Industrial Project

Tree Adoption Application for MoT

Final Documentation

Computer Science Department

City College, International Faculty of University of Sheffield

Antonios Antoniadis

Evangelos Barmpas

Lorik Korça

Rei Manushi

Vijon Baraku

February, 2022

Abstract

This paper is the final iteration of the industrial project. Specifically, the development of the tree adoption application for the Municipality of Thessaloniki. The iteration discusses the progress made since the project was given to our team. Ideas discussed, user requirements, front and back end design. Moreover, the paper discusses ideas for what needs to be completed by the next team.

Contents

1 Revision History

Date	Version	Description	Author
18/10/2021	1.0	Creating the documentation report	All group members.
19/10/2021	1.0	User requirement section.	All group members.
21/10/2021	1.0	Finalyzing use case diagram and its description	All group members.
25/10/2021	1.0	UI prototypes for main functionalities.	All group members.
26/10/2021	1.0	Finishing the documentation of the UI prototypes.	All group members.
2/11/2021	1.0	Began experimenting with React and Material UI in order to get more familiar and apply them to the project.	All group members.
3/11/2021	1.0	Started implementing a UI with React and Material UI.	All group members.
27/11/2021	2.0	New backend and frontend implementation features added to the documentation.	All group members.
29/11/2021	3.0	Tree adoption map linked with Database	Baraku, Antoniadis
29/11/2021	3.0	Sign in and sign up implementation	Antoniadis, Barmpas
7/12/2021	3.0	Tree adoption implementation	Antoniadis, Barmpas
7/12/2021	3.0	Home page connected to the database	Antoniadis
13/12/2021	3.0	Tree deletion implemented	Antoniadis, Barmpas
13/12/2021	3.0	Tree adoption map clustering implemented	Baraku
13/12/2021	3.0	Starting to work on watering	Antoniadis, Barmpas
09/01/2022	4.0	New Redesigned UI and Node JS Backend	Antoniadis
13/01/2022	4.0	Added UI screenshots.	Antoniadis, Manushi
17/01/2022	4.0	New Node JS backend REST API implementation, HTTP requests.	Antoniadis, Manushi
23/01/2022	4.0	Node JS Detailed Controllers documentation.	Korça
25/01/2022	4.0	New efficient map created.	Baraku
26/01/2022	4.0	PWA requirements	Barmpas

2 Introduction

This is the third year industrial project from the MoT team at City College. The project is being developed in accordance with the Municipality of Thessaloniki. Due to the lack of green spaces in the city, the Municipal office has proposed the development of a web application enabling the citizens to adopt trees. The adoption entails a citizen selecting a tree from the city to take care of. Overall the idea is based on prompting volunteer work from the citizens in order to protect the already small green space of the city. Besides simply facilitating the adoption process, the application will ideally include gamified features to further encourage citizen participation.

After a formal meeting with the municipality, our team was able to gather the aforementioned objectives for the application. Based on the statistics there was found to be only 2.5 square meters of green space per person in Thessaloniki. Therefore the development of this app can end up being a crucial part in the procession of the available green space. Considering that MoT is the first team in starting this project, our development team has the great responsibility of making sure that the application provides a solid foundation for the upcoming teams that are going to further improve and extend the scope and functionality of the project.

3 User Requirements and System Functionalities

As previously mentioned, the main functionality of this application is going to be the adoption of a tree from a user. By adopting a tree a user is able to take care of it. This leads to the second general functionality the application should incorporate. That would be the documentation of the user actually volunteering to care for the tree / trees that they have adopted. After having accomplished the implementation of the adoption and volunteer work documentation features, another important aspect that the user required was the gamification of the latter. The main objective behind this requirement is the idea of stimulating citizen participation by adding game features to the process of a user taking care of a tree. In forthcoming iterations, a social aspect can also be implemented, with each user seeing their friends and neighbours adopting and taking care of trees. This will encourage the users to perform their responsibilities. Below each requirement will be expanded and analysed further.

1. **Login / register :** In order for users to interact with the system, first they have to

register to the platform. This is a simple but necessary requirement because it allows for information tracking and processing of this information for further requirements.

2. **Main page :** This is the main page that the user will be visiting most frequently. It is in this feature where the user will be able to view how many trees they currently have adopted. Here the user can also see general information about the tree and when it was last watered.

3. **Tree details :** Every tree should have its own page where the user can see information about the tree, such as species, watering requirements, age etc.

4. **See map with available trees :** Through this functionality, the users will be able to open a google maps page where trees that are available for adoption are shown with their precise location on the map.

5. **Interact with map :** Each tree should be intractable and clickable so that users can adopt directly from the map.

6. **Distinguish differently-aged trees on map :** Trees less than three years old should be a different color than trees over three years of age (blue on the map).

7. **Distinguish adopted trees by other users :** These trees are unavailable and cannot be adopted (grey on the map).

8. **Distinguish your adopted trees :** These trees are the ones that you have adopted (magenta on the map).

9. **Adopt a tree :** When browsing through the map, once the user finds a tree they like best, they can click on the tree on the map and select the tree to be adopted.

10. **Tree sharing :** Make it so that the same tree can be adopted by a lot of people. (Max 2, Only will be implemented if there are a lot of users and no tree is left not adopted)

11. **Ceremonial process of adoption :** After selecting a tree to be adopted, a screen with a certificate is displayed, thus making the process of adopting a tree more ceremonial.

12. **Renaming of adopted trees :** After adopting a tree the user will be able to personalize their tree by being able to rename it according to their choice.

13. **Log watering of tree :** The user is able to easily report each time they water the tree that the tree has been watered.

14. **Adaptive watering needs :** The amount of water a tree needs should be dependent on some factors regarding the tree. Such as the season, and the age

15. **Report problem :** This functionality is used when the user notices something wrong with the tree they have adopted. For example if the tree needs to be maintained or if the tree does not look healthy, the user will report this issue to the municipality.

16. **Log removal of waste :** A feature where if the user notices piled up waste in the trees compartment, they could log that they have cleaned up the waste.

17. **Notifications :** Whenever a tree can be watered again, send a notification to the user as a reminder to water the tree. In case someone has not watered the tree for a long time, send a reminder notification too.

18. **Abandon tree :** If someone doesn't wish to take care of a tree for whatever reason, they should be able to abandon the tree, at which point the tree gets returned into the pool of available trees.

19. **Max limit of adopted trees :** There should be a limit to how many trees a user can adopt.

20. **Make it both mobile and desktop friendly :** The desktop website should be optimized for computer monitors while the mobile version should be a progressive web application, which is formatted for mobile devices.

21. **Make it a mobile app :** Turn the application into a downloadable mobile app through PWAs.

22. **Deploy the application :** The application should be deployed so feedback can be gathered and people can use it.

23. **Contact us page :** Add a page with contact information of the municipality.

24. **Administration page :** System administrators should have their own home page where they can manage the application as well as other users. Possible actions include:

25. **Revoke trees :** If users are neglecting their trees, the administrator should be able to revoke their tree privileges

26. **Respond to requests :** Administrators should be able to see problems reported by users, resolve them and get back at the users either with a notification of success, or a query for more information.

27. **Make other users admins :** Administrators should be able to christen other users as administrators.

28. **Weather tracking :** Track the weather so that in case it rains, the watering can auto-update to indicate that the trees got watered by the rain.

29. **Gamification aspect :** This feature is an extension to the features regarding taking care of the tree. These elements would serve as an extra motivation for a volunteer to take care of their trees. There are many ways that this could be done. For example :

30. **Points :** Whenever a user performs an action, they earn points. An action can be anything from watering a tree, reporting a problem or simply logging in the app for the day.

31. **Prizes :** Points can be exchanged for prizes. These prizes could range from in-app rewards to real life ones. However, this will depend on what the municipality is willing to give away.

32. **Competition :** Competitive aspects could be also integrated into the app, such as leaderboards.

33. **Social aspect :** The application could also serve as a hub for socializing between volunteers. Some examples are :

34. **See other adoptions :** See which tree is adopted by whom

35. **Commenting :** Commenting on each-others' trees

36. **Regional scoring :** Combined neighborhood scores

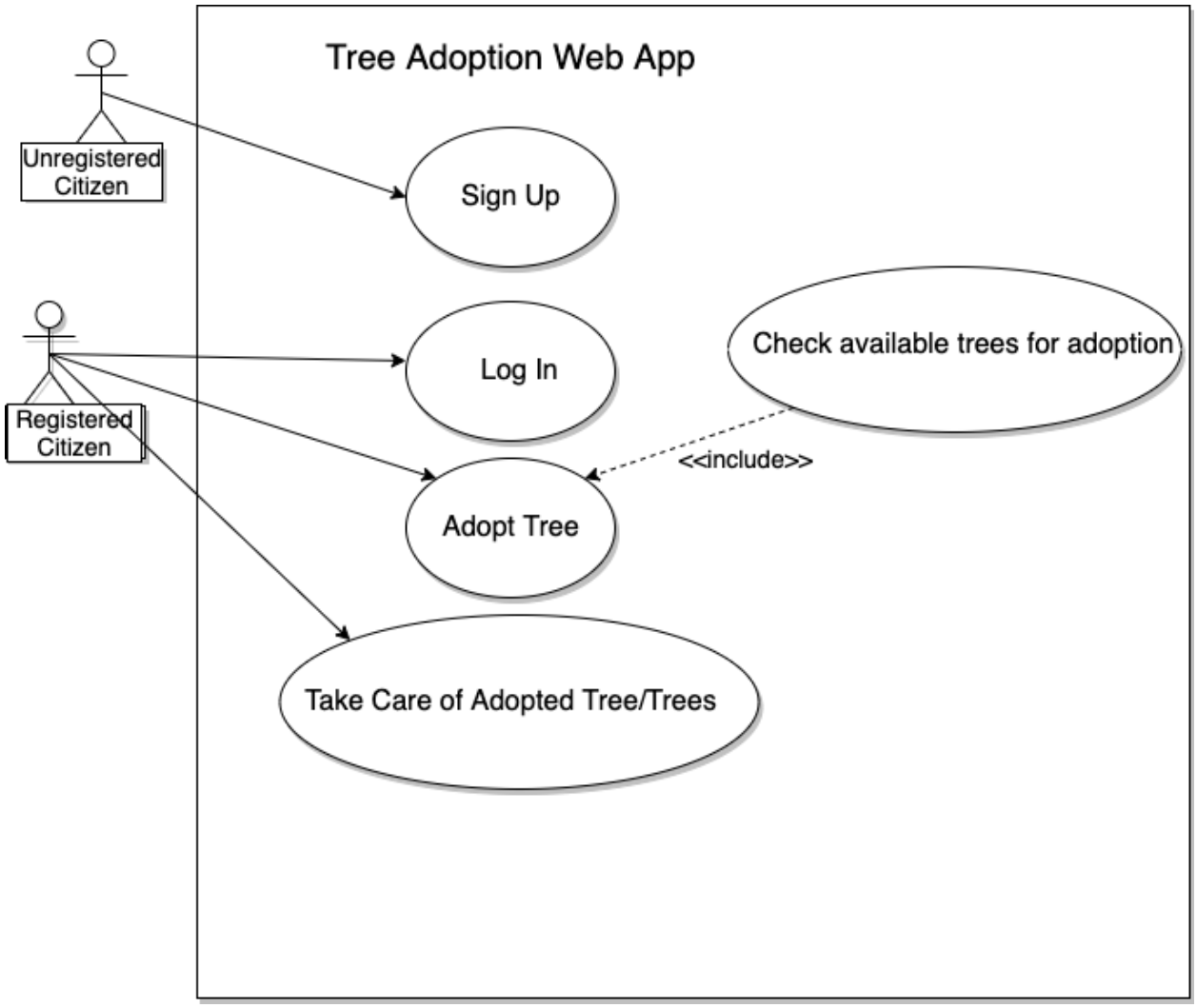
3.1 Requirement priority

To decide what the team will work on first, a table with the priority of the tasks can be seen below.

Requirement	Hours	Importance	Implemented
Login / register	5	Very high	Yes
My trees page	4	Very high	Yes
Adopt a tree	6	Very high	Yes
Make it both mobile and desktop friendly	20	Very High	Yes
Tree details	2	High	Yes
See map with available trees	8	High	Yes
Have interactable map	8	High	Yes
Distinguish differently-aged trees on map	8	High	Yes
Tree adoption confirmation screen	9	High	Yes
Log watering of tree	7	High	Yes
Report problem	6	High	No
Distinguish already adopted(unavailable trees)	8	High	Yes
Distinguish your adopted trees from the rest of the trees	8	High	Yes
Abandon tree	2	High	Yes
Have a landing page	3	High	No
Make it a mobile application	8	High	Yes
Deploy the appliction	8	High	Yes
Administration page	12	High	No
Revoke trees	4	High	No
Respond to requests	10	High	No
Ceremonial process of adoption	3	Medium	No
Log removal of waste	4	Medium	No
Max limit of adopted trees	1	Medium	Yes
Contact us page	1	Medium	No
Tree sharing	3	Low	No
Renaming of adopted trees	3	Low	No
Notifications	7	Low	No
Make other users admins	1	Low	No
Adaptive watering needs	14	Very low	No
Weather tracking	6	Very low	No
Gamification aspect	16	Very low	No
Points	4	Very low	No
Prizes	4	Very low	No
Competition	8	Very low	No
Social aspect	16	Very low	No
See other adoptions	4	Very low	No
Commenting	4	Very low	No
Regional scoring	8	Very low	No

Tasks under high priority, our team considers to be crucial to the system. These tasks will be worked on first, and as it can be seen, those were the ones worked on first. Once tasks of the higher priority are finished, the tasks under lower priorities will be implemented and tasks under very low priority could be finished in the next year when the project goes into its second cycle.

