

Project Report

Plant Watering Reminder

Hannemari Kuusisto

TIEVA31 – Principles of Programming Graphical User Interfaces, project work, 5cu

30.9.2021

Introduction

Plant Watering Reminder is a web application that helps users keep track of their plants' watering and fertilizing schedules. The user can add all their houseplants into the application with their watering and fertilizing frequency. The application will show which plants need to be watered each day. After watering or fertilizing a plant, the schedule will update to show the new time until each action.

Functionality

The application has the following main functionality:

- When adding a plant into the application, the user can add a picture, a species name, a name for the plant, additional notes, a watering frequency, and a fertilizing frequency.
- User can delete or edit plants.
- Application has keyboard shortcuts for adding and editing plants.
- User can undo/redo actions edit plant, add plant, and remove plant.
- Application includes a table of upcoming watering and fertilizing times.
- User can sort table according to days until next water or days until next fertilize.

In addition, it will have the following features:

- User can print the list of plants.
- Pictures can be cut/copied/pasted.

User Interface

Homepage

Homepage has two different states. Figure 1 show the beginning state where no plants have been added. There is a button linking the user to My Plants -page where they can add plants. Otherwise, the homepage shows which plants need to be watered and fertilized today as shown in Figure 2. The user can also move to different pages through the navigation bar.

