

BILKENT UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING

CS 299
SUMMER TRAINING
REPORT

Deniz Tuna Onguner

22001788

Performed at

Cubicl

15.08.2023 - 13.09.2023

Table of Contents

1	Introduction	3
2	Company Information.....	4
2.1	About the company.....	4
2.2	About my department	4
2.3	About the hardware and software systems	4
2.4	About my supervisor.....	5
3	Work Done	5
3.1	How Monkedo works	5
3.2	Used technologies on Monkedo	5
3.3	Developing components	6
3.4	Developing integrations	7
4	Performance and Outcomes	8
4.1	Solving Complex Engineering Problems	8
4.2	Recognizing Ethical and Professional Responsibilities	8
4.3	Making Informed Judgments	8
4.4	Acquiring New Knowledge by Using Appropriate Learning Strategies.....	9
4.5	Applying New Knowledge as Needed	9
4.6	Awareness About Diversity, Equity, and Inclusion	9
5	Conclusions	10
	References.....	11
	Appendix A.....	12
	Appendix B.....	13
	Appendix C	14

1 Introduction

The objective of this report is to document and demonstrate the experiences and issues I encountered during my internship in the summer of 2023 and provide overall information about the company I worked for and the work I have accomplished there [1].

I had my summer internship at the firm Cubicl in their software development team. My motivation for pursuing an internship there was to participate in a project set for launch, creating an application meant to be utilized while collaborating with a diverse team of experienced senior developers and junior colleagues. As a relatively small firm, I could assume greater responsibility and acquire more knowledge at Cubicl than a larger corporation [1].

At Cubicl, I took part in developing a web application named "Monkedo," often called the automation project on the site. The purpose of Monkedo is to automate tasks that people frequently repeat on their computers via connecting third-party applications. For instance, many companies use dozens or hundreds of software programs to monitor their operations. Much of the work involves constant repetition, requiring transfer between repeated tasks or actions when an event occurs. These tasks can take up hours, causing to waste both time and productivity. Monkedo makes the automation of such repetitive tasks possible. For example, it can automate various actions, such as sending individual emails and SMS messages to a customer list, transferring data between applications, or receiving notifications when a significant event occurs.

In the team, I was assigned to integrate various third-party applications into Monkedo via their API endpoints. Thus, the services provided by these applications have become usable within Monkedo. The applications to integrate in were decided by the seniors and then assigned to me as tasks. My duty was to read the assigned programs' official API documentation, integrate them into Monkedo, and finally test them on my local before deploying them into the actual program. Occasionally, I was asked to revise my work by the other team members before publishing it.

In the following sections, I will provide more detailed information about Cubicl with my team and supervisor. Then, I will explain and exemplify the tasks I have completed during my internship and the issues I needed to deal with during the process. I will also list the technologies, programming languages, frameworks I used and the applications I integrated into Monkedo. In the end, I will finalize my report with the contributions of this internship to me and my overall learning outcomes.

2 Company Information

2.1 About the company

Cubicl is a software company located in ODTÜ Teknokent, Ankara, which provides project and task management services to firms and teams. It was officially founded in 2017, and as of 2023, it has around 50 employees [2].

2.2 About my department

I worked in the automation team at Cubicl, which was the team responsible for the development of Monkedo. To be more specific, the team I worked in was developing the API integrations of Monkedo or frequently dealing with anything regarding the backend.

Although the frontend part of the program was not one of the responsibilities of the automation team, we were the ones deciding the explanations that were to appear on the UI since we were the ones developing them.

There were also other teams working on different projects but Monkedo; though, I never worked with someone from these other teams.

2.3 About the hardware and software systems

Interns working on Monkedo were allowed to use their devices, though we are asked to work either on MacOS or Ubuntu – briefly, a Unix-based operating system, which was not a problem for me since I do use a MacBook.

For internal company communication, our primary communication channel was Cubicl – a program that shares the exact same name with the company. It sends notifications for activities and messages related to the member. It is also used to keep track of the tasks and set up meetings.