3.2 Use Cases Diagram



The above diagram is used to represent how the users can actually interact with the application. An unregistered citizen can only be able to actually sign up by creating an account in the app. Nothing else can be done without having an account. On the other hand there are the registered users, citizens who have created an account. These users can log into their account and actually be able to adopt a tree. Part of this process will also be

looking at the available trees for adoption. A registered user can also be able to take care of their adopted trees by logging the activities they performed during the care taking process. It is also assumed that a citizen can have the ability of adopting more than one tree.

4 Frontend

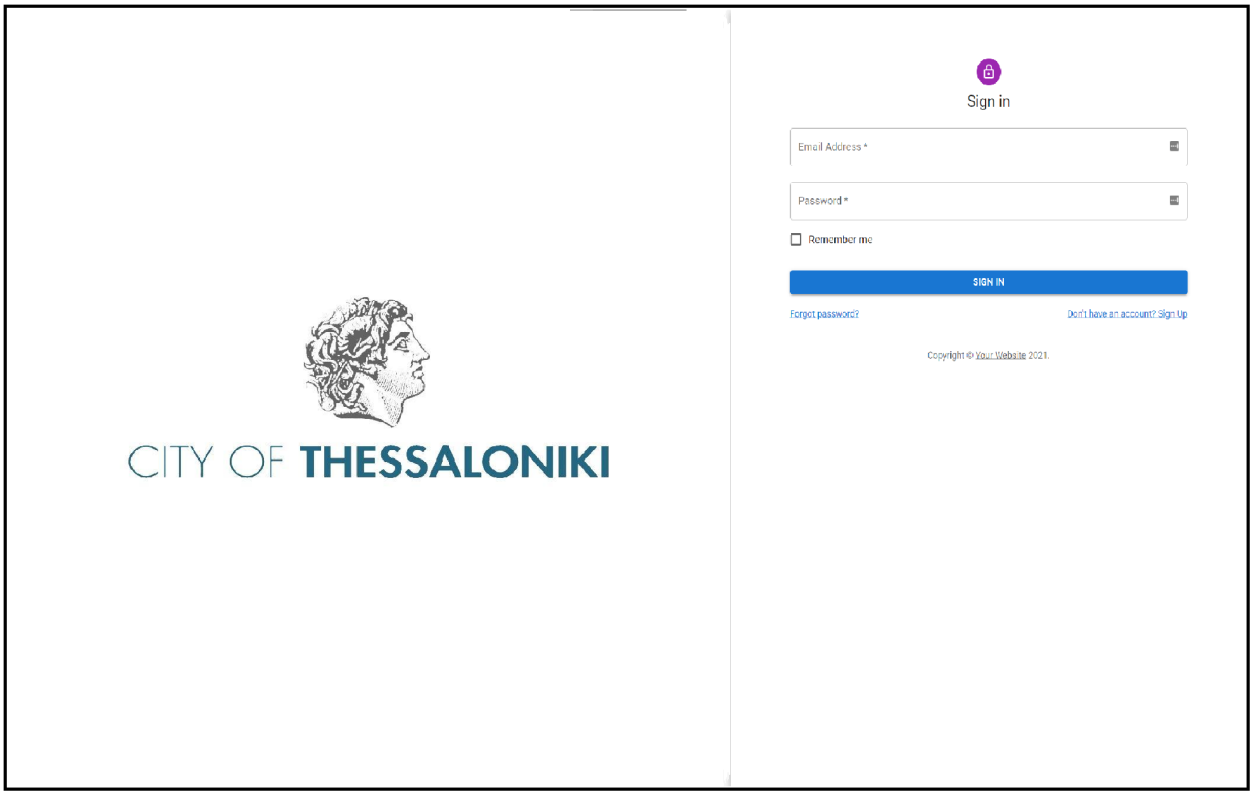
Considering the fact that this application will have a very broad target audience from young to senior citizens, one should not neglect how critical it is that a user-friendly UI is set in place. This is the part of the application that the users are going to actually interact with. For this reason its design should be simple and straightforward. Below initial ideas have been explored for the first UI prototypes. These initial prototypes display the basic user journey with the main functionalities defined in the previous section.

The frontend was iterated and improved on each cycle by gathering feedback from the professor and the client. This improvement is demonstrated below:

4.1 Iteration 1 - Initial UI Prototypes

Before writing code, the team developed some initial ideas about how the UI will look like.

4.1.1 Log in and Sing Up



Both login and signup will have a simple design displaying the logo of the municipal office. The necessary text fields will be found in the forms that require the necessary user input for both procedures.

4.1.2 Main page

This is the first page a user will be once they are logged in the application. In here they will be able to see all of their adopted trees and various information regarding them. For example the last time one of the trees was actually watered. At the same time in this page a trash shaped button will be used in case the user wants to actually remove one of the trees from their adoption list. This would set that particular tree as an available tree for adoption yet again. At the same time this page also provides a button that allows the user to add another tree to their adoption list. The status bars in blue are not buttons, they actually let the user know that a tree has been watered enough. The reb status bar is also a button and is clickable, which enables the user to log themselves watering their particular tree. In other words, this section of the app will enable the functionality of the user logging the actions they take in order to take care of their adopted trees. Lastly, because the target audience are greek citizens, the language first implemented will be greek.

4.1.3 Adopting a Tree

After pressing the adoption button the user will be directed to another page of the application, which consists of a map displaying all of the trees in the city. Trees that are available for adoption are represented by the little green circles, while the others that are taken are grey. By clicking or tapping into one of the green circles, the above pop up screen appears. This will allow the user to confirm the adoption of the particular tree they pressed on.

4.2 Iteration 2 and 3 - Implementing the frontend

After designing the prototypes the frontend development team started with its actual implementation. Material UI templates were used for the login and the sign up in order to speed up the process. Meanwhile the main page and tree adoption map were developed specifically to meet the needs of the application. For the moment, all the content found in both these pages is static or in other words, not yet retrieved from the actual database. This was done for testing purposes in order to understand how the content is going to be displayed to the users once the app is released.

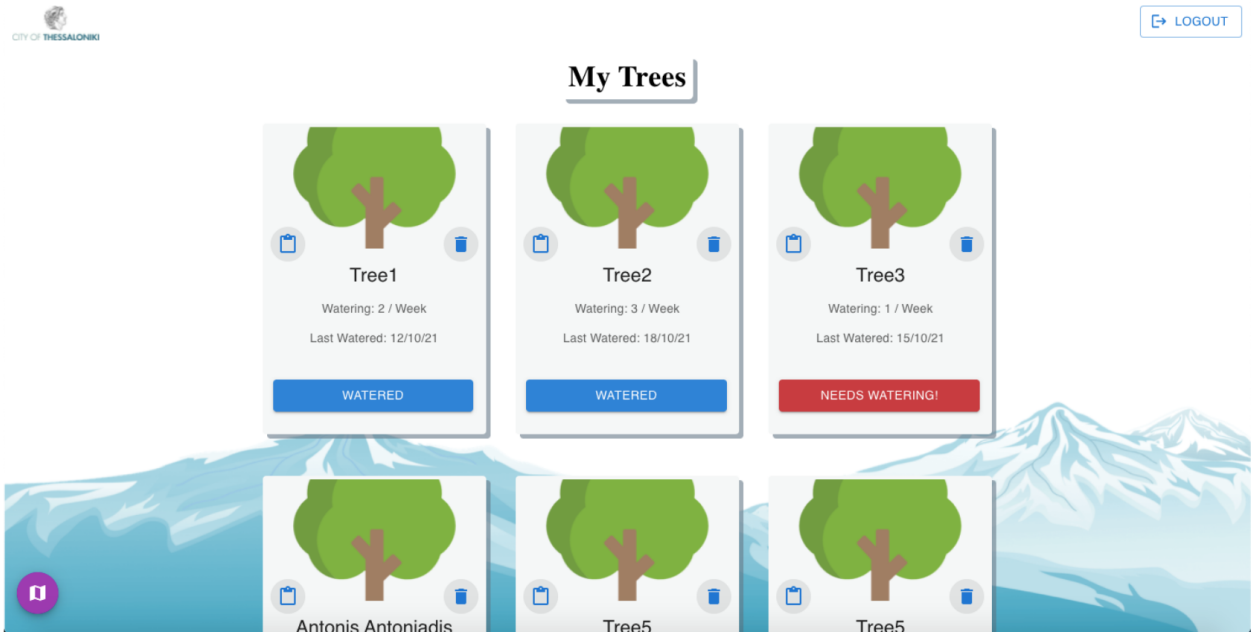


Figure 1:

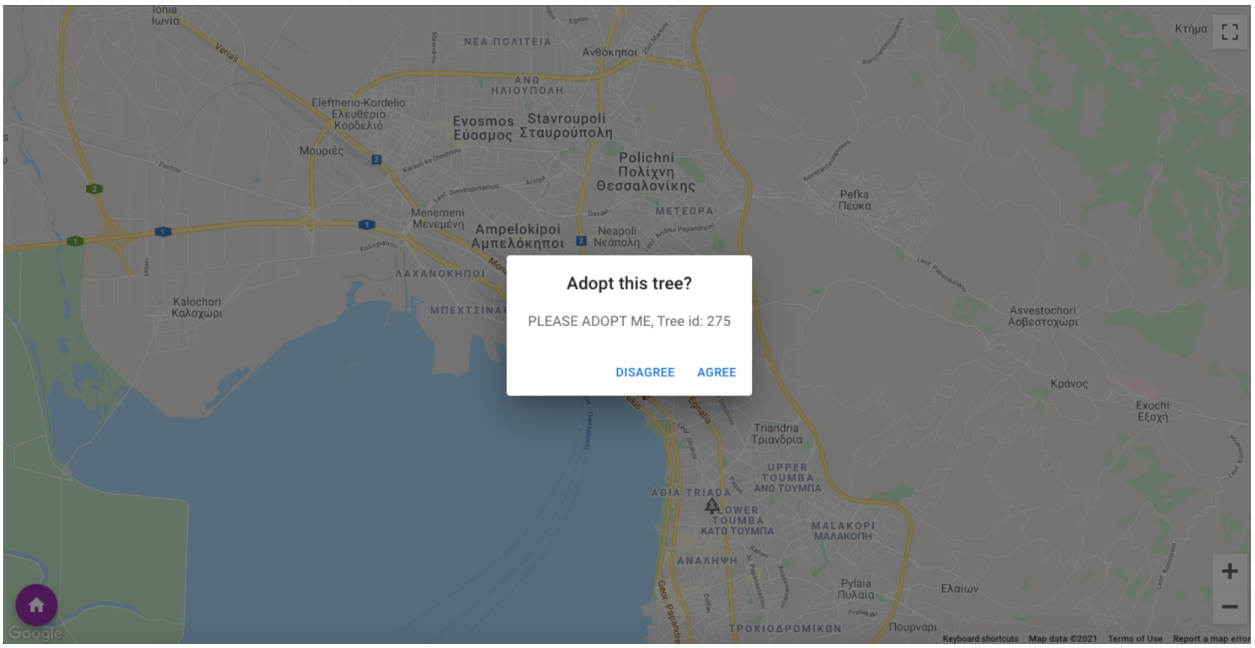


Figure 2:

In figures 1 and 2 one can observe how the main page will be displayed to the user on a desktop format. Considering that the application will also be a progressive web app it

was crucial that the necessary measures be taken so that it could be compatible with mobile format. Instances of this compatibility can be seen in the screenshots below:

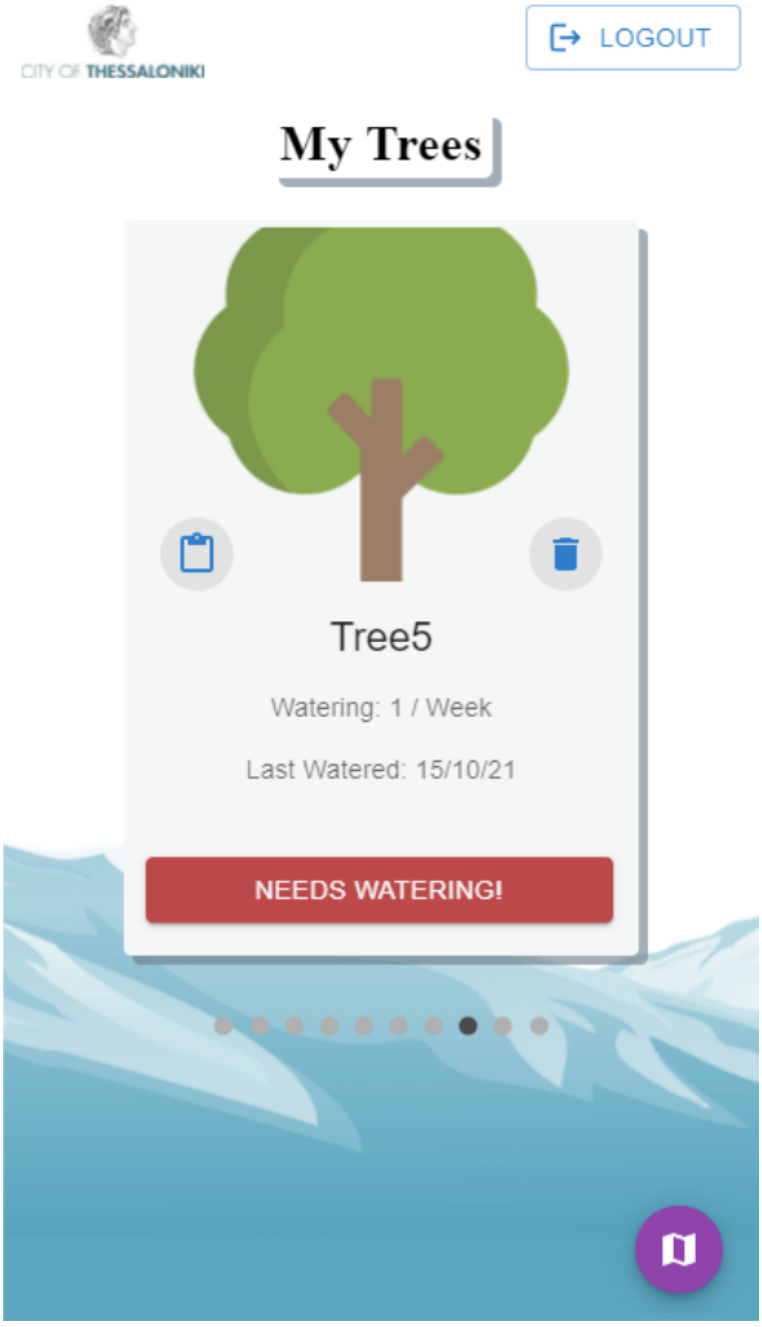


Figure 3:

