by the `taskDetails` controller function.
- *GET* `"/:id/edit"` renders a form for editing an existing task. Requires the user to be logged in, checks if the user is the author of the task (`isTaskStudent` middleware), and is handled by the `editTaskForm` controller function.
- *PUT* `"/:id"` handles updating an existing task. Requires the user to be logged in, checks if the user is the author of the task (`isAuthor` middleware), validates the task data (`validateTask` middleware), and handles file upload (if any). It is processed by the `editTask` controller function.

- *DELETE "/:id"* handles deleting a task. Requires the user to be logged in and checks if the user is the author of the task. It is handled by the 'deleteTask' controller function.
- *GET "/:id/removeFile"* handles removing a file associated with a task. Requires the user to be logged in and checks if the user is the author of the task. It is handled by the 'removeTaskFile' controller function.

exporting router: the router instance is exported, making these routes available for use in other parts of the application. To summarize, this code sets up routes for managing tasks, ensuring proper authentication, validation, and file handling along the way.

routes/task.js

```
1  const express = require("express");
2  const router = express.Router({ mergeParams: true });
3  const tasksC = require("../controllers/tasks");
4  const catchAsync = require("../utils/catchAsync"); //error handling requirement
5  const { isLoggedIn, validateTask, isAuthor, isTaskStudent } = require("../Middlewares");
6
7  // multer addition -> for image upload
8  const multer = require("multer");
9  const { storage } = require("../cloudinary");
10 const upload = multer({ storage });
11
12 // CRud task
13 router.get("/new", isLoggedIn, catchAsync(tasksC.newTaskForm));
14
15 router.post(
16   "/",
17   isLoggedIn,
18   validateTask,
19   upload.single("document"),
20   catchAsync(tasksC.addTask)
21 );
22
23 router.get("/", isLoggedIn, catchAsync(tasksC.showTasks));
24
25 router.get("/:id", isLoggedIn, catchAsync(tasksC.taskDetails));
26
27 // crUd task
28 router.get("/:id/edit", isLoggedIn, isTaskStudent, catchAsync(tasksC.editTaskForm));
29
30 router.put(
31   "/:id",
32   isLoggedIn,
33   isAuthor,
34   validateTask,
35   upload.single("document"),
36   catchAsync(tasksC.editTask)
37 );
38
39 // cruD tasks
40 router.delete("/:id", isLoggedIn, isAuthor, catchAsync(tasksC.deleteTask));
41
42 // Route to remove task file
43 router.get("/:id/removeFile", isLoggedIn, isAuthor, catchAsync(tasksC.removeTaskFile));
44
45 module.exports = router;
46
```

Users routes

Users routes are the ones relative to the users' URLs, allow user management actions like add new ones or getting their data.

dependencies: it imports necessary libraries such as Express for route handling, controllers for handling user-related actions, Passport.js for authentication, and multer for handling file uploads.

router initialization: it initializes an Express Router by calling `express.Router()`.

file upload setup: multer is configured to handle file uploads. It utilizes a storage configuration provided by the function named `storage2` imported from `../cloudinary`. The `upload` variable is then initialized with this storage configuration.

routes definition

- *POST* `/register`: handles user registration. Renders a registration form on GET request (`userC.showRegisterPage`) and processes registration data on POST request, adding a new user to the database (`userC.addUser`).
- *POST* `/login`: handles user login. Renders a login form on GET request (`userC.showLoginPage`) and processes login data on POST request, authenticating the user using Passport.js local strategy. If authentication fails, it redirects to the login page with a flash message. If successful, it redirects to the account page (`userC.login`).
- *GET* `/logout`: handles user logout, invoking the `userC.logout` controller function to log the user out of the application.
- *GET* `/account`: renders the user's account page, displaying account details (`userC.account`).
- *PUT* `/account`: handles updating the user's profile picture. Processes a file upload of a new profile picture (`accountImage`) and updates the user's account picture in the database (`userC.updateAccountPicture`).
- *GET* `/account/remove`: handles removing the user's profile picture. Invokes the `userC.removeAccountPicture` controller function to remove the user's account picture from the database.

