



Eötvös Loránd University
Faculty of Informatics
Department of Data Science
& Engineering

Web Application for Grapevine Image Annotation

Supervisor:

Dr. Tomas Horvath
Head of Department

Author:

Ahmed Blej
Computer Science BSc

Budapest, 2019

Contents

1. Introduction:

1.1. Precision farming leading the next agricultural revolution.	1
1.2. Shortcomings of precision viticulture.	2
1.3. Thesis project introduction.	2
1.4. A brief look into the tools used.	3

2. User Documentation:

2.1. The solved problem – Data sets for viticulture.	4
2.2. The implementation.	4
2.3. Starting the App.	5
2.4. Login/Registration.	6
2.5. About page.	7
2.6. Annotate page.	8
2.7. Menu page.	14
2.8. Sequence diagram.	17

3. Developer Documentation:

3.1. The detailed specification of the problem.	18
3.2. Class diagrams.	18
3.3. Methods and notions used to annotate.	22
3.4. Methods and notions used to prune.	32
3.5. Methods and notions used to login/register.	39
3.6. Testing.	42

4. Conclusion.	44
----------------	----

5. Figures.	45
6. Bibliography.	48

Chapter 1:

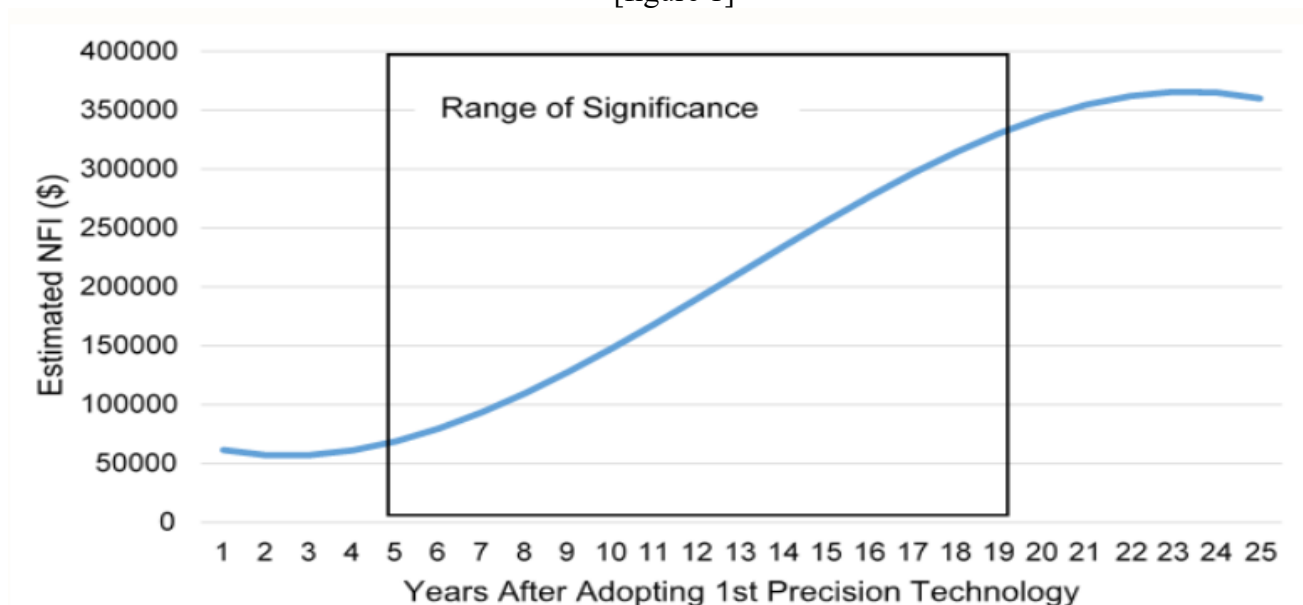
Introduction:

It is to no one's surprise that we live in an age of great automation and that it has affected many areas of our lives with improvements, facilitation, and simplifications. One of these areas is agriculture; the majority of research and development regarding automation in this area is focusing mostly on data collection, aggregation, and visualization which offer large possibilities of educated decision-making to the farmers. One focal point, however, is that the farmers have to make these decisions themselves.

1.1 – Precision farming leading the next agricultural revolution:

Precision farming is currently leading the third great revolution for agriculture ^[1], and it is believed to have the most transformative impacts out of the agricultural revolutions that happened, not only on agriculture but also on the food value chain. Also known as satellite farming, or site-specific crop management ^[2], it aims to maximize the returns while minimizing the inputs in a strategic way that is based on observations and measurements of the inter and intra-field variability in crops. Research and analyses are predicting that the use of precision agriculture technology is promising continuously increasing profits ^[3]:

[figure 1]



1.2 – Shortcomings of precision viticulture:

Precision viticulture uses precision agriculture methods to increase grape production and quality by monitoring and reducing the effects of outside factors ^[4]. It focuses more on the growth environment and vineyard management than the farmer expertise, which is often overlooked especially when it comes to pruning techniques that are necessary to prevent infections and ensure a healthy and beneficial growth of the tree.

1.3 – Thesis project introduction:

My thesis project deals with viticulture, and specifically with grapevine pruning. As there are many styles and techniques for growing and pruning grapevines that require deep expertise, they do not have a straight-forward way to represent, encode and implement them. This has resulted in a lack of data sets that are necessary for the development of more efficient machine learning prediction models, that would help digitalize the pruning procedure and provide algorithms capable of giving pruning models and notes for a given grapevine.

And thus, I am developing a web application for data collection and annotation in which a user can upload an image of the grapevine, annotate its structure using image processing techniques and store the annotated data in a repository. The gathered images and related data can be browsed in the system, giving the opportunity to mark the pruning points, write comments justifying the decisions for pruning, and provide insight that could be helpful firstly for the users that will have the possibility to discuss and rate these comments and annotations, and secondly to build expert systems, machine learning modules, or frameworks for grapevine pruning.

The gathering of this data is not only going to contribute in a practical manner to the field it relates to but it also opens more research aspects alongside the guaranteed benefits of data gathering ^[5] that facilitate the visibility and tracking of the user-made contributions.

1.4 – A brief look into the tools used:

This application relies on the Model-View-Controller architecture and the image processing tools that will be implemented with Javascript, which is also responsible for back-end and client-side scripting along with PHP, coupled with HTML and CSS for responsive web design. All the data gathered is going to be stored in a database using MySQL, and as of now, the web application is hosted on a local web server using Xampp.

Chapter 2:

User Documentation:

2.1. The solved problem – data sets for viticulture:

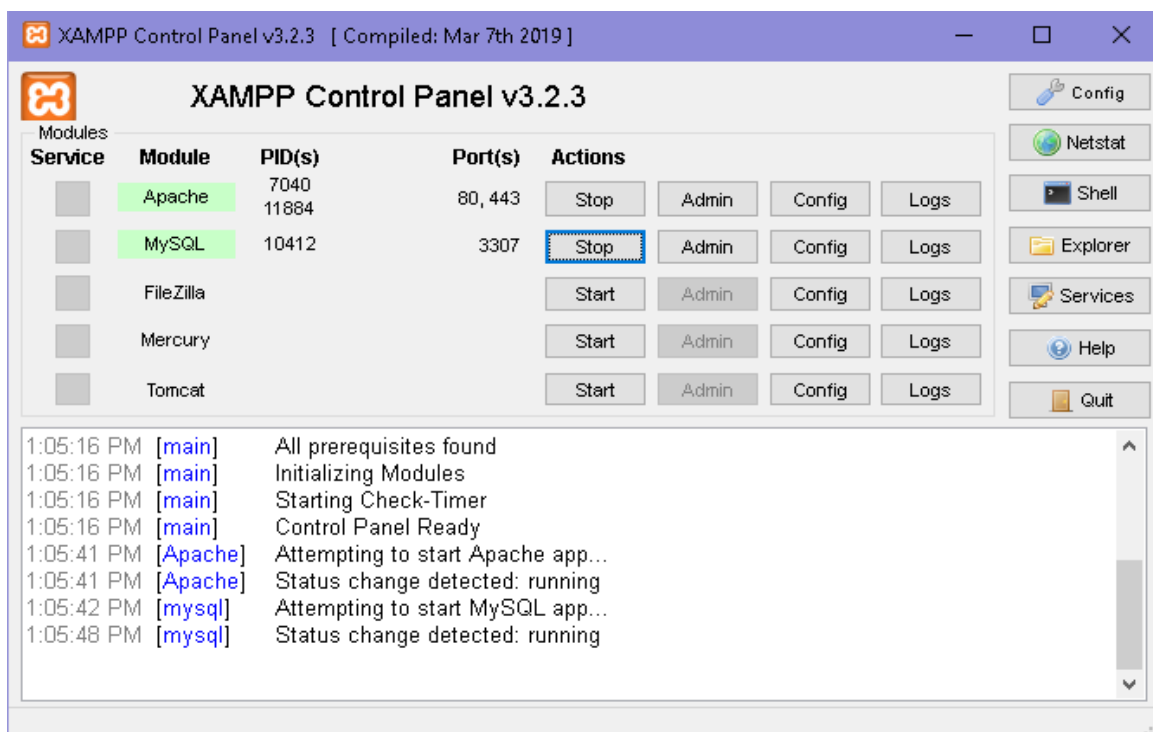
As mentioned earlier, the lack of data sets regarding precision viticulture is dealt with in this application by allowing the user to annotate the structure of the tree and mark the pruning points. These are stored as coordinates in a repository, which allows us to analyze the patterns of annotation and different techniques of pruning, giving us thus more data regarding the whole procedure.

