Multer

Objective: *Be able to upload and store files in the database.*

Uses of Multer

Why is an external library such as multer needed to handle input type file in form submissions?

Files are stored inside the database as a reference, but the file in itself is not stored. When a file is sent to the server, it is stored inside a folder { avatar } and is referenced inside the database.

Multer

- is a nodejs middleware that helps us achieve the above functionality.
- helps in handling multipart form-data i.e. data from a form that has input fields that collect both text and files.

If the form data is multipart, it won't be accessible using the *body property (req.body)* of the request that goes to the controller after submitting the form. This happens because the request can not read the input type file. To achieve this, multer middleware is used.

How does multer middleware helps read input type file?

Middleware is a function that is executed before the controller accesses the input data submitted. It has access to the request, and response objects and can analyse, and process data present in request and response.

As a middleware can edit the request object, multer adds a `body` object and a `file` object to the `request` object.

- The `body` object contains the values of the text fields of the form.
- The `file` object contains the files uploaded via the form.

This is how multer helps the request get access to the file submitted as input. Thereafter, this request is sent to the controller which can further access it using ***req.file.***

***NOTE:** Multer will NOT process any form which is not multipart. It means `enctype="multipart/form-data"` should be set for the form.*

Setting and Configuring Multer

In our project, Codeial, multer is required to store profile images (or avatars) of the users. So we will set up and configure multer inside the users schema as corresponding to each user, an avatar will be created which needs to be processed and stored. So another field, 'avatars' with type String will be added to the users schema as well.

Setting up

```
const multer = require('multer');
const path = require('path');
const AVATAR_PATH = path.join('/uploads/users/avatars');
```

Configuring

```
const storage = multer.diskStorage({
    destination: function (req, file, cb) {
        cb(null, path.join(__dirname, '..', AVATAR_PATH));
    },
    filename: function (req, file, cb) {
        const uniqueSuffix = Date.now() ;
        cb(null, file.fieldname + '-' + uniqueSuffix);
    }
});

userSchema.statics.uploadedAvatar =  multer({storage:
storage}).single('avatar');

userSchema.statics.avatarPath = AVATAR_PATH;
```

- *multer.diskStorage()* is a function which tells multer that disk storage will be used to store the files. It takes in an argument which is an object with destination and filename properties.

    - *destination* specifies the folder path for the folder where files will be stored.
    - *filename* specifies the name of the file which will be stored in the destination folder.

- *schemaName.statics* helps in creating static methods or variables for the schema. Here static methods/variables are those that will have a single instance of the occurrence for every user i.e. they are common for all the users (instead of defining them individually for every user, we define them once and this same definition will be used by all the users(user objects)).

    - *usersSchema.statics.uploadedAvatar* is storing a static function which sets the *storage* property of multer to the diskStorage that was earlier defined, which tells it where the uploaded files will be stored.
    - *single('avatar')* means only one file can be uploaded for the field 'avatar' of the users schema, i.e. one user can upload only one avatar.
    - *usersSchema.statics.avatarPath* is storing a static variable which tells the folder path of the folder where files will be stored.

Using Multer

Once the user submits the multi-part form, the action of the form gets triggered. And the specified route is called, which maps the post request to the associated controller.

- Here, in project Codeial, when the submit button is clicked on the users_profile page to update the user details, the control goes to the update function of the users_controller.
- This update function further uses the *uploadedAvatar()* static function which was defined earlier, to store the input file from the *req.file* into the specified folder *(which was specified using multer.diskStorage()).*