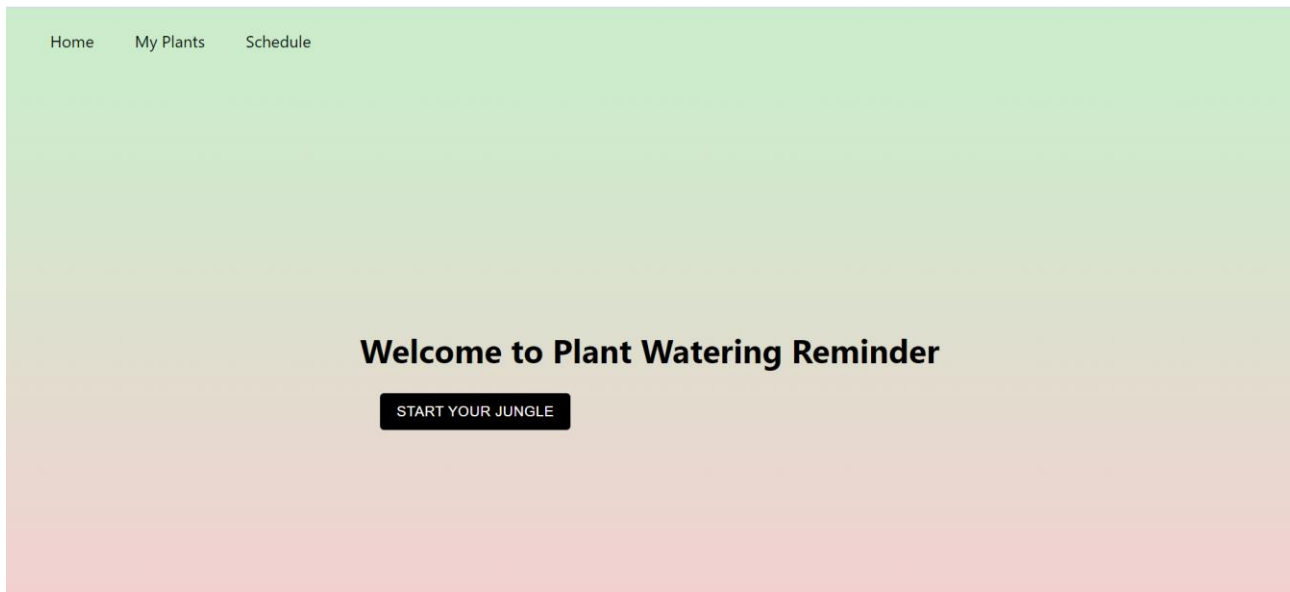


Figure 1 Homepage in the beginning

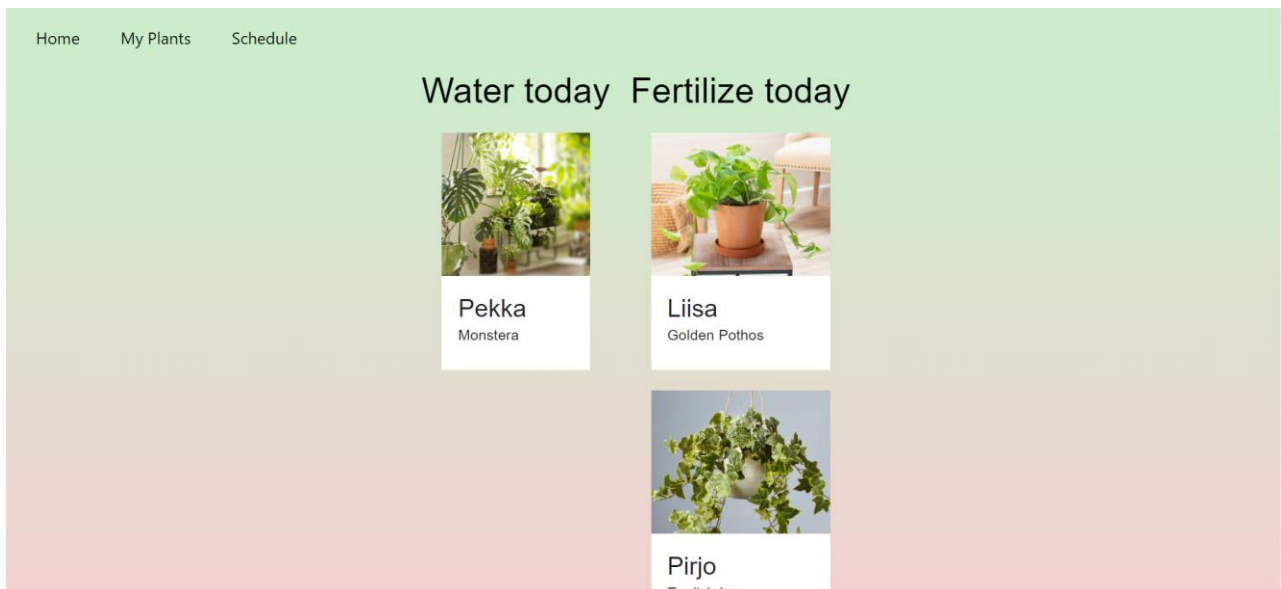


Figure 2 Homepage with plants that need to be watered/fertilized

[My Plants page](#)

My Plants page has a list of all the plants the user has added (Figure 3). More plants can be added with the Add Plant button. It opens the form shown in Figure 4. The form can also be opened with the keyboard shortcut "Ctrl+Q". In the form the user can add a picture by clicking the camera button or dragging a picture over to the image on the left. Name, species, watering frequency and fertilizing frequency are required fields, but notes is not.

