Module 3

Purwadhika
Digital Technology School

Full Stack Web Development

Advanced Topic

Data Validation, File Upload & Emailer

Job Connector Program

Outline

- Data Validation
- File Upload
- Emailer

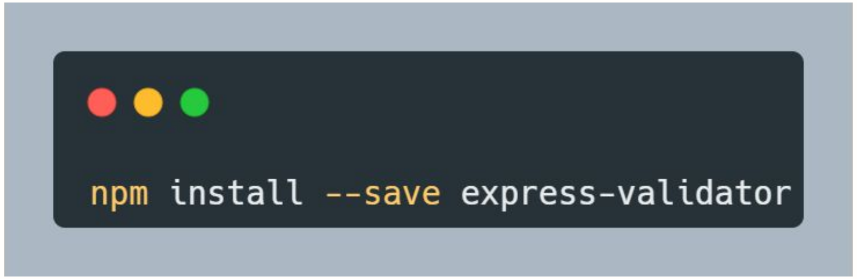
Data Validation

In computer science, data validation is the process of ensuring data has undergone data cleansing to ensure they have data quality, that is, that they are both correct and useful. It uses routines, often called "validation rules", "validation constraints", or "check routines", that check for correctness, meaningfulness, and security of data that are input to the system.

Data Validation

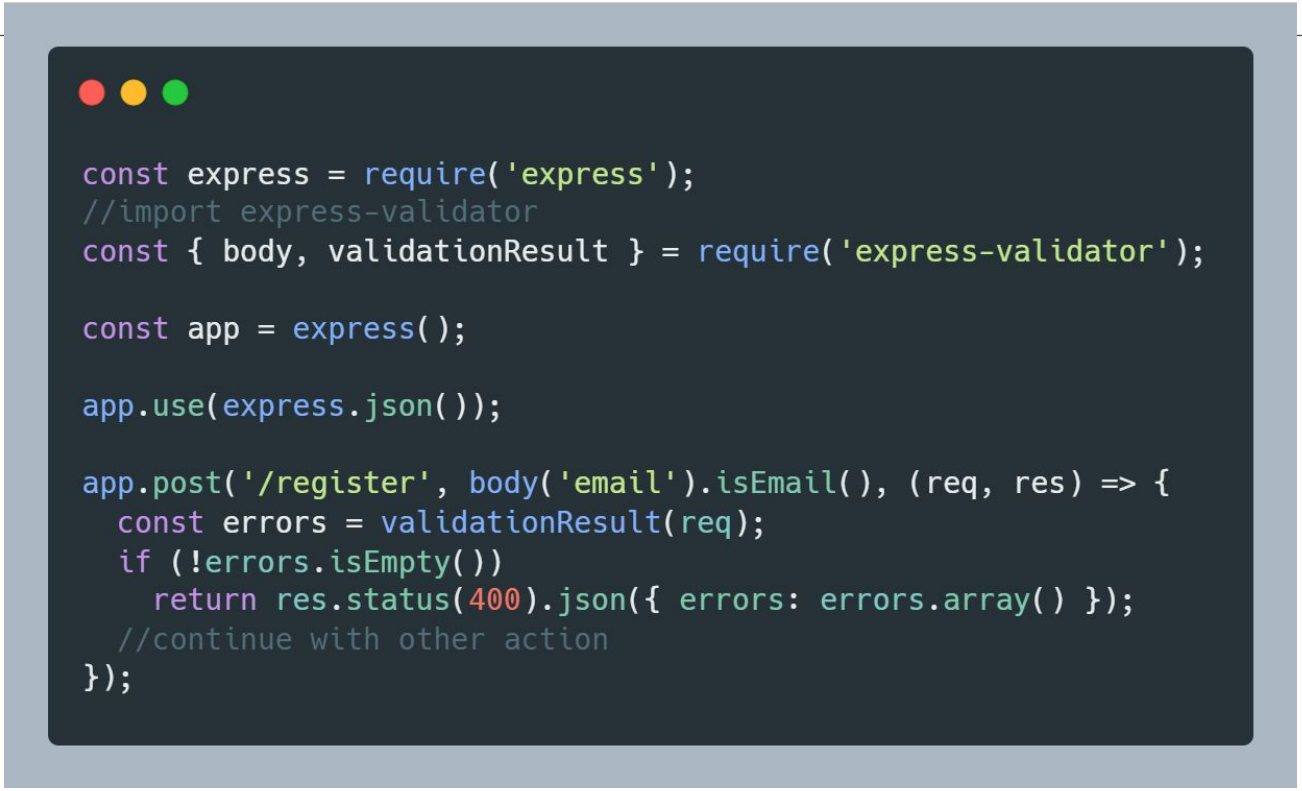
There are so many validator package you can use, but for this time we will learn about **express-validator** for our data validation.

express-validator is a set of express.js middlewares that wraps validator.js validator and sanitizer functions.