As the user annotates the tree, the coordinates for each branch along with the parent-child relationship between the branches of the tree are being stored in the database. After the user's submission of their annotation, we end up with the image in its raw form and the annotated version of it.

2.2. The implementation:

This is a web application that is implemented with the use of HTML, CSS, Javascript, and PHP. I use the canvas element for the image annotations that are facilitated through the use of Javascript methods, and PHP scripts with SQL that are responsible for storing the data in the database. The application is hosted on Xampp server:

[figure 2]



It is a web server developed by Apache Friends and consists of the Apache HTTP server, MariaDB database, and interpreters for scripts written in PHP^[6].

The database and the tables used can be accessed and administrated through phpMyAdmin which is an open source administrative tool for MySQL and MariaDB^[7]:

[figure 3]

Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> branches	★ Browse Structure Search Insert Empty Drop	25	InnoDB	utf8_general_ci	32 KiB	-
<input type="checkbox"/> coordinates	★ Browse Structure Search Insert Empty Drop	257	InnoDB	utf8_general_ci	32 KiB	-
<input type="checkbox"/> person	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8_general_ci	16 KiB	-
<input type="checkbox"/> prunings	★ Browse Structure Search Insert Empty Drop	14	InnoDB	utf8_general_ci	16 KiB	-
<input type="checkbox"/> reviews	★ Browse Structure Search Insert Empty Drop	5	InnoDB	utf8_general_ci	48 KiB	-
<input type="checkbox"/> trees	★ Browse Structure Search Insert Empty Drop	2	InnoDB	utf8_general_ci	3.5 MiB	-
6 tables	Sum	306	InnoDB	utf8_general_ci	3.7 MiB	0 B

2.3. Starting the App:

As the application is hosted on Xampp server, for now, the Apache and MySQL modules should be started and connected on free ports as shown in figure 2, after that the user will have many functionalities available to them.

In case the default MySQL port configured isn't free, there will be an error message when starting it, however, the port number can be modified from the my.ini file by clicking on config:

[figure 4]



[figure 5]

```
# The following options will be passed to all MySQL clients
[client]
# password          = your_password
port                = 3307
socket              = "C:/xampp/mysql/mysql.sock"
```

When both the Apache and MySQL modules are running, the status change will be shown in the console:

[figure 6]

```
1:05:41 PM [Apache] Attempting to start Apache app...
1:05:41 PM [Apache] Status change detected: running
1:05:42 PM [mysql]  Attempting to start MySQL app...
1:05:48 PM [mysql]  Status change detected: running
```

2.4. Login/Registration:

When the app is started the user will be directed to a login/registration page first:

[figure 7]



The image shows a dark-themed login and registration form. It features two input fields: "Email:" and "Password:". Below these fields is a button labeled "Login/Register". The form is set against a light gray background.

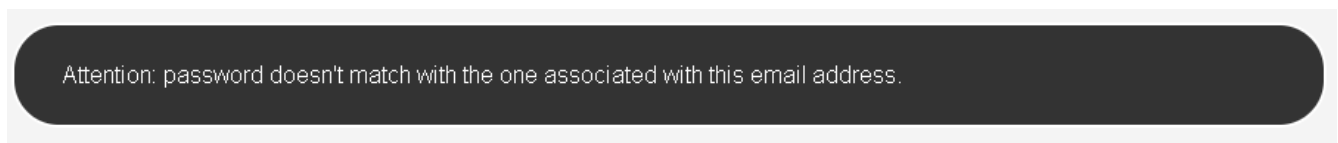
When the user enters their email and password, if no account is associated with that email address then a new one will be created – registration, but if there is already one, then there will be a password check and if the password matches then the user will be logged in.

There are also prompts that will be shown in case the fields are left empty or if the password doesn't match:

[figure 8]



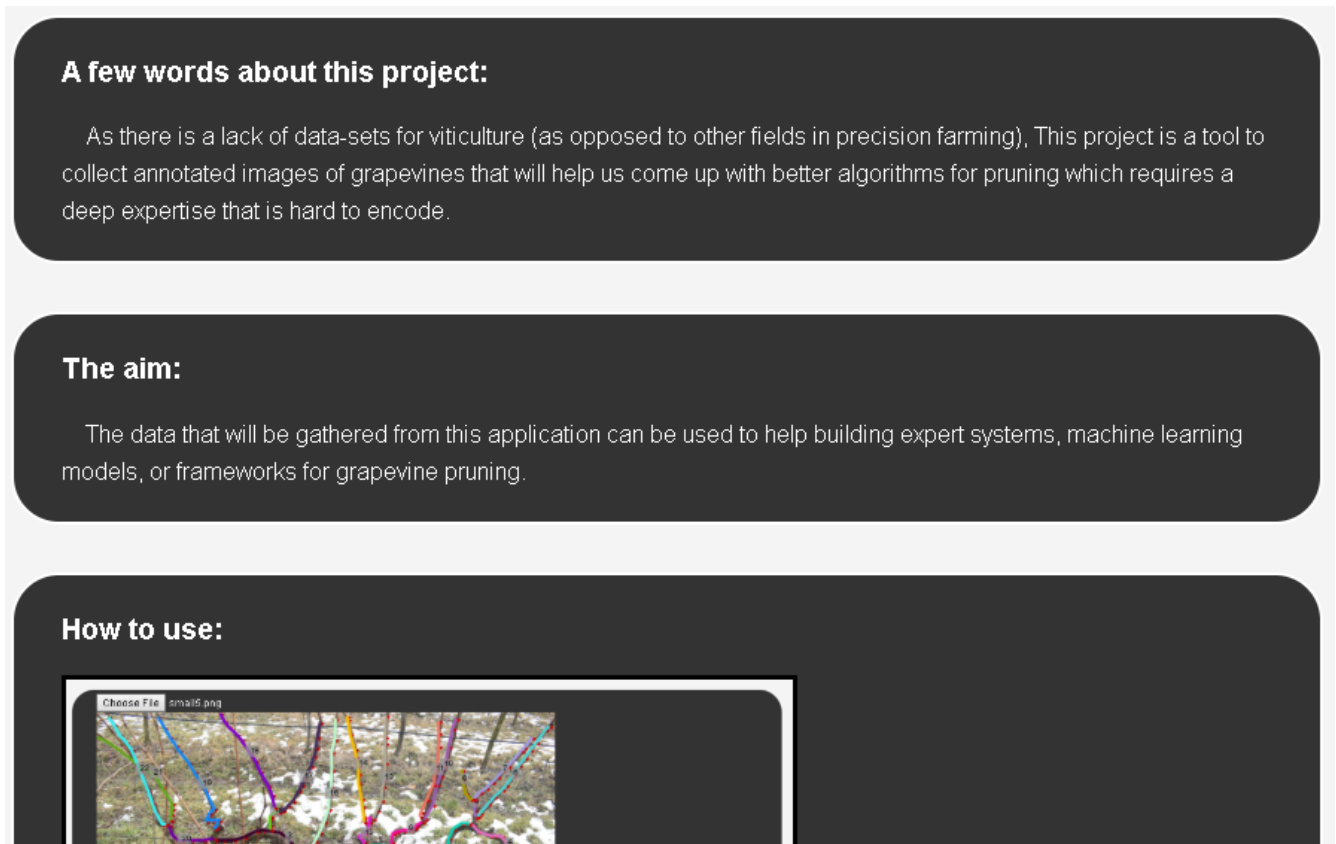
[figure 9]



2.5. About page:

After a successful login the user will be able to see a page mainly with information about the project and how to use it:

[figure 10]



The user can go to the other pages with the user of the navigation bar:

[figure 11]



2.6. Annotate page:

This is where the user can upload an image of the grapevine and annotate it. There are three main panes here, the canvas on which the image will be displayed and annotated, one that will show the textual representation of the tree once submitted, and one that keeps track of the coordinates of the markers put by the user.