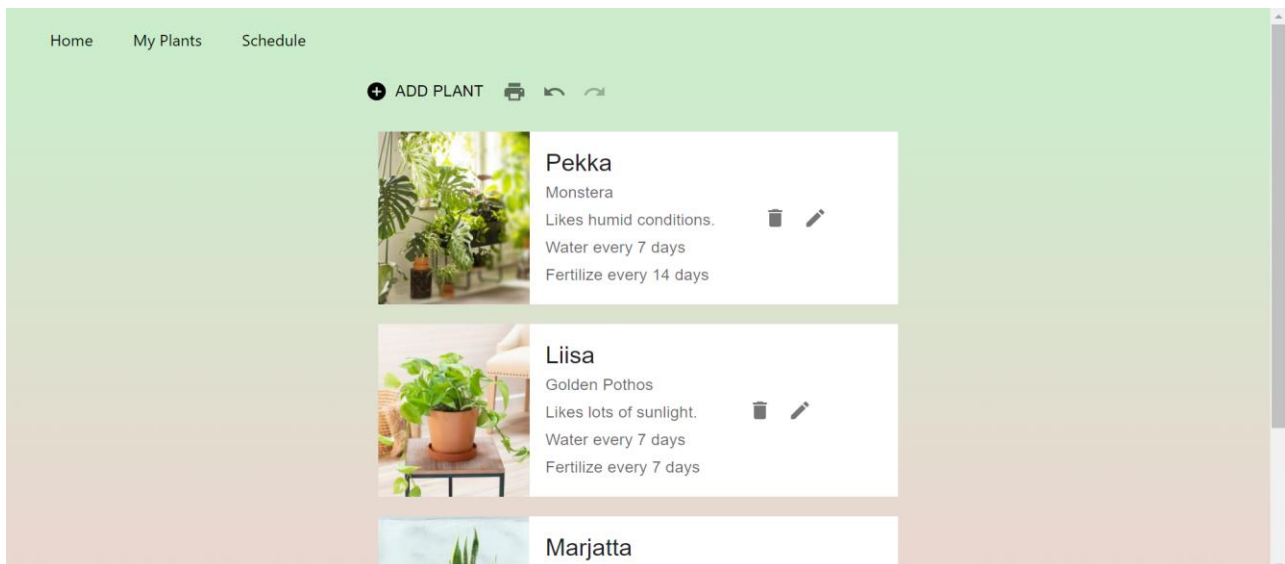


Figure 3 My Plants page with plant list

The screenshot shows the 'Add Plant' form with a green header bar. The form has a large image placeholder on the left and several input fields on the right. The fields are: Name *, Species *, Notes, Watering frequency *, and Fertilizing frequency *. There are also CANCEL and ADD PLANT buttons at the bottom right.


Field	Value
Name *	
Species *	
Notes	
Watering frequency *	0
Fertilizing frequency *	0

Figure 4 Form for adding a new plant

A plant's information can be edited by clicking the edit icon. This opens the form with the plant's current information. Plants can also be edited with the keyboard shortcut "Ctrl+Y". A plant can be removed by clicking the delete icon. All the actions can be undone and redone with the arrow icons at the top of the page. The print icon opens the print window with the buttons removed from the list (Figure 6).

Home My Plants Schedule

+ ADD PLANT

 Name * Pekka Watering frequency * 7

Species * Monstera Fertilizing frequency * 14

Notes Likes humid conditions.

CANCEL SAVE CHANGES



 Liisa Golden Pothos Likes lots of sunlight. Water every 7 days Fertilize every 7 days


Figure 5 Plant edit mode


Apps How to


Home M

10/12/1, 9:07 AM Read App

 Pekka Monstera Likes humid conditions. Water every 8 days Fertilize every 15 days

 Liisa Golden Pothos Likes lots of sunlight. Water every 7 days Fertilize every 7 days

 Marjatta Snakeplant Let soil dry between waterings. Water every 14 days Fertilize every 30 days

 Pirjo English ivy Water every 10 days Fertilize every 10 days

Print 1 sheet of paper

Destination Adobe PDF

Pages All

Layout Portrait

Color Color

More settings

Print Cancel

Snakeplant

Figure 6 Print mode

Schedule

Figure 7 shows the Schedule page which has a table of all the plants with the days until their watering and fertilizing. This page still has some problems, and the days don't always show correctly. In Figure 8 the rows have been sorted in ascending order by days until next water. This can be done by clicking the arrow icon next to the column header. The arrow icons only show when the mouse hovers over them.

