FOTOTECA EXPLAINED

1. Initial Setup
   1. Use Express to create routes and to set the server’s port (app.listen)
      1. Set express.static to use “public” as a folder for public files
      2. Set express urlencoded to be able to submit POST requests
   2. Use Morgan middleware to log requests received by the server (Morgan is npm package; a popular HTTP request logger middleware, which provides a simple way to log HTTP requests and their details; useful for monitoring and debugging)
   3. Create tagsArr as a new Set() - Set is a built-in object in JS that allows you to store only unique values of any type;
   4. Set “view engine” to EJS - templating engines used in JS web development. Allows you to generate HTML dynamically, rendering templates with data passed from your server. EJS syntax is similar to plain HTML with embedded JavaScript code that runs on the server side. Other templating engines are Handlebars (used in our ArtBox project) and Pug (formerly known as Jade).
2. Setting up the routes and rendering a template and passing the necessary data to the view
   1. Home / get route
   2. Search / get route
      1. Set a search form in the partial nav view
      2. use the submit button to send the query string to the server
      3. Get req.query and compare it to the objects title using filter and includes methods -> const searchResult = images.filter(i => i.title.toUpperCase().includes(keyword.toUpperCase() );
   3. Filter / get route
      1. Another form on home.ejs which renders only if the tagArr set.size is more than 0; then onchange (so that there is no need of submit button to pass info to server) the filterImages function is executed (triggers submit) -> document.getElementById(“filterForm”).submit();
         1. add-image-form / get route
            1. Create a new Date variable for the current day (to limit the calendar to today)
            2. Render image-form and pass the variables needed by the view
         2. add-image-form / Post route
            1. Set today variable to today via new Date
            2. Deconstruct req.body coming from client’s form
            3. Error Management

Server side validation for title presence and title length

Client side validation for title symbols via regex pattern (pattern="[a-zA-Z0-9\_ ]+"); imageUrl url (pattern="[https://.\*](about:blank)") ; date type “date”)

* + - * 1. Check if image url already exists isImageAlreadyAdded = images.some(i => i.imageUrl === imageUrl) -> if true -> triggers error message that it already exists which disappears by clicking on it;
        2. If imageUrl is not duplicated, we proceed to use color-thief-node nm package to get the dominant color and add it to the object alongside with the rest of properties coming from the image-form view using promises
        3. in the Second then, receiving and parsing the tags JSON string that comes from tagify (cool tags input component that lets you add multiple tags in a form using tab or space bar (you can adapt it to your needs), also lets you delete a tag if you have committed an error). Adds separately the tags from the parsed tags array (as they are added to an object declared as Set, they are only added if they don’t already exist)
        4. On the next .then use .sort to order the images by date
        5. On the last .then render image-form and pass all necessary variables to the view.
        6. Catch error in case of error
        7. In the “else” manages the case of duplicated url, setting the variable “isImageAlreadyAdded” to true and triggering an error to inform the user
      1. /images/:id/delete / Post Route
         1. Sets a dynamic route, thanks to which we get the image id on the server side from req.params and apply the following logic to remove it from the images array that is shown on home -> images = images.filter(i => i.id != id)