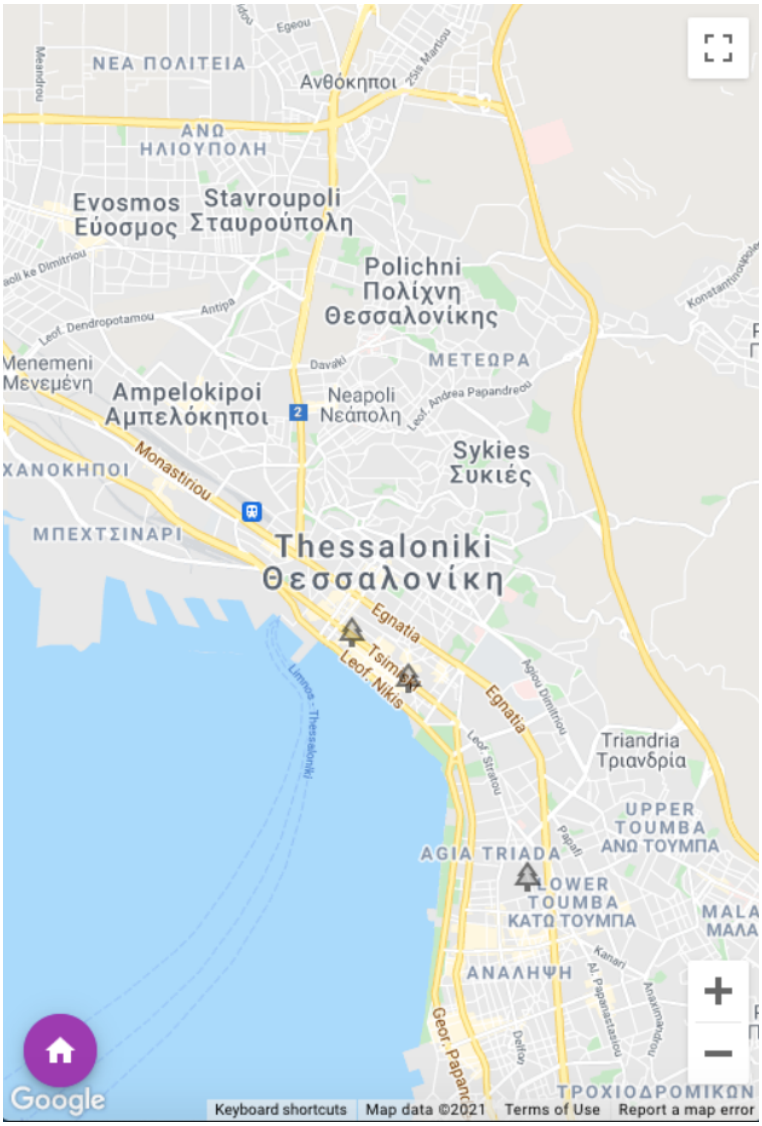


Figure 4:

Another major update to the frontend has been the clustering of the available trees in the map of Thessaloniki. This method provides a more sophisticated way of presenting the available trees for adoption to the users. By clicking or tapping on the clustered circles the users continually zoom until they get the detailed street view and tree positions on the area that they are interested in. A clustered view is able to remove the lagging from the application especially when a big amount of data is to be displayed.

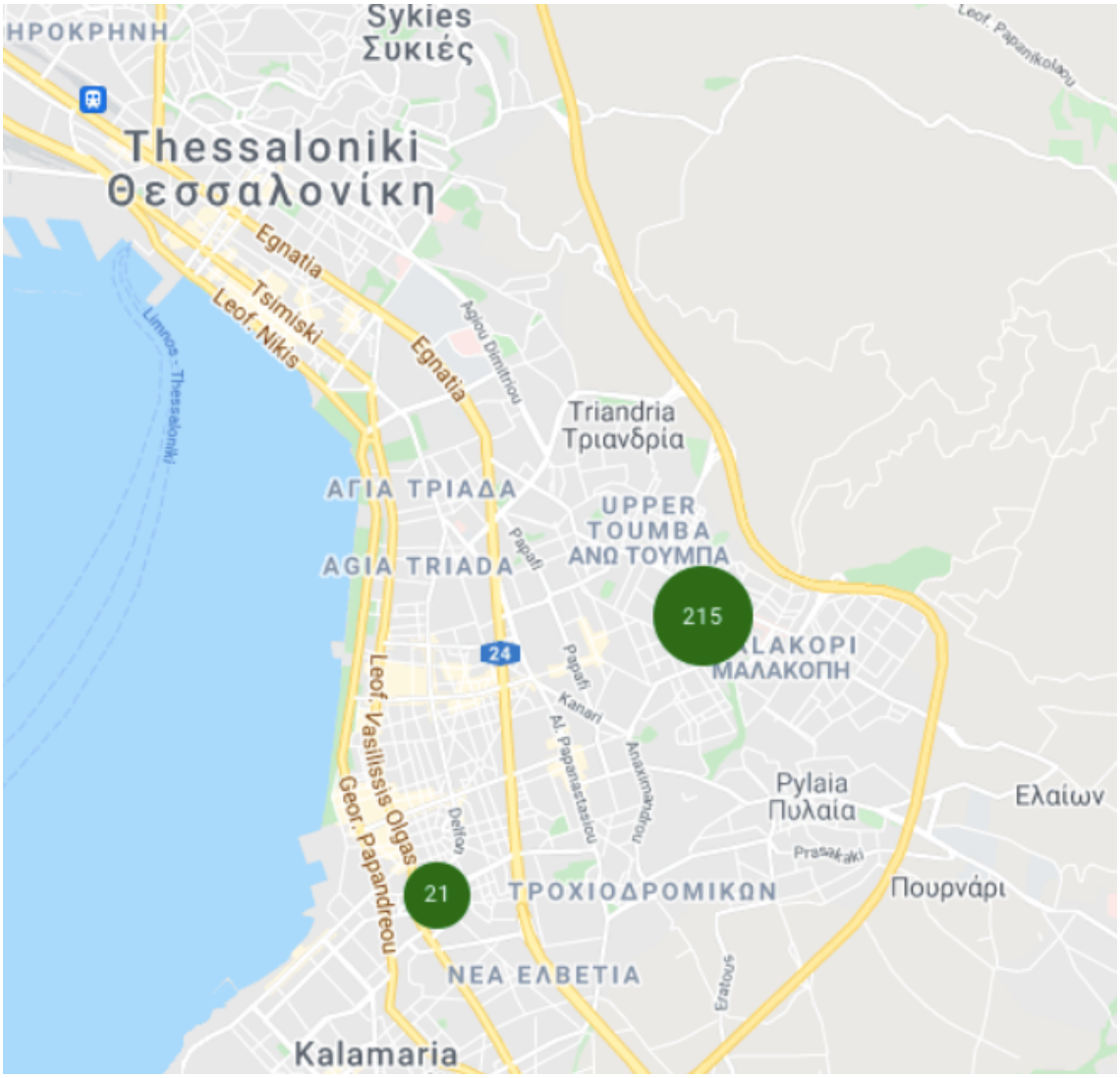


Figure 5:

4.3 Iteration 4 - Redesigned Frontend

The development team felt the need that a new design should be implemented in the final stages of the project. This new design was purposely put in place to give the application a more modern look and feel.

A modern application layout is important in order to provide the users with a better experience and overall satisfaction while using the product. Despite the initial prototypes being simple and easy to use, they still lacked the feel of a complete application that is ready to be deployed in the market. For this reason it was decided that a redesign of the frontend

would positively affect the overall project.