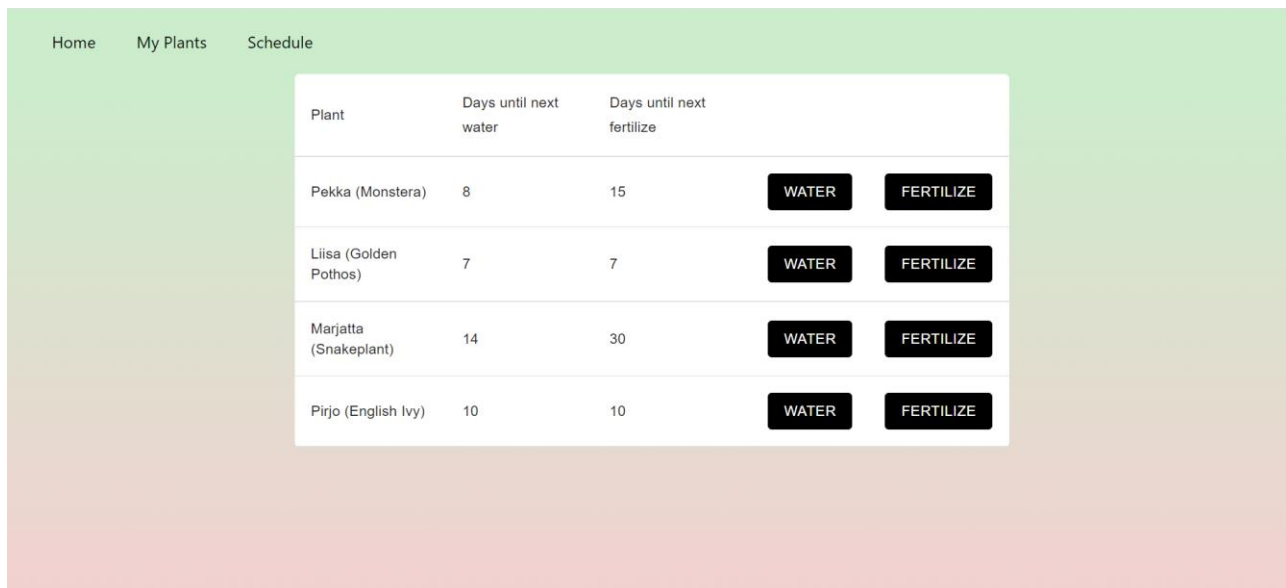


Figure 7 Schedule page

Plant	Days until next water ↑	Days until next fertilize		
Liisa (Golden Pothos)	7	7	WATER	FERTILIZE
Pekka (Monstera)	8	15	WATER	FERTILIZE
Pirjo (English Ivy)	10	10	WATER	FERTILIZE
Marjatta (Snakeplant)	14	30	WATER	FERTILIZE

Figure 8 Schedule sorted by days until next water

Software Structure

Files are within the “src” folder. “Index.js” and “App.js” are directly inside “src”. Rest of the files are divided into assets, components, pages, and styles folders. Assets folder has the default image used in plant form. Components folder includes “Navbar.js”, “NewPlantForm.js”, “Plant.js” and “PlantInfoCard.js”. Pages folder includes “Home.js”, “MyPlants.js” and “Schedule.js”. Styles folder includes “Styles.js”.

“App.js”

Component state

Plants

- State for storing all the plant objects in an array

- Initian state: empty array

Functionality

updatePlants

```
const updatePlants = (updatedPlants) => {
  | setPlants(updatedPlants);
}
```

- Receives updatedPlants from child component MyPlants and sets them in plants state

“MyPlants.js”

Component state

plantItems

- State for storing plants
- Initial state: empty array

formOpen

- State for storing Boolean value which determines if NewPlantForm is shown
- Initial state: false

undoStack

- State for storing plantItems arrays so user can return to previous state
- Initial state: empty array

redoStack

- State for storing plantItems arrays before undo so undo can be redone
- Initial state: empty array

Functionality

handleKeyDown

```
const handleKeyDown = (event) => {
  | if (event.ctrlKey && event.code === "KeyQ") {
  |   | setFormOpen(true);
  | }
  | if (event.code === "Escape") {
  |   | setFormOpen(false);
  | }
}
```

- Sets new plant form open if keyboard shortcut “Ctrl+Q” is pressed
- Closes form if escape is pressed on the keyboard

undo

```
const undo = (event) => {
  | setRedoStack(redoStack.concat([plantItems]));
  | setPlantItems(undoStack[undoStack.length-1]);
  | setUndoStack(undoStack.slice(0, undoStack.length-1));
}
```

- Adds current plantItems to redo stack
- Reverts plantItems to previous state
- Removes last item from undo stack

redo

```
const redo = (event) => {
  setUndoStack(undoStack.concat([plantItems]));
  setPlantItems(redoStack[redoStack.length-1]);
  setRedoStack(redoStack.slice(0, redoStack.length-1));
}
```

- Adds current plantItems to undo stack
- Reverts plantItems to the state before last undo
- Removes last item from redo stack

handleSubmit

```
const handleSubmit = (event, newPlant) => {
  event.preventDefault();

  const plantObject = {
    image: newPlant[0],
    name: newPlant[1],
    species: newPlant[2],
    water: parseInt(newPlant[3]),
    fertilize: parseInt(newPlant[4]),
    notes: newPlant[5],
    id: Date.now().toString(),
    nextWater: Date.now() + newPlant[3]*(24*60*60*1000),
    nextFertilize: Date.now() + newPlant[4]*(24*60*60*1000),
  }

  const newPlantList = plantItems.concat([plantObject])
  setPlantItems(newPlantList);
  updatePlants(newPlantList);
  setUndoStack(undoStack.concat([plantItems]));
  setRedoStack([]);
}
```

