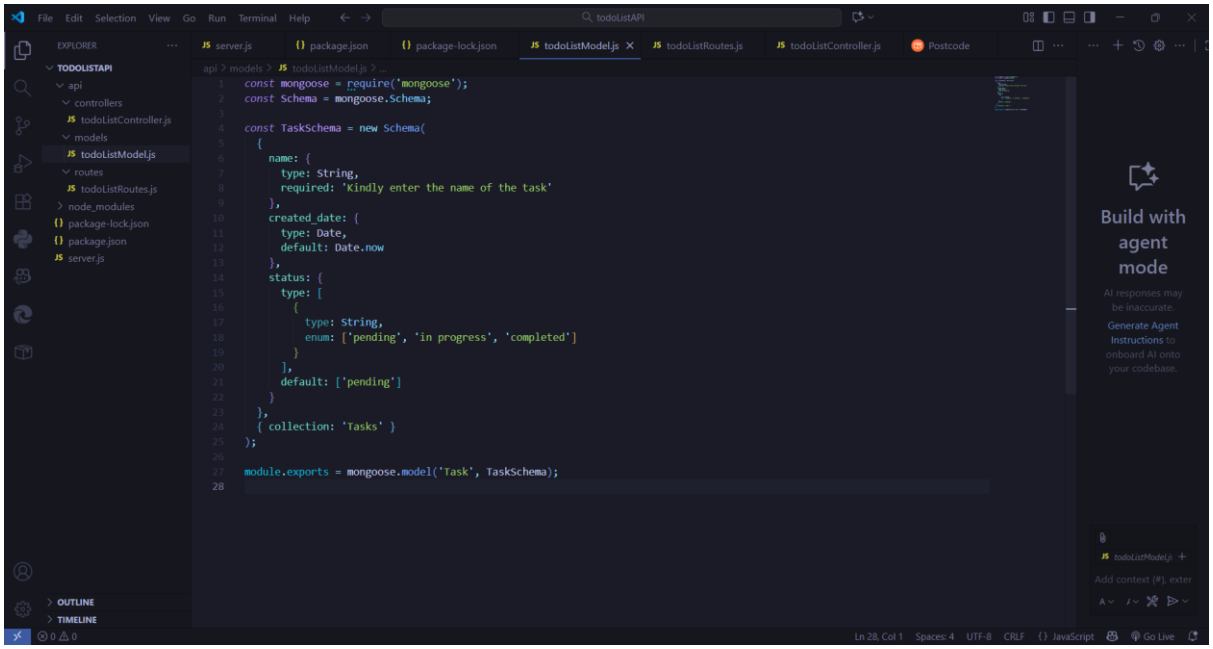
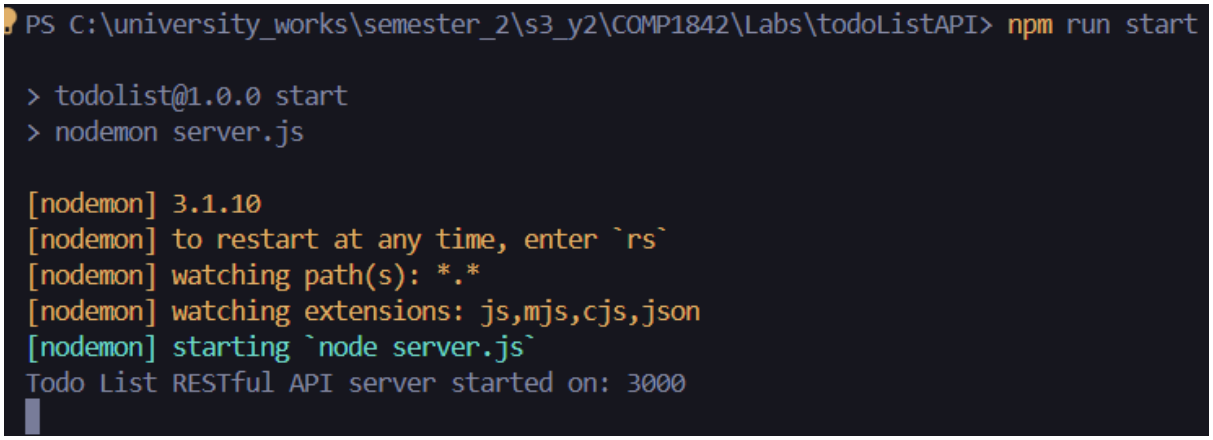
# Part I – Evidence

