1. Folder Structure

bash
CopyEdit
```
land-verification-backend/
|── src/
|   ├── config/            # Configuration files (DB, environment
variables)
|   ├── controllers/       # Handles API request logic
|   ├── middlewares/       # Authentication, authorization,
validation
|   ├── models/            # Database models using Knex.js or
Sequelize
|   ├── routes/            # API routes for different resources
|   ├── services/          # Business logic services
|   ├── utils/             # Helper functions
|   ├── index.ts           # Entry point
|── tests/                 # Test cases for the API
|── .env                   # Environment variables
|── package.json           # Project dependencies
|── tsconfig.json          # TypeScript configuration
|── knexfile.ts            # Knex configuration (if using Knex)
|── Dockerfile             # Containerization (if needed)
|── README.md              # Documentation
```

---

2. API Endpoints

Resource	Method	Endpoint	Description
Auth	POST	`/api/auth/register`	Register users (buyers, sellers, admins)
Auth	POST	`/api/auth/login`	User login with JWT
Users	GET	`/api/users/:id`	Get user profile
Users	PUT	`/api/users/:id`	Update user profile
Properties	POST	`/api/properties`	List a new property
Properties	GET	`/api/properties`	Get all listed properties

Properties	GET	`/api/properties/:id`	Get property details
Properties	DELETE	`/api/properties/:id`	Remove a property
Verification	POST	`/api/verification/request/:propertyId`	Request land verification
Verification	PUT	`/api/verification/approve/:propertyId`	Approve property after verification
Payments	POST	`/api/payments/verify`	Verify payment for land verification
Messaging	POST	`/api/messages`	Buyer-seller chat messages

---

3. Key Components

(a) Authentication (JWT)

- Uses `bcrypt` for password hashing.
- Uses `jsonwebtoken` for secure authentication.

(b) Database Models (Knex.js / Sequelize)

- **User model** (buyers, sellers, verification team, admin)
- **Property model** (land details, documents, images, location)
- **Verification model** (status, assigned verification agent)
- **Payment model** (transaction records, payment verification)

(c) Middleware

- `authMiddleware.ts`: Protects routes, allows only verified users.
- `errorHandler.ts`: Handles global API errors.
- `validator.ts`: Ensures inputs are properly formatted.

(d) Services Layer

- **PropertyService**: Manages property listing, retrieval, and deletion.
- **VerificationService**: Handles property verification workflow.
- **PaymentService**: Integrates payment gateways like Paystack/Flutterwave.

(e) File Uploads

- Uses `Cloudinary` for storing land documents and images.

(f) Real-time Messaging (WebSocket)

- Enables direct communication between buyers and sellers.

(g) Geolocation API

- Uses Google Maps API for verifying land location accuracy.

---

4. Technologies Used

- **Backend Framework:** Node.js (Express.js)
- **Database:** PostgreSQL/MySQL (Knex.js or Sequelize ORM)
- **Authentication:** JWT & Bcrypt
- **File Uploads:** Cloudinary
- **Payment Gateway:** Paystack/Flutterwave
- **Real-time Features:** WebSocket for chat
- **Environment Management:** Dotenv
- **Validation:** Joi / Express Validator
- **Testing:** Jest / Mocha

---

5. Deployment

- **Docker**: For containerization.
- **CI/CD Pipeline**: GitHub Actions for automated deployment.
- **Cloud Hosting**: AWS, DigitalOcean, or Heroku.
- **Load Balancing**: Nginx for handling multiple requests.

---

6. Next Steps

- Implement user authentication.
- Set up the database and ORM models.
- Build API endpoints for listing and verifying properties.
- Integrate third-party services for geolocation and payments.
- Deploy the backend and test with frontend/mobile.

Would you like a sample `index.ts` file and a basic `routes` implementation? 🚀

To **encrypt uploaded documents** in your land verification platform, follow this approach:

Encryption Strategy

1. **Encrypt files before storing them in Cloudinary or local storage.**
2. **Use AES (Advanced Encryption Standard) with a secret key.**

3. **Decrypt files only when authorized users access them.**
4. **Store encryption metadata (IV, encrypted key) securely.**

Implementation Steps

1. Install Required Packages

sh
CopyEdit
```
npm install crypto fs multer multer-storage-cloudinary dotenv
```