exporting router: the router instance is exported, making these routes available for use in other parts of the application.

routes\users.js

```
1  const express = require("express");
2  const router = express.Router();
3  const userC = require("../controllers/users");
4  const passport = require("passport");
5  const catchAsync = require("../utils/catchAsync");
6  const multer = require("multer");
7  const { storage2 } = require("../cloudinary");
8  const upload = multer({ storage: storage2 });
9
10 router // Register
11   .route("/register")
12   .get(userC.showRegisterPage)
13   .post(catchAsync(userC.addUser));
14
15 router // Login
16   .route("/login")
17   .get(userC.showLoginPage)
18   .post(
19     passport.authenticate("local", {
20       failureFlash: true,
21       failureRedirect: "/login",
22     }),
23     userC.login
24   );
25
26 // logout
27 router.get("/logout", userC.logout);
28
29 router.get("/account", userC.account);
30
31 // edit profile picture
32 router.put(
33   "/account",
34   upload.single("accountImage"),
35   catchAsync(userC.updateAccountPicture)
36 );
37
38 // Route to remove account picture
39 router.get("/account/remove", userC.removeAccountPicture);
40
41 module.exports = router;
42
```

5.4.5 Controllers

Controllers define the specific functions called where a route where they are applied is triggered, and HTTP verbs are executed. In Costabex checklist, controllers are directly connected with task and user routes and consist in different related actions.

Tasks controllers

Tasks controllers consist in tasks management functions relative to addition, editing, showing and deleting actions. ShowTasks and taskDetails are the ones exposed.

- *newTaskForm*: renders a form for creating a new task and fetches necessary data.
- *addTask*: processes form data to create a new task, including file uploads, and saves it to the database.
- *showTasks*: fetches and displays tasks based on specified criteria.
- *taskDetails*: retrieves and displays detailed information about a specific task.
- *editTaskForm*: renders a form for editing an existing task and fetches task details.
- *editTask*: processes form data to edit an existing task, including file uploads, and updates it in the database.
- *deleteTask*: deletes a task from the database.
- *removeTaskFile*: removes the file associated with a task from Cloudinary and updates the task in the database.

controllers\tasks.js

```
1 module.exports.showTasks = async (req, res) => {
2   const { period, category } = req.query; // Extract both period and category from the
  query string
3
4   try {
5     let tasks;
6
7     if (period && category) {
8       // Filter tasks by both period and category
9       tasks = await Task.find({ period, category }).populate("author", "role");
10    } else if (period) {
11      // Filter tasks by period only
12      tasks = await Task.find({ period }).populate("author", "role");
13    } else {
14      // Fetch all tasks if no period specified
15      tasks = await Task.find({}).populate("author", "role");
16    }
17
18    // Filter out tasks without an associated author
19    tasks = tasks.filter((task) => task.author);
20    res.render("tasks/tasksIndex", {
21      tasks,
22      period: period || "all",
23      category: category || "all",
24      categoryParam: category || null,
25      periodParam: period || "all",
26    });
27  } catch (error) {
28    console.error("Error fetching tasks:", error);
29    res.status(500).send("Internal Server Error");
30  }
31 };
32
33 // /:id get
34 module.exports.taskDetails = async (req, res, next) => {
35   const { id } = req.params;
36   const task = await Task.findById(id).populate("author student");
37   console.log(task);
38   if (!task) {
39     next(new ExpressError("Page not found -> all route", 404));
40     return res.redirect("/tasks");
41   }
42   res.render("tasks/show", { task });
43 };
```

Users controllers

Users controllers enabled user management functions relative to users' registration, data getting or showing, login, logout, deleting actions and handles user-related actions within the application. The ones shown below are the showRegisterPage and addUser ones.