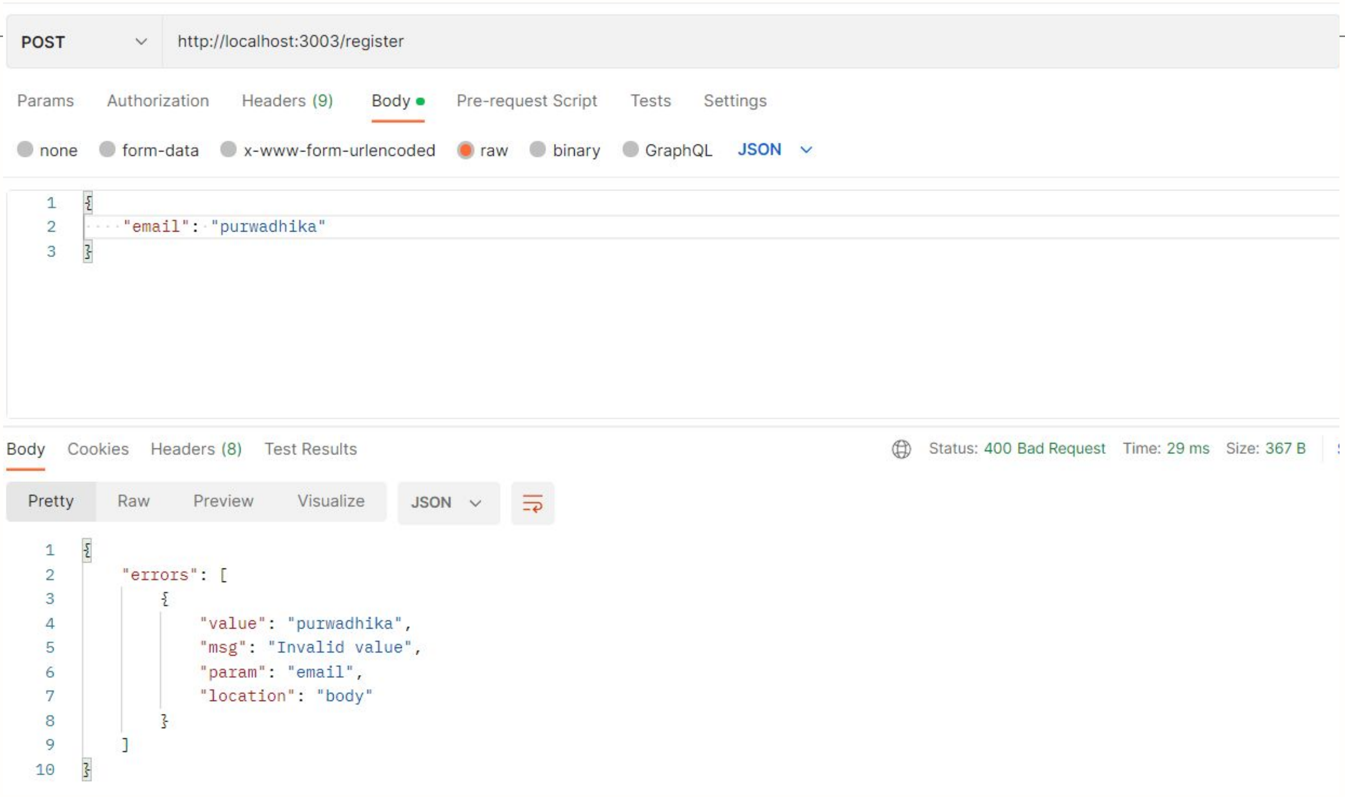
```
npm install --save express-validator
```

Basic Usage Express Validator as Middleware



```
const express = require('express');
//import express-validator
const { body, validationResult } = require('express-validator');

const app = express();

app.use(express.json());

app.post('/register', body('email').isEmail(), (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty())
    return res.status(400).json({ errors: errors.array() });
  //continue with other action
});
```

Test on Postman



POST http://localhost:3003/register

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests | Settings

```
{
    "email": "purwadhika"
}
```

Body | Cookies | Headers (8) | Test Results — Status: 400 Bad Request Time: 29 ms Size: 367 B

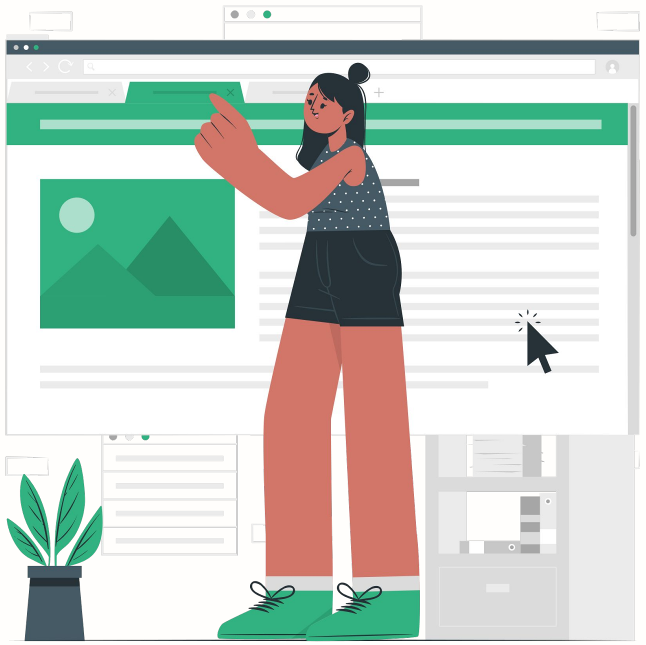
Pretty | Raw | Preview | Visualize | JSON

```
{
    "errors": [
        {
            "value": "purwadhika",
            "msg": "Invalid value",
            "param": "email",
            "location": "body"
        }
    ]
}
```

File Upload

File upload is an activity to transmitting a file from one computer system to another. Uploaded files can be in any format, such as images, documents, audio, video, etc.

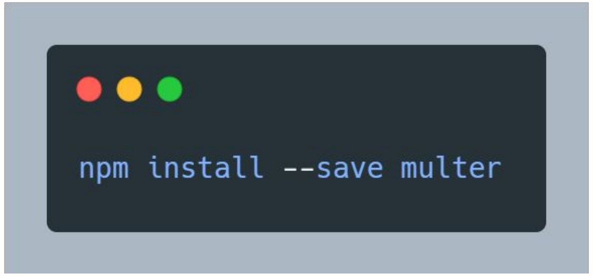
To be able to create a file upload feature, we can use a package called Multer. Multer is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files.