We utilized the Nuclino application for knowledge sharing in the team and documentation.

Furthermore, to foster development within our team, we used Git and GitHub. For each task assigned, a new branch is created, and then the new features are deployed via the creation of pull requests.

Our designated development environment was Visual Studio Code (VS Code).

2.4 About my supervisor

My supervisor at Cubicl was Erkan Pulat. He graduated from the Department of Computer Engineering of İnönü University in 2017. After his graduation, he started to work at Cubicl as a full-time web developer in the same year [3].

3 Work Done

As stated earlier, my main responsibility at Cubicl was to integrate third-party applications into Monkedo. In this section, I will explain how Monkedo works to make the work I have done clear, and then I will detail my contributions to the project and provide code samples that I developed to demonstrate my work.

The demo version of Monkedo has been publicly launched on the 10th of September 2023 and is accessible via the following link: <https://monkedo.com>.

3.1 How Monkedo works

In a nutshell, Monkedo works via its components. Each component is a separate program which is designed to fulfil a specific task. Components can be used by themselves, or they can be connected to other components, so they work together. They include components for retrieving data from the resources you use, processing data, performing mathematical operations, sending emails, and many other operations.

Each flow created on the platform is called an automation within the application. To create an automation, you need to open the automation units. Here, for the operations you want to perform, you place components and connect them to each other (refer to Appendix B, Image I).

These components can be easily found based on their purposes. The first category is Triggers, which are components that determine when an automation will run. For example, they can be set to run manually, at specific time intervals, or in response to various events or incidents within an application to use. The second category is Action components, which allow to perform operations within the applications and process data. Data processing components, as the name suggests, allow to save data within the automation and access it later. Browser Action components, on the other hand, are to open web pages in a web browser and perform actions on them as if they were a real user.

Components have inputs and outputs. They take input data from inputs, perform their respective tasks, and then provide the output in the output section. When a component is clicked, an information panel that provides details about the component is opened. Users can see names, types, options, and other

settings of the inputs and outputs that will be received on this panel. Outputs of a component can be used as input for another component (refer to Appendix B, Image II).

One can apply this video on YouTube for a visualized tutorial which explains everything and more I discussed above. But please note that it is an unofficial tutorial created to inform new members at Cubicl rather than the target users of Monkedo.

Monkedo is developed on NestJS with JavaScript and TypeScript (refer to Appendix A, Table I for the full list of the technologies used). We used Docker to run it on our local machines. Although the set-up of Docker was not my responsibility, I utilized it often to start the application while testing the codes I wrote.

3.2 Developing components

Components serve as fundamental units in the construction of an automation, each designed to fulfil a specific role within the system. During my internship at Cubicl, I contributed to the development of action-type components. An action component functions akin to a typical software function, accepting a set of inputs and producing an output [4]. However, what sets an action component apart is its capacity to not only format the inputs but also generate an API call to a designated endpoint. This distinctive feature renders the API accessible to individuals without programming expertise, effectively bridging the gap between technical and non-technical users [4]. Moreover, an action component retains the results from previous executions and can provide them upon request, enhancing its functionality and reusability. Additionally, these action components can be interconnected, allowing the return value of one component to serve as input for another, thereby creating a modular and scalable approach to an automation design [4].

Each action component is developed as a separate object and extends another object `ActionComponent`, so each component developed fulfils a criterion. Each component has an ID separating it from others, a name, a description explaining what it does, an icon, an authentication method since it will create an API call to another app, an app name that the component belongs to and an app key to this app, keywords for searching purposes, input fields, outputs fields, and a function that will run when the component is run (refer to Appendix C, Image I).

The components I developed were assigned to me by senior team members as part of my internship responsibilities. To begin the development process, I initiated by accessing the official API documentation of the designated application. In cases where it was necessary, I created an account to gain access to the required resources [5].

Subsequently, I meticulously followed the guidelines and specifications outlined in the documentation. This involved writing code that adhered to the

documented API endpoints, parameters, and data formats. For each component, I provided a descriptive name and a clear, informative description to facilitate its integration and understanding within the automation system [5].