- Receives newPlant information from NewPlantForm and makes them into a plantObject
- Adds new plantObject to plantItems
- Sends newPlantList to parent component App
- Adds current plantItems to undo stack
- Clears redo stack

handleClose

```
const handleClose = () => {
  setFormOpen(false);
}
```

- Closes NewPlantForm

handleEdit

```
const handleEdit = (event, editedInfo) => {
  event.preventDefault();

  const editedPlant = {
    image: editedInfo[0],
    name: editedInfo[1],
    species: editedInfo[2],
    water: editedInfo[3],
    fertilize: editedInfo[4],
    notes: editedInfo[5],
    id: editedInfo[6]
  }
  let index = editedInfo[7];
  let editedPlants = plantItems.slice(0, index).concat([editedPlant]).concat(plantItems.slice(index+1));
  setPlantItems(editedPlants);
  updatePlants(editedPlants);
  setUndoStack(undoStack.concat([plantItems]));
  setRedoStack([]);
}
```

- Receives edited plant info from NewPlantForm and makes it into Object
- Removes old plant object and adds the new one in its place
- Sends edited plant list to parent component App
- Takes care of undo and redo stacks

removePlant

```
const removePlant = (index) => {
  let newArr = plantItems.slice(0, index).concat(plantItems.slice(index+1));
  setPlantItems(newArr);
  updatePlants(newArr);
  setUndoStack(undoStack.concat([plantItems]));
  setRedoStack([]);
}
```

- Deletes a plant from list of plants
- Takes care of undo and redo stacks

printPage

```
const printPage = () => {
  window.print();
}
```

- Opens print window

useEffect

```
useEffect(() => {
  let newArr = [...plants];
  newArr.forEach(plant => {
    plant.nextWater = new Date().getTime() + plant.water*(24*60*60*1000);
    plant.nextFertilize = new Date().getTime() + plant.fertilize*(24*60*60*1000);
  })
  setPlantItems(newArr);
}, []);
```

- Once on render calculates the time for each plant's next watering and fertilizing based on watering frequency and fertilizing frequency.
- Probably this shouldn't be done here, and it is causing problems in the schedule.

useEffect

```
useEffect(() => {
  window.addEventListener("keydown", handleKeyDown);

  return () => window.removeEventListener("keydown", handleKeyDown);
}, [])
```

- Adds event listener for keydown

"Plant.js"

Component state

editOpen

- State for storing Boolean which determines if NewPlantForm is shown
- Initial state: false

Functionality

handleKeyDown

```
const handleKeyDown = (event) => {
  if (event.ctrlKey && event.code === "KeyY") {
    setEditOpen(true);
  }
  if (event.code === "Escape") {
    setEditOpen(false);
  }
}
```

- Sets all edit forms open if keyboard shortcut "Ctrl+Y" is pressed
- Not optimal but couldn't figure out how to open just one
- Closes forms if escape is pressed on the keyboard

handleSave

```
const handleSave = (e, newInfo) => {  
  handleSubmit(e, newInfo);  
}
```

- Receives edited plant information from NewPlantForm and sends it to parent component App

handleClose

```
const handleClose = () => {  
  setEditOpen(false);  
}
```

- Closes edit form

useEffect

```
useEffect(() => {  
  window.addEventListener("keydown", handleKeyDown);  
  
  return () => window.removeEventListener("keydown", handleKeyDown);  
}, [])
```

- Adds event listener for keydown

"NewPlantForm.js"

Component state

plantInfo

- State for storing individual plant's information
- Initial state: props.info
 - has plantinfo if called from Plant component
 - empty if called from MyPlant

Functionality

handleSaveClick

```
const handleSaveClick = (e, plantInfo) => {  
  console.log(plantInfo, "plantInfo")  
  props.handleSubmit(e, plantInfo);  
  setPlantInfo([TestImage, "", "", 0, 0, "", ""]);  
  props.handleClose();  
}
```