Multter

Usage:

Multer adds a body object and a file or files object to the request object. The body object contains the values of the text fields of the form, the file or files object contains the files uploaded via the form.



```
npm install --save multer
```

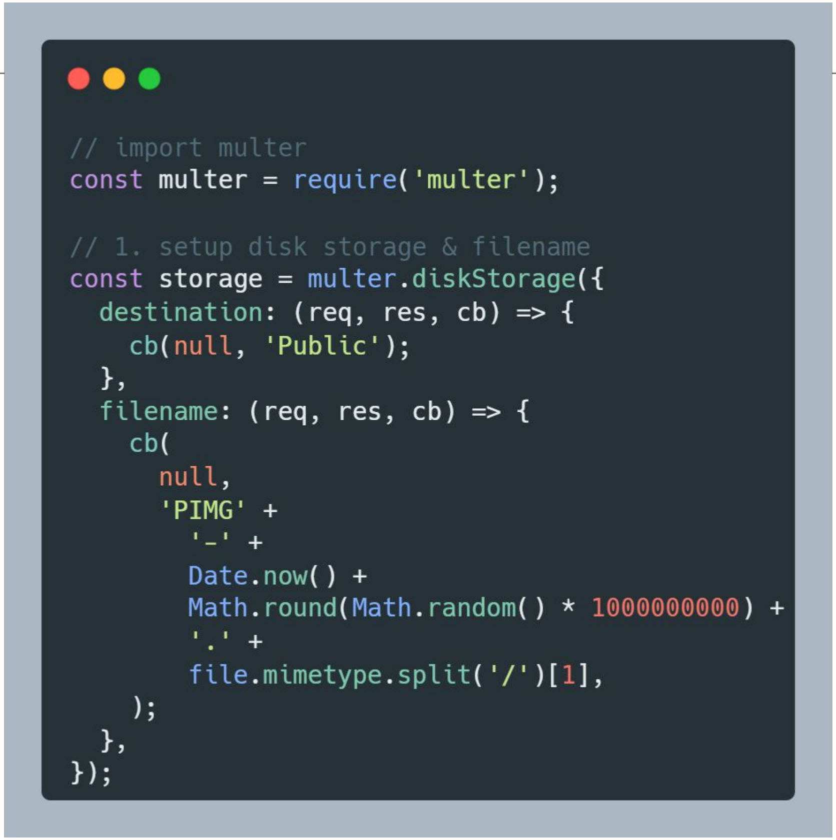
File Upload

File upload is an activity to transmitting a file from one computer system to another. Uploaded files can be in any format, such as images, documents, audio, video, etc.

To be able to create a file upload feature, we can use a package called Multer. Multer is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files.

Setup Multer-1

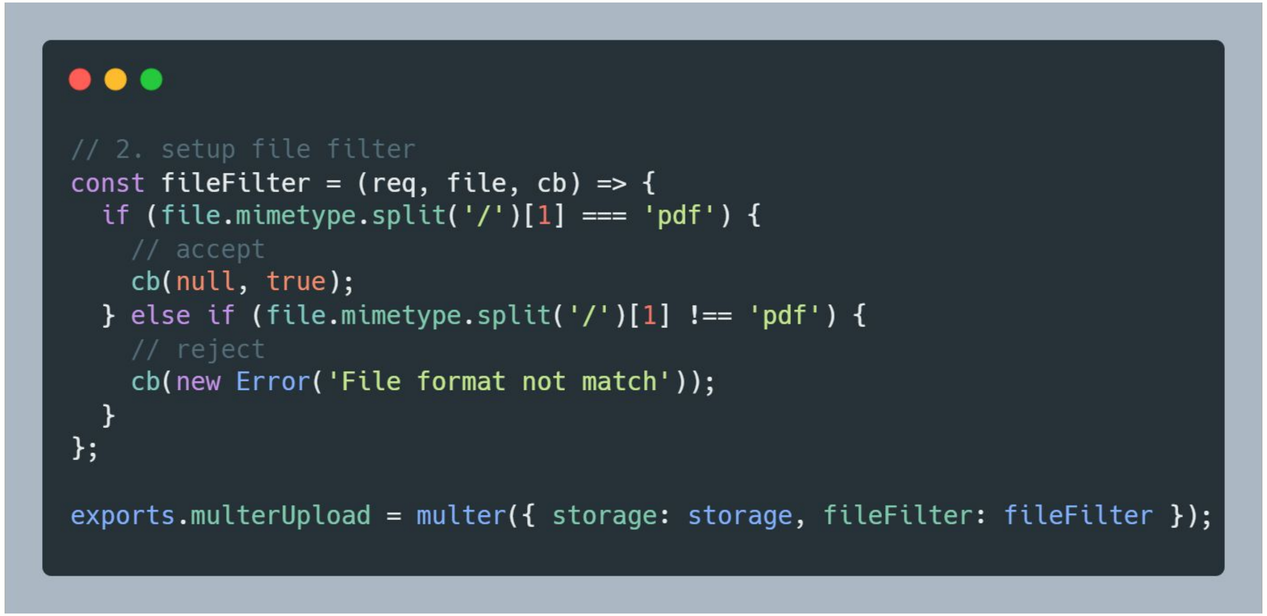
Put this code into one file (ex. Multer.js)



```javascript
// import multer
const multer = require('multer');

// 1. setup disk storage & filename
const storage = multer.diskStorage({
  destination: (req, res, cb) => {
    cb(null, 'Public');
  },
  filename: (req, res, cb) => {
    cb(
      null,
      'PIMG' +
        '-' +
        Date.now() +
        Math.round(Math.random() * 1000000000) +
        '.' +
        file.mimetype.split('/')[1],
    );
  },
});
```

Setup Multer-2

Put this code below into one file (ex. Multer.js)



```
// 2. setup file filter
const fileFilter = (req, file, cb) => {
  if (file.mimetype.split('/')[1] === 'pdf') {
    // accept
    cb(null, true);
  } else if (file.mimetype.split('/')[1] !== 'pdf') {
    // reject
    cb(new Error('File format not match'));
  }
};

exports.multerUpload = multer({ storage: storage, fileFilter: fileFilter });
```

Options: diskStorage

Destination is used to determine within which folder the uploaded files should be stored. This can also be given as a string (ex. 'Public'). If no destination is given, the operating system's default directory for temporary files is used.

Note: You are responsible for creating the directory when providing destination as a function. When passing a string, multer will make sure that the directory is created for you.

Filename is used to determine what the file should be named inside the folder. If no filename is given, each file will be given a random name that doesn't include any file extension.