The first thing a user wanting to contribute to this data set has to do is to choose the image that they want to annotate:

[figure 12]



After choosing the image, it will be displayed on the canvas:

[figure 13]



Now the user has multiple functionalities to annotate this tree and submit it to the database:

[figure 14]



The user can click on the Start Branch button to start annotating the image of the tree, if the user clicks on the Add Markers button, a quick tutorial on how to annotate (thus add markers) will appear:

[figure 15]

- 1 - click on start branch and select the parent.
- 2 - annotate the branch in question.
- 3 - click on finish branch.

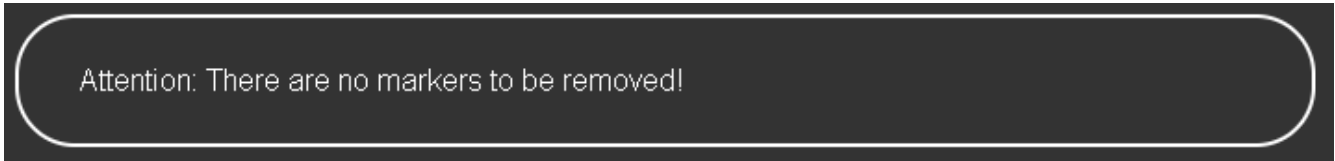
When annotating the branch, the user will be able to see red markers appear where they click on the go:

[figure 16]



The user has the possibility erase a marker and go back one step at a time by clicking on the Remove Marker button, the button can also display warnings when the removal is not possible, either when no markers are available or when the branch has already been finalized:

[figure 17]

A dark gray rectangular box with rounded corners and a thin white border. Inside, the text "Attention: There are no markers to be removed!" is written in a light gray, sans-serif font.

[figure 18]

A dark gray rectangular box with rounded corners and a thin white border. Inside, the text "Attention: You cannot remove a marker of an already finished branch!" is written in a light gray, sans-serif font.

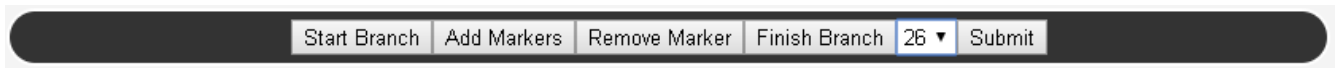
Once the user is done annotating said branch, they can click on the Finish Branch button, thus adding it to the parents option list and finalizing it which is shown by coloring the branch a unique color and adding an indicator with its identifier so that the branch is recognizable from the list:

[figure 19]



The user can proceed with annotating the other branches following the same steps, except, now that the first branch has been annotated, it can be picked as a parent from the options selector using the identifier in the indicator that is close to the branch:

[figure 20]



Whenever a branch is finished, it will be added to the options, so that the user can pick it if it is the parent of any other branches.

Along with the ability to choose the branch as a parent upon finalizing it, the user can also view all the coordinates of the markers related to it on the bottom-most pane:

[figure 21]

```
coordinates  
  
Branch - 26:  
x = 320.65625, y = 316  
x = 318.65625, y = 298  
x = 315.65625, y = 281  
x = 316.65625, y = 261  
x = 327.65625, y = 237  
x = 332.65625, y = 216  
x = 338.65625, y = 202  
x = 330.65625, y = 189  
x = 319.65625, y = 171  
x = 308.65625, y = 163  
x = 294.65625, y = 154
```

Once the annotation of all branches is finished, the user can click on the Submit button to add the image, the annotation, and all coordinates and data related to it to our repository.

[figure 22]



The user can also see a textual representation of the annotated tree, emphasizing the parent-child relationships between the annotated branches, on the middle pane:

[figure 23]

```
branches  
  
Branch - 26  
----Branch - 27  
-----Branch - 29  
-----Branch - 30  
-----Branch - 31  
-----Branch - 32  
----Branch - 28  
-----Branch - 33  
-----Branch - 34  
-----Branch - 35  
----Branch - 36  
----Branch - 37
```


2.7. Menu page:

The user can browse a list of all the annotated images that have been uploaded to the server on the Menu page. The page also offers the user the possibility to mark the pruning points and add insightful comments on each tree.

Each tree had two main panes, the first one with the tree image and various buttons for the pruning functionality, and the second one for the reviews and comments:

[figure 24]



Each image is identified with a tree id and the id of the user who annotated it.

[figure 25]



A dark-themed user interface for adding comments. It features a large white text input field with the placeholder text "add a comment regarding this annotation". Below the input field is a small "Share" button. At the bottom of the section is the word "Comments" in a bold, white font.

All the comments belonging to a certain annotation will appear below it.

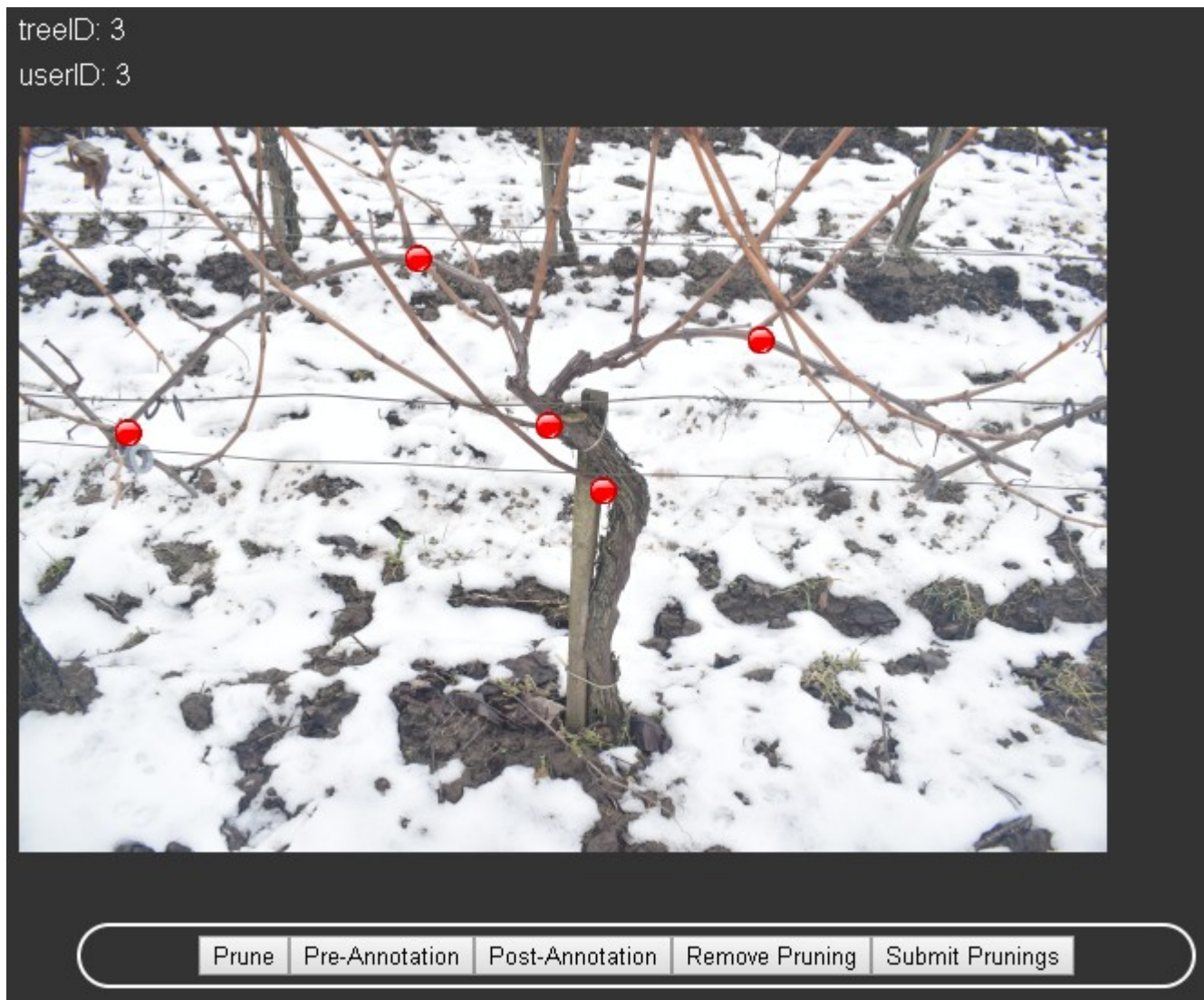
The user can check both the raw image of the tree and the annotated one by clicking on either the Pre-Annotation button or the Post-Annotation one respectively:

[figure 26]



When the user would like to prune, they can click on the Prune button and proceed with marking the pruning points on the image of the tree that will automatically be switched to the pre-annotated version for better vision:

[figure 27]



The user can, of course, erase the pruning markers by clicking on the Remove Pruning button, and once done, the user can submit these pruning points by clicking on the Submit button.