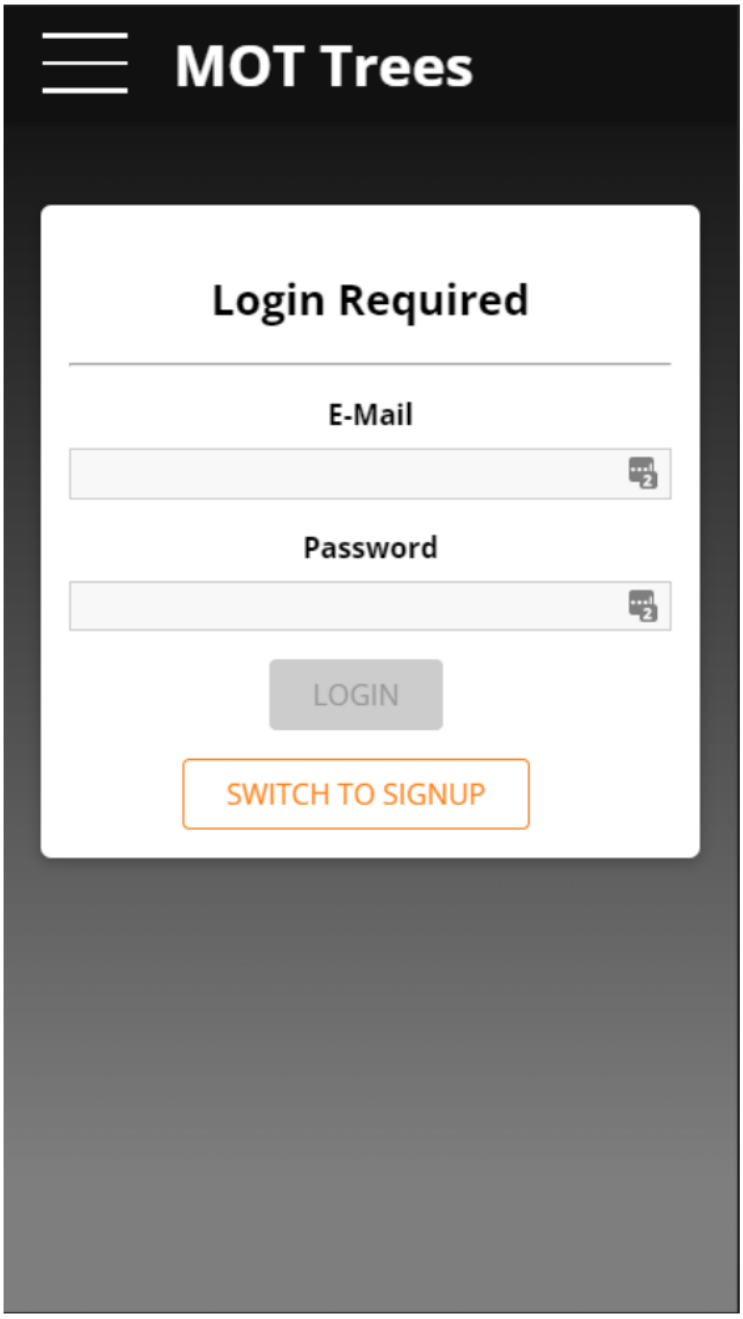


Figure 6: New login screen

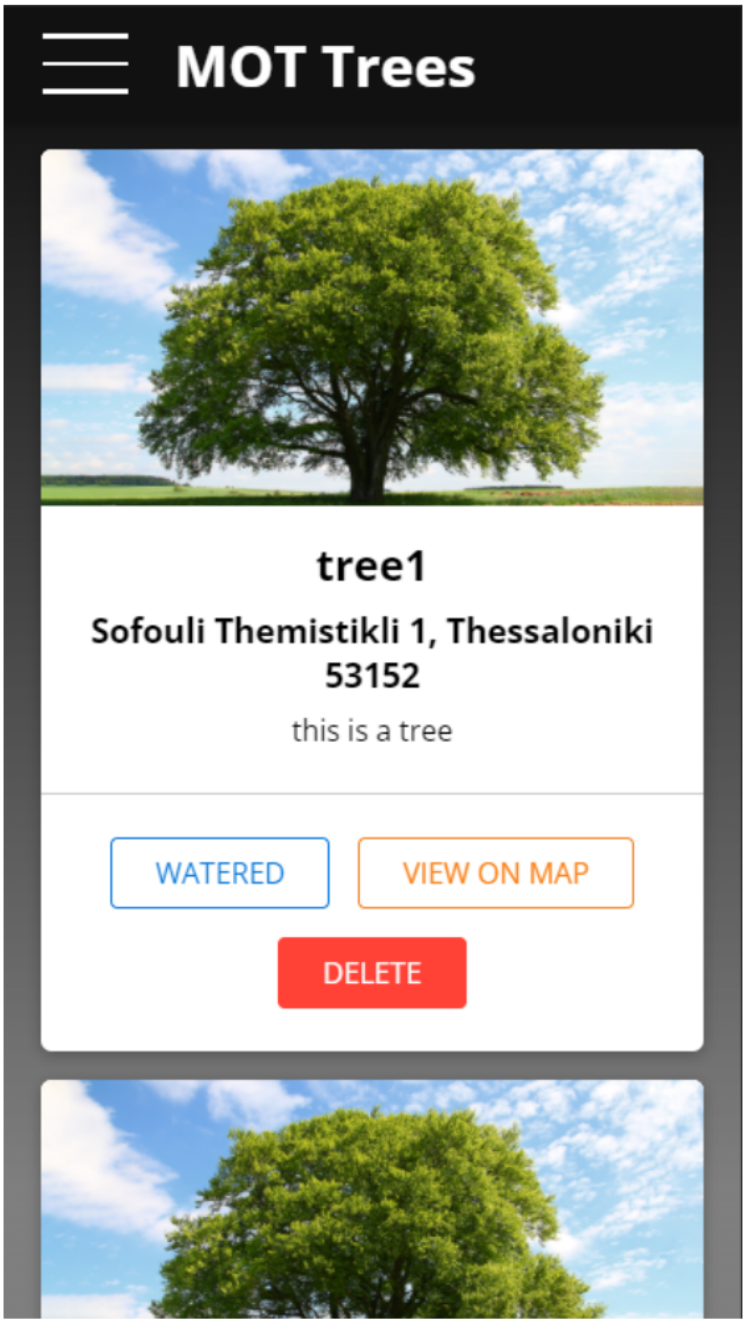


Figure 7: New main page showing user trees

The new main page has also a new implemented feature that allows a user to view the location of one of their specific trees on a map. This can be achieved by the user pressing on the view on map button. After pressing on the button a pop up appears with a map view displaying the location of the tree. This functionality can be seen in detail in 8 below.

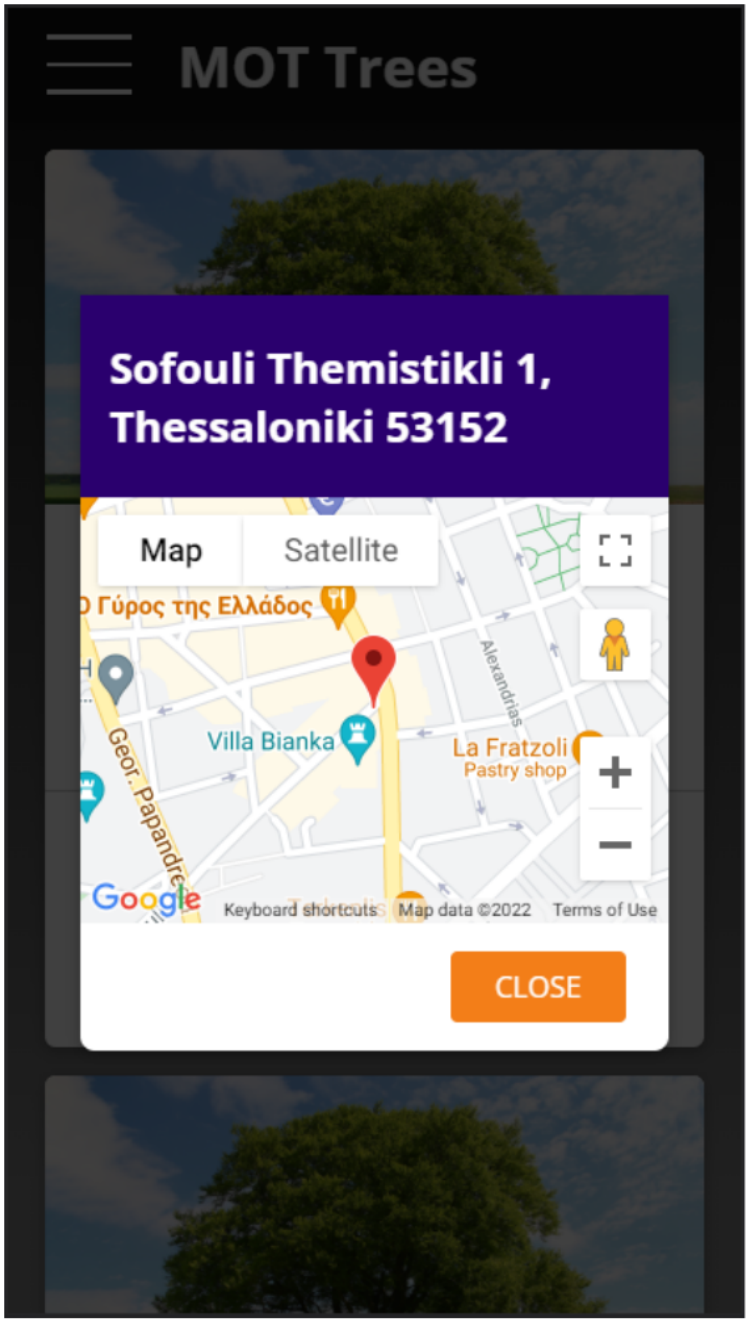


Figure 8: Location pop up for a specific tree

This functionality is added in order to allow a user to view their tree location easier in case they wish to adopt multiple trees that might not be in their vicinity. Therefore allowing the user to be able to adopt trees from different areas of the city without worrying about remembering where their specific trees are located.

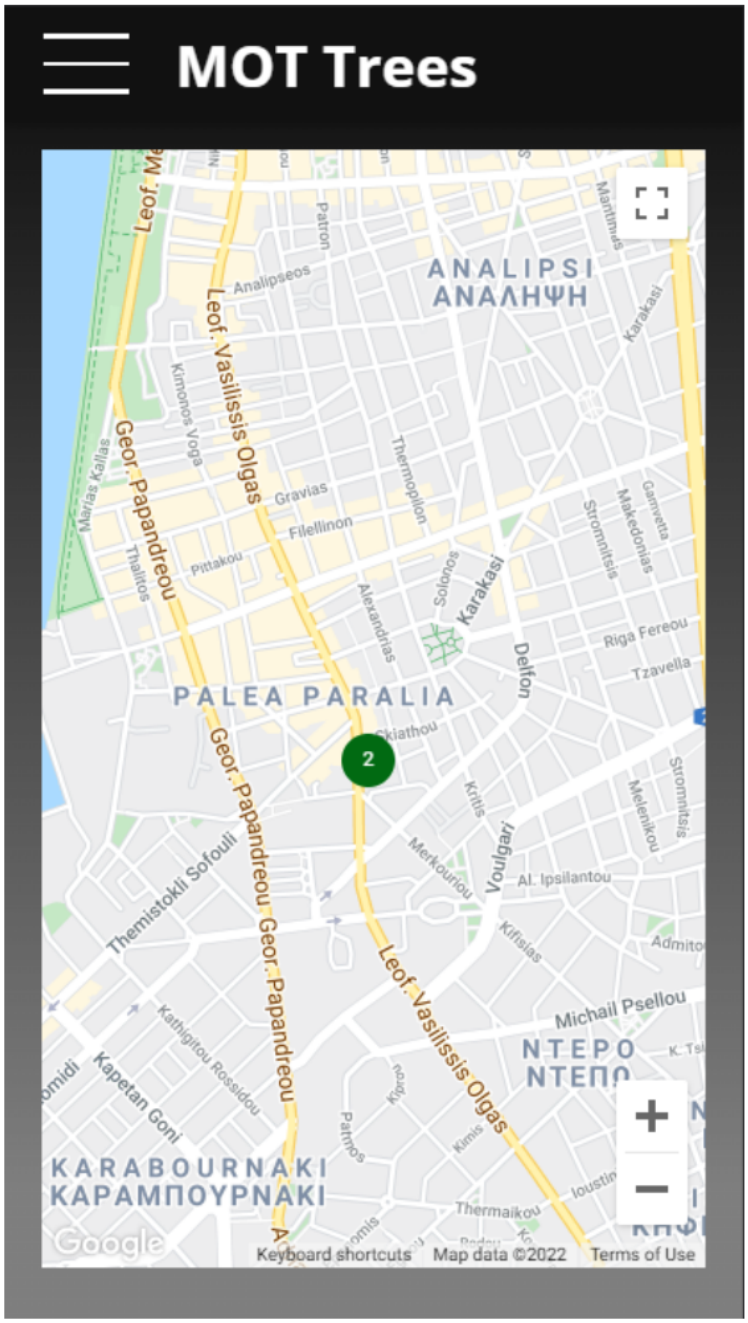


Figure 9: Map view

The tree map allowing users to adopt trees from around the city has also a new implemented feature. Besides clustering trees in order to load them easier without lagging the map also provides a new way to differentiate between new trees and old trees based on the date they were planted. This functionality can be observed below.

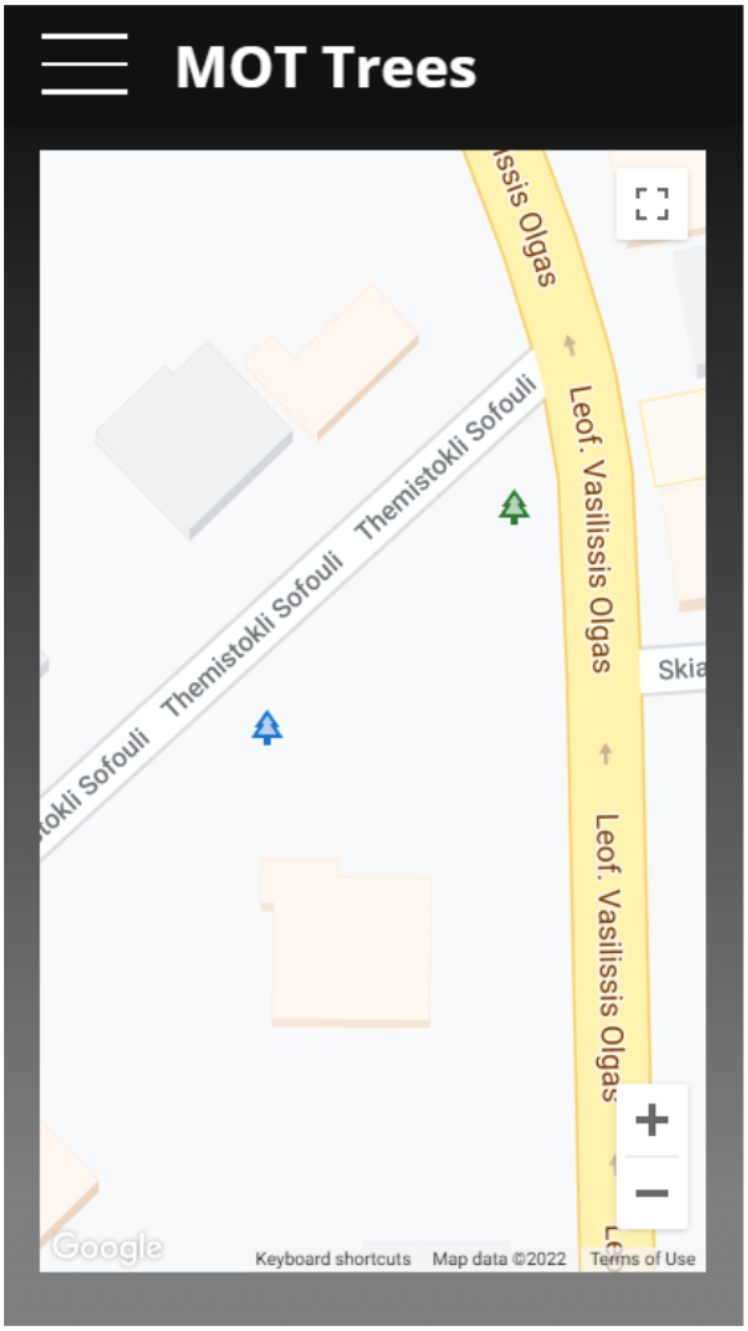


Figure 10: Tree age differentiation in the map view

The old trees are displayed in green while the young ones are displayed in a blue color. This feature was a new requirement specified by the client, due to the worry that new trees require extra attention in order to grow into healthy ones in the future. In other words, the user should be able to understand that adopting a young tree comes with a greater

responsibility and greater risk as opposed to adopting an old tree. Meanwhile the rest of the functionalities remain the same with the main differences being the new UI and Node JS backend.