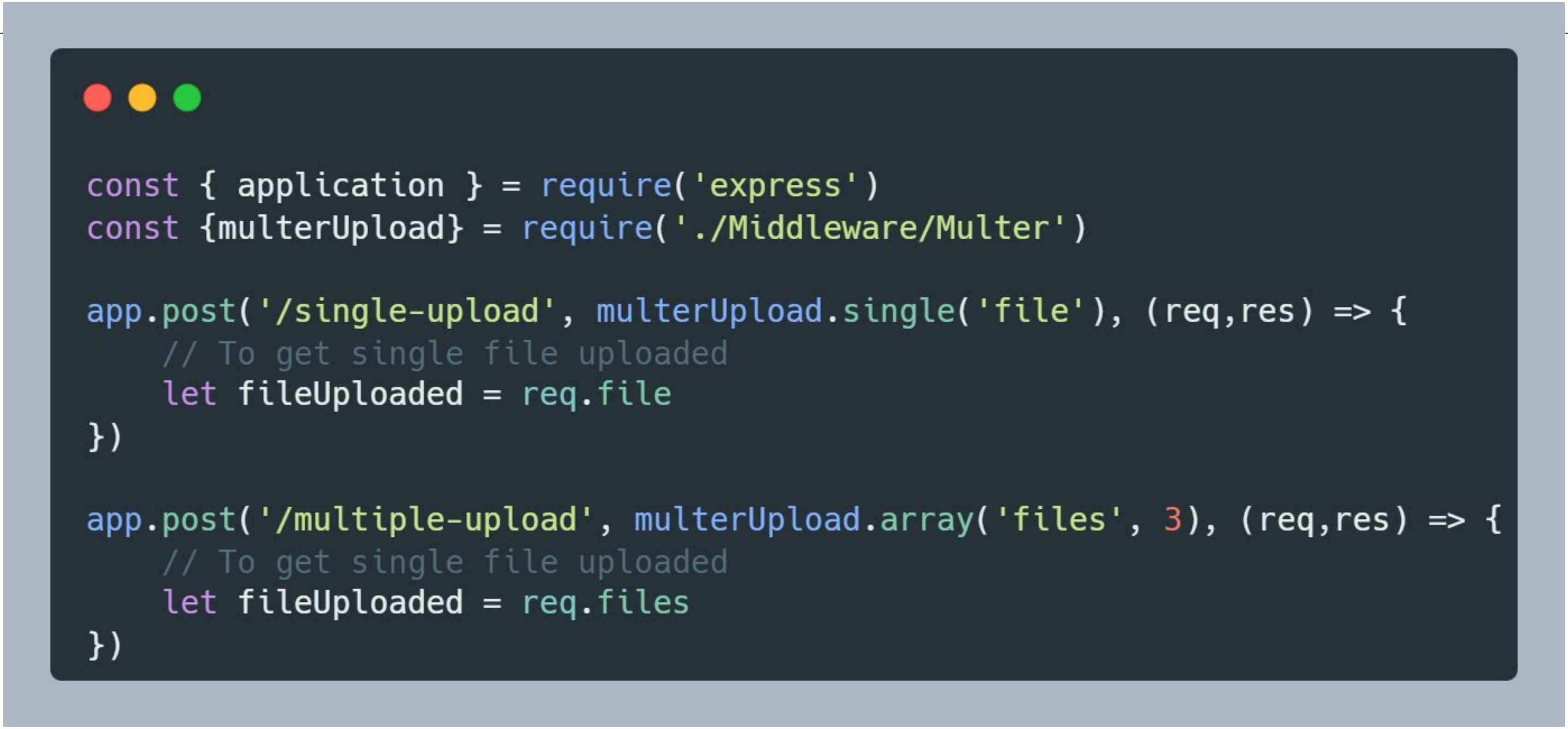
Note: Multer will not append any file extension for you, your function should return a filename complete with an file extension.

Options: fileFilter

FileFilter is used to filter the file format.

Note: Function to filter files must be defined in multer's option object.

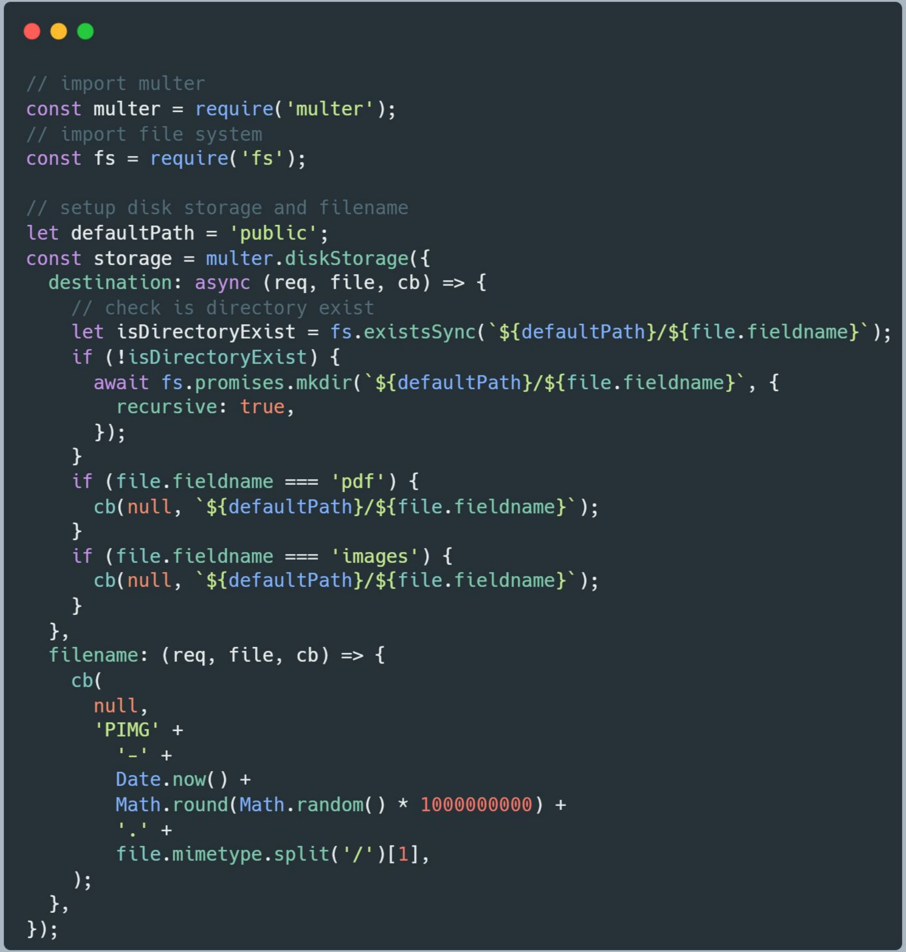
Multter as Middleware



```
const { application } = require('express')
const {multerUpload} = require('./Middleware/Multer')

app.post('/single-upload', multerUpload.single('file'), (req,res) => {
    // To get single file uploaded
    let fileUploaded = req.file
})

app.post('/multiple-upload', multerUpload.array('files', 3), (req,res) => {
    // To get single file uploaded
    let fileUploaded = req.files
})
```