Fundamentally, each component is designed to accept specific inputs. Within the 'run' function of these components, I employed a systematic approach to format these inputs, whether they originated from user interactions or were passed from other components. This formatting process aimed to ensure compatibility with the expected format of the API endpoint [5].

Once the inputs were appropriately formatted, the component was programmed to create an API call to the designated endpoint, utilizing the formatted input data. The resulting return value from the API call was also subject to formatting to ensure that it provided data in a structured and user-friendly manner, optimizing its utility for downstream processes [5].

I have developed around 30 action components during my internship for ten different applications (refer to Appendix A, Table II for the full list).

3.3 Developing integrations

In Monkedo, an integration is an authentication method that the components work on. So, users only need to authenticate an application once, and all the components belonging to this application become usable without requiring login action each time they use it.

I developed one integration while working on Monkedo, which was an application named Phrase.

While developing it, I first decided on the authentication method -- API key; then, I wrote the code accordingly. A user needs to create an API key from the official website once and log in with it on Monkedo. After that, each component of Phrase will be available to use for that user.

One can refer to Appendix C, Image II for the actual code running on Monkedo.

4 Performance and Outcomes

4.1 Solving Complex Engineering Problems

Although I completed all the tasks assigned to me and followed the general principles of programming and engineering during the process, I do not think that I solved any complex engineering problems.

I served as an intern on a web application project that employed high-level programming languages and frameworks. It is important to note that the project's complexity and demanding requirements were primarily handled by senior members of the development team. The project, known as Monkedo, is not exclusively assigned to interns; rather, it is a fully-fledged, real-time application developed by a comprehensive team consisting of full-time programmers, senior engineers, and junior engineers [5].

4.2 Recognizing Ethical and Professional Responsibilities

In terms of ethical and professional responsibilities, I can say that I followed the requirements and expectations of my team and the company. I attended all the team meetings either on-site or online (A few meetings had happened on Discord). Furthermore, I have tried to complete my tasks on the time given to me – each task assigned had its own deadline, and after completing a task, a new one was assigned.

My approach was rooted in collaboration, and I sought help and guidance from colleagues whenever necessary. This collaborative spirit not only facilitated problem-solving but also fostered a sense of mutual respect within the team. I consistently demonstrated professionalism and courtesy in all my interactions, which contributed to a positive and productive working environment during the project's development process [5].

4.3 Making Informed Judgments

When a task was assigned to me, I needed to decide on numerous stuff on my own. For instance, the types of input fields while developing a component are decided by the one developing it. So, I made the decisions regarding that while I was developing my components. However, when feedback was provided and asked for a revision, I made the required alterations as asked.

Moreover, I can say that not every API documentation is as clear as it should be, and this problem forces developers, including me, to make a bunch of judgements. Since I was responsible for developing API calls, I have dealt with this problem quite often.

4.4 Acquiring New Knowledge by Using Appropriate Learning Strategies

I worked on numerous new technologies during my internship, including TypeScript, Docker, and Nest. This situation forced me to apply different learning strategies. I often listened to the courses on the Udemty account of the company. Besides, I frequently applied AI tools – ChatGPT and GitHub's Copilot. I asked for help from my team members when AI and online sources failed me.

4.5 Applying New Knowledge as Needed

As stated, I needed to gain new knowledge often since I was working with technologies that I was not familiar with before. Each time I faced a bug or a syntax error, I searched them on Stackoverflow or sometimes other web pages like Geeksforgeeks.

Though I must admit that I never needed printed sources, everything I needed was findable on online sources.

When an error occurred related to something that I was not a part of, I asked for the help of the ones who were responsible for it. In this way, I got the solution rapidly but also informed the team of a possible error.

4.6 Awareness About Diversity, Equity, and Inclusion

In the context of my academic internship at Cubicl, I had the opportunity to gain valuable insights into the company's approach to fostering awareness about diversity, equity, and inclusion (DEI). Cubicl, as an organization, places a strong emphasis on promoting DEI principles in the workplace, creating an environment that is both enriching and empowering [6].