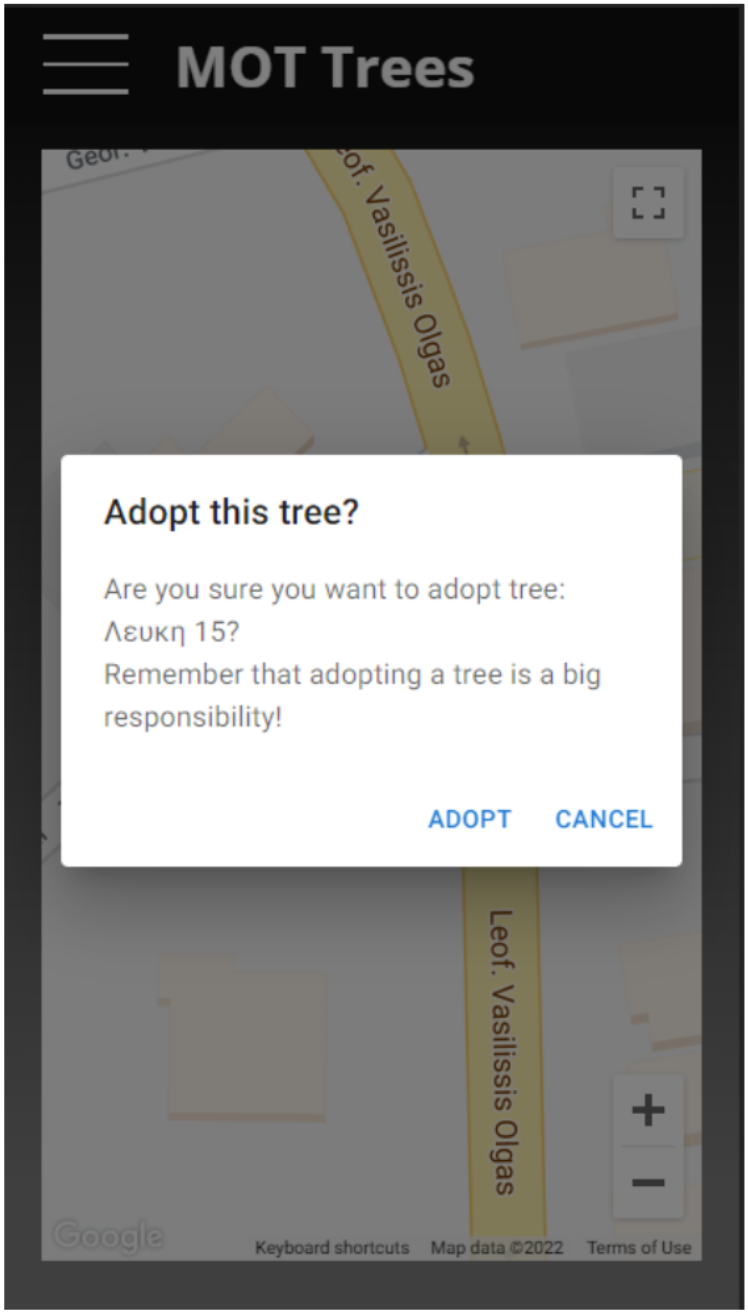


Figure 11: Adoption pop up

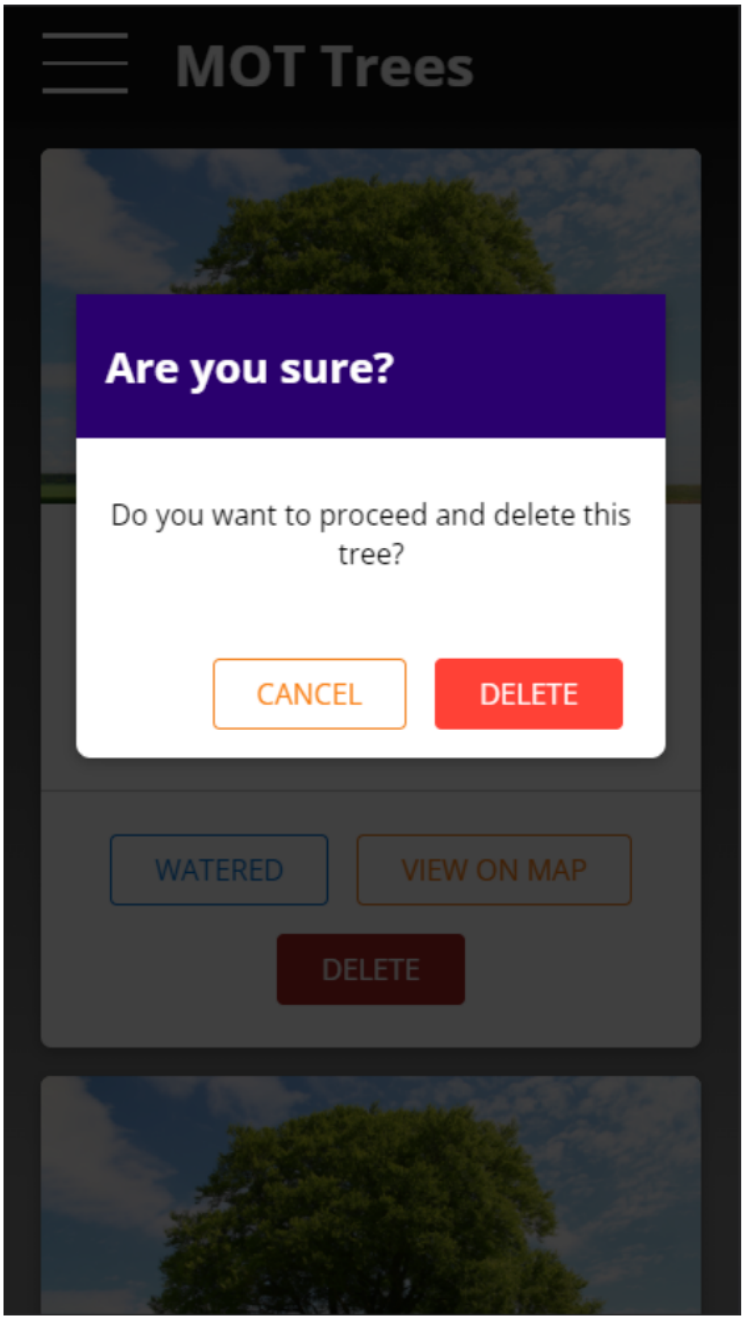


Figure 12: Tree removal pop up

Due to the new functionalities it was decided that a new way to navigate the different sections of the application was also required. This new navigation would make it easier for the user to move around the different pages in the application and at the same time would make it clearer for them to understand in which section they are located. Below is an

example of side drawer that allows the user to move in different parts of the application

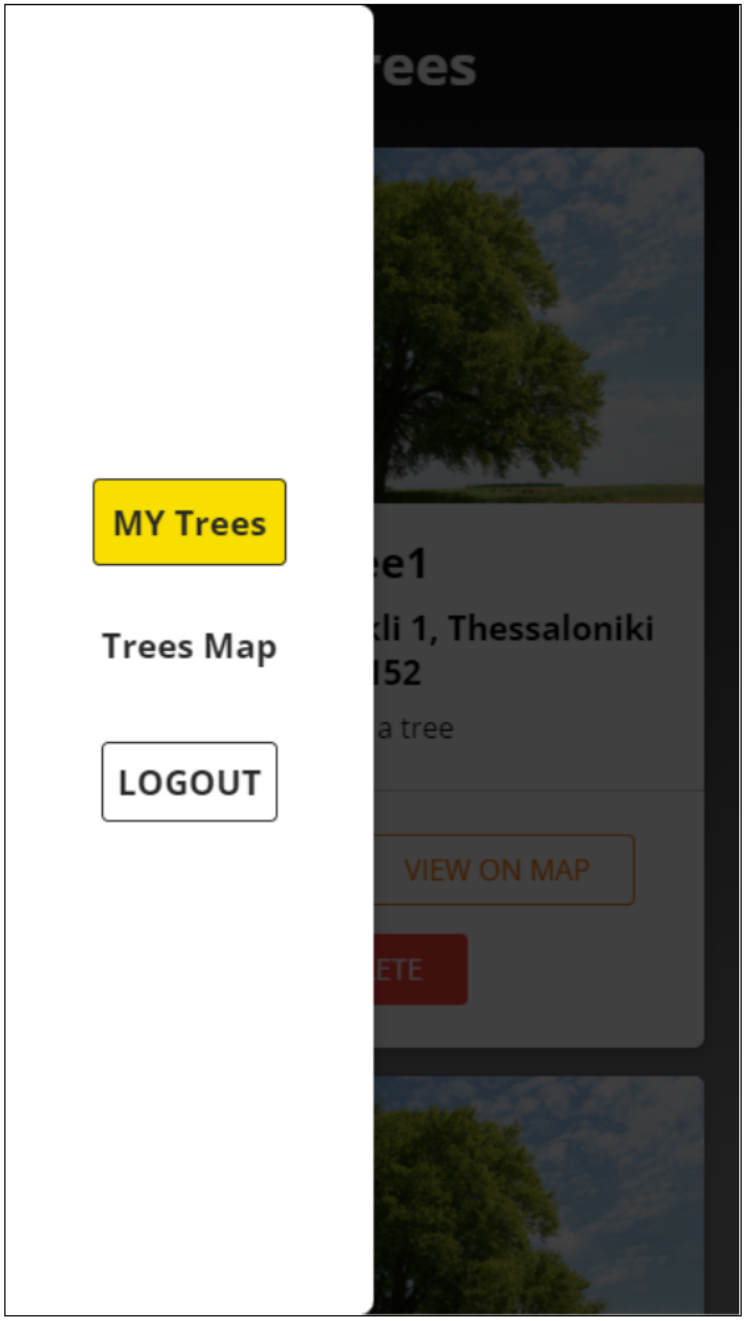


Figure 13: Side drawer navigation

In order for this side drawer to show up the user must press on the top left side menu button found in all of the pages of the application. Therefore keeping the navigation pattern consistent throughout the application.

4.4 Final Iteration - Efficiency, distinguishing, Greek language

This is the final iteration and how the current state of the application is when the first team delivers the product. The biggest issue that was fixed from iteration 4, is efficiency. Although clustering was an effective solution for a couple of thousand trees, when 38k were placed on the map the efficiency dropped a lot. The map became near impossible to run smoothly and a new approach to loading the data needed to be found.

4.4.1 Deck.gl

Deck.gl was the solution to that problem. Deck is a WebGL-powered framework for visual exploratory data analysis specifically designed for large datasets. The difference in efficiency comes in the way they are implemented. On previous iterations the markers were put directly on the map resulting in one map holding a lot of data. This became inefficient as the data set was too large. Although Deck.gl still uses Google Maps Api, it differs in implementation. Deck uses a two-layer approach, so it generates one layer with the data (dots for trees) and sticks it to the second layer (the map). This means that the map can load easily and the data sticks to it on the second layer. More on this under code documentation under section 4.4.2.

Moreover, in this iteration the distinguishing of trees on the map had to be redesigned and improved. Currently the map distinguishes between:

- Available trees (green)
- Unavailable trees adopted by someone else (grey)
- Young trees (blue)
- Your adopted trees (magenta)

Additionally, while waiting for the backend to fetch the trees from the database a loading screen was also implemented

These implementations can be seen below:

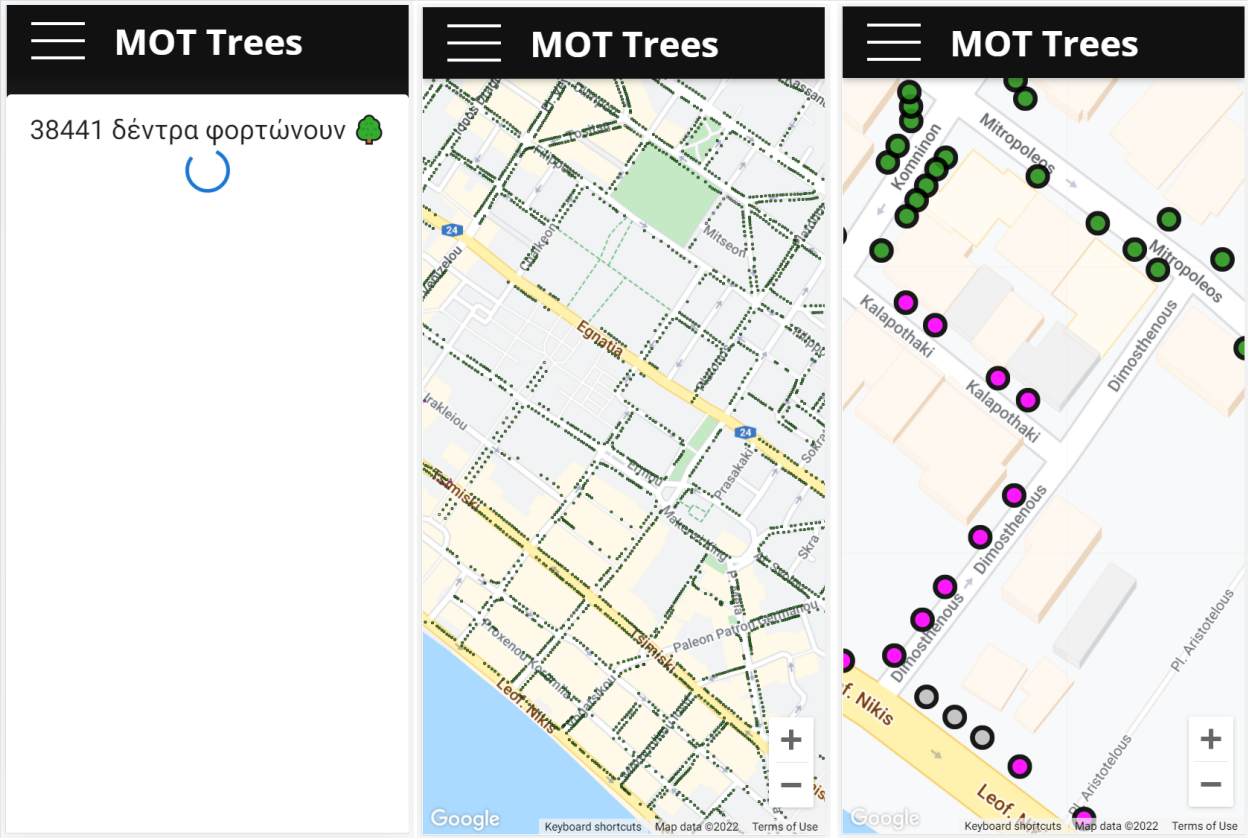


Figure 14: Loading screen, Deck.Gl implementation, and tree distinguishing

Finally since the system was designed for the Greek municipality the whole application was translated to greek and can be seen below:

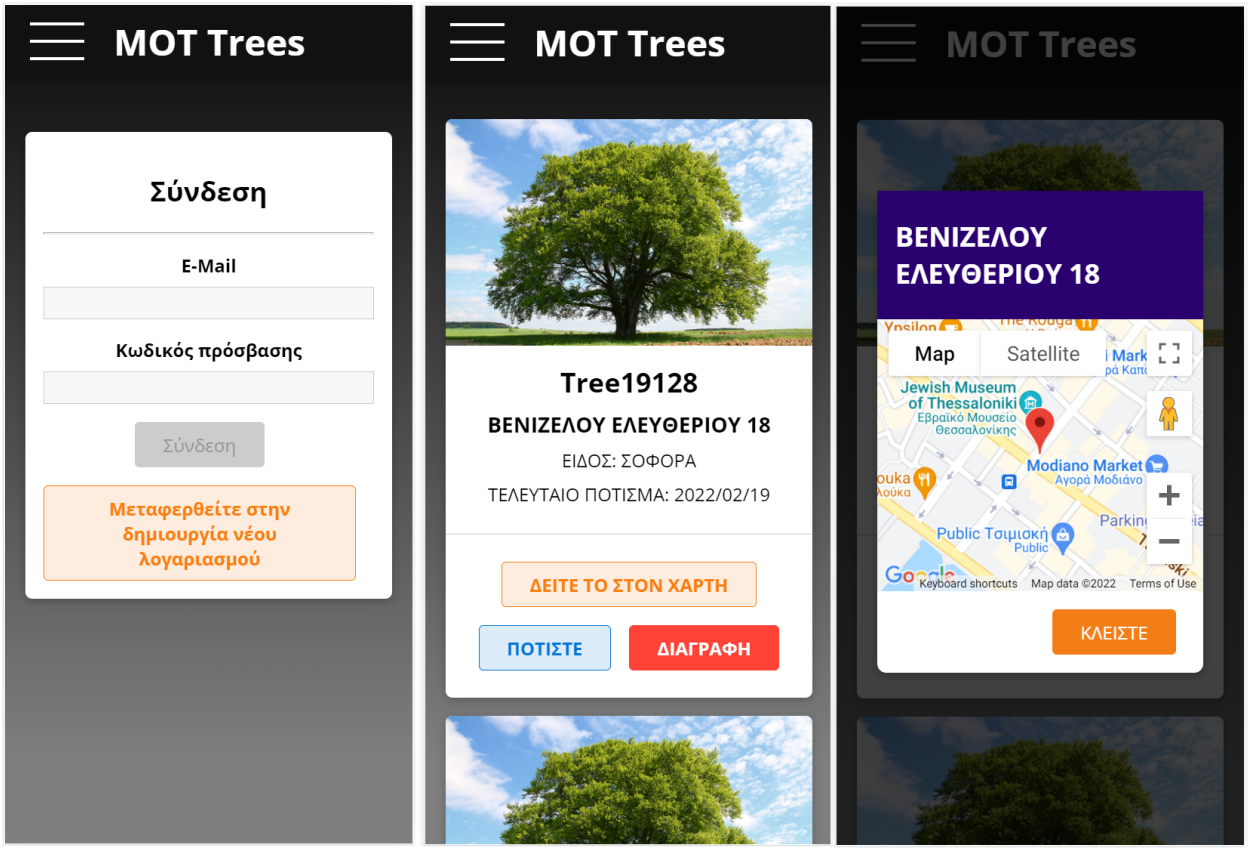


Figure 15: Greek translation

4.4.2 Code documentation

Since this is the final iteration, a more detailed code documentation for the front-end will be presented for the next teams to read.

The frontend directory contains the following folders:

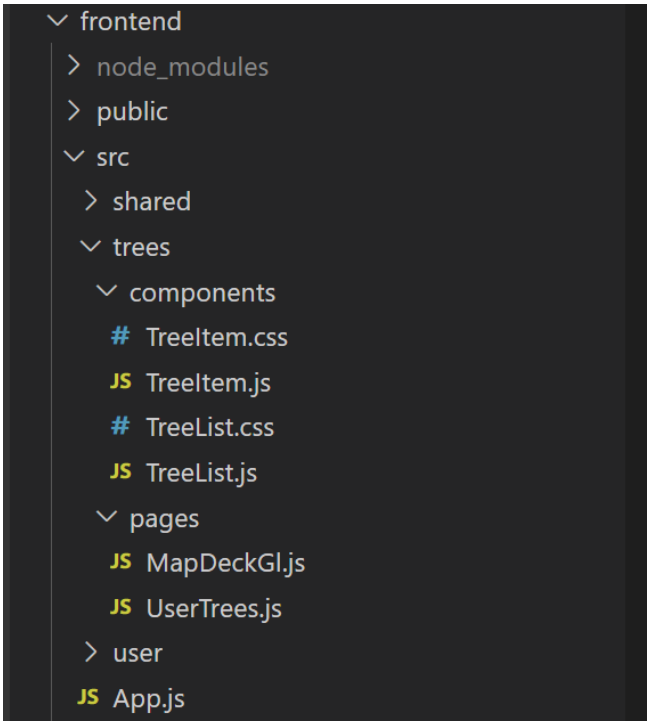


Figure 16: Frontend directory

Consider modules to be the same as JavaScript libraries. A set of functions you want to include in your application. Here you will find. Under public, you can find logos and pictures used in the application. The src folder is where the code is and where future components will be added. TreeItem.js holds the individual trees, these are the trees that show up in the home page where users view their trees. TreeList.js takes all the trees and wraps them in a list. Combined with UserTrees.js these 3 components make up the page users see when they view their trees.

MapDeckGl.js is perhaps the most complex page in the system. This is where new technologies and frameworks come together to display each tree efficiently. Under this page, you can find the Deck.gl implementation and the Google Maps api interaction. To avoid cluttering the document with code, MapDeckGl.js page has been well commented and each function has been described in detail. If there is some confusion, a look at the comments should make things clearer.

5 Backend

5.1 Iteration 1

The backend is an aspect that will be focused more on the second iteration. However, some things need to be kept in mind when we are designing the UI and working generally on the front end. The municipality has provided our team with a spreadsheet of all the trees and their location in Thessaloniki. To use this spreadsheet, initial ideas are that the spreadsheet data should be moved to a database and a backend technology to interact with this database should be implemented.

However, one thing to keep in mind is that the municipality is working on an API for their database, meaning that at some point the project should be restructured to work with their api. With this in mind, our team is trying to design a backend that is isolated enough and can be transformed into the desired implementation whenever the municipality provides the API. Lastly, there will be an implementation of a way to store the users. To be compliant with the General Data Protection Regulation, the information stored from the users will be kept at a minimum.

5.1.1 Backend technologies considered

In the beginning the development team decided to use Firebase in order to implement the backend of the application. Firebase allowed for a serverless backend by syncing JSON data between users [1]. The problem with Firebase is that it does not allow for creating an isolated API environment. In other words if this technology was to be used in the implementation it would not allow for the incorporation of another API between the app and its database. This is a major drawback considering that the Municipality is currently working on developing their own API for the database they have shared with the development team. For this reason the team decided on considering another technology which would be better suited in order to facilitate these changes.

The next technology considered to implement the backend was Node JS. This is a JavaScript framework specifically built to handle backend requests for a web application [2]. The main reason why this technology was the next to be considered is due to its compatibility with the other UI framework that is being used for the application which is React. Different from Firebase, Node also provides a more isolated backend environment that can potentially allow for another API to be used in the future for managing the database.

5.1.2 Backend implementation

After considering both backend technologies from above, it was decided that using Firebase is the most appropriate approach. In order to achieve the goal of having an isolated environment, the backend development team decided to create separate files with functions making calls to firebase. This was done so that there were no calls to firebase found in the application's frontend. One core file is created in order to include all Firebase configurations and modules such as its database (Firestore) and authentication (auth). The figure below displays how this was achieved:

```
1   import { initializeApp } from "firebase/app"
2   import { getFirestore } from "firebase/firestore"
3   import {getAuth} from "firebase/auth"
4   const firebaseApp = initializeApp({
5       apiKey: "AIzaSyAUMauD-NU1lHaVtplTLyZl3Z1OgGILvWA",
6       authDomain: "fir-tutorial-9f91d.firebaseapp.com",
7       databaseURL: "https://fir-tutorial-9f91d-default-rtdb.firebaseio.com",
8       projectId: "fir-tutorial-9f91d",
9       storageBucket: "fir-tutorial-9f91d.appspot.com",
10      messagingSenderId: "50629745045",
11      appId: "1:50629745045:web:840e25062f1d53da9bbe9c",
12      measurementId: "G-JVPP0NNRRK"});
13
14  const db = getFirestore();
15  const auth = getAuth();
16  export {db, auth};
```

Figure 17:

After the modules were imported from firebase they were afterwards exported in order to be used in the subsequent files that actually make use of them. At present, two specific files have been created in order to deal with such an issue. The first one is the authentication file specifically called "auth.js". In this file, all the necessary authentication functions have been implemented while making use of the firebase modules as can be seen in figure 6.

```
import {auth} from "./firebaseCore";
import {createUserWithEmailAndPassword, signInWithEmailAndPassword, signOut} from "firebase/auth";

async function signUp(email, password){
    await createUserWithEmailAndPassword(auth, email, password).then(()=>{
        alert("New user profile created!");
    }).catch((error)=>{
        alert("Error while creating user: " + error + ".");
    });
}

async function signIn(email, password){
    await signInWithEmailAndPassword(auth, email, password).then(()=>{
        alert("User logged in!");
        console.log("user in!");
    }).catch((error) => {
        alert("Error on sing in: " + error + ".");
    });
}

async function logOut(){
    await signOut().then(() => {
        alert("User logged out!");
    }).catch((error) => {
        alert("Error on log out: " + error + ".");
    });
}

//Exporting the funcitons
export {signUp, signIn, logOut};
```

Figure 18:

This way, the functions can be exported and called into the frontend as simple js functions without having to make actual calls to firebase directly. The same process has also been used for getting the trees from the firestore database displaying their data into the main page of the application.

```
import {db} from "./firebaseCore";
import { collection, query, where, getDocs } from "firebase/firestore";

  async function readDB (coll, field, operator, value){
  const querySnapshot = await getDocs(query(collection(db, coll), where(field, operator, value)))
  let results = [];
  querySnapshot.forEach((doc) => {
      results.push(doc.data());
    });
    return results;
}
export default readDB;
```

Figure 19:

This is only the first example of actually being able to get data from the database and display it in the console log. More work has to be done so that the data is displayed dynamically in the frontend. Nonetheless, what has been implemented so far is quite an achievement considering that the backend development team was also trying to get familiar with the firebase technology.

5.2 Iteration 2

5.2.1 CRUD Operations

After authentication was finished the backend team focused on creating the necessary crud operation for the trees in the database. These functionalities will be needed to further implement the rest of the requirements in our application. Small examples of code will be shown for each CRUD operation below.

```
1  import db from "./firestoreCore";
2    import { doc, setDoc, collection } from "firebase/firestore";
3
4  async function addDocToDB (coll, data){
5
6        await setDoc(doc(collection(db, coll)), data);
7    }
8    export default addDocToDB;
```

Figure 20: Create

```
import db from "./firestoreCore";
import { collection, query, where, getDocs } from "firebase/firestore";

async function readDB (coll, field, operator, value, type){
const querySnapshot = await getDocs(query(collection(db, coll), where(field, operator, value)))
let results = [];
//You can choose between getting the ID of the documents you are querying, its content or both. leaving the type
if(type == "id")
  querySnapshot.forEach((doc) => {
      results.push(doc.id);
    });
else if(type == "both")
  querySnapshot.forEach((doc) => {
    results.push([doc.id, doc.data()]);
  });
else
  querySnapshot.forEach((doc) => {
    results.push(doc.data());
  });

return results;
}
export default readDB;
```

Figure 21: Retrieve

```
src > components > firebase > JS updateDocument.js > [e] default
1   import db from "./firestoreCore";
2   import { doc, updateDoc } from "firebase/firestore";
3
4   async function updateDocument(coll, docName, data){
5
6       await updateDoc(doc(db, coll, docName), data);
7   }
8   export default updateDocument;
```

Figure 22: Update

Figure 23: Delete

By having these functions ready, new functionalities that have to operate with cordiance to the database can be implemented much easier in the future. Similar to the authentication functions these operations were also isolated extensively in the form of javascript functions that make calls to the firebase database. This way moving towards other backend technologies can be easier if required.