- *showRegisterPage*: renders the registration page. If a user is authenticated, it logs them out before rendering.
- *addUser*: processes registration form data to create a new user. If the role is "tutor", it checks the tutor key before registration. After successful registration, it logs the user in and redirects them to the tasks page.
- *showLoginPage*: renders the login page. If a user is authenticated, it logs them out before rendering.
- *login*: processes login form data, flashes a welcome message, and redirects the user to the tasks page or the previously visited page.
- *logout*: logs the user out and redirects them to the home page ("/").
- *account*: renders the user's account page.
- *updateAccountPicture*: handles updating the user's account picture. It uploads the new picture to Cloudinary, deletes the previous picture, updates the user's account with the new picture details, and redirects to the account page.
- *removeAccountPicture*: removes the user's account picture from Cloudinary, updates the user's account with the default picture details, and redirects to the account page.

controllers\users.js

```
1  const User = require("../models/user");
2  // needed to remove the file from cloudinary
3  const cloudinary = require("cloudinary").v2;
4
5  // /register get
6  module.exports.showRegisterPage = async (req, res) => {
7    // Log out the user if there's an active session
8    if (req.isAuthenticated()) {
9      await req.logout(() => {});
10   }
11   console.log("register page");
12   res.render("users/register");
13 };
14
15 // /register post
16 module.exports.addUser = async (req, res, next) => {
17   try {
18     const { role, tutorKey } = req.body; // Extract role and tutorKey from req.body
19     let registeredUser;
20
21     // Check if the role is "tutor" and if the tutorKey is correct
22     if (role === "tutor" && tutorKey !== "tutorpass") {
23       req.flash("error", "Invalid tutor key");
24       return res.redirect("/register"); // Redirect back to the registration page
25     }
26
27     // Create user object based on extracted userData
28     const user = new User(req.body);
29
30     // If role is "tutor", set additional property for tutorKey
31     if (role === "tutor") {
32       user.tutorKey = tutorKey;
33     }
34
35     // Register user
36     registeredUser = await User.register(user, req.body.password);
37
38     // User login
39     req.login(registeredUser, (err) => {
40       if (err) return next(err);
41       res.redirect("tasks");
42     });
43   } catch (e) {
44     req.flash("error", e.message);
45     res.redirect("tasks");
46   }
47 };
```


5.4.6 Middlewares

The middlewares implemented in the Costabex checklist app are mainly used for tasks and users' authorization and validation.

- *isLoggedIn*: checks if the user is authenticated. If not, it redirects them to the login page with an error message. It also stores the original requested URL in the session for redirection after login.
- *validateTask*: validates the task data received in the request body using Joi schema. If there's an error, it throws an `ExpressError` with the details of the validation error.
- *isAuthor*: checks if the authenticated user is the author of the task or has appropriate permissions. If not, it redirects with an error message.
- *isTaskStudent*: checks if the authenticated user is the student assigned to the task. If not, it redirects with an error message.

Middlewares.js

```
1  const Task = require("./models/task"); //require the model
2  const { taskJoiSchema } = require("./joiSchemas");
3  const ExpressError = require("./utils/ExpressError");
4
5  // login middleware
6  module.exports.isLoggedIn = (req, res, next) => {
7    if (!req.isAuthenticated()) {
8      req.session.returnTo = req.originalUrl;
9      req.flash("error", "You must be signed in first!");
10     return res.redirect("/login");
11   }
12   next();
13 };
14
15 // Joi schema middleware
16 module.exports.validateTask = (req, res, next) => {
17   const { error } = taskJoiSchema.validate(req.body.task);
18   if (error) {
19     const msg = error.details.map((el) => el.message).join(","); //to show the message in
the error template
20     throw new ExpressError(msg, 400);
21   } else {
22     next();
23   }
24   console.log(error);
25 };
26
27 // Permission middleware
28 module.exports.isAuthor = async (req, res, next) => {
29   const { id } = req.params;
30   const task = await Task.findById(id);
31   if (!task.author.equals(req.user._id) && req.user.role !== "student") {
32     req.flash("error", "You do not have permission to do this action.");
33     return res.redirect(`/tasks/${id}`);
34   }
35   next();
36 };
37
38 // Middleware to check if the current user is the student assigned to the task
39 module.exports.isTaskStudent = async (req, res, next) => {
40   const { id } = req.params;
41   const task = await Task.findById(id).populate("student");
42   console.log(task);
43   // Check if there is a current user and if they are the student assigned to the task
44   if (!req.user || !task.author) {
45     req.flash("error", "You do not have permission to edit this task.");
46     return res.redirect(`/tasks/${id}`);
47   }
48   next();
49 };
50
```