One notable aspect of Cubicl's commitment to DEI is the diversity of its workforce. The company prides itself on employing individuals with diverse educational backgrounds, hailing from various universities and academic disciplines. This diversity of thought and experience is a driving force behind Cubicl's innovative solutions and collaborative work culture [6].

Cubicl understands that diversity alone is not enough; it must be complemented by equity and inclusion. Through ongoing education, training, and open dialogue, Cubicl strives to ensure that every employee feels valued, heard, and respected. This commitment to equity and inclusion enhances our ability to attract, retain, and nurture top talent from all walks of life [6].

5 Conclusions

My internship experience at Cubicl has been incredibly enriching and educational. Throughout this short term, I have had the privilege of gaining insights into new technologies and honing my collaborative skills within a team of professionals. This experience has been instrumental in my personal and professional growth [7].

One of the most exciting aspects of my internship was the opportunity to dive into the world of programming. I worked extensively with two programming languages and two web frameworks that were entirely new to me (JavaScript, TypeScript, NestJS, and NodeJS). This hands-on experience not only expanded my technical repertoire but also deepened my understanding of software development and its practical applications [7].

Additionally, my internship at Cubicl provided me with the chance to learn how to effectively utilize Git and GitHub within a team setting. While there were some initial hiccups, including a branch mishap, I persevered and eventually mastered these essential tools. Understanding Git and GitHub's collaborative capabilities has proven invaluable, as they are foundational to modern software development practices and team collaboration [7].

In summary, my time at Cubicl has been a transformative learning experience. I've not only acquired new technical skills but also gained a deeper appreciation for the importance of teamwork and version control in the software development process [7].

References

- [1] "OpenAI's ChatGPT AI language model," response to question from author, 4 October 2023. <https://chat.openai.com/chat>
- [2] "Cubicl," LinkedIn, <https://www.linkedin.com/company/cubicl/about/> (accessed Oct. 8, 2023).
- [3] "Erkan Pulat," LinkedIn, <https://www.linkedin.com/in/erkanpulat/> (accessed Oct. 8, 2023).
- [4] OpenAI's ChatGPT AI language model," response to question from author, 5 October 2023. <https://chat.openai.com/chat>
- [5] "OpenAI's ChatGPT AI language model," response to question from author, 6 October 2023. <https://chat.openai.com/chat>
- [6] "OpenAI's ChatGPT AI language model," response to question from author, 7 October 2023. <https://chat.openai.com/chat>
- [7] "OpenAI's ChatGPT AI language model," response to question from author, 8 October 2023. <https://chat.openai.com/chat>

Appendix A

Table I
List of the technologies used to develop Monkedo

Technology (PL, Framework, Program etc.)	Version
NodeJS	18
NPM	8
Angular	CLI
NestJS	CLI
Docker	<latest>
MongoDB	6.0

Table II
List of the applications that I integrated into Monkedo

Application	Components Developed
Axonaut	List Events, Create Event, Create Employee, List Tasks, Create Task
Baserow	Delete Row, Update Row, Get Row, List Rows
eSputnik	Send Event, Update Contact
MailerSend	Send Email, Send Email from Template, Verify Email
Phrase	List Accounts, List Projects
ProfitWell	Create Subscription, Update Subscription
QStash	Create Topic, List Endpoints, List Topics, Publish Endpoint Message, Publish Topic Message
Snov.io	Add Prospect to List, Create Prospect List, Email Verifier, Find Prospect, See Campaign Replies, See User Lists, View All Campaigns
Timecamp	Create Time Entry
Zenler	Enroll User, Unenroll User

