UDACITY

‹ Return to Classroom

Image Processing API

REVIEW

CODE REVIEW 9

HISTORY

▾ src/file.ts 4

```
import { promises as fs } from 'fs';
import path from 'path';
import imageProcess from './processing';


interface ImageQuery {
```

AWESOME

Great use of interface to define a custom data type.

```
    filename?: string;
    width?: string;
    height?: string;
}

export default class control {
    // Default paths
    static imagesFullPath = path.resolve(__dirname, '../images/full');
    static imagesThumbPath = path.resolve(__dirname, '../images/thumb');

    //image paths
    static async getImagePath(params: ImageQuery): Promise<null | string> {
```

AWESOME

Great job defining a return type for functions by leveraging typescript features.

```
    if (!params.filename) {
      return null;
    }
    const filePath: string =
      params.width && params.height
        ? path.resolve(
          control.imagesThumbPath,
            `${params.filename}-${params.width}x${params.height}.jpg`
          )
        : path.resolve(control.imagesFullPath, `${params.filename}.jpg`);
    try {
      await fs.access(filePath); //check if file exists
      return filePath;
    } catch {
      return null;
    }
  }
  //image availability check
  static async isImageAvailable(filename = ''): Promise<boolean> {
    if (!filename) {
      return false;
    }

    return (await control.getAvailableImageNames()).includes(filename);
  }
  //Get  Images available
  static async getAvailableImageNames(): Promise<string[]> {
    try {
      return (await fs.readdir(control.imagesFullPath)).map(
        (filename: string): string => filename.split('.')[0]
      );
    } catch {
```

AWESOME

Using try/catch blocks goes a long way with error handling. It helps to intercept errors as and when they a

```
      return [];
    }
  }

  //check if there was a thumb created for image
  static async isThumbAvailable(params: ImageQuery): Promise<boolean> {
    if (!params.filename || !params.width || !params.height) {
      return false;
    }
    const filePath: string = path.resolve(
      control.imagesThumbPath,
      `${params.filename}-${params.width}x${params.height}.jpg`
    );

    try {
      await fs.access(filePath);
      return true;
    } catch {
      return false;
    }
  }

  static async createThumbPath(): Promise<void> {
    try {
      await fs.access(control.imagesThumbPath);
    } catch {
      fs.mkdir(control.imagesThumbPath);
    }
  }
  //create a thumb file
  static async createThumb(params: ImageQuery): Promise<null | string> {
    if (!params.filename || !params.width || !params.height) {
      return null;
    }

    const filePathFull: string = path.resolve(
      control.imagesFullPath,
      `${params.filename}.jpg`
    );
    const filePathThumb: string = path.resolve(
      control.imagesThumbPath,
      `${params.filename}-${params.width}x${params.height}.jpg`
    );

    console.log(`Creating thumb ${filePathThumb}`);
```

SUGGESTION

Remove console statements after debugging as a good coding practice.

```

  // Resize & store as thumb
    return await imageProcess({
      source: filePathFull,
      target: filePathThumb,
      width: parseInt(params.width),
      height: parseInt(params.height)
    });
  }
}

```

▸ src/processing.ts 2

▸ README.md 1

▸ src/routes/api/images.ts 1

▸ src/routes/index.ts 1

▶ **dist/file.js**

▶ **dist/index.js**

▶ **dist/processing.js**

▶ **dist/routes/api/images.js**

▶ **dist/routes/index.js**

▶ **dist/tests/fileSpec.js**

▶ **dist/tests/helpers/reporter.js**

▶ **dist/tests/indexSpec.js**

▶ **dist/tests/routes/api/imagesSpec.js**

▶ **dist/tests/routes/indexSpec.js**

▶ **src/index.ts**

▶ **src/tests/fileSpec.ts**

▶ **src/tests/helpers/reporter.ts**

▶ **src/tests/indexSpec.ts**

▶ **src/tests/routes/api/imagesSpec.ts**

▶ **src/tests/routes/indexSpec.ts**

RETURN TO PATH