5.4.7 Views

Views are the files used for displaying different tasks, users and app pages. Within these files we have: boilerplate, partials, tasks views, users views, error view and landing page view.

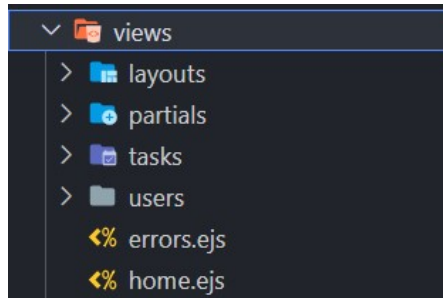


Figure 5.3: Views folder

Boilerplate

The boilerplate is the basic template included in all other views, it constitutes the basic layout for all other pages, the skeleton of the app. It includes navbar, flash messages and bootstrap styles embedding, validation scripts and includes the necessary metadata, such as character set and viewport settings. Additionally, it imports Bootstrap CSS and JavaScript files from CDN (Content Delivery Network) sources to style and add interactive elements to the webpage. The '`<body>`' element is configured with classes for flexbox layout ('`d-flex flex-column vh-100`') to ensure proper alignment and responsiveness. Within the body, it includes a navigation bar using an EJS partial `<%- include("../partials/navbar") %>`, and the main content area ('`<main>`') where the dynamic content will be injected. At the bottom of the body, it imports a custom JavaScript file ('`validateForms.js`') for form validations using Bootstrap's validation features.

views\layouts\boilerplate.ejs

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Costabex checklist</title>
8   <!-- bootstrap -->
9   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel=
"stylesheet" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxS
R1GAsXEV/Dwvykc2MPK8M2HN" crossorigin="anonymous">
10
11   <script src="
https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js" integrity="
sha384-C6RzsynM9kWDrmNeT87bh950GNYZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL" crossorigin="
anonymous" defer"></script>
12
13   <!-- popper -->
14   <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.8/dist/umd/popper.min.js"
integrity="sha384-I7E8VVD/ismYTF4hNIPjVp/Zjvgyo16VfVrKX/vR+Vc4jQkC+hVqc2pM80Dewa9r"
crossorigin="anonymous"></script>
15
16   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.min.js"
integrity="sha384-BBt1+eGJRgqQAUMxJ7pMwbEyER411g+015P+16Ep7Q9Q+zzqX6gSbd85u4mG4QzX+"
crossorigin="anonymous" defer"></script>
17
18   <!-- ----- -->
19
20
21 <body class="d-flex flex-column vh-100">
22   <%- include("../partials/navbar") %>
23   <main class="container mt-3">
24     <%- body %>
25   </main>
26   <!-- bootstrap validations -->
27   <script src="/javascripts/validateForms.js"></script>
28 </body>
```

Partials

Partials in Costabex checklist are view modules mainly used in the boilerplate. Navbar or flash view page are an example of a partial.

task views

Task views are the files that allow the display of the tasks' management pages. NewTask view file is the one exposed below:

- **edit task form (editTask.ejs):** this file represents the view for editing a task. It includes a form with fields to modify various aspects of a task, such as its status, category, deadline, and document upload. The form is conditionally rendered based on the user's role and authorization to edit the task. If the user is the author or a tutor, they can edit all fields, including the assigned student and file approval status. If the user is the student assigned to the task, they can only view certain fields and upload a document if the task is not yet checked.
- **new task form (newTask.ejs):** this file represents the view for creating a new task. It contains a form with fields to input task details, including the task name, assigned student (for tutors), status, category, document upload, and deadline. It also includes logic to conditionally display the "Assign to Student" dropdown only if the user is a tutor.
- **task details page (showTask.ejs):** this file represents the view for displaying detailed information about a single task. It includes details such as the task name, status, period, category, author, assigned student (if applicable), deadline, and file preview link. Additionally, it provides options for editing the task (if authorized), deleting the task, and removing the associated file.
- **dashboard (tasksIndex.ejs):** this file represents the main dashboard view, displaying a summary of tasks based on different periods and categories. It includes options to filter and sort tasks by period and category and others features, as well as buttons to view all tasks, navigate to the checklist, and add a new task. The dashboard also displays a list of tasks with options to edit or delete each of them, depending on the user's role and authorization. Additionally, it includes notification alerts for tutors to review tasks uploaded by students.

views\tasks\new.ejs

