🔐 Security in NoSQL (MongoDB) & Project Structure

🔹 1. Introduction to NoSQL

NoSQL databases like **MongoDB**, **Cassandra**, and **CouchDB** are known for:

- Flexible schemas
- Easy horizontal scaling
- High performance for large and dynamic datasets

Unlike traditional relational databases (SQL), NoSQL is optimized for **speed, scalability, and schema flexibility**, making it ideal for modern web applications.

🔹 2. Security in NoSQL

While powerful, many NoSQL databases **don't enforce strict security by default**. For example, early versions of MongoDB left ports open, exposing sensitive data.

Modern MongoDB has improved with:

- **Authentication & Authorization**
- **Transport encryption** using TLS/SSL
- **Role-Based Access Control (RBAC)**
- **IP Whitelisting**
- **Audit logging**

🔹 3. Common Attack Vectors

Here are some key threats to MongoDB systems:

🔥 **Attack Type**	💥 **Description**
Injection Attacks	Unsanitized input can be used to manipulate queries
Unauthorized Access	Poor permissions or no password setup
Denial of Service (DoS)	Flooding the DB with expensive queries
Data Leakage	Lack of encryption in transit or at rest

🔹 4. Solutions Proposed in Literature

Several researchers and MongoDB's official team have proposed solutions:

- **Santosh et al. (2020)**:
  - Enforce role-based access
  - Use encrypted authentication tokens
- **Chen & Zhao (2021)**:
  - Introduce secure API gateways
  - Use automated vulnerability scanning tools
- **MongoDB Official Docs**:
  - Enable **TLS/SSL encryption**
  - Use **IP whitelisting**
  - Turn on **audit logging**
  - Avoid running with default ports & credentials

🔹 5. Open Challenges

Despite advances, securing NoSQL systems comes with challenges:

- ⚡ Performance tradeoffs: encryption vs speed
- 🌐 No uniform standards across DB platforms
- ☁️ Cloud configurations can be complex and error-prone
- 🔌 Third-party plugins often introduce risk

🛠️ Your Proposal

> Develop lightweight middleware or plug-ins tailored to **small-to-medium dev teams**:

- **Auto-sanitize user input** against injection
- **Monitor suspicious usage patterns**
- **Warn developers of insecure configurations**

This can serve as a security-first bridge for startups and early-stage devs using NoSQL.

📁 Suggested Project Structure

```
inventory-app/
│
├── models/                 # MongoDB schemas (if using ODM like PyMODM or MongoEngine)
│   └── Product.py
├── routes/                 # Flask route definitions
│   └── product_routes.py
├── controllers/            # Business logic (optionally split from routes)
│   └── product_controller.py
│
├── app.py                  # Main Flask app
├── .env                    # Environment variables
├── postman_collection.json
└── README.md               # Documentation
```