house-rental-app/

├── public/

│ └── uploads/ # Uploaded house images (auto-created by Multer)

│

├── views/

│ ├── login.ejs # Login form

│ ├── dashboard.ejs # Admin dashboard (show all properties)

│ ├── add-property.ejs # Form to add new property

│ ├── confirm-delete.ejs # Prompt for re-login before deletion

│

├── routes/

│ └── admin.js # Routes for login, add/delete/view properties

│

├── models/

│ └── Property.js # Mongoose schema for property

│ └── Admin.js # Admin credentials schema

│

├── controllers/

│ └── adminController.js # Logic for each route (auth, add, delete)

│

├── .env # Store DB URI and admin credentials

├── .gitignore

├── app.js # Main server file

├── package.json

npm init -y

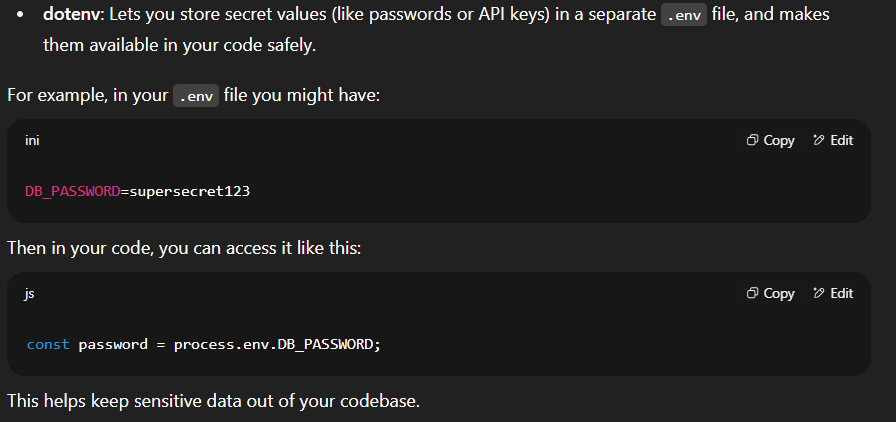
npm install express mongoose dotenv bcryptjs body-parser

 **express**: Web framework for Node.js to build APIs and web servers easily.

 **mongoose**: ODM (Object Data Modeling) library to interact with MongoDB using schemas.

 **bcryptjs**: Library to hash and compare passwords securely in JavaScript.

*  **body-parser**: Middleware to parse incoming request bodies (JSON, URL-encoded, etc.).



app.use(bodyParser.urlencoded({ extended: true }));

👈 **It tells your Express app to parse incoming form data (from HTML forms) and make it available in req.body.**

**Explanation:**

* bodyParser.urlencoded(...): Parses URL-encoded data (like from <form> submissions).
* { extended: true }: Allows nested objects (e.g., user[name]=John becomes { user: { name: 'John' } }).

**Example:**

If a form sends:

name=Alice&age=25

Then in your route, you can access it like:

app.post('/submit', (req, res) => {

console.log(req.body); // { name: 'Alice', age: '25' }

});

So this line makes it easier to handle form data in your backend.

This line:

app.use(express.static(path.join(\_\_dirname, 'public')));

👉 **It serves static files (like HTML, CSS, JS, images) from the public folder.**

**Explanation:**

* express.static(...): Tells Express to look in a folder for static files.
* path.join(\_\_dirname, 'public'): Makes sure the path to the public folder works on any system.
* \_\_dirname: Refers to the current directory of your file.

**Example:**

If you have public/style.css, it will be available at:

http://localhost:3000/style.css

So you can keep your frontend files (like HTML, CSS, JS) in public/ and let Express serve them directly.

These two lines do the following:

const authRoutes = require('./routes/auth');

app.use('/', authRoutes);

**Simple Explanation:**

👉 **They load your auth route file and use its routes for the base URL (/).**

**Breakdown:**

* require('./routes/auth'): Imports the routes you defined in routes/auth.js.
* app.use('/', authRoutes): Tells Express to use those routes for any path that starts with /.