Setup Multer with Default Field/Directory #1

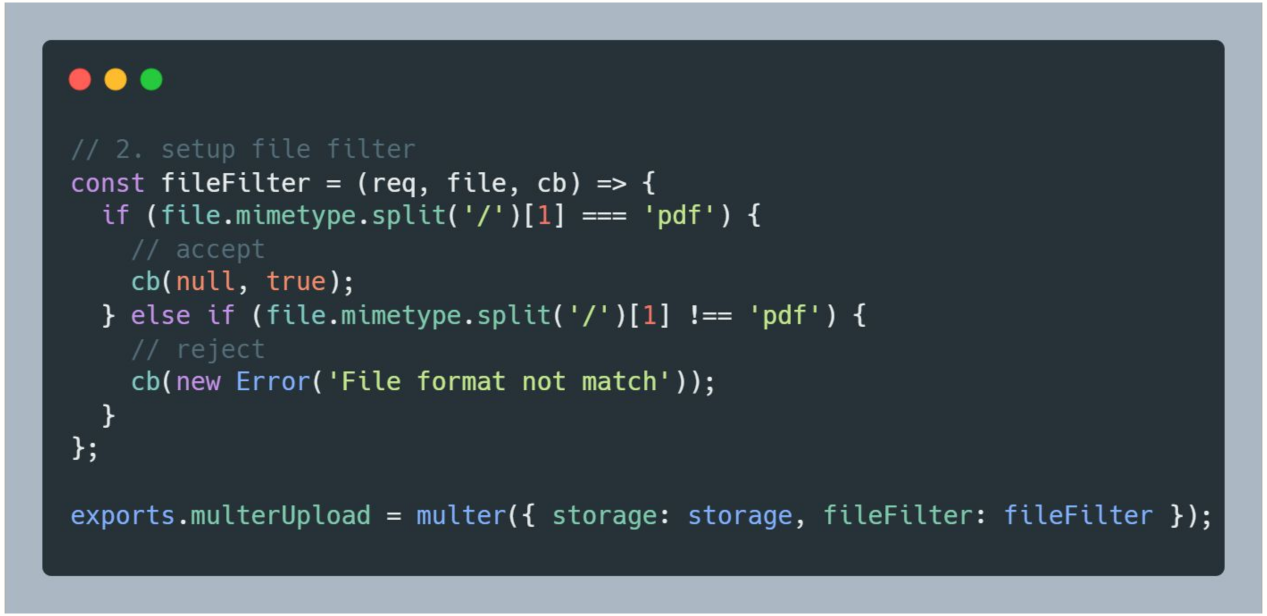
Put this code below into one file (ex. Multer.js)



```javascript
// import multer
const multer = require('multer');
// import file system
const fs = require('fs');

// setup disk storage and filename
let defaultPath = 'public';
const storage = multer.diskStorage({
  destination: async (req, file, cb) => {
    // check is directory exist
    let isDirectoryExist = fs.existsSync(`${defaultPath}/${file.fieldname}`);
    if (!isDirectoryExist) {
      await fs.promises.mkdir(`${defaultPath}/${file.fieldname}`, {
        recursive: true,
      });
    }
    if (file.fieldname === 'pdf') {
      cb(null, `${defaultPath}/${file.fieldname}`);
    }
    if (file.fieldname === 'images') {
      cb(null, `${defaultPath}/${file.fieldname}`);
    }
  },
  filename: (req, file, cb) => {
    cb(
      null,
      'PIMG' +
        '-' +
        Date.now() +
        Math.round(Math.random() * 1000000000) +
        '.' +
        file.mimetype.split('/')[1],
    );
  },
});
```

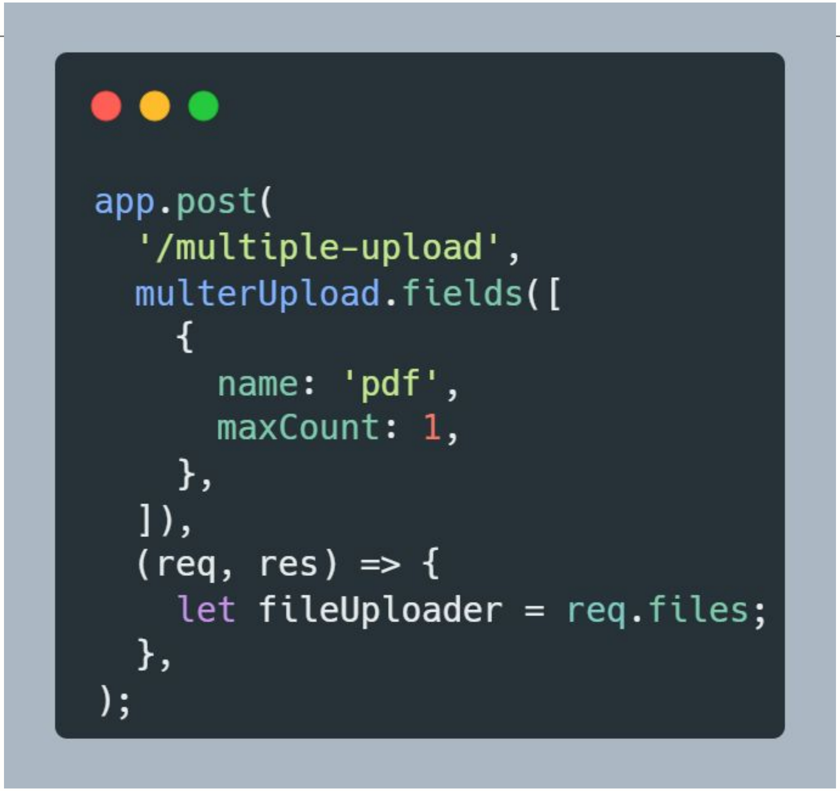
Setup Multer-2

Put this code below into one file (ex. Multer.js)



```
// 2. setup file filter
const fileFilter = (req, file, cb) => {
  if (file.mimetype.split('/')[1] === 'pdf') {
    // accept
    cb(null, true);
  } else if (file.mimetype.split('/')[1] !== 'pdf') {
    // reject
    cb(new Error('File format not match'));
  }
};

exports.multerUpload = multer({ storage: storage, fileFilter: fileFilter });
```