5.3 Iteration 3

5.3.1 Adopt a Tree Function

The next main function that was implemented was the tree adoption function. This is the most important aspect of the application alongside taking care of the adopted trees. A snippet of code has been provided below in order to demonstrate how this functionality was added to the application.

```
export default function AdoptPopUp({ id }) {
    const [open, setOpen] = React.useState(false);
    const user = useAuthState();
    const handleClickOpen = () => {
        setOpen(true);
    };

    const handleClose = () => {
        setOpen(false);
    };

    const handleAdopt = () => {
        readDocument("Trees", "Id", "==", id, "both")
        .then((tree) =>{
            let msg = "An error has occured";
            if(tree[0][1].Owner == user.user.uid){
                msg = "You have already adopted this tree";
            }else if(tree[0][1].Owner === "-1"){
                updateDocument("Trees", tree[0][0], {
                    "Owner" : user.user.uid
                });
                msg = "Congratulations you have successfully adopted a new tree!!!";
            }else if(tree[0][1].Owner != user.user.uid){
                msg = "This tree has been adopted by someone else";
            }
            console.log(msg);
        });
        setOpen(false);
    };
```

Figure 24: Delete

The function checks three possibilities. First it checks whether or not the tree is already adopted by the certain user. Secondly it checks if the tree has been adopted by another user. If both checks work then the application allows the user to adopt the certain tree by updating the "Owner" value of the tree to the UID of the user. This function can potentially be changed in the future by being developed further in more robustness.

5.3.2 Node and MongoDB Rest api

Due to the fact that firebase functions come at a price after a certain threshold, the development team also decided to focus on other technologies in order to create a backend that the application can use in the future. It was decided that the new backend would be

implemented in Node JS and MongoDB in the form of a rest api to handle http requests to the backend of the application.

5.4 Iteration 4

After a thorough discussion with both the project supervisor and clients it was decided that the backend was going to be migrated using the Node JS framework in order. This decision was taken due to the disadvantage that Firebase required a paid update while the number of users increased in the future. Having a free of charge backend would provide a much more flexible maintenance fee in the future for both upcoming developers and their clients. For this reason the development team started migrating the app towards a new free of charge technology.

Another reason as to why Node JS was chosen also lies in the fact that the rest of the application is developed with React, a javascript framework. Therefore having both frontend and backend implemented with the same similar syntax structure would allow the final project to be more consistent, and at the same time would make it easier for upcoming developers to work on it in the future. At the same time, a new design was put in place for the frontend of the application. Making it feel more modern and mobile friendly.

5.4.1 Node JS Backend

In order to migrate to a Node JS backend our team decided to create a REST API in order to handle each functionality in the form of incoming HTTP requests coming from the frontend. These HTTP requests would be handled in order to also communicate with the database of the application which is now implemented in the form of a MongoDB database server. There are 5 main types of HTTP request, each of them is explained in the table below.

HTTP Request Name	CRUD Function
POST	Creating or inserting a new entry into a collection.
GET	Retrieving data from a collection.
PUT	Entirely updating or replacing a collecting entry.
PATCH	Partially updating a collecting entry, used when only a specific attribute of the entry is to be changed.
DELETE	Deleting an entry from the collection

By using this format the REST API can be able to handle the different requests and alter them in order to meet the specific requirements of the Tree Adoption Application. An example of how this can be done is shown below.

- **Creating a new user profile (Sign Up)** : A POST request can be used in order to create a new user entry and insert it into the database
- **Viewing the all available trees on the map and differentiating them by age** : A GET request can retrieve all of the available trees from the database, and using their age the front end can differentiate between new and old trees.
- **Adopting a tree** : A PATCH request can be used in order to update the owner field for the specific tree entry that is being adopted. The owner field would equal the ID of the owner that is logged in and trying to adopt the tree.

These are only some basic examples that demonstrate how the REST API can handle all of the necessary features that will be included in the final submission of the application. A more detailed insight into the backend implementation is going to be presented in the form of code snippets in the final submission of the application.

This iteration saw a further development of the MoT Adopt a tree project. The already implemented features were all given a more modern look, and the requirement to distinguish the new trees and the old trees was fully implemented. The backend was switched and this process went smoothly without much time being lost. Switching the whole backend mid production was quite the challenge but the team managed to coordinate and migrate the whole application to a new backend fairly quickly.

5.5 Final Iteration

5.5.1 Mongo Schemas

Schemas are crucial components when it comes to the representation of data into a database. Two different schemas were used in the application in order to represent both a user entity and a tree entirety into the database. The information that is stored for each entity can be viewed in their MongoDB schemas below.

```
const mongoose = require("mongoose");

const Schema = mongoose.Schema;

const treeSchema = new Schema({
        title: { type: String, required: true, default: "Tree" },
        type: { type: String, required: true },
        address: { type: String, required: true },
        location: {
                lat: { type: Number, required: true },
                lng: { type: Number, required: true },
        },
        zip :{type: String, required: false},
        date: { type: String, required: true },
        lastWatered: { type: String, required: false },
        needsWatering: { type: Boolean, required: true, default: false },
        owner: { type: mongoose.Types.ObjectId, required: false, ref: "User" },
        isAlive: { type: Boolean, required: true, default: true },
});

module.exports = mongoose.model("Tree", treeSchema);
```

Figure 25: Tree Schema

```
const mongoose = require("mongoose");
const uniqueValidator = require("mongoose-unique-validator");

const Schema = mongoose.Schema;

const userSchema = new Schema({
        email: { type: String, required: true, unique: true },
        password: { type: String, required: true, minlength: 6 },
        trees: [{ type: mongoose.Types.ObjectId, required: true, ref: "Tree" }],
});

userSchema.plugin(uniqueValidator);

module.exports = mongoose.model("User", userSchema);
```

Figure 26: User Schema

5.5.2 Backend Controllers

While developing the Node JS backend in the form of a REST API multiple controllers had to be set in place in order to accommodate the different backend related functionalities. Such functionalities include tree adoption, tree retrieval, tree adanopment, and tree watering. A specific controller has been developed for each in order to handle all potential errors and to excenture the needed operations on the actual application database which is constructed on MongoDB.

5.5.3 Retrieving all trees for the database

Getting the tree information from the database is an important feature for the application. Both the tree adoption map and also the my trees page of the application make use of this feature. In terms of the map a controller that retrieves all three data from the database is needed. This is because the map offers different distinguishing features to differentiate between user trees, other adopted trees, and even younger new trees. Besides only performing the actual tree retrieval operation, the controller also tries to handle possible errors that might occur. The simple implementation of the get all function from the controller is displayed below.

```
const getTrees = async (req, res, next) => {
        let allTrees;
        try {
                allTrees = await Tree.find();
        } catch (err) {
                const error = new HttpError(
                        "Something went wrong, could not find trees.",
                        500
                );
                return next(error);
        }

        if (!allTrees) {
                const error = new HttpError("Could not find trees", 404);
                return next(error);
        }

        res.json({
                trees: allTrees.map((tree) => tree.toObject({ getters: true })),
        });
};
```

Figure 27: Get all trees controller

It should also be stated that the controller makes use of already defined functions associated with the Tree schema mentioned above. For example in this case the find() method is used in order to retrieve all tree data from the database connection.

5.5.4 Retrieving trees for a specific user

Besides retrieving all trees in order to display them into the information map, another controller was needed specifically to retrieve trees that were adopted by a specific user. The reason for this controller is because every user should have quick access to the specific tree entities they own. At the same time the user should be able to take care of their own tree. By developing this controller the trees of a specific user can be displayed to them directly into the my trees section of the application. The way in which trees are retrieved for a specific user can be observed in the following code snippet.

```
30  const getTreesByUserId = async (req, res, next) => {
31          const userId = req.params.uid;
32          let userWithTrees;
33          try {
34                  userWithTrees = await User.findById(userId).populate("trees");
35          } catch {}
36          res.json({
37                  trees: userWithTrees.trees.map((tree) => tree.toObject({ getters: true })),
38          });
39  };
```

Figure 28: Get user trees controller

The controller uses a uid parameter passed by the HTTP request which is then passed to the findById() function in order to retrieve the specific user data from the database. Similar to find, this is an already defined MongoDB function from the schema. Part of the user schema is also a trees array which stores the ids of all the trees the user has adopted. These ids are then used in order to map the data for each tree adopted by the user into the my trees section in the app.

5.5.5 Adoption

The tree adoption controller is used to provide one of the most important functionalities in the application, allowing users to adopt a tree in Thessaloniki. In order to do this the controller extracts both a tree id and a user id from the HTTP request sent. Both the ids are then used by the findById() function of both schemas in order to retrieve the specific user requesting the adoption and specific tree that is being requested for adoption. In case no error is occurring the owner field in the requested tree is updated to the user's id and the trees array for the requesting user is updated, with the tree id being added to it. More specifically the process can be seen below.

```
154  const adoptTree = async (req, res, next) => {
155          const treeId = req.body.tid;
156          const userId = req.body.uid;
157          let tree;
158          let user;
159
160          try {
161                  tree = await Tree.findById(treeId);
162                  user = await User.findById(userId);
163          } catch (err) {
164                  const error = new HttpError(
165                          "Something went wrong, could not adopt tree.",
166                          500
167                  );
168                  return next(error);
169          }
170
171          if (!treeId) {
172                  const error = new HttpError("Could not find tree for this id.", 404);
173                  return next(error);
174          }
175
176          user.trees.push(treeId);
177          tree.owner = userId;
```

Figure 29: Adoption controller

5.5.6 Watering

The watering controller similar to the adoption controller also takes both a tree id and a user id from the HTTP request. The user id is only used to make sure that the user taking care of this tree is actually its rightful owner. Meanwhile the tree id is used to retrieve the specific tree so that its data can be updated when watering occurs. When a user waters the tree its lastWatered date is updated to the current date that the user is performing the watering on the tree. At the same time the needsWatering status on the tree is updated to false since the tree has been watered and no watering will be required until a specific amount of time passes.

```
const waterTree = async (req, res, next) => {
        const treeId = req.body.tid;
        const userId = req.body.uid;

        let tree;
        try {
                tree = await Tree.findById(treeId);
        } catch (err) {
                const error = new HttpError(
                        "Something went wrong, could not water tree.",
                        500
                );
                return next(error);
        }

        if (!treeId) {
                const error = new HttpError("Could not find tree for this id.", 404);
                return next(error);
        }

        if (tree.owner != userId) {
                const error = new HttpError("You are not allowed to water this tree.", 401);
                return next(error);
        }

        var today = new Date();
        var dd = String(today.getDate()).padStart(2, "0");
        var mm = String(today.getMonth() + 1).padStart(2, "0");
        var yyyy = today.getFullYear();
        today = yyyy + "/" + mm + "/" + dd;
        tree.needsWatering = false;
        tree.lastWatered = today;
```

Figure 30: Watering controller

5.5.7 Schedule event

For the final iteration a new file has been added to the backend of the application. This is a scheduled event designed specifically to update the watering status of the trees in the database. This function is scheduled to run everyday in order to check the trees found in the database. Its main task is to check and update the watering status on all of the adopted trees. Before going into more detail it is important to explain the differentiation between old trees and new ones when it comes to the watering process. Based on the information given by the client, old trees required regular watering within a time frame of 4 days, meanwhile

the time frame that young trees can go without watering is 2 days.

Knowing this the function was purposely designed so that it can differentiate between the trees by calculating their age. When an old tree is analyzed, the function checks whether or not more than 4 days have passed since it's last watering date. If yes the needsWatering status is changed to true. When it comes to young trees (less than 3 year old) the function checks whether more than 2 days have passed since the last watering date and follows the same procedure. Currently, the needsWatering value of the tree is only used during the watering process where the value is changed to false and during the execution of the scheduled job occurring everyday. There are no other frontend features that make use of this value. However, this field combined with the process of watering and the scheduling event can lead to many future improvements such as the creation of a notification system that let's the user know that a long time has passed since their tree was last watered. In even more severe occurrences, when over a month has passed since the last time the tree has been watered, the application might be upgraded to remove the tree from the specific user's ownership. In other words, despite its lack of usage in the current application this function has very high premises for the future of the application.

6 Deploying

Deployment of the application was done with the help of the free web app deployment service Heroku. In order to do that, some major changes were made to the application file structure in the code. First and foremost, for security reasons all the API credentials and every other sensitive information that could compromise security, were hidden with the use of .env files both for the production and development build of the application. Subsequently, for performance reasons, the front-end part of the application was confessed in a packaged version with the command "npm run build", and that build was moved to the public folder of the backend so that it can be served to the client by the backend API and ExpressJS.

Additionally React.lazy was also used for improved performance. This built-in function works together with React Routers and helps with performance by only requesting from the back-end the current page of the app, instead of fetching all React pages that exist in the routing process. Finally, an account and a project were created in Heroku and with the help of git and Heroku CLI the code was deployed to the public under: https://mottrees.herokuapp.com/ . It is also worth mentioning that the initial plan was for the deployment to take place in an already provided service called CPanel, but unfortu-

nately after several attempts by the team, it turned out that some of the technologies used in the app like the MongoDB database, were not compatible with this deployment service.

7 Mobile Application

In order to meet the requirement of the client so that the final application is a mobile one, our team has decided on developing a progressive web app or PWA. These types of applications allow web apps to function more like native apps on a user's mobile device. A user can access the app through the web and once there they can be able to install the web app into their mobile as a native app for easier access. Such a functionality can be easily implemented into an application by configuring its manifest appropriately as well as writing a service worker, which acts as a middleman between the frontend and the backend, providing functionalities like caching. This is also the main plan our team is going to follow in order to enable native mobile functionality to the tree adoption app. Currently, the application is fully installable through browsers that support PWAs, such as Google Chrome. Deployment on the Google play store by the client is also possible.

To demonstrate this is how the PWA app looks on a real phone.