Appendix B

Image I
The automation development page of Monkedo

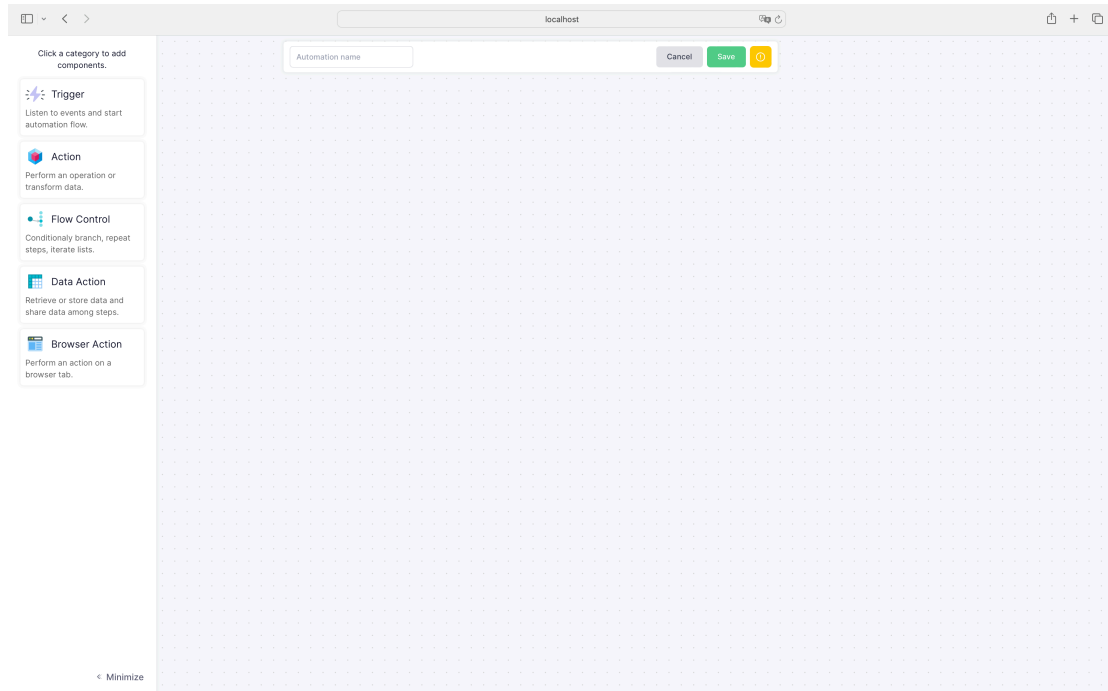
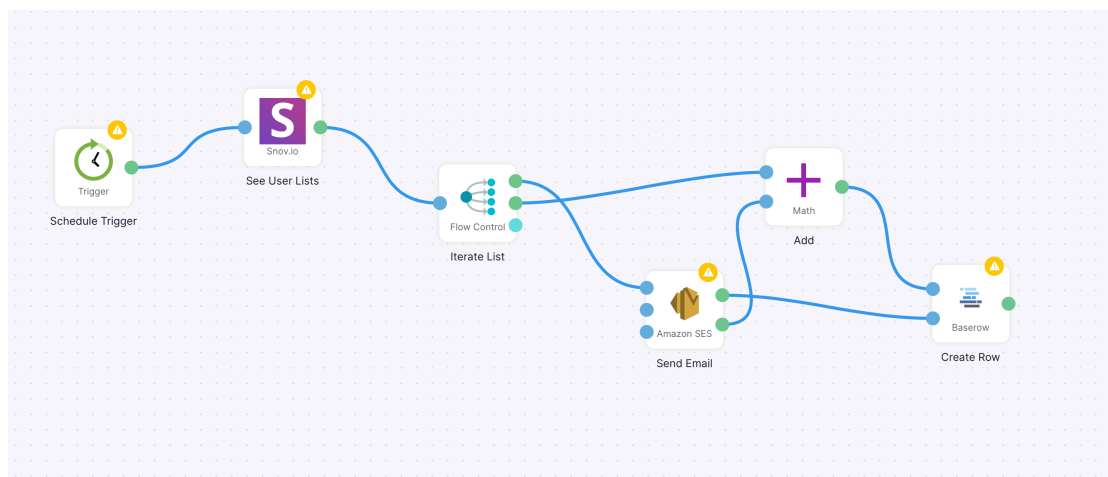


