MERN Stack Developer – Technical Assignment
Experience Required: 2+ Years
Submission Deadline: Within 24 Hours of Receiving the Assignment
Submit to: info@easesmith.com
Subject Line: MERN Assignment – [Your Full Name]
Submission Format: Public GitHub repository + Detailed README

Assignment Title:

User Submission Tracker with File Management & Analytics

Purpose of the Assignment:

The goal of this assignment is to evaluate your practical skills in building backend applications using the MERN stack with a particular focus on:

- Designing and connecting interrelated MongoDB schemas.
- Implementing business logic and efficient data querying using aggregation pipelines.
- Creating scalable and modular Node.js code.
- Handling and storing images and PDF files with metadata extraction.
- Delivering a clean, production-like backend system with appropriate structure and documentation.

What You Are Expected to Build:

You are required to build a backend system that allows:

1. **Users** to submit structured information (forms) along with multiple files (PDFs and images).
2. **File handling system** to validate, store, and extract metadata from the uploaded files.
3. **Reporting APIs** to analyze user activities and generate summaries through MongoDB aggregations.

What We Expect from You:

1. Clarity in Schema Design

- Your MongoDB schema should be logically structured with relationships across User, Submission, and FileUpload.
- Design schemas that are **normalized** and **scalable**, using ObjectId references between collections.

2. Efficient Use of MongoDB Aggregation

- You must use **MongoDB aggregation pipelines** to generate analytics such as:
    - Top users by number of submissions.
    - Number of PDF and image files grouped by category.

3. Well-Structured Node.js Code

- Maintain modular folder structure:

```
/controllers

/routes

/models

/services

/middleware

/utils

/config
```

- Avoid writing business logic inside routes. Follow a service-controller-repository pattern.

4. Clean Algorithm Implementation

- Your code should follow separation of concerns, reusable methods, clear naming conventions, and well-structured APIs.
- Apply meaningful logic when linking data across models and during metadata extraction.

5. Proper File Handling

- You must:
    - Accept only .jpg, .png, and .pdf files.
    - Use Multer for upload handling.
    - Extract metadata:

        - **Image files:** width and height using sharp or image-size
        - **PDF files:** page count using pdf-parse or pdfjs-dist
    - Save file data and metadata into the FileUpload schema.

6. Professional Delivery

- The system must be well-documented.
- The code should be readable, modular, and testable.
- The logic should be efficient, not redundant or over-engineered.

Features to Implement

A. User Management

POST /api/users

- Create a new user
- Request: name, email
- Response: saved user data

B. Submissions Management

POST /api/submissions

- Create a new form submission linked to a user
- Accept title, description, category (e.g. "Research", "Application"), and multiple files
- Store reference to user and files
- Extract metadata after upload and save in the FileUpload schema

GET /api/submissions/:id

- Get full details of a submission
- Include:
    - Submission data
    - Associated user info
    - File data with metadata (size, dimensions/page count)

C. Analytics and Reporting (using Aggregation Pipelines)

GET /api/analytics/top-users

- Return top 3 users who made the most submissions
- Output: userId, name, totalSubmissions

GET /api/analytics/files-report

- Return report grouped by submission category and file type
- Example Response:

json

```
{
  "Research": { "pdf": 12, "image": 6 },
  "Application": { "pdf": 8, "image": 9 }
}
```

Schema Definitions (Expectation)

User Schema

js

```
{
  name: String,
  email: String,
  createdAt: Date
}
```

Submission Schema

Js

```
{
  title: String,
  description: String,
  category: String,
  userId: ObjectId, // Reference to User
  files: [ObjectId], // References to FileUpload
  submittedAt: Date
}
```

FileUpload Schema

js

```
{
  submissionId: ObjectId, // Reference to Submission
  fileType: String, // 'image' or 'pdf'
  fileUrl: String,
  fileMeta: {
    size: Number,
    dimensions: { width: Number, height: Number }, // for image
    pageCount: Number // for pdf
  },
  uploadedAt: Date
}
```

Technical Stack Guidelines

- **Node.js + Express**
- **MongoDB + Mongoose**
- **Multer** for file upload
- **pdf-parse / pdfjs-dist** for PDF page count
- **sharp / image-size** for image dimensions
- **Validation:** Joi or express-validator
- Use dotenv for environment config

Optional Enhancements (Bonus)

- Swagger documentation

- Authentication with JWT
- Dockerfile for local setup
- Database seeding script
- Rate limiting middleware
- Basic testing (unit or integration)

Deliverables

1. A public **GitHub repository** containing:
   - Complete codebase
   - .env.example file
   - Postman collection (optional)
   - Swagger docs (optional)
2. A **README.md** file including:
   - How to run the project
   - MongoDB schema structure
   - How file uploads and metadata extraction work
   - Explanation of aggregation logic
   - Assumptions made
   - Time taken and pending tasks (if any)
3. Submit the GitHub repository **public link** to
   info@easesmith.com
   with subject line: **MERN Assignment – [Your Full Name]**

Evaluation Criteria

Criteria	**Weightage**
Schema Design & Relationships	20%
Aggregation Query Quality	20%
Backend Architecture & Modularity	20%

Criteria	**Weightage**
File Upload & Metadata Handling	15%
Algorithm Clarity & Reusability	15%
Validation & Error Handling	10%

If anything is unclear, you may send a clarification email to **info@easesmith.com**. Your ability to reason through the assignment and follow best practices matters more than covering every optional feature.