Multer as Middleware with Default Field/Directory #3

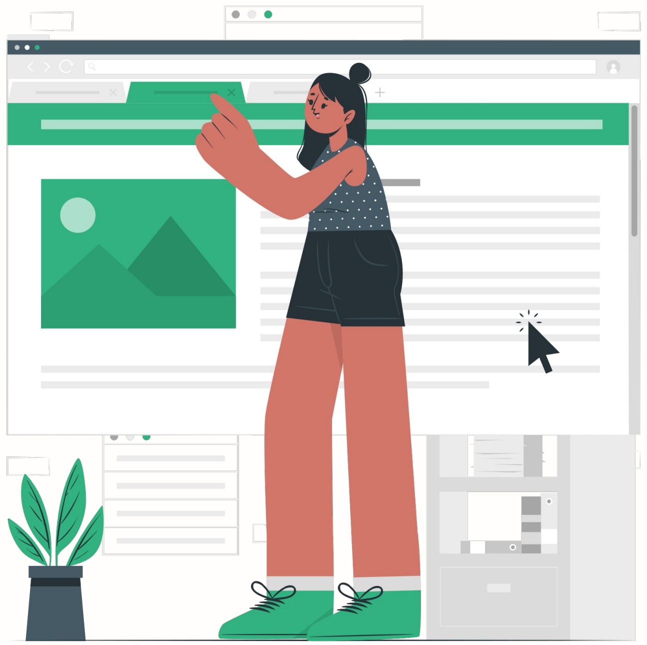


```
app.post(
  '/multiple-upload',
  multerUpload.fields([
    {
      name: 'pdf',
      maxCount: 1,
    },
  ]),
  (req, res) => {
    let fileUploader = req.files;
  },
);
```

Emailer

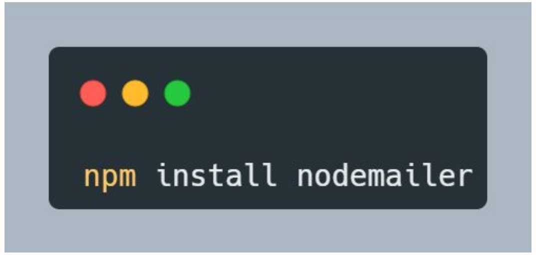
In some features we need to perform activities that send emails to users, for example such as verification, forgot password, etc. Therefore we can use a package called Nodemailer.

Nodemailer is a module for Node.js applications to allow easy as cake email sending.



Nodemailer

Requirements:
If you are able to run Node.js version 6 or newer, then you can use Nodemailer. There are no platform or resource specific requirements. All public Nodemailer methods support both callbacks and Promises (if callback is comitted). You need to have at least Node v8.0.0 if you want to use *async..await* with Nodemailer.



```
npm install nodemailer
```

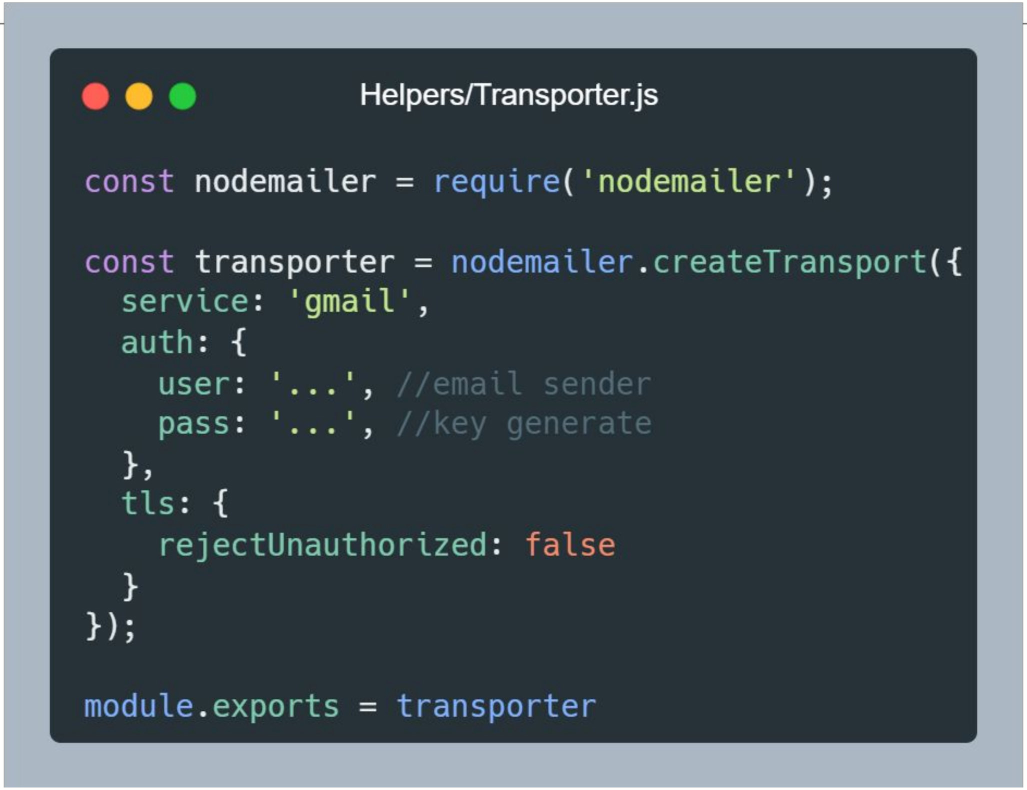
Setup Nodemailer

In short, what you need to do to send messages, would be the following:

1. Create a Nodemailer transporter using either SMTP or some other transport mechanism
2. Set up message options (who sends what to whom)
3. Deliver the message object using the **sendMail()** method of your previously created transporter

Nodemailer with Gmail as Transporter

Create file Transporter.js and write down this code to setup to handle email process

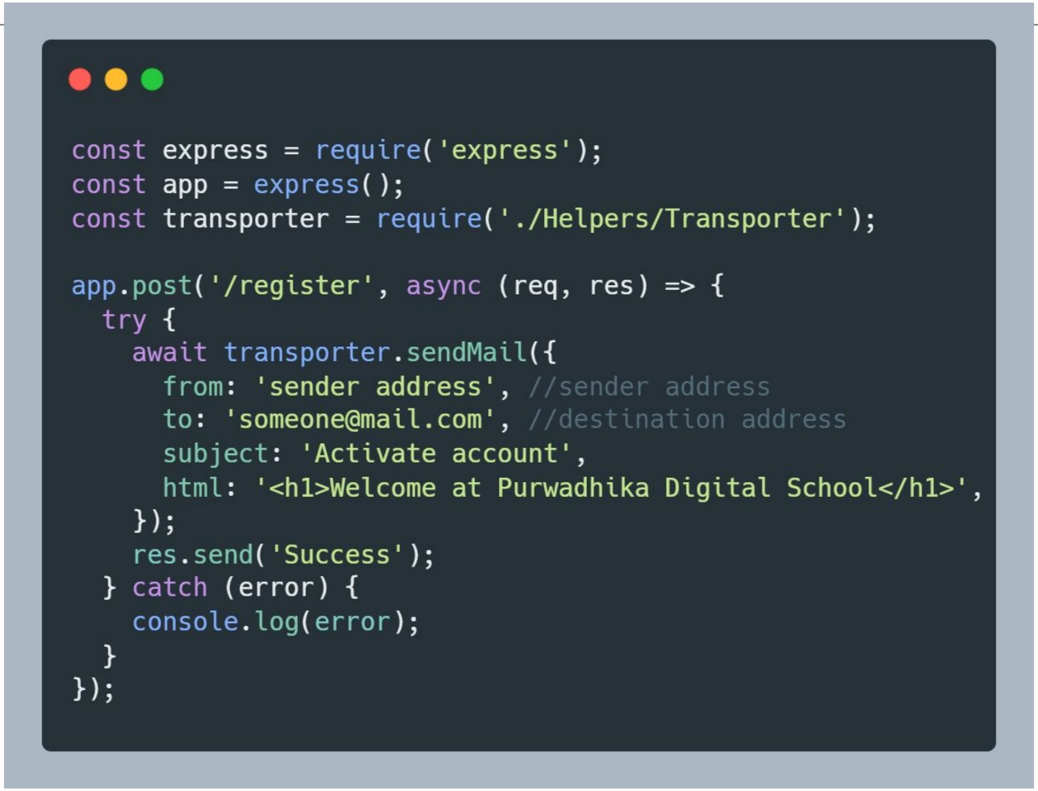


Helpers/Transporter.js

```javascript
const nodemailer = require('nodemailer');

const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: '...', //email sender
    pass: '...', //key generate
  },
  tls: {
    rejectUnauthorized: false
  }
});

module.exports = transporter
```

Send Email with Defined Transport Object

Purwadhika
Digital Technology School



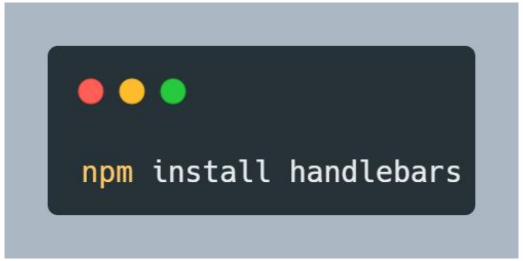
```javascript
const express = require('express');
const app = express();
const transporter = require('./Helpers/Transporter');

app.post('/register', async (req, res) => {
  try {
    await transporter.sendMail({
      from: 'sender address', //sender address
      to: 'someone@mail.com', //destination address
      subject: 'Activate account',
      html: '<h1>Welcome at Purwadhika Digital School</h1>',
    });
    res.send('Success');
  } catch (error) {
    console.log(error);
  }
});
```

Handlebars for Email Templating

Handlebars is a simple templating language. It uses a template and an input object to generate HTML or other text formats. Handlebars templates look like regular text with embedded Handlebars expressions like this:

```
t<p>{{firstname}} {{lastname}}</p>
```