```
1 <% layout("layouts/boilerplate") %>
2
3 <head>
4   <meta charset="UTF-8" />
5   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6   <title>New task</title>
7   <link rel="stylesheet" href="/stylesheets/newTask.css">
8 </head>
9
10 <body>
11   <div class="container mt-3">
12     <div class="row justify-content-center">
13       <div class="card p-4">
14         <h3 style="color: #3B7D8C;" class="card-title">Add Task</h3>
15         <section>
16           <form action="/tasks" method="post" class="needs-validation" novalidate enctype="
multipart/form-data">
17             <div class="mb-3">
18               <label for="name"><b>Insert Task</b></label>
19               <input class="form-control rounded-pill" type="text" name="name" id="name"
required maxlength="40" placeholder="Task name">
20             </div>
21             <% if (currentUser && currentUser.role === "tutor") { %>
22             <!-- Show the "Assign to Student" dropdown only if the user is a tutor -->
23             <div class="mb-3">
24               <label for="student"><b>Assign to Student</b></label>
25               <select class="form-control form-select rounded-pill" name="student" id="
student">
26                 <option value="">Not Assigned</option> <!-- Option for not assigning to any
student -->
27                 <% students.forEach(student => { %>
28                   <option value="<%= student._id %>"><%= student.firstname %> <%=
student.lastname %></option>
29                 <% }); %>
30               </select>
31             </div>
32             <% } %>
33             <div class="mb-3">
34               <label for="checked"><b>Done?</b></label>
35               <input type="checkbox" name="checked" id="checked">
36               <label for="checked">YES</label>
37             </div>
38             <div class="mb-3">
39               <label for="category"><b>Select the Category</b></label>
40               <select class="form-control form-select rounded-pill" name="category" id="
category">
41                 <% for (let category of categories) { %>
42                   <option value="<%= category %>"> <%= category %> </option>
43                 <% } %>
44               </select>
45             </div>
46             <div class="mb-3">
47               <label for="pdf"><b>pdf upload (optional)</b></label>
48               <input class="form-control rounded-pill" type="file" name="document" id="
document" accept=".pdf">
49             </div>
50             <div class="mb-3">
51               <label for="deadline"><b>Deadline</b></label>
```

```
52     <input class="form-control rounded-pill" type="date" name="deadline" id="
deadline">
53   </div>
54   <div class="text-center">
55     <button class="btn btn-light mt-3 px-4 rounded-pill" id="addButton">Add<
/button>
56   </div>
57 </form>
58 <div class="mt-3">
59   <a class="btn btn-light" href="/tasks">Back</a>
60 </div>
61 </section>
62 </div>
63 </div>
64 </div>
65 </body>
66
```

User views

User views are the files that allow the display of the users' management pages. Login one is exposed below.

- **accountInfo.ejs:** this file represents the user account information page. It presents the user's profile picture, name, and institution. Users can edit their account picture by uploading a new image or removing the existing one. The page includes a form for uploading a new picture and a button to remove the current picture. JavaScript is used for form validation to ensure that only JPEG or PNG files are uploaded, and a message is displayed to guide users on the file format and recommended dimensions.
- **login.ejs:** the Login.ejs file serves as the login page for users. It presents a form where users can input their username and password to log in. Additionally, it utilizes Bootstrap icons for visual elements. JavaScript functionality is included to allow users to toggle the visibility of the password field, enhancing user experience by giving them the option to view or hide their entered password.
- **register.ejs:** register.ejs represents the registration page for new users. It displays a form where users can input their first name, last name, university, username, email, password, and role (either student or tutor). The form dynamically adjusts based on the selected role, showing an additional field for the tutor key if the user selects the role of a tutor. JavaScript functionality is included to enable users to toggle the visibility of the password and tutor key fields, providing convenience and usability during the registration process. Additionally, Bootstrap icons are utilized for visual elements throughout the form.

views\users\login.ejs