Sharing a comment follows the same principle, by typing the comment on the text area and clicking share, it will show up in the comments section with the user's ID:

[figure 28]

Comments

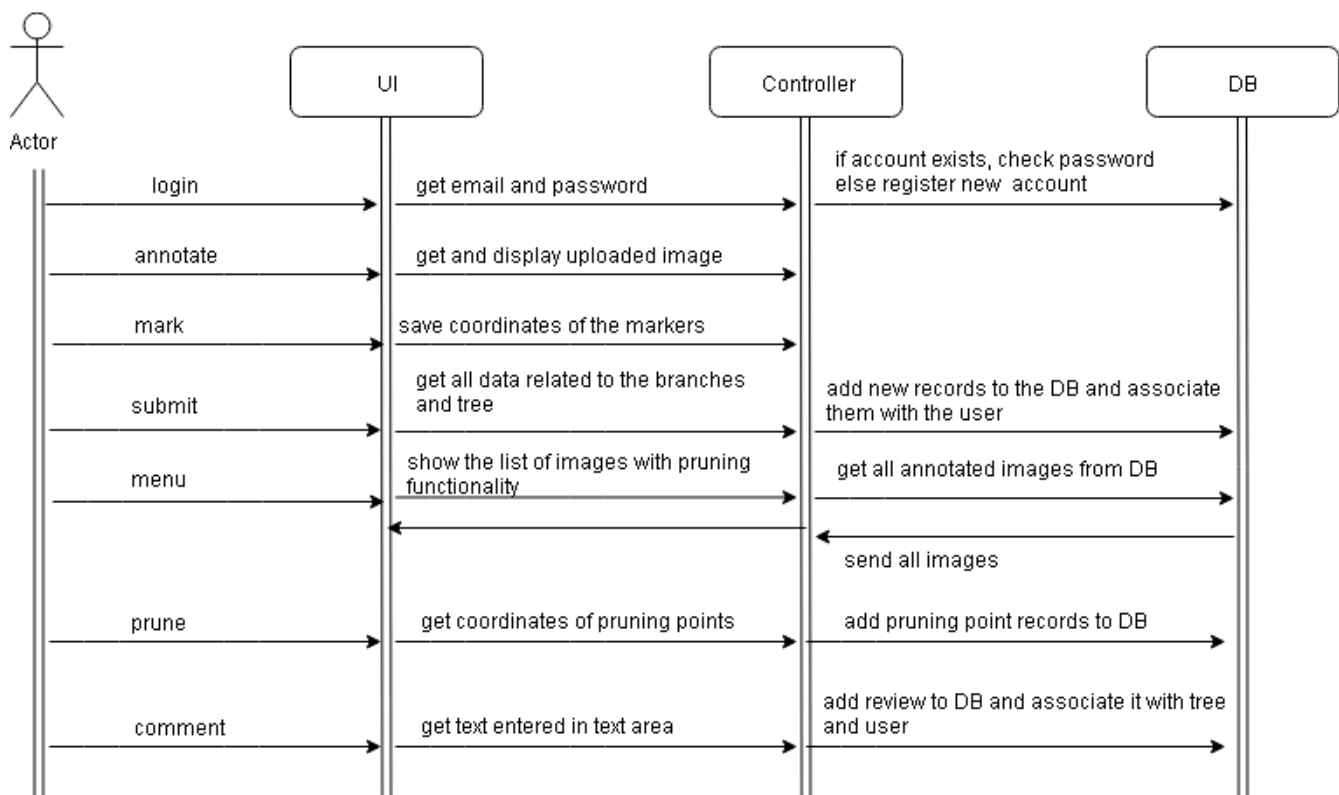
id - 3: i think most of the outer most branches should be pruned in this tree

The moment the user clicks on Share or Submit, the entered comments and pruning points are added to the database and associated with the user.

And thus we have detailed data on the structure of the tree, its pruning points according to the user and potentially insightful remarks regarding the pruning and annotation.

2.8. Sequence diagram:

[figure 29]



Chapter 3:

Developer Documentation:

3.1. The detailed specification of the problem:

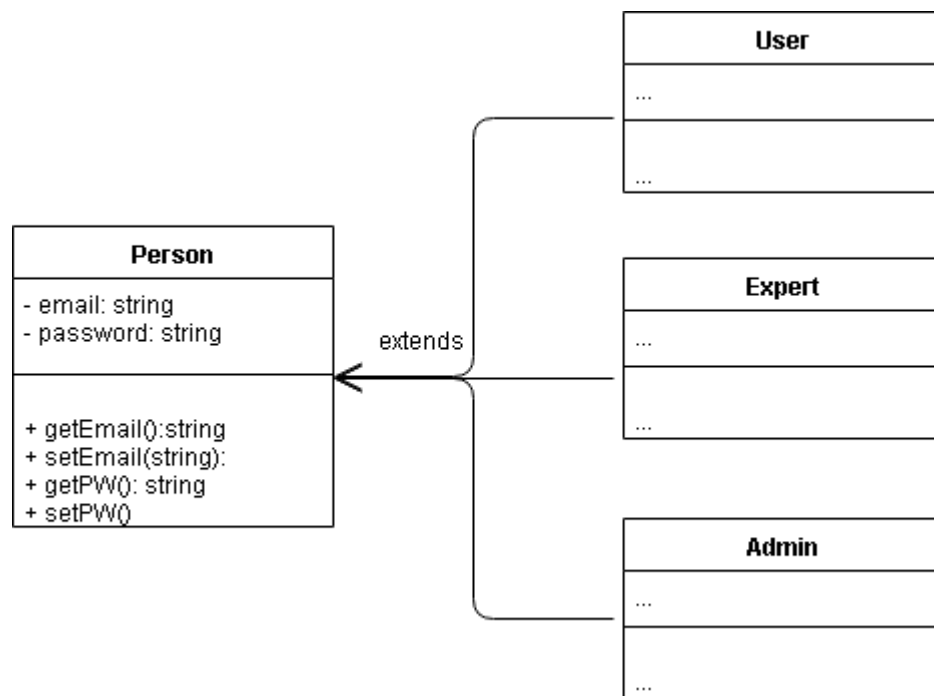
As said before, the structures of the tree and the pruning procedures are not easily implemented and encoded in a straight-forward manner. Moreover, the expertise needed for pruning is not meticulously documented as the farmers learn these techniques through years of experience.

By using this application, we will be able to gather information in a more-or-less uniform way as the tree structure is digitized and stored in an easily accessible way.

3.2. Class diagrams:

Each component and agent of this application is represented in a class.

[figure 30]



The person class represents the users that will be using this application, they are divided into 3 types: normal users, experts, and admins. Initially, all users are created as User and the distinction is made later on. The aim of making each user be their own separate agent is to find patterns and techniques and also for monitoring purposes; if a user's annotations are especially insightful, then they can easily be noticed in the database and thus be more useful.

[figure 31]

Coordinate
- x: int - y: int - branchID: int
+ getX(): int + setX(int) + getY(): int + setY(int) + getBranchID(): int + setBranchID(int)

The Coordinate class represents the markers and the branches they belong to.

[figure 32]

Pruning
- x: int - y: int - treeID: int
+ getX(): int + setX(int) + getY(): int + setY(int) + getTreeID(): int + setTreeID(int)

The Pruning class represents the pruning points and the tree they belong to.

The coordinates are going to be used as arrays in each branch in the Annotate page while the prunings will be used with each tree in the Menu page.

[figure 33]

Branch
<ul style="list-style-type: none"> - ID: int - parentID: int - treeID: int - coordinates: int[] - children: int[]
<ul style="list-style-type: none"> + getID(): int + setID(int) + getParentID(): int + setParentID(int) + getTreeID(): int + setTreeID(int) + getCoordinates(): int[] + setCoordinates(int[]) + getChildren(): int[] + setChildren(int[])

The Branch class contains all the data fields to store the data related to each annotated branch in the Annotate page. Each tree will have an array of branches. Each branch knows its parent branch and children branches as it will store their IDs upon creation.

[figure 34]

Review
<ul style="list-style-type: none"> - userID: int - comment: string - treeID: int - importance: int
<ul style="list-style-type: none"> + getUserID(): int + setUserID(int) + getTreeID(): int + setTreeID(int) + getComment(): string + setComment(string) + getImportance(): int + setImportance(int)

The Review class represents the comments that can be added by the users in the Menu page. The importance helps to determine how much weight should be given to a review by a specific user; an expert's remark has more weight than that of a regular user.

When doing an overall analysis, these reviews can help determine patterns, techniques, and different styles in pruning and annotating.

