1 Setting up the project

Make sure you have NodeJS installed on your machine.

- Install electron globally: `npm install -g electron`
- Clone the git repo: `git clone https://github.com/FelixStarke/e-portfolio.git`
- Navigate to the project folder: `cd e-portfolio/hands-on/example`
- Initialize node project: `npm init`
  - Follow the instructions and set the properties according to your needs.
  - Change *entry point* to `app/main.js`
- Install electron for that project: `npm install electron --save`
- Within the created *package.json* add `"start": "electron ."` inside *scripts* (you can remove *test* for this demo) so you start your application with the `npm start`-command

2 Getting started

Create the *main.js* file and and proceed by importing all the modules that are used in this demo as well as create a first window.

2.1 Import modules

```
const {app, BrowserWindow, Menu, dialog, nativeImage} = require('electron');
const path = require('path');
const url = require('url');
```

2.2 Create window

The class *BrowserWindow* represents one of the applications windows. You can set many different properties when creating the window, however we are only using a few simple ones, like the window size and background color. If you want to know all the other options, visit `https://github.com/electron/electron/blob/master/docs/api/browser-window.md`.
Next we are loading an html-file to display. For that you should create an *index.html* in the *app*-folder and add the html-structure as described in step 2.3. We are loading our html from the file system, but you could also load it from an external source.

```
let window;

function createWindow() {
    window = new BrowserWindow({
        width: 1280,
        height: 720,
        backgroundColor: '#000'
    });

    window.loadURL(url.format({
        pathname: path.join(__dirname, 'index.html'),
        protocol: 'file',
        slashes: true
    }));

    window.on('close', () => {
        window = null;
    });
}

app.on('ready', createWindow);

app.on('window-all-closed', () => {
    if (process.platform !== 'darwin') {
        app.quit();
    }
});
```

2.3 Add HTML and renderer.js

Create the *index.html* and *renderer.js* files for the frontend of the application.
Add a basic HTML-structure, link the provided stylesheet (*styles.css*) and import *renderer.js*, which you can leave empty for now.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Simple Slideshow</title>
    <link rel="stylesheet" type="text/css" href="styles.css" />
  </head>
  <body>
    <script>
      require('./renderer.js');
    </script>
  </body>
</html>
```

3 Custom menu

Create a new menu template and add it to your application with
`Menu.setApplicationMenu(Menu.buildFromTemplate(menuTemplate));`
when creating your window.

```
const menuTemplate = [
    {
        label: 'File',
        submenu: [
            {
                label: 'Open Images',
                accelerator: 'CmdOrCtrl+O'
            }
        ]
    },
    {
        label: 'View',
        submenu: [
            {
                label: 'Slow',
                accelerator: 'CmdOrCtrl+1'
            },
            {
                label: 'Medium',
                accelerator: 'CmdOrCtrl+2'
            },
            {
                label: 'Fast',
                accelerator: 'CmdOrCtrl+3'
            },
            {
                type: 'separator'
            },
            {
                role: 'togglefullscreen',
                accelerator: 'F'
            },
            {
                role: 'toggledevtools',
                accelerator: 'F12'
            }
        ]
    }
];
```

4 Inter-Process Communication (IPC)

Send the message *resize* and the current window size to the renderer process.

```
function onResize() {
    window.webContents.send('resize', { width: window.getContentSize()[0], height: window.getContentSize()[1] ↙
        ↳ } );
}
```

Receive the message *resize* in the renderer process and handle it.

```
const ipc = require('electron').ipcRenderer;

ipc.on('resize', (event, data) => {
    console.log(data);
});
```

Add the following to statements to the window creation process to emit the *resize*-event when resizing and once the frontend is loaded.

```
window.on('resize', onResize);
window.webContents.on('did-finish-load', onResize);
```

5 Displaying an image

5.1 Add the img into your html

Add the following into your html-body. Make sure to use the same id's to make it work with the provided stylesheet.

```
<div id="container">
    <img id="image" />
</div>
```

Replace the console log: resize the container every time the window resizes.

```
ipc.on('resize', (event, data) => {
    document.getElementById('container').style.width = data.width + 'px';
    document.getElementById('container').style.height = data.height + 'px';
});
```

5.2 Selecting images

Extend the *Open Images* menu item with the following click-event-handler.

```
{
    label: 'Open Images',
    accelerator: 'CmdOrCtrl+O',
    click () {
        dialog.showOpenDialog(window, { filters: [ { name: 'Images', extensions: ['jpg', 'png'] } ], ↙
            ↳ properties: [ 'openFile', 'multiSelections' ] }, openImages);
    }
}
```

Create the function *openImages* that handles the result of the open dialog. Also add a variable to store your image-paths. We are only going to show a single image for now.

```
let imageQ = [];

function openImages(filePaths) {
    if (!filePaths) return;
    imageQ = filePaths;

    window.webContents.send('show', imageQ[0]);
}
```

Implement the receiver in the renderer process and display the image.

```
ipc.on('show', (event, data) => {
    document.getElementById('image').src = data;
});
```

6 Randomly iterate through the image queue

Extract displaying the image into a new function *update* and pick a random image out of the image queue every update.

```
function openImages(filePaths) {
    if (!filePaths) return;
    imageQ = filePaths;

    update();
}

function update() {
    if (!imageQ.length) return;

    const i = Math.floor(Math.random() * imageQ.length);
    window.webContents.send('show', imageQ[i]);
}
```

Start an interval that updates the image every 2 seconds.

```
function openImages(filePaths) {
    if (!filePaths) return;
    imageQ = filePaths;

    update();
    setInterval(update, 2000);
}
```

7 Changing interval

Add the variables *interval* and *intervalTime* and add the function *updateInterval*.

```
let interval = null;
let intervalTime = 10000;

function updateInterval(iTime) {
    intervalTime = iTime;
    if (!imageQ.length) return;

    clearInterval(interval);
    interval = setInterval(update, intervalTime);
}
```

Instead of calling *setInterval* directly, use *updateInterval* in *openImages*.

```
function openImages(filePaths) {
    if (!filePaths) return;
    imageQ = filePaths;

    update();
    updateInterval(intervalTime);
}
```

Update the interval times whenever you click on the corresponding menu item.

```
{
    label: 'Slow',
    accelerator: 'CmdOrCtrl+1',
    click () {
        updateInterval(30000);
    }
    },
    {
    label: 'Medium',
    accelerator: 'CmdOrCtrl+2',
    click () {
        updateInterval(10000);
    }
    },
    {
    label: 'Fast',
    accelerator: 'CmdOrCtrl+3',
    click () {
        updateInterval(2000);
    }
}
```

Additionaly make sure to clear the interval when the window closes before you set it to *null*.

```
window.on('close', () => {
    clearInterval(interval);
    window = null;
});
```

8 Display image info

To display info on the image, add the following *span*-tag after the container *div*-tag containing the *img*-tag. Make sure to use the same id.

```
<span id="info-text"></span>
```

After updating the image, use the module *nativeImage* to load the image and send the size information to the renderer process.

```
const img = nativeImage.createFromPath(imageQ[i]);
window.webContents.send('info', img.getSize().width + ' x ' + img.getSize().height + ' px');
```

Implement the corresponding receiver in the renderer process.

```
ipc.on('info', (event, data) => {
    document.getElementById('info-text').textContent = data;
});
```