**Example:**

If auth.js contains:

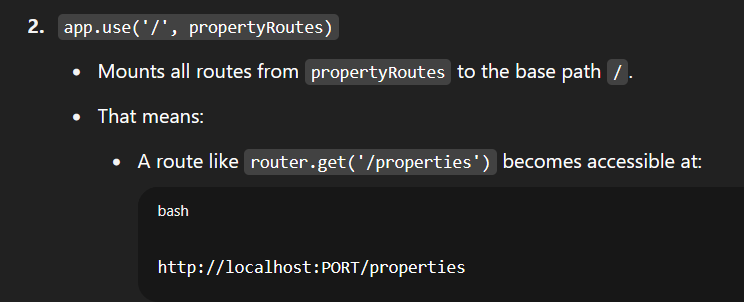
router.post('/login', ...);

Then your app will respond to:

POST /login

This is how you organize and separate route logic into different files for cleaner code.

npm install multer



 .then(() => {

    return bcrypt.hash("4321", 10);

  })

  .then(hashedPassword => {

    return Admin.create({ username: "santhoshi", password: hashedPassword });

  })

Seee here the first then() returns a promise(a value) that will be used by 2nd then() -> .then() is used to synchronize the operation

| **Path** | **Meaning** |
| --- | --- |
| './file' | Same folder |
| '../file' | Go up one folder |
| '../../file' | Go up two folders |
| 'models/property' | (Works only if you're in root or using a bundler with aliasing) |

Soo to connect Compass to atlas

Just replace in .env the connection string got from atlas.(in atlas create a collection to get the string)

But still only the framework gets reflected, the data that are inserted after the connection is only reflected in atlas.

So export the already available data in compass to atlas

1. go to compass and to the particular connection (Export -> export full coneection) and save it anywhere ( do for all connection)
2. then install MongoDB Tool(<https://www.mongodb.com/try/download/database-tools>)
3. Add it to environmental variable
4. C:\Users\santh>mongoimport --uri "mongodb+srv://Santho2917:Rishabh%231729@demo.bnrdj5i.mongodb.net/houseRental" --collection=properties --file=C:\Users\santh\OneDrive\Desktop\houseRentalDB.properties.json –jsonArray
5. C:\Users\santh>mongoimport --uri "mongodb+srv://Santho2917:Rishabh%231729@demo.bnrdj5i.mongodb.net/houseRental" --collection=admins --file=C:\Users\santh\OneDrive\Desktop\houseRentalDB.admins.json --jsonArray

.env variables shld always be capital

The difference between **PATCH** and **PUT** methods in HTTP mainly lies in **how they update resources** on a server.

**🔄 PUT (Full Update)**

* **Definition**: Replaces the entire resource with the new data.
* **Use case**: When you want to **completely update** or **replace** a resource.
* **Idempotent**: ✅ Yes (Calling it multiple times results in the same state).
* **Data required**: Requires the **full representation** of the resource, even unchanged fields.

**🔧 Example:**

If you're updating a user profile:

PUT /users/123

{

"name": "Alice",

"email": "alice@example.com",

"age": 25

}

If you omit age, it may be removed or reset, depending on the API.

**✂️ PATCH (Partial Update)**

* **Definition**: Updates only **specific fields** in the resource.
* **Use case**: When you want to **modify just part** of the resource.
* **Idempotent**: ✅ Often treated as idempotent, but technically it's not guaranteed.
* **Data required**: Only the **fields to be updated**.

**🔧 Example:**

If you're only updating the email:

PATCH /users/123

{

"email": "new-email@example.com"

}

Only the email is changed; other fields remain untouched.

**⚖️ Summary:**

| **Feature** | **PUT** | **PATCH** |
| --- | --- | --- |
| Type of update | Full (replaces entire resource) | Partial (modifies selected fields) |
| Idempotent | Yes | Often (but not strictly) |
| Payload | Entire resource | Only changed fields |
| Risk | May overwrite data if not careful | Safer for small changes |

Let me know if you want examples using code (like with Python's requests or JavaScript's fetch).