Image II
Interconnected components of an automation in Monkedo



Appendix C

Image I
Code sample to action component

```
1 import _ from 'lodash';
2 import axios from 'axios';
3
4 import MailerSendIntegration from '@integrations/apps/mailerSend';
5 import { typeFor, valueAs } from '@automation/helpers/value';
6 import { ActionComponent } from '@automation/types';
7
8 export const definition: ActionComponent = {
9   id: 'action-mailerSend-send-an-email',
10   name: 'Send an email',
11   desc: 'Send an email. [See docs here](https://developers.mailerSend.com/api/v1/email.html#send-an-email).',
12   icon: 'mailerSend.png',
13   auth: 'apikey',
14   appKey: 'mailerSend',
15   appOwner: 'MailerSend',
16   keywords: ['post an email', 'create an email'],
17   inputs: [
18     {
19       name: 'from',
20       label: 'Sender',
21       type: typeFor('EntityEmail: Text, name: Text'),
22       desc: 'The **from** email address used to deliver the message. This address should be a **verified sender** in your **MailerSend account**. And, optionally, a name or title associated with the sending email address.',
23       properties: [
24         {
25           name: 'email',
26         },
27         {
28           name: 'name',
29           optional: true,
30         },
31       ],
32     },
33     {
34       name: 'to',
35       label: 'Recipients',
36       type: typeFor('TableEmail: Text, name: Text'),
37       desc: 'Email addresses and the names of the intended recipients.',
38       properties: [
39         {
40           name: 'email',
41         },
42         {
43           name: 'name',
44           optional: true,
45         },
46       ],
47     },
48     {
49       name: 'subject',
50       type: typeFor('Text'),
51       desc: 'The subject line of the email.',
52     },
53     {
54       name: 'contentType',
55       type: typeFor('Text'),
56       desc: 'The content type of the email. Either 'text/plain' or 'text/html'.',
57       options: [
58         { label: 'Text', value: 'text/plain' },
59         { label: 'HTML', value: 'text/html' },
60       ],
61     },
62     {
63       name: 'text',
64       type: typeFor('Text'),
65       desc: 'Email represented in a text ('text/plain') format.',
66       showWhen: {
67         key: 'contentType',
68         value: 'text/plain',
69       },
70     },
71     {
72       name: 'html',
73       label: 'HTML',
74       type: typeFor('Text'),
75       desc: 'Email represented in HTML ('text/html') format, you can add variables such as '{company}' to be replaced.',
76       showWhen: {
77         key: 'contentType',
78         value: 'text/html',
79       },
80     },
81     {
82       name: 'variables',
83       label: 'Replace Variables',
84       type: typeFor('TableEmail: Text, field: Text, value: Text'),
85       desc: 'Dynamic variables that should be replaced on the email. For example, a field of 'company' and a value of 'Monkodo' will replace '{company}' in the template with 'Monkodo'.',
86       optional: true,
87       properties: [
88         {
89           name: 'email',
90         },
91         {
92           name: 'field',
93         },
94         {
95           name: 'value',
96         },
97       ],
98     },
99   ],
100   outputs: [
101     {
102       name: 'messageId',
103       type: typeFor('Text'),
104     },
105   ],
106   run: async (inputs) => {
107     let rawInputs = _mapValues(inputs, (v) => v.value);
108     rawInputs.to = rawInputs.to.map(({email, name}: {string, string}) => ({email, name}));
109
110     if (rawInputs.variables) {
111       let variables: {email: string; substitutions: {var: string; value: string}[]} = [];
112       for (const {email, field, value} of rawInputs.variables) {
113         const existingVariable = variables.find((variable) => variable.email === email);
114
115         if (existingVariable) {
116           existingVariable.substitutions.push({var: field, value});
117         } else {
118           variables.push({email, substitutions: [{var: field, value}]});
119         }
120       }
121       rawInputs.variables = variables;
122     }
123
124     const credentials = await MailerSendIntegration.getCredentialsFromConnection(
125       rawInputs['$connection'],
126     );
127
128     const response = await axios.post('https://api.mailerSend.com/v1/email', rawInputs, {
129       headers: {
130         Authorization: `Bearer ${credentials.api_token}`,
131       },
132     });
133
134     return {
135       name: 'Result',
136       outputs: {
137         messageId: valueAs(typeFor('Text'), response.headers['x-message-id']),
138       },
139     };
140   },
141 };
142
```