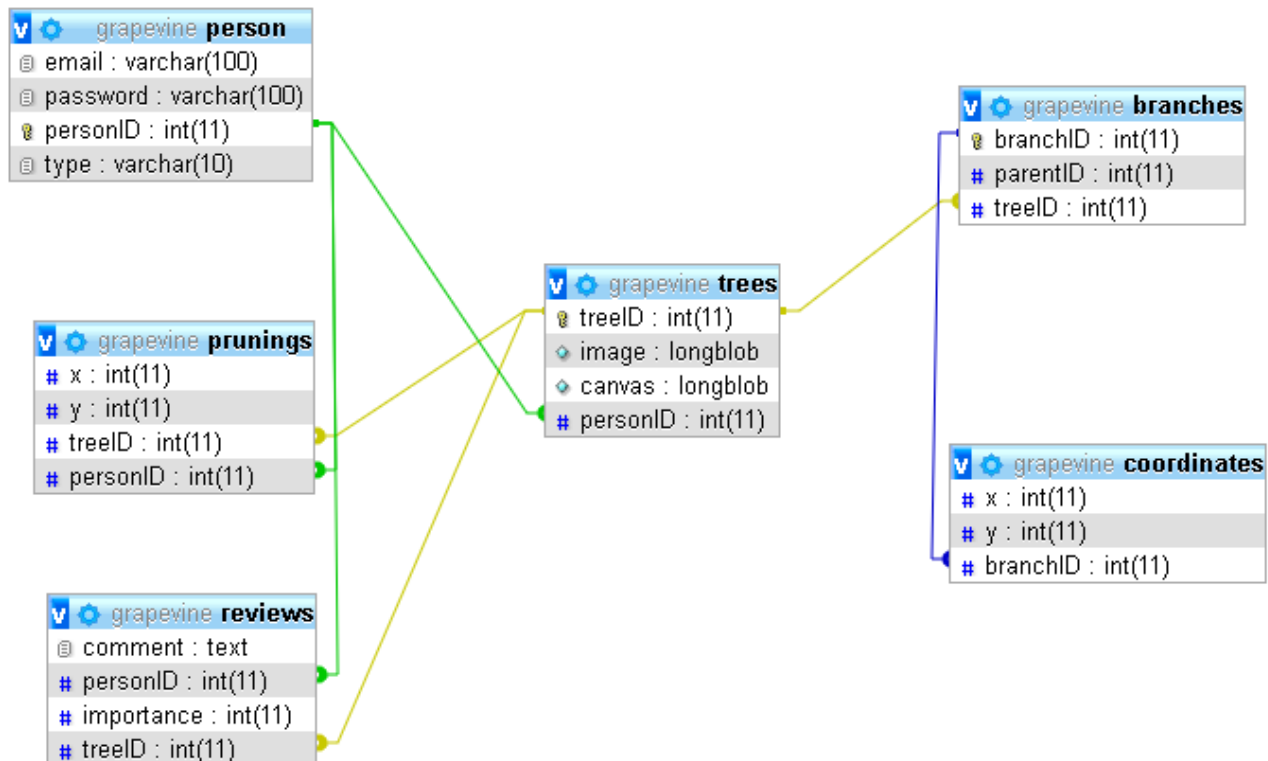
[figure 35]

Tree
<ul style="list-style-type: none"> - treeID: int - image: string - branches: int[] - reviewList: string[]
<ul style="list-style-type: none"> + getTreeID(): int + setTreeID(int) + getImage(): string + setImage(string) + getBranches(): int[] + setBranches(int[]) + getReviewList(): string[] + setReviewList(string[])

The Tree class represents the annotated tree when submitted by the user, here, the image is represented by the blob data type in the database.

These classes are all represented by their respective tables in the database and are related as follows:

[figure 36]



3.3. Methods and notions used to annotate:

The overall annotation procedure is done through multiple steps and functionalities embodied in the different buttons shown before in the Annotate page.

The classes are implemented independently from the controller of this page, and thus the first step is to import the file with the model part into the controller:

[figure 37]

```
let imported = document.createElement('script');
imported.src = 'model.js';
document.head.appendChild(imported);
```

model.js contains nothing but the implementation of all the classes detailed in the previous sub-chapter, the following is an example of what it contains:

[figure 38]

```
class Pruning
{
  constructor(x, y, treeId)
  {
    if(typeof x === 'number' && typeof y === 'number'
    && typeof treeId === 'number')
    {
      this._x = x;
      this._y = y;
      this._treeId = treeId;
    }
  }
}
```

Each button on the view part of the Annotate page has its onClick event specified:

[figure 39]

```
<div class = "buttons">
  <button onClick = "createBranch()"> Start Branch </button>
  <button onClick = "addCoordinates()"> Add Markers </button>
  <button onClick = "removeCoordinate()"> Remove Marker </button>
  <button onClick = "addBranch()"> Finish Branch </button>
  <select id = "parents"> Select Parent </select>
  <button onClick = "createTree()"> Submit </button>
</div>
```

These functions are defined in the controller part.

Another important HTML element in the view part is the Canvas element:

[figure 40]

```
<canvas id = "canvas" width = "600" height = "400">
```

We used `document.getElementById` to get the canvas element and be able to load content into the canvas from the controller using Javascript:

[figure 41]

```
let canvas = document.getElementById('canvas');  
let context = canvas.getContext("2d");
```

The same thing is done for the parents selector and various other HTML elements so that we can manipulate them using Javascript.

The image selected by the user is then loaded into the canvas using `FileReader` which makes accessing files easier in Javascript:

[figure 42]

```
let uploadedImage = document.getElementById('uploadedImage');  
uploadedImage.addEventListener('change', showImage, false);  
  
function showImage(e)  
{  
  let reader = new FileReader();  
  reader.onload = function(event)  
  {  
    let img = new Image();  
    img.onload = function()  
    {  
      context.drawImage(img, 0, 0);  
    }  
    img.src = event.target.result;  
  }  
  reader.readAsDataURL(e.target.files[0]);  
}
```

Then there is the initialization of some indices and arrays that are going to be used in the annotation:

[figure 43]

```
let counter = 0;
let coordinates = [];
let branchId = 0;
let branchCounter = 0;
let branches = [];
let marker = new Image();
marker.src = "marker.png";
let treeId = 0;
let treeCounter = 0;
let trees = [];
let userId = 0;
```

The coordinates array will be filled with the coordinates of the markers added by the user, and whenever a branch is finalized it will be added to the branches array. The marker image is given its source so that it will be drawn using the coordinates of the marker.

Another important array is the canvasLayers array which will be used to store frames of the canvas so that when the user wants to remove a marker, a previous frame of the canvas could be loaded on it which is a technique used in game development that uses the canvas element:

[figure 44]

```
let canvasLayers = [];
```

The Start Branch button has the following function associated with its onClick event:

[figure 45]

```
function createBranch()
{
    let parentId = Number(parents.value);
    let children = []
    let branch = new Branch(coordinates, parentId, branchId, 0, treeId, children);
    branches.push(branch);
}
```

It gets the value of the parent branch selector and creates a new branch that is added to the branches array. Besides the parentId and branchId, the remaining fields are just filled in with redundant values as they will be modified later when the actual values are obtained.

The clickMouse function is responsible for drawing the markers and creating the coordinates and is an event listener of our canvas element:

[figure 46]

```
function clickMouse(mouse)
```

[figure 47]

```
canvas.addEventListener("mousedown", clickMouse, false);
```

It gets the x and y coordinates of where the user clicked and makes them relative to the canvas, creates a Coordinate with the current branch ID and adds it to the coordinates array:

[figure 48]

```
let bounds = canvas.getBoundingClientRect();  
let mouseXPos = (mouse.x - bounds.left);  
let mouseYPos = (mouse.y - bounds.top);  
  
let coordinate = new Coordinate(mouseXPos, mouseYPos, branchId);  
coordinates.push(coordinate);
```

It also draws the marker on that coordinate so that the annotation is visible to the user:

[figure 49]

```
let markerX = coordinates[counter].getX;  
let markerY = coordinates[counter].getY;  
context.drawImage(marker, markerX, markerY, 5, 5);
```

One last thing it does is that it saves that frame in the canvasLayers array so that the removal of the marker is possible with the use of the canvas element:

[figure 50]

```
let imageCanvas = new Image();  
imageCanvas.src = document.getElementById("canvas").toDataURL();  
canvasLayers.push(imageCanvas);
```

At the start of a branch annotation, the very first frame is also saved so that all markers of that branch can be removed:

[figure 51]

```
if(coordinates.length === 0)
{
    let imageCanvas = new Image();
    imageCanvas.src = document.getElementById("canvas").toDataURL();
    canvasLayers.push(imageCanvas);
}
```

The addCoordinates function associated with the Add Markers button displays a small tutorial on how to annotate:

[figure 52]

```
<div>
  <p id = "tuto"> </p>
</div>
```

[figure 53]

```
function addCoordinates()
{
    let htmlId = "tuto";
    let p = document.getElementById(htmlId);
    if(p.innerHTML === "")
    {
        p.innerHTML = "1 - click on start branch and select the parent." +
            "<br> 2 - annotate the branch in question." +
            "<br> 3 - click on finish branch.";
    }
    else
    {
        p.innerHTML = "";
    }
}
```

The removeCoordinate function removes the last coordinate from the coordinates array and the last frame from the canvasLayers array and displays the frame prior to drawing the marker:

[figure 54]

```
coordinates.pop();  
counter--;  
  
canvasLayers.pop();  
let canvasIndex = canvasLayers.length - 1;  
let canvasImage = canvasLayers[canvasIndex];  
  
context.drawImage(canvasImage, 0, 0, 600, 400);
```

it also shows some warnings in case the removal is not possible as shown previously.