- send information to parent component after save click
- clears form and closes it

handleInfoChange

```
const handleInfoChange = (event, id) => {  
  let newArr = [...plantInfo];  
  newArr[id] = event.target.value;  
  setPlantInfo(newArr);  
}
```

- updates form fields

handleImage

```
const handleImage = (event) => {  
  let file = event.target.files[0];  
  if (file && file.name.match(/\.(jpg|jpeg|png|gif)$/)) {  
    let reader = new FileReader()  
    reader.onloadend = function(evt) {  
      let newArr = [...plantInfo];  
      newArr[0] = reader.result;  
      setPlantInfo(newArr);  
    }  
    reader.readAsDataURL(file);  
  }  
}
```

- saves image

ImageDragStart, ImageOnDrop, ImageDragEnter, & ImageDragOver

```
const imageDragStart = (event) => {
  event.dataTransfer.setData("image/jpeg", event.target.src);
  event.dataTransfer.dropEffect = "copy";
}

const ImageOnDrop = (event) => {
  event.preventDefault();
  let file = event.dataTransfer.files[0];
  if (file && file.name.match(/\.(jpg|jpeg|png|gif)$/)) {
    let reader = new FileReader()
    reader.onloadend = function(evt) {
      let newArr = [...plantInfo];
      newArr[0] = reader.result;
      setPlantInfo(newArr);
    }
    reader.readAsDataURL(file);
  }
}

const imageDragEnter = (event) => {
  event.preventDefault();
  let file = event.dataTransfer.files[0];
  if (file && file.name.match(/\.(jpg|jpeg|png|gif)$/)) {
    event.preventDefault();
  }
}

const imageDragOver = (event) => {
  event.preventDefault();
  event.dataTransfer.dropEffect = "copy";
}
```

- handles image drag

“Schedule.js”

Component state

plantSchedule

- state for storing plants
- initial state: empty array

sortCol

- state for storing column to be sorted
- initial state: 0

SortAsc

- State for storing Boolean which tells if sort order is ascending
- Initial state: false

Functionality

waterPlant & fertilizePlant

```
const waterPlant = (event, id) => {
  let index = plantSchedule.findIndex(elem => elem.id === id);
  let untilWater = plantSchedule[index].nextWater;
  let newWater = untilWater + plantSchedule[index].water;
  let newArr = [...plantSchedule];
  newArr[index].nextWater = newWater;
  setPlantSchedule(newArr);
}

const fertilizePlant = (event, id) => {
  let index = plantSchedule.findIndex(elem => elem.id === id);
  let untilFertilize = plantSchedule[index].nextFertilize;
  let newFertilize = untilFertilize + plantSchedule[index].fertilize;
  let newArr = [...plantSchedule];
  newArr[index].nextFertilize = newFertilize;
  setPlantSchedule(newArr);
}
```

- Gets the index of the plant that is being watered/fertilized
- Adds up remaining days until next water and watering frequency
- Sets new remaining days until water

updateSorting

```
const updateSorting = (event, column) => {
  if (column === sortCol) {
    setSortAsc(!sortAsc);
  }
  setSortCol(column);
}

const sortedPlants = plantSchedule.sort((a, b) => {
  let dirMult = (sortAsc?1:-1);
  if (sortCol === 1)
    return (a.water-b.water)*dirMult;
  if (sortCol === 2)
    return (a.fertilize-b.fertilize)*dirMult;
  return 0;
});
```

- Sorts table

useEffect

```
useEffect(() => {  
  let newArr = [...plants];  
  newArr.forEach(plant => {  
    plant.nextWater = Math.round((plant.nextWater - Date.now()) / 1000 / 60 / 60 / 24);  
    plant.nextFertilize = Math.round((plant.nextFertilize - Date.now()) / 1000 / 60 / 60 / 24);  
  })  
  setPlantSchedule(newArr);  
}, [])
```

- Once on render replaces nextWater date with calculated days until next water
- Maybe should rethink this

Used Tools

This project was developed using React create-react-app and Material UI library with Visual Studio Code.

Third-party libraries:

- Material UI (npm install @mui/material @emotion/react @emotion/styled)
- React Router Dom (npm install react-router-dom)

Project Timeline

Project started beginning of September and finished 30.9.2021.