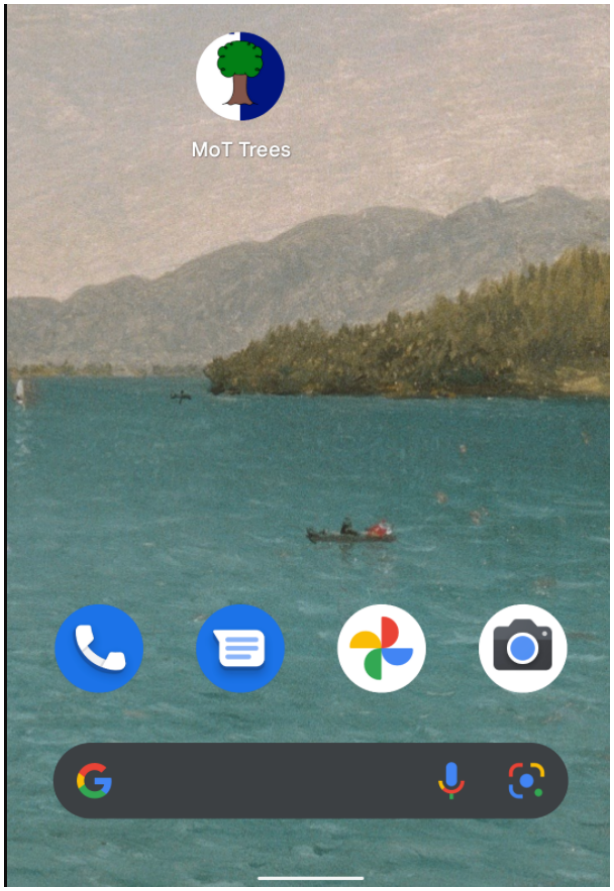


Figure 31: Mobile app

8 Gamification

In order to make the application more interactive and to also encourage citizen participation, the Municipality also required that gamification aspects are included in the application. This can be done by implementing a system where the user is assigned certain amounts of points as a result of their interaction with the Tree Adoption app. After some discussion the initial idea for this point system was as follows.

User interaction	**Points awarded**
Adopting a tree	10
Watering a tree	20
Reporting a problem with a tree	30

Adding a gamification feature to the application creates new possibilities for the client to further develop this initiative even further in the future. A good example would be by setting

a certain maximum point limit which a user can later on exchange for a price. Another option would be by creating some sort of competition between users where the user with the highest amount of points wins a big prize, such as an electric scooter from the municipality, or tickets to a special event taking place in Thessaloniki. Despite the fact that these ideas might seem controversial due to the fact that it tries to attribute compensation to volunteer work, one should acknowledge that even certain citizens, who might not be interested in volunteering are intrigued to participate in the community after all.

In order to implement gamification in the application a new section in the user collection should be used to keep track of a user's points after their interaction with the application. New function will also have to be implemented which in turn updates a users score each time they have one of the aforementioned interactions with the app. Also another point to be considered is whether or not there is going to be a maximum point limit after which a user's score is reset to 0.

9 Conclusion

After nearly 6 months of work, the project has now finished the first cycle and is ready to be picked up by the next group of students. This project has been challenging and engaging providing our team with a great set of knowledge in a lot of fields. From React and Google maps, backend technologies such as Node and Firebase, to ways to communicate with clients.

The progress made was a successful one, setting a great foundation for the platform to build on. The number of requirements met is pleasing and the core requirements were all tackled first. This resulted in a system that is ready to use and that encapsulates the original idea of the municipality (adopt trees).

Finally, we wish the following teams the best of luck as they embark on this endeavor. It will not be simple, but we believe you will enjoy working on it and learning from it as much as we did.

References

[1] Firebase. 2021. Firebase Products. [online] Available at: ¡https://firebase.google.com/products-build¿ [Accessed 7 November 2021].

[2] Node.js. 2021. About — Node.js. [online] Available at: ¡https://nodejs.org/en/about/¿ [Accessed 7 November 2021].

10 INSTALLATION MANUAL

The system was designed with the capability to run on mobile devices and this was achieved, therefore the Hardware System Requirements are very minimal. The operating system which the platform was designed is Windows, but the application works as a PWA on both IOS and Android as well.

To install the software and run it locally, follow these steps:

0. Get familiar with React, Node, Mongo, Express, Heroku.
1. Download Node.js - `https://nodejs.org/en/`
2. Clone the repository: git clone https://github.com/city-cs-msc/ccp3300-21-mot.git
3. Open folder in editor (VSCode)
4. Open a terminal, run the command: **Cd frontend**
5. run the command: **Npm install**
6. Open a terminal, run the command: **Cd backend**
7. run the command: **Npm install**
8. Open terminal on the FrontEnd directory, run command: **Npm update**
    * To open a terminal in the frontEnd directory: right click the directory in VSCode and click Open in Integrated Terminal
9. Open terminal on the BackEnd directory, run command: **Npm update**
    * To open a terminal in the backEnd directory: right click the directory in VSCode and click Open in Integrated Terminal
10. On frontend terminal run: **Npm start**
11. On backend terminal run: **Npm run dev**

* Note to install CORS disabler[1]: hyphenshttps://chrome.google.com/webstore/detail/moesif-origin-cors-change/digfbfaphojjndkpccljibejjbppifbchyperref

To delete, just delete the folder containing everything.

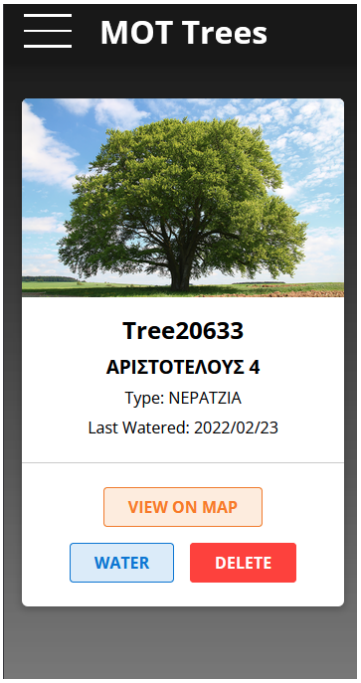
To view the deployed version check `https://mottrees.herokuapp.com/`

[1]Read what CORS is: `https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS`

USER MANUAL

The application is currently available at https://mottrees.herokuapp.com/

Page descriptions



Τα δέντρα μου

"Τα δέντρα μου" is the main page of the application, in which users manage their trees. There are currently three main functions per tree:

- Watering
- Deleting
- Viewing on map

Χάρτης δέντρων

The "Χάρτης δέντρων" page is a map of Thessaloniki, in which all the trees of the city are shown. Trees are displayed in different colors for different situations:

- Green for non-adopted trees over three years old
- Blue for non-adopted trees under three years old
- Gray for trees adopted by other users.
- Fuchsia for trees adopted by the user himself.

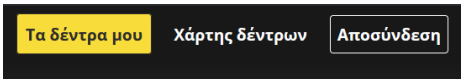
(Currently there are no blue trees in our database because the data in it are very old)

Use

Accessing

1. Go to https://mottrees.herokuapp.com/, click "Μεταφερθείτε στην δημιουργία νέου λογαριασμού" and create an account with an email account and password of your choice
2. If the credentials are valid, hit “Δημιουργία νέου λογαριασμού”
3. If you already have an account, enter your credentials and hit “Σύνδεση”

Navigation

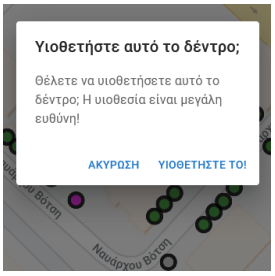


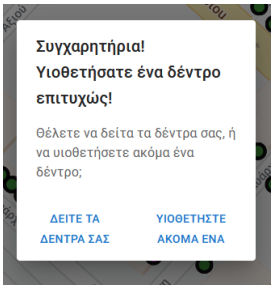
- If you are on a computer, the top right of the web page has three buttons that you can use to navigate or to log out

- If you're on a mobile phone or tablet, press the three horizontal bars to reveal the navigation buttons

Adoption

1. Navigate to "Χάρτης δέντρων” to go to the map from which you can adopt a tree.
2. Once loaded, select which tree you want to adopt by clicking on it. You can move the map in all directions by tapping and dragging in the direction you want either with your finger on touch screen devices or with your computer mouse.
3. Click on the tree you want and hit the "ΥΙΟΘΕΤΗΣΤΕ ΤΟ" button if you are sure you want the that tree.
4. If you want to adopt other trees, hit "ΥΙΟΘΕΤΗΣΤΕ ΑΚΟΜΑ ΕΝΑ", otherwise, hit on "ΔΕΙΤΕ ΤΑ ΔΕΝΤΡΑ ΣΑΣ" to be taken to the page of your adopted trees.





Watering

1. On the "Τα δέντρα μου" page, you will see all the trees you have adopted. If it does not adopt a tree yet, follow the adoption steps described above.
2. Simply hit the "ΠΟΤΙΣΤΕ" button, and you will see the "ΤΕΛΕΥΤΑΙΟ ΠΟΤΙΣΜΑ" field updated with the current date

Others

- if you do not wish to have a tree anymore, simply hit the “ΔΙΑΓΡΑΦΗ” button
- If you want to see where a tree is on the map, simply hit the “ΔΕΙΤΕ ΣΤΟΝ ΧΑΡΤΗ” button

Installation

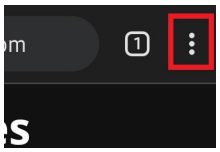
The application can be installed on Android and iOS devices

Android

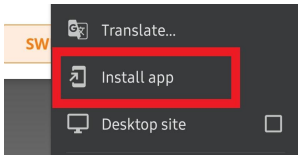
The instructions are specifically for Google Chrome, the steps may not be the same if you use another web browser, or it may not even support it.

Instructions

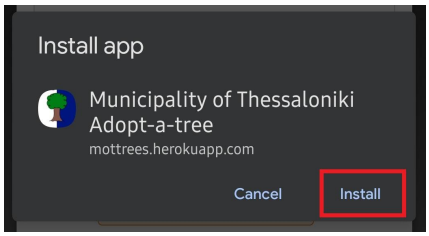
1. Go to https://mottrees.herokuapp.com/
2. Hit the three horizontal dotson the top right



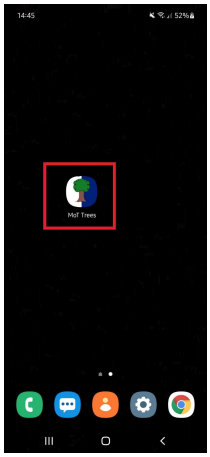
3. Click “Install app”/”Εγκατάσταση εφαρμογής”



On the menu that will come up, hit “Install”/”Εγκατάσταση”



4. The application is now installed and you can treat it as you would any other application on your Android device. If you click on it, it will open just like any regular app would.

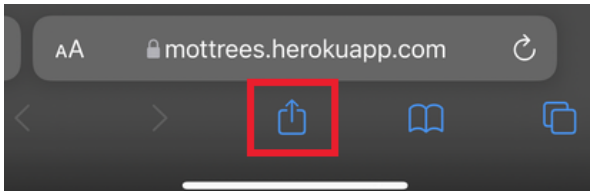


IOS

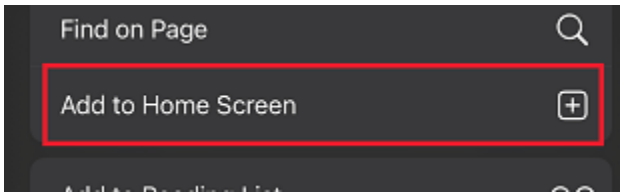
The instructions are specifically for Safari, the steps may not be the same if you use another web browser, or it may not be supported by it.

Instructions

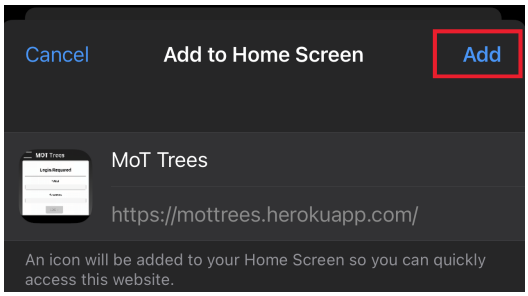
1. Go to https://mottrees.herokuapp.com/
2. Hit the middle button on the bottom



3. Click “Add to home screen”



4. On the menu that will pop up click “Add”



5. The application is now installed and you can treat it as you would any other application on your iOS device.

Glossary

Translation of Greek terminology. The capitalization is maintained as in the original application.

Greek	English
Τα δέντρα μου	My trees
Χάρτης δέντρων	Trees map
Μεταφερθείτε στην δημιουργία νέου λογαριασμού	Switch to account creation
Δημιουργία νέου λογαριασμού	Create new account
Σύνδεση	Log in
ΥΙΟΘΕΤΗΣΤΕ ΤΟ	Adopt it
ΥΙΟΘΕΤΗΣΤΕ ΑΚΟΜΑ ΕΝΑ	Adopt one more
ΔΕΙΤΕ ΤΑ ΔΕΝΤΡΑ ΣΑΣ	View your trees
ΠΟΤΙΣΤΕ	Water (verb)
ΤΕΛΕΥΤΑΙΟ ΠΟΤΙΣΜΑ	Last watering
ΔΙΑΓΡΑΦΗ	Delete
ΔΕΙΤΕ ΣΤΟΝ ΧΑΡΤΗ	View on map
ΑΚΥΡΩΣΗ	Cancel