The addBranch function is associated with the Finish Branch button, It sets the coordinates of the current branch to the array of coordinates that the user has been adding, then resets the coordinates array and the canvasLayers array. It, then, selects a random color:

[figure 55]

```
context.strokeStyle = '#' + Math.random().toString(16).substr(-6);
```

and draws the branch following the markers:

[figure 56]

```
context.lineWidth = 4;  
context.beginPath();  
context.moveTo(tempCoordinates[0].getX, tempCoordinates[0].getY);  
  
for(let i = 0; i < tempCoordinates.length; i++)  
{  
    context.lineTo(tempCoordinates[i].getX, tempCoordinates[i].getY);  
}  
context.stroke();
```

an indicator with the branch ID is then drawn next to the branch, and the branch ID is added to the list of options in the selector. The branch is added as a child to the parent that was chosen for it from the selector:

[figure 57]

```
let bParentId = Number(parents.value);
let parentIndex = 0;

for(let i = 0; i < branches.length; i++)
{
    if(bParentId === branches[i].getId)
    {
        parentIndex = i;
    }
}

branches[parentIndex].addChild(branchId);
```

The generateTextBranches function is then called to add the coordinates of the added branch to the third pane of the Annotate page:

[figure 58]

```
function generateTextBranches(branch)
{
    let text = document.getElementById('textBranches');
    let coordinates = branch.getCoordinates();
    text.innerHTML += "Branch - " + branch.getId + "<br>";

    for(let i = 0; i < coordinates.length; i++)
    {
        let x = coordinates[i].getX;
        let y = coordinates[i].getY;
        text.innerHTML += "&nbsp; &nbsp;" + "x = " + x + ", y = " + y + "<br>";
    }
}
```

The submit button has the createTree function as its onClick event. It creates a new tree and sets its branches array to the branches array that the user has been finalizing.

At this point, all of the data related to the annotation is also present in the controller in the instances of the classes we have been working with, so this function also adds them to the database by calling a PHP script responsible for that. First, the URL of the PHP script is set using its name:

[figure 59]

```
let url = 'addTree.php';
```

We get the image uploaded using the form HTML element and the FormData object:

[figure 60]

```
const form = document.querySelector('form');

const files = document.querySelector('[type=file]').files;
let formData = new FormData();

for (let i = 0; i < files.length; i++) {

    let file = files[i];

    formData.append('files[]', file);
}
```

We get the final frame of the annotated image:

[figure 61]

```
let imageCanvas = document.getElementById("canvas").toDataURL();
```

All the remaining necessary info to add a record in the trees table is appended using the FormData object:

[figure 62]

```
formData.append('canvas', imageCanvas);

formData.append('tId', tId);

formData.append('personId', userId);
```

and then using the Fetch API, we send this data the previously specified URL of the PHP script:

[figure 63]

```
fetch(url, {

    method: 'POST',
    body: formData,

}).then(response => {

    console.log(response);
})
```


In the PHP file, in this case addTree.php, we define new variables and give them the values of the ones we just sent:

[figure 64]

```
$tId = $_POST['tId'];  
$personId = $_POST['personId'];
```

[figure 65]

```
$imageName = mysqli_real_escape_string($connection, $_FILES['files']['name'][0]);  
$imageData = mysqli_real_escape_string($connection, file_get_contents($_FILES['files']['tmp_name'][0]));  
$imageType = mysqli_real_escape_string($connection, $_FILES['files']['type'][0]);
```

[figure 66]

```
$canvas = $_POST['canvas'];
```

then we set up the connection to the database:

[figure 67]

```
// establishing database connection details:  
  
$host = "localhost";  
$dbusername = "tanulo";  
$dbpassword = "asd123";  
$dbname = "grapevine";  
  
// creating connection to the database:  
  
$connection = new mysqli ($host, $dbusername,  
    $dbpassword, $dbname);
```

we, then, check if there were any errors regarding the connection to the database and if not, we write the SQL query to add the new record to the table, which in this case is the trees table. Any errors that may happen are displayed and the connection is then closed:

[figure 68]

```
$sql = "INSERT INTO trees (treeID, image, canvas, personID)
      values ('$tId', '$imageData', '$canvas','$personId')";

if($connection->query($sql))
{
    echo "New record added successfully!";
}
else
{
    echo "Error: " . $sql . "<br>" .
        $connection->error;
}

$connection->close();
```

After adding this new record to the trees table, there is a for loop in the Javascript file, that goes through the branches of the tree we just added and does the same thing to add them to the branches table with the use of a PHP script, and within that for loop, there is another one that does the same for the coordinates of each branch.

The addTree function calls the generateTextTree function which is responsible for displaying the textual representation of the tree structure in the second pane of the Annotate page:

[figure 69]

```
for(let i = 0; i < tempBranches.length; i++)
{
    if(!order.includes(i))
    {
        order.push(i);
        let id = tempBranches[i].getId;
        text.innerHTML += "Branch - " + id + "<br>";
        let index = i;
        recursiveChildren(tempBranches, index, "----", text, order);
    }
}
```

the recursiveChildren function is called recursively so that each branch is associated with its parent in order to keep the parent-child relationship between the branches:

[figure 70]

```
function recursiveChildren(branchesArray, index, tab, text, order)
{
    let branch = branchesArray[index];
    let children = branch.getChildren();
    let id = branch.getId;

    if(children.length > 0)
    {
        for(let i = 0; i < children.length; i++)
        {
            let child = children[i];
            let childIndex = 0;

            for(let j = 0; j < branchesArray.length; j++)
            {
                if(child === branchesArray[j].getId)
                {
                    childIndex = j;
                }
            }

            if(child !== id)
            {
                text.innerHTML += "" + tab + "Branch - " + child + "<br>";
                let index = childIndex;
                order.push(index);
                recursiveChildren(branchesArray, index, tab + "----", text, order);
            }
        }
    }
}
```

And with this, we have successfully added an annotated image of a tree to the database and along with a digital representation of its branches. This tree can now be viewed on the Menu page.

3.4. Methods and notions used to prune:

The user had the ability to prune in the menu page, and they can do so for all the trees in the menu, thus this page generates a menu of all the trees and gives the possibility to annotate each one of them.

First, the connection to the database is established in the PHP script:

[figure 71]

```
// establishing database connection details:

$host = "localhost";
$dbusername = "tanulo";
$dbpassword = "asd123";
$dbname = "grapevine";

// creating connection to the database:

$connection = new mysqli ($host, $dbusername,
    $dbpassword, $dbname);
```

then we get the number of the trees from the trees database and store it in a file for later use:

[figure 72]

```
$treeNum = 0;

$sql = "SELECT max(treeID) FROM trees";

if($connection->query($sql))
{
    if ($result=mysqli_query($connection,$sql))
    {
        while ($row=mysqli_fetch_row($result))
        {
            $treeNum = $row[0];
        }
    }
}
else
{
    echo "Error: " . $sql . "<br>" .
        $connection->error;
}

file_put_contents("TreeNum.txt", $treeNum);
```

after that, we get all the trees in the database and go through each one of them using a loop:

[figure 73]

```
$sql = "SELECT * from trees";

if($connection->query($sql))
{
    $result = mysqli_query($connection, $sql);
    while ($row = mysqli_fetch_array($result))
    {
```

A collection of IDs is then defined:

[figure 74]

```
$commentId = "comment" . $row['treeID'];
$canvasId = "canvas" . $row['treeID'];
$treeId1 = "tree" . $row['treeID'] . "1";
$treeId2 = "tree" . $row['treeID'] . "2";
$preA = "preA" . $row['treeID'];
$postA = "postA" . $row['treeID'];
```

we generate the HTML part for each tree with the use of PHP:

[figure 75]

```
echo "<canvas id = '" . $canvasId . "' width = '600' height = '400'>";
echo "<img id = '" . $treeId2 . "' src='" . $row['canvas'] . "'/>";
echo "<img id = '" . $treeId1 . "' src='data:image/jpeg;base64,'" . base64_encode($row['image']) . "'/>";
echo "</canvas>";
```

Here we made it possible for both the raw image and the annotated image to be loaded onto the canvas by having two image elements with their sources defined, the annotated one (canvas, in the figure) is already encoded so we only need to use the `base64_encode` with the source if the raw image.