Image II
Code sample to integration

```
1 import axios from 'axios';
2
3 import { APIKeyIntegration } from '@integrations/api-key-integration';
4 import { CredentialInfo } from '@integrations/api-types';
5 import { ConnectionModel } from '@database/connection';
6 import { PDFFunctionInputs } from '@automation/api-types';
7 import { convertToOptions } from './helper';
8
9 export default class PhraseIntegration extends APIKeyIntegration {
10   apiKey = 'phrase';
11   appName = 'Phrase';
12
13   getCredentialInfo(): CredentialInfo {
14     return {
15       appName: this.appName,
16       fields: [{ name: 'access_token' }],
17       desc: credentialsDesc,
18     };
19   }
20 }
21
22 const credentialsDesc = `To connect your **Phrase account** with **Monkodo**:  

23 - [Sign in](https://eu.phrase.com/idm-ui/signin) to your **Phrase Account**. If you haven't registered yet, you can [create a new account](https://eu.phrase.com/idm-ui/signup).  

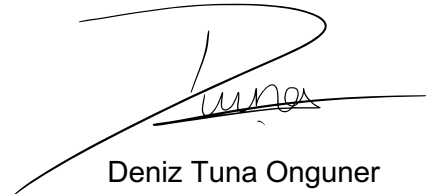
24 - Once you're signed in, head over to your `Strings account/product`.  

25 - Within your **Strings account**, access the `Strings` section from the Dashboard. Then, proceed to the upper-right corner/icon and select `Settings > Profile`.  

26 - In your **Profile settings**, locate the `Access Tokens` tab. Click on the `Generate Token` button to create a new access token.`;
27
28 const apiURL = 'https://api.phrase.com/v2';
29
30 export async function getAccessToken(connectionId: string): Promise<string> {
31   const connection = await ConnectionModel.findById(connectionId);
32   return connection!.keys!.access_token;
33 }
34
35 export async function getAccounts(params: AccountParams, connectionId: string): Promise<any[]> {
36   const access_token = await getAccessToken(connectionId);
37   params.access_token = access_token;
38
39   const response = await axios.get(`${apiURL}/accounts`, {
40     params,
41   });
42
43   return response.data;
44 }
45
46 export async function getProjects(params: ProjectParams, connectionId: string): Promise<any[]> {
47   const access_token = await getAccessToken(connectionId);
48   params.access_token = access_token;
49
50   const response = await axios.get(`${apiURL}/projects`, {
51     params,
52   });
53
54   return response.data;
55 }
56
57 export async function getAccountIDs(inputs: PDFFunctionInputs) {
58   const accounts = await getAccounts(inputs as AccountParams, inputs.$connection);
59   return convertToOptions(accounts);
60 }
61
62 export type AccountParams = {
63   access_token: string;
64   page: number;
65   per_page: number;
66 };
67
68 export type ProjectParams = {
69   access_token: string;
70   page: number;
71   per_page: number;
72   account_id?: string;
73   sort_by?: string;
74   filter?: string;
75 };
76
```

Self-Checklist for this Report

- ☒ Did I provide detailed information about the work I did?
- ☒ Is supervisor information included?
- ☒ Did I use the Report Template to prepare my report, so that it has a cover page, has all sections and subsections specified in the Table of Contents, and uses the required section names?
- ☒ Did I follow the style guidelines?
- ☒ Does my report look professionally written?
- ☒ Does my report include all necessary References, and proper citations to them in the body?
- ☒ Did I remove all explanations from the Report Template, which are marked with yellow color? Did I modify all text marked with green according to my case?



Deniz Tuna Onguner