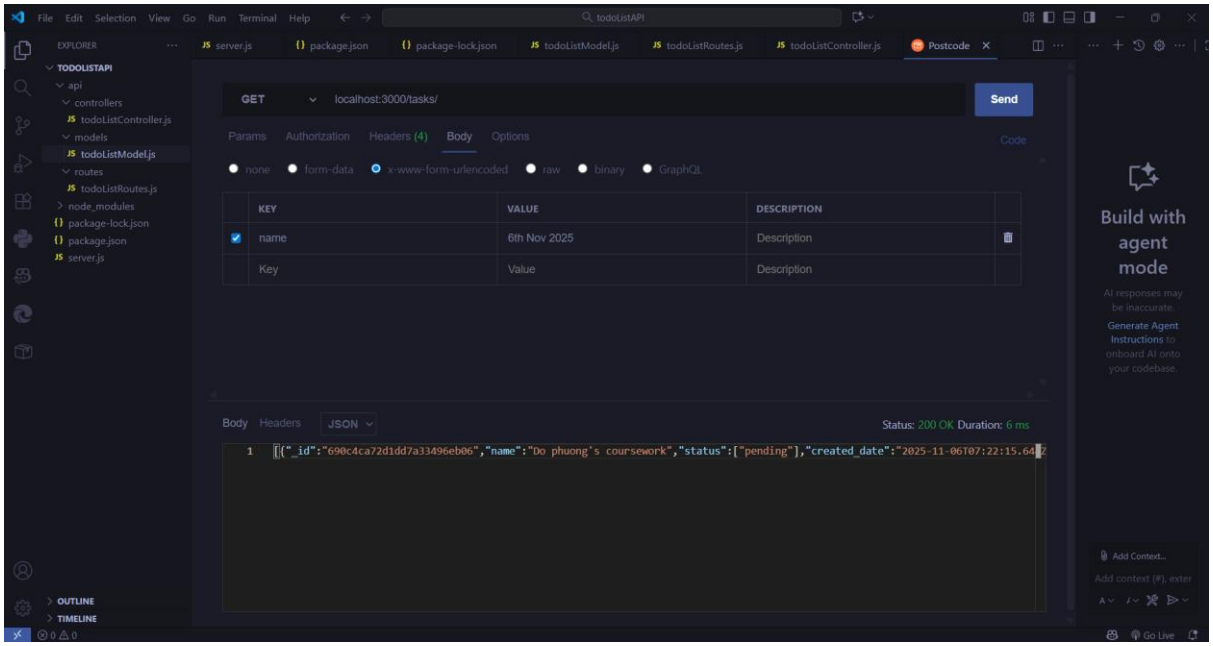
Screenshot 1 – Evidence of source code



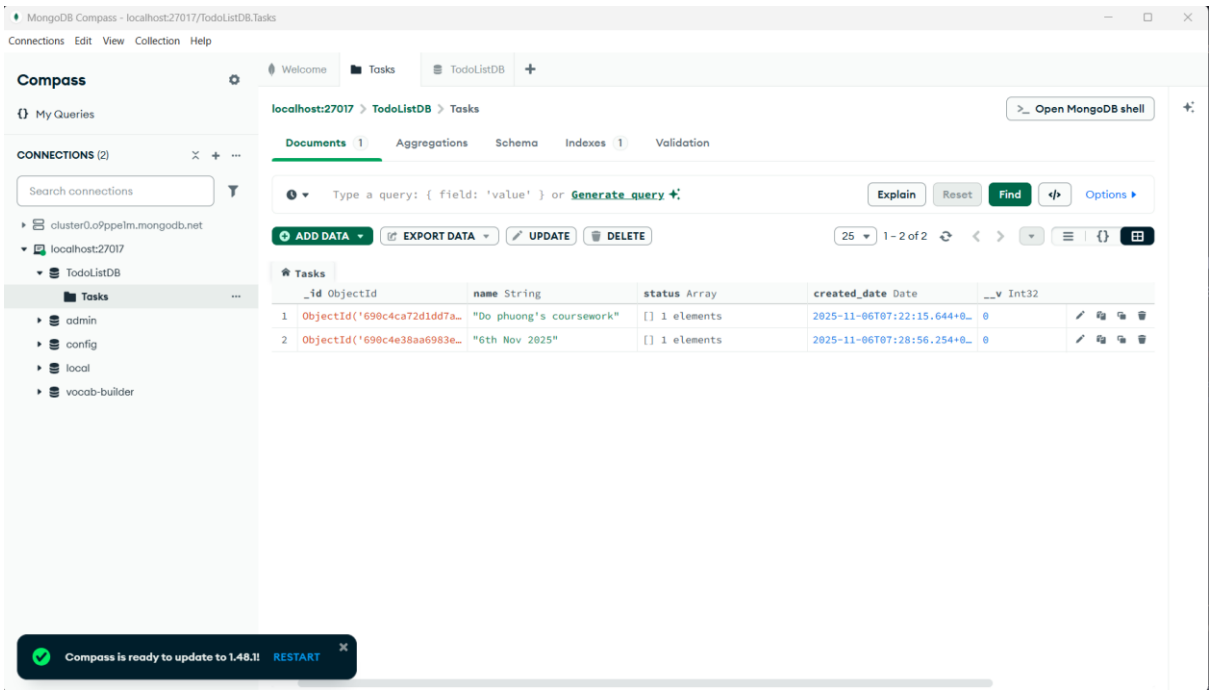

Screenshot 2 – Evidence of server running on with Nodemon on Port 3000




Screenshot 3 – Evidence of API connected in Postcode showing some data from my database




Screenshot 4 – Evidence of data in my MongoDB Compass




# Part II – Understanding

Through this lesson, I learned how RESTful APIs serve as a structured way for applications to communicate between client and server, building directly on the Node.js server work from Lecture 7. While the previous lecture introduced how to create a simple server that responds to requests, this session expanded on the concept to develop a complete RESTful API that connects with a MongoDB database through Express and Mongoose.

I now understand that REST, or Representational State Transfer, is an architectural style that defines how data resources are accessed and manipulated over HTTP using specific verbs, such as GET, POST, PUT, and DELETE, which correspond to CRUD (Create, Read, Update, Delete) operations. Each endpoint represents a resource, and data is typically returned in JSON format for flexibility and readability.

By setting up models, controllers, and routes, the API follows a clear structure that separates logic and improves maintainability. Testing the API with Postcode enabled me to observe how different HTTP methods interact with the server and database in real-time. At the same time, MongoDB Compass provided me with visual confirmation that the data was being created correctly in the database.

This lesson strengthened my understanding of HTTP communication, status codes, and the importance of stateless design. Compared to the simple Node.js server from Lecture 7, the RESTful API provides a more dynamic and reusable way to manage data, laying the groundwork for full-stack integration for the future.