We also generate the buttons responsible for the pruning functionality:

[figure 76]

```
echo "<div class = 'buttons'>";
echo "<button onClick = 'prune(\". $row['treeID'] .\")'> Prune </button>";
echo "<button onClick = 'preAnnotate(\". $row['treeID'] .\")'> Pre-Annotation </button>";
echo "<button onClick = 'postAnnotate(\". $row['treeID'] .\")'> Post-Annotation </button>";
echo "<button onClick = 'removePruning(\". $row['treeID'] .\")'>Remove Pruning </button>";
echo "<button onClick = 'submitPrunings()'>Submit Prunings </button>";
echo "</div>";
```

We also add the text area and the sharing button for the comments, and for each tree, we generate its reviews list so that it is available for the users to see:

[figure 77]

```
$sql2 = "SELECT * from reviews WHERE treeID = '$tId'";
$result2 = mysqli_query($connection, $sql2);
while($row2 = mysqli_fetch_array($result2))
{
    echo "<p id = 'com'> <b> id - " . $row2['personID'] .
        ": </b>" . $row2['comment'] . "</p>";
}
```

At this point, when the user enters the Menu page, all images along with their buttons and comment sections will be generated.

Now for the functionalities of the buttons, we start by importing the model in the Javascript part as done previously:

[figure 78]

```
let imported = document.createElement('script');
imported.src = 'model.js';
document.head.appendChild(imported);
```

We, then, initialize the pruning points array and canvas frames array along with the pruning marker image:

[figure 79]

```
let pruneHere = new Image();
pruneHere.src = "pruneHere.png";
```

As we had previously stored the number of the trees in a file, we fetch it and set out trees number variable to it:

[figure 80]

```
let treeNum = 0;

fetch('TreeNum.txt')
  .then(response => response.text())
  .then(text => {
    console.log(text);
    treeNum = Number(text);
  });
```

At this point we need to iterate through the canvases we have in our page and load the images onto them:

[figure 81]

```
for(let i = 1; i <= treeNum; i++)
{
    let canvasId = "canvas" + i;
    canvas = document.getElementById(canvasId);
    let context = canvas.getContext("2d");

    let treeId = "tree" + i + "2";
    let tree = document.getElementById(treeId);

    context.drawImage(tree, 0, 0);
}
```

In order to give the possibility to view both the annotated and raw images of the trees, there are two buttons that, once clicked, will load the respective image for that:

[figure 82]

```
function preAnnotate(tId)
{
    let canvasId = "canvas" + tId;
    canvas = document.getElementById(canvasId);
    let context = canvas.getContext("2d");

    let treeId = "tree" + tId + "1";
    let tree = document.getElementById(treeId);

    context.drawImage(tree, 0, 0);
}
```

[figure 83]

```
function postAnnotate(tId)
{
    let canvasId = "canvas" + tId;
    canvas = document.getElementById(canvasId);
    let context = canvas.getContext("2d");

    let treeId = "tree" + tId + "2";
    let tree = document.getElementById(treeId);

    context.drawImage(tree, 0, 0);
}
```

In order to add a comment, we get the active user ID in the same way we did with the number of the trees, get the entry in the comments text area and send it to a PHP script that is going to add that comment to the reviews table and associate it with the user.

We append all the necessary data for the record addition using the FormData object:

[figure 84]

```
comment = document.getElementById(commentId).value;

formData.append('comment', comment);
formData.append('id', id);
formData.append('importance', importance);
formData.append('tId', tId);
```

and then using the Fetch API, we send it to URL of the PHP script:

[figure 85]

```
const url = "addComment.php";

fetch(url, {

  method: 'POST',
  body: formData,

}).then(response => {

  console.log(response);
})
```

The pruning functionality resembles that of the annotation, we get the x and y coordinates of where the user clicked, create a Pruning instance and add it the prunings array, then we save the canvas frame for the possibility of removal, the clickMouse function is then added as the event listener for the canvas:

[figure 86]

```
function clickMouse(mouse)
{
    let bounds = canvas.getBoundingClientRect();
    let mouseXPos = (mouse.x - bounds.left);
    let mouseYPos = (mouse.y - bounds.top);

    let pruning = new Pruning(mouseXPos, mouseYPos, tId);
    prunings.push(pruning);

    context.drawImage(pruneHere, mouseXPos, mouseYPos, 15, 15);

    let imageCanvas = new Image();
    imageCanvas.src = canvas.toDataURL();
    canvasLayers.push(imageCanvas);
}

canvas.addEventListener("mousedown", clickMouse, false);
```

in order to remove a pruning point, we simply remove the last addition to the pruning array and to the canvasLayers array and load the previous frame of the canvas:

[figure 87]

```
function removePruning(tId)
{
    let canvasId = "canvas" + tId;
    canvas = document.getElementById(canvasId);
    let context = canvas.getContext("2d");

    if(prunings.length > 0)
    {
        prunings.pop();

        canvasLayers.pop();
        let canvasIndex = canvasLayers.length - 1;
        let canvasImage = canvasLayers[canvasIndex];

        context.drawImage(canvasImage, 0, 0, 600, 400);
    }
}
```

When the user is done marking the pruning points and is ready to submit, he can click on the Submit button, the submitPrunings function will go through the pruning points and add them to the database:

[figure 88]

```
url = 'addPruning.php';

let x = prunings[i].getX();
let y = prunings[i].getY();
let tId = prunings[i].getTreeId();

formData.append('x', x);
formData.append('y', y);
formData.append('tId', tId);
formData.append('id', id);

fetch(url, {
  method: 'POST',
  body: formData,
}).then(response => {
  console.log(response);
})
```

After the PHP script executes, we will have pruning points associated to the tree and the user who made them which is helpful to extract more information regarding the patterns and techniques of pruning.

3.5. Methods and notions used to Login/Register:

The login and registration page are the same as the registration is done automatically in case there is no account associated with the given email address in the database.

First, we get the email and password in the PHP script when the user clicks on the submit button defined in the HTML part, and initialize a counter for errors:

[figure 89]

```
$errorCount = 0;

$email = mysqli_real_escape_string($connection, $_POST['email']);

$password = mysqli_real_escape_string($connection, $_POST['password']);
```

We perform an initial check on whether the user entered the required values:

[figure 90]

```
if(empty($email))
{
    echo "<div>";
    echo "<p> Attention: email is required.";
    echo "</div>";

    $errorCount++;
}

// checking if password field was filled in:

if(empty($password))
{
    echo "<div>";
    echo "<p> Attention: password is required.";
    echo "</div>";

    $errorCount++;
}
```

We then perform a check on whether there are any accounts associated with the email address entered, if there are, we encrypt the password and check if it is equal with the one in the database and if the passwords match, we log the user in and forward him to our About page, otherwise we output a warning to the user saying that the entered password does not match the one associated with the email address:

[figure 91]

```
$checkQuery = "SELECT * FROM person where email = '$email' LIMIT 1";
$result = mysqli_query($connection, $checkQuery);

$row = mysqli_fetch_assoc($result);

if($row)
{
    // encrypting the password:

    $password = md5($password);

    // checking if password matches:

    if($password == $row['password'])
    {
        // storing the userID in a file:

        file_put_contents("activeUser.txt", $row['personID']);

        // logging in and moving to the next page:

        $_SESSION['username'] = $email;
        $_SESSION['success'] = "You are now logged in";
        header('location: about.html');
    }
    else
    {
        echo "<div>";
        echo "<p> Attention: password doesn't match with the one associated with this email address.";
        echo "</div>";
    }
}
```

If there are no accounts with the email address entered, we encrypt the password, add a new account to the database and forward our new user to the About page:

[figure 92]

```
$password = md5($password);

// adding record to person table:

$sql = "INSERT INTO person (email, password, personID, type)
VALUES ('$email', '$password', '$userId', 'user')";

if($connection->query($sql))
{
    echo "New record added successfully!";

    // storing the userID in a file:

    file_put_contents("activeUser.txt", $userId);

    // logging in and moving to the next page:

    $_SESSION['username'] = $email;
    $_SESSION['success'] = "You are now logged in";
    header('location: about.html');
}
```