```
1 <% layout("layouts/boilerplate") %>
2 <%- include("../partials/flash") %>
3
4 <head>
5   <meta charset="UTF-8" />
6   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7   <title>Login</title>
8   <link rel="stylesheet" href="/stylesheets/login.css">
9 </head>
10
11 <body>
12   <div class="container mt-3">
13     <div class="row justify-content-center">
14       <div class="col-md-12">
15         <div class="card p-3">
16           <div class="card-body">
17             <h5 class="card-title">Login</h5>
18             <form action="/login" method="POST" class="needs-validation" novalidate>
19               <div class="mb-2">
20                 <label class="form-label" for="username">Username</label>
21                 <input class="form-control rounded-pill" type="text" id="username" name="
username" autofocus required>
22               </div>
23               <div class="mb-2">
24                 <label class="form-label" for="password">Password</label>
25                 <div class="input-group">
26                 <input class="form-control rounded-pill" type="password" id="password"
name="password" required>
27                 <button class="input-group-addon" type="button" id="togglePassword">
28                   <i class="bi bi-eye-slash"></i>
29                 </button>
30                 </div>
31               </div>
32               <div class="text-center">
33                 <button class="btn btn-light mt-2 px-3 rounded-pill">Login</button>
34               </div>
35             </form>
36           </div>
37         </div>
38       </div>
39     </div>
40   </div>
41
42   <!-- Bootstrap icons -->
43   <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.8.1/font/bootstrap-icons.css"
rel="stylesheet">
44   <script src="/javascripts/login.js"></script>
45 </body>
```

Error view

The error page template file sets up the basic structure of an error page within an EJS document. It includes a layout directive to use the common "boilerplate" layout for consistency across pages and provides a structured layout for presenting error messages in a visually consistent manner.

Landing page view

This file serves as the landing page template for the application utilizing the common boilerplate layout file too for consistent page structure and creates a visually appealing landing page that invites users to engage with the application by registering or logging in.

5.4.8 Joi validation and utils

The 'taskJoiSchema' is used for generic validation of task data within the context of an application that involves users and tasks. It ensures that the task object follows a predefined structure and constraints, which are defined based on the Task model. This schema validates properties such as the task name, whether it's checked, its period, category, author, student, document details, creation date, and deadline. By enforcing these validation rules, the application can maintain data integrity and consistency, enhancing overall reliability and usability.

The 'catchAsync' function is a middleware wrapper designed to handle asynchronous functions in Express routes. It takes another function 'func' as an argument, which is typically an asynchronous route handler. Inside 'catchAsync', it returns a new function that accepts 'req', 'res', and 'next' as parameters. This returned function invokes the original 'func' function with these parameters. However, it adds a 'catch()' method to the invocation, passing 'next' as the error handler. This allows any errors that occur within the asynchronous function to be caught and forwarded to Express's error handling middleware ('next'). Essentially, 'catchAsync' simplifies error handling for asynchronous operations within Express routes, making the code cleaner and more maintainable.

The ExpressError class handling file is designed to create custom error objects specifically for use within an Express.js application. It extends the built-in Error class. When instantiated, it takes a message and a status code as parameters, allowing developers to create specific error instances with custom messages and HTTP status codes. This facilitates error handling and response customization within the application.

joiSchemas.js