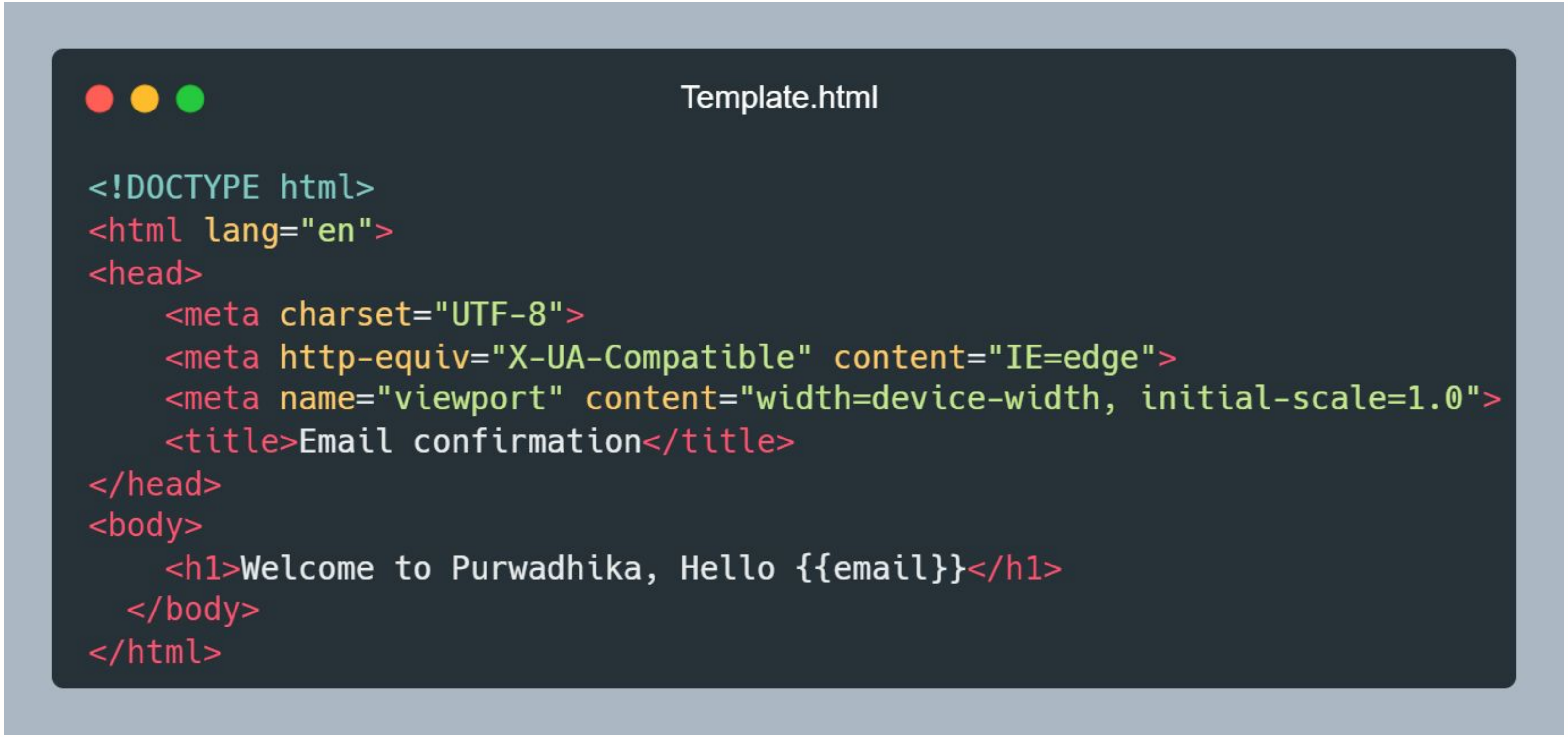
When the template is executed, these expressions are replaced with values from an input object.



```
npm install handlebars
```

Email Template

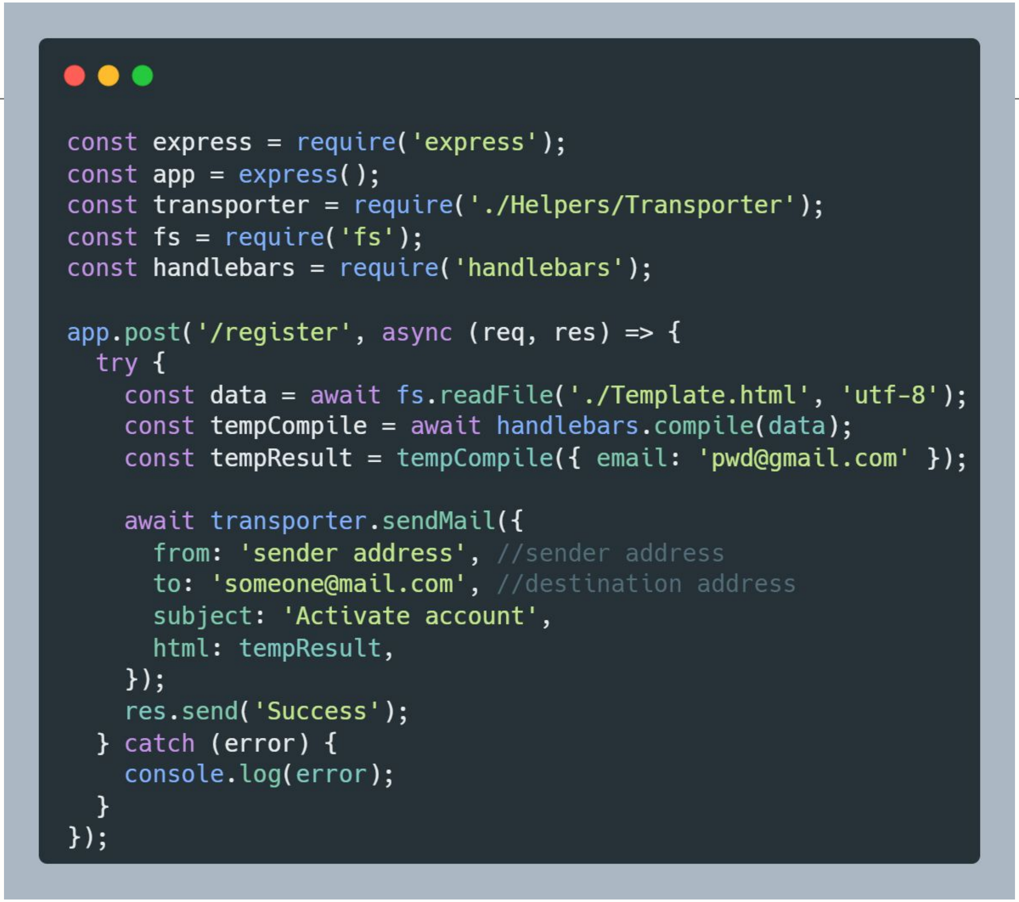
Content inside {{. . .}} will replace with users data



Template.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Email confirmation</title>
</head>
<body>
    <h1>Welcome to Purwadhika, Hello {{email}}</h1>
  </body>
</html>
```

Send Email Template



```
const express = require('express');
const app = express();
const transporter = require('./Helpers/Transporter');
const fs = require('fs');
const handlebars = require('handlebars');

app.post('/register', async (req, res) => {
  try {
    const data = await fs.readFile('./Template.html', 'utf-8');
    const tempCompile = await handlebars.compile(data);
    const tempResult = tempCompile({ email: 'pwd@gmail.com' });

    await transporter.sendMail({
      from: 'sender address', //sender address
      to: 'someone@mail.com', //destination address
      subject: 'Activate account',
      html: tempResult,
    });
    res.send('Success');
  } catch (error) {
    console.log(error);
  }
});
```

Thank You!