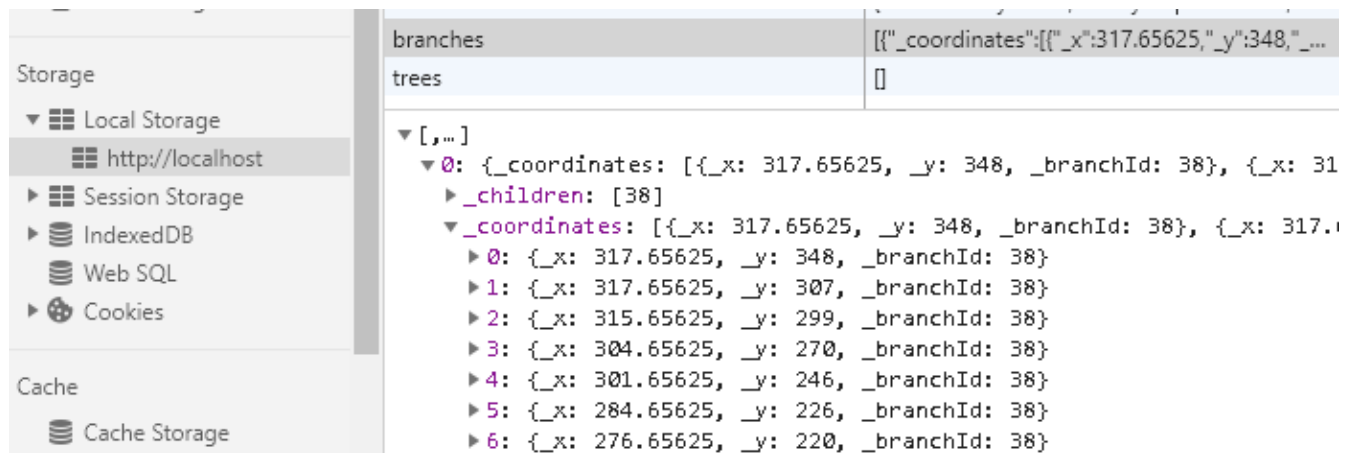
3.6. Testing:

The testing was not done with the use of unit tests but throughout the development of the application, first by using the local Storage of the current page and checking on the application storage using the browser inspect option:

[figure 93]

```
localStorage.setItem('branches', JSON.stringify(branches));
localStorage.setItem('trees', JSON.stringify(trees));
localStorage.setItem('coordinates', JSON.stringify(coordinates));
```

[figure 94]











This way, we could test in real time whether the coordinates and the branches were being added to their respective arrays. As for adding records to the database, a response could be read from the console and the details of it could be checked from the Network window in the browser inspect option:

[figure 95]



If the status is equal to 200 then we know that the data has been sent to the PHP script for the record addition, and by checking the response of the PHP script, we can see if said record addition was successful or not:

[figure 96]

Name	×	Headers	Preview	Response	Timing
 pruneHere.png	▲			1 x: 154.046875	
 data:image/png;base...				2 y: 161	
 data:image/png;base...				3 tree Id: 1	
 data:image/png;base...				4 person Id: 3	
 data:image/png;base...				5 New record added successfully!	
 activeUser.txt					
 addPruning.php					
 addPruning.php	▼				

Conclusion:

My thesis project is a take on one possibility for digitizing the tree structure and extracting the encoding necessary for the pruning procedure. This can open possibilities to develop more machine learning algorithms that can study the positioning of the pruning points with regards to the structure of the tree and the generation of the branch they are on using the data in the repository which can be extracted to give more insight on this.

Of course, more features can be added to this application and it can be used in more ways than one. The motivation behind it was to create a tool that helps us gather data on the pruning techniques, styles, and patterns of grapevines. Developing it as a web application helps make it more accessible as it can be deployed on the university server, for example, to allow said access.

One more way this application can be used is in education with fields related to arboriculture.

Overall, this is a project that will always be in continuous progress with hopes of it shining a light on more ways to encode data and information that is not majorly digitized.

Figures:

- [1] – Statistics regarding the profits associated with the use of precision farming techniques.
- [2] – Xampp Server Control Panel.
- [3] – Tables in the grapevine database using phpMyAdmin.
- [4] – Configuration button for MySQL on Xampp.
- [5] – Modifying MySQL port number if necessary.
- [6] – Status change on the Xampp server console.
- [7] – Login/Registration page.
- [8] – Login/Registration warnings.
- [9] – Login warning.
- [10] – About page.
- [11] – Navigation bar.
- [12] – Upload file button.
- [13] – Image uploaded.
- [14] – Annotation buttons.
- [15] – Small tutorial.
- [16] – Annotation markers appear.
- [17] – Marker removal warning.
- [18] – Marker removal warning.
- [19] – First branch annotated.
- [20] – Annotation buttons.
- [21] – 3rd pane in Annotate page.
- [22] – Annotated tree in 1st pane of the Annotate page.
- [23] – Textual representation of the tree in the 2nd pane of the Annotate page.
- [24] – Example tree in Menu page.
- [25] – Comment section.
- [26] – Starting the pruning procedure.
- [27] – Pruning points appear.
- [28] – Comment appears.
- [29] – Sequence diagram.
- [30] – Person class diagram.

- [31] – Coordinate class diagram.
- [32] – Pruning class diagram.
- [33] – Branch class diagram.
- [34] – Review class diagram.
- [35] – Tree class diagram.
- [36] – Relationships between the database tables.
- [37] – Importing model.js.
- [38] – Example in model.js.
- [39] – Buttons in Annotate page.
- [40] – Canvas in Annotate page.
- [41] – Getting canvas image in the Javascript file.
- [42] – Loading the uploaded image.
- [43] – Initializing indices and arrays.
- [44] – Re-setting the canvasLayers array.
- [45] – The createBranch function.
- [46] – The clickMouse function.
- [47] – Adding event Listener to the canvas.
- [48] – Creating Coordinate instance.
- [49] – Drawing the annotation marker.
- [50] – Saving the canvas frame.
- [51] – Saving the starting canvas frame.
- [52] – The p tag in the HTML part.
- [53] – The addCoordinates function.
- [54] – Removing a marker.
- [55] – Choosing a random color.
- [56] – Drawing a line on the annotated branch.
- [57] – Adding current branch as a child to its parent.
- [58] – The generateTextBranch function.
- [59] – Setting the PHP script to URL.
- [60] – Appending the uploaded image to the FormData object.
- [61] – Getting the annotated image.
- [62] – Appending data to the FormData object.
- [63] – Using the Fetch API to send data to the PHP script.

- [64] – Getting sent data in the PHP script.
- [65] – Getting sent data in the PHP script.
- [66] – Getting sent data in the PHP script.
- [67] – Establishing database connection.
- [68] – Adding tree record.
- [69] – The generateTextTree function.
- [70] – The recursiveChildren function.
- [71] – Establishing database connection.
- [72] – Getting number of trees from the database.
- [73] – Selecting all the trees and iterating through them.
- [74] – Initializing IDs.
- [75] – Generating canvas element using PHP.
- [76] – Generating buttons using PHP.
- [77] – Generating the comments associated with a tree.
- [78] – Importing model.js.
- [79] – Initializing the pruning image.
- [80] – Fetching the trees number.
- [81] – Loading the image of each canvas.
- [82] – Loading the raw image.
- [83] – Loading the annotated image.
- [84] – Appending comment data using the FormData object.
- [85] – Sending data to the PHP script using the Fetch API.
- [86] – Adding pruning points.
- [87] – Removing pruning points.
- [88] – Sending data to the PHP script using the Fetch API.
- [89] – Getting the email and password.
- [90] – Checking if the email and password are empty.
- [91] – Checking if the account already exists to log the user in.
- [92] – Adding new account for user registration.
- [93] – Adding the arrays to the local storage to test element additions.
- [94] – Checking element additions through the browser inspect option.
- [95] – Checking the response through the console.
- [96] – The checking the response of the PHP script.

Bibliography:

- [1] – “Digital agriculture: Helping to feed a growing world”:
<https://consulting.ey.com/digital-agriculture-helping-to-feed-a-growing-world/>
- [2] – <https://eos.com/blog/precision-agriculture-from-concept-to-practice/>
- [3] – <https://agecon.unl.edu/cornhusker-economics/2017/precision-agriculture-adoption-profitability>
- [4] – https://en.wikipedia.org/wiki/Precision_viticulture#cite_note-1
- [5] – “The 4 key advantages of digital data collection”:
<https://medium.com/@Progressly/the-4-key-advantages-of-digital-data-collection-c839a89f168d>
- [6] – <https://en.wikipedia.org/wiki/XAMPP>
- [7] – <https://www.phpmyadmin.net/>