```
1  const Joi = require("joi");
2
3  // Joi schema for tasks model
4  module.exports.taskJoiSchema = Joi.object({
5    task: Joi.object({
6      name: Joi.string().required(),
7      checked: Joi.boolean().required(),
8      period: Joi.string().valid("before", "during", "after").required(),
9      category: Joi.string().valid(
10       "mobility process",
11       "formalities",
12       "finance management",
13       "language development",
14       "accomodation",
15       "cultural adaptation",
16       "students helping students",
17       "it's packing time",
18       "what could go wrong",
19       "study advice",
20       "stress management",
21       "how to manage a crisis",
22       "work-rest balance",
23       "re-entry shock and how to deal with it",
24       "personal development"
25     ).required(),
26     author: Joi.string().required(),
27   }).required(),
28 });
29
30 // utils/catchAsync.js
31 module.exports = (func) => {
32   return (req, res, next) => {
33     func(req, res, next).catch(next);
34   };
35 };
36
37 // utils/ExpressError.js
38 class ExpressError extends Error {
39   constructor(message, statusCode) {
40     super(); //it's going to call the Error constructor
41     this.message = message;
42     this.statusCode = statusCode;
43   }
44 }
45
46 module.exports = ExpressError;
47
48
49
```

Chapter 6

Conclusions

As this thesis draws to a close, it is evident that the development of the Costabex checklist represents just the initial steps in a journey toward empowering students during their Erasmus exchanges. User-friendly design, and practical implementation has yielded a robust platform poised to support the way students manage their academic, personal, and administrative tasks. However, it is important to recognize that this is merely the beginning of a foundation upon which further enhancements, refinements, and iterations can be built to better serve the evolving needs of students in the dynamic landscape of international education.

The Costabex checklist operates at the intersection of document management and task organization, offering a seamless and intuitive user experience tailored specifically to the challenges faced by students during their Erasmus journeys. At its core, the platform provides users with a centralized hub for storing and accessing documents, such as visa applications, academic transcripts, learning agreements and others allowing users to create, prioritize, and track their academic assignments, extracurricular activities, and personal commitments. Central to the overarching purpose of the app is the enhancement of student well-being related with the main Costabex project.

In conclusion, the development of the Costabex checklist represents a first step in the ongoing quest to enhance the student experience during Erasmus exchanges. Future developments will regard costabex website completion thought continuously partners and targets emphasize, define, ideate and testing design thinking's processes. Subsequently, future checklist service's deployment, mail management and notification system integration in addition to service workflow explanation displayed in the landing page and more optimization based on improving security issues to be safer for sensitive data ensuring the privacy of all users.

Bibliography

- [1] URL: <https://www.esn.org/news/report-mental-health-and-international-mobility> (cit. on p. 1).
- [2] Donald A. Norman. *The Design of Everyday Things*. 2011 (cit. on pp. 6, 10).
- [3] F.Moschetto G.Malnati F.Bobba. *Digital interaction design Introduction - digital interaction design course*. 2017-2023 (cit. on pp. 9, 13).
- [4] F.Moschetto G.Malnati. *User centered design - digital interaction design course*. 2019-2022 (cit. on p. 9).
- [5] F.Moschetto G.Malnati. *Modello di interazione (don Norman reference) - digital interaction design course*. 2019-2022 (cit. on p. 10).
- [6] Bottà. D. *User Experience Design*. 2019 (cit. on p. 10).
- [7] F.Moschetto G.Malnati. *Design thinking - digital interaction design course*. 2019-2023 (cit. on p. 11).
- [8] Jan Nikka A. Estefani. *The Future of UX Design in 2024*. 2023. URL: <https://raw.studio/blog/the-future-of-ux-design-in-2024/#:~:text=In%20the%20future%2C%20we%20can%20expect%20to%20see%20more%20and,seamless%20and%20intuitive%20for%20users>. (cit. on p. 13).
- [9] URL: <https://costabex.eu/> (cit. on p. 15).