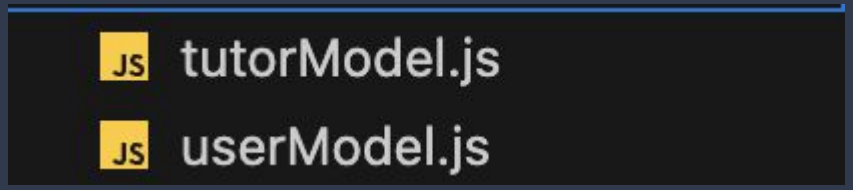
# Bright Academics – Dashboard


T3A2 - Full Stack Web Application

# Who is Bright Academics and why do they need this application?

- Bright Academics is a tutoring business who offer 1:1 sessions to high school students sitting their HSC
- They are a relatively new business and are looking to conduct expansion in the number of tutors as well as the number of clients they have.
- As such, the owner has approached us to create a dashboard for them, where new clients can view tutors and search for them via the corresponding subject.
- This dashboard will also double up on the admin side to act as an employee management tool for the owner and designated employees.

# Backend - Models

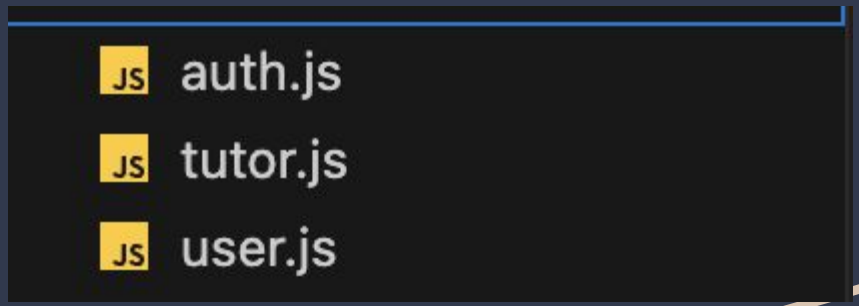



```
const userSchema = new Schema(
  {
    firstname: {
      type: String,
      required: true,
    },
    lastname: {
      type: String,
      required: true,
    },
    email: { type: mongoose.SchemaTypes.Email, required: true },
    mobileNr: {
      type: String,
      match: /^(\()?\d{3}(\))?(-|\s)?\d{3}(-|\s)\d{4}$/,
    },
    password: { type: String },
    isAdmin: { type: Boolean, required: false },
  },
  { timestamps: true }
);
```

```
  name: {
    type: String,
    required: true,
  },
  subjects: {
    type: [String],
    required: true,
  },
  rate: {
    type: Number,
    required: true,
  },
},
{ timestamps: true }
```

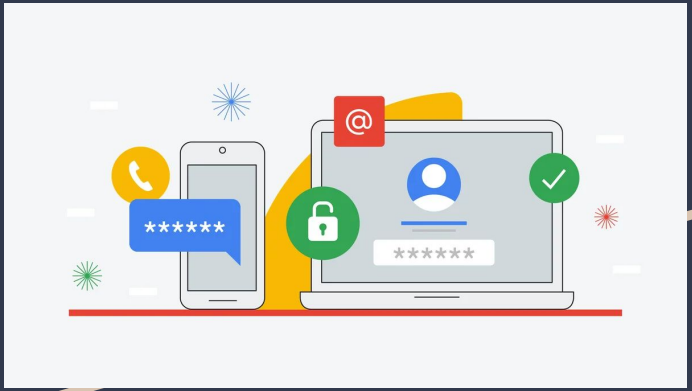
- There are two models which make up the backend of this application, these include;
  - Tutor model, which includes:
    - Name : String
    - Subjects: String, Required
    - Rate: Number, Required
  - User model
    - Firstname: String, Required
    - Lastname: String, Required
    - Email: String, Required,
    - Password: String, Required
    - isAdmin: Boolean

# Backend – Routes




- There are a total of three different route types, each with their own designated CRUD routes.
- **These include;**
  - **auth.js**
    - Handles routes to assist users to log in, log out, reset password and recover account.
  - **tutor.js**
    - Handles routes which manipulate data relating to the tutor model
  - **user.js**
    - Handles routes which manipulate data relating to the user model

# R14 - Account Recovery




- This feature is located with the the auth.js file. It was a requisite of the owner of Bright Academics to have a feature which allowed both clients and admins to recover their account.
- This feature works by utilising nodemailer which is a package which allows emails to be sent from route handlers within express programs.
- Works by sending an email to the one entered within the body of the request with a link if they are a user. This link will be one which is linked to the resetPassword route which will allow the user to send a request with a brand new password.
- The old password in the database will be rewritten to the new one and user will have
- **We will now complete a  walkthrough of this feature**

```
router.post("/recoverAccount", async (req, res) => {
  try {
    const emailHome = req.body.email;

    // Check if email is provided
    if (!emailHome) {
      return res.status(400).json({ message: "Email is required" });
    }

    // Check if user exists
    const user = await User.findOne({ email: emailHome });
    if (!user) {
      return res.status(401).json({ message: "User not found" });
    }

    // Generate a unique reset token
    const resetToken = jwt.sign({ _id: user._id }, process.env.JWT_SECRET_KEY, {
      expiresIn: "10m",
    });
```

```
// Send email with reset link
const transporter = nodemailer.createTransport({
  service: "gmail",
  auth: {
    user: process.env.EMAIL_USER,
    pass: process.env.EMAIL_PASSWORD,
  },
});

const mailOptions = {
  from: "noreply@brightacademics.com",
  to: emailHome,
  subject: "Password Reset",
  text: `You are receiving this email because you requested a password reset for your Br
        Please click on the following link to reset your password: \n\n
        http://${process.env.FRONTEND_URL}/recoverAccount/${resetToken}\n\n
        If you did not request this password reset, please ignore this email and your pa
};

await transporter.sendMail(mailOptions);
```

```
    res.json({
      status: "success",
      message: "Recovery email sent successfully",
    });
  } catch (err) {
    return res.status(500).json({ error: err.message });
  }
});
```

# R15 - Fetching from the Frontend

**Problem?**

- **Writing a fetch request for every component that needs one is not very DRY**
- **Need to add error and loading state to all those components**

**Solution**

- **Custom fetching hooks that handle error and loading state**
- **Makes adding fetch requests very easy**
- **Less duplicate code, easier to change fetching for the entire app (e.g. adding JTW fetches)**

# R15 - Fetching from the Frontend

## Custom Hooks

- useFetchData
  - used to fetch from a useEffect
  - Used for GET requests
  - Returns { data, loading, error }

- useFetchFunc
  - Used for fetches called from user input
  - Used for PATCH, DELETE and POST requests
  - Returns a function
  - Returns { fetchData, loading, error }

# R15 - useFetchFunc

- Takes a path that later gets added on to the end of the hard coded backend url
- State for loading and error

```
import { useState } from "react";

export function useFetchFunc(path, method, token = false) {
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  const backend_url = "http://127.0.0.1:8000";
```

```
const fetchData = async (json) => {
  setLoading(true);
  setError(null);
  try {
    const response = await fetch(backend_url + path, {
      method: method,
      headers: {
        "Content-Type": "application/json",
        ...(token && { "Authorization": `Bearer ${token}` }),
      },
      ...(json && { body: JSON.stringify(json) }),
    });
    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.error || response.statusText);
    }
    const result = await response.json();
    setError(null);
    setLoading(false);
    return result;
```

# R15 - useFetchFunc

```
    } catch (error) {
      setError(error);
    }
    setLoading(false);
  };

  return { fetchData, loading, error };
}
```