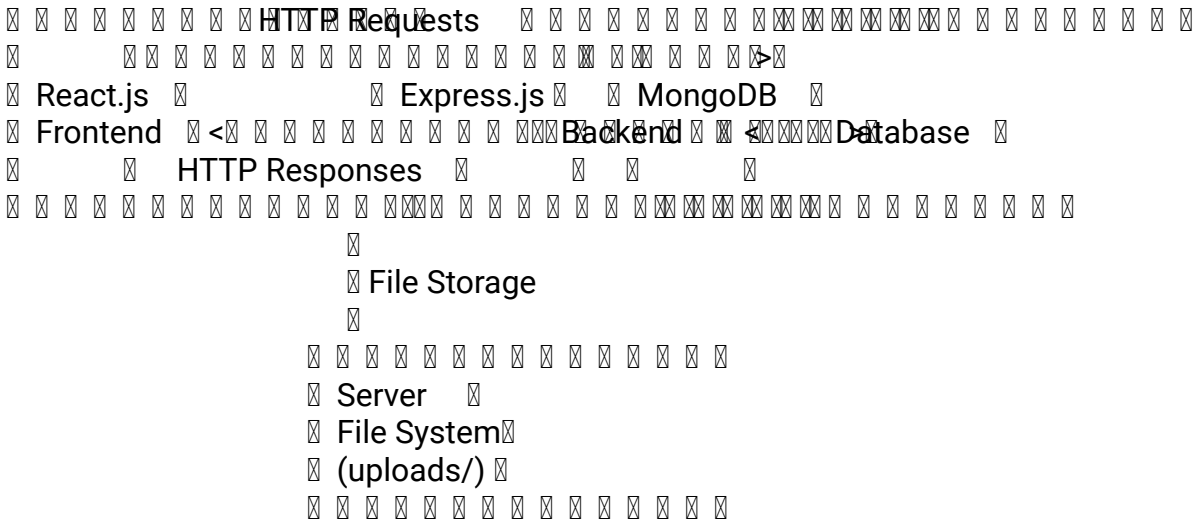
File Sharing Application - Technical Documentation

1. Project Overview

This file sharing application allows users to upload files, generates shareable links, and creates QR codes for easy access. The application is built using the MERN stack (MongoDB, Express.js, React.js, Node.js) and follows a client-server architecture.

2. System Architecture

2.1 High-Level Architecture



Copy

Insert

2.2 Component Breakdown

1. **Frontend (React.js)**

   - User interface for file upload
   - Display of shareable links
   - QR code generation
   - Tabbed interface for sharing options

2. **Backend (Express.js)**

   - RESTful API endpoints
   - File upload handling
   - File metadata storage
   - File serving

3. **Database (MongoDB)**

   - Stores file metadata
   - Tracks download counts
   - Records access timestamps

4. **File Storage (Server File System)**

   - Physical storage of uploaded files
   - Organized in the 'uploads/' directory

3. Data Flow

3.1 File Upload Process

1. User selects a file through the React UI
2. React creates a FormData object and sends it to the server
3. Server receives the file via Multer middleware
4. Server saves the file to the 'uploads/' directory
5. Server extracts metadata (size, type, name) from the file
6. Server generates a unique UUID for the file
7. Server creates a document in MongoDB with file metadata
8. Server returns the shareable link to the client
9. React displays the link and generates a QR code

3.2 File Download Process

1. User accesses the shareable link
2. Server receives the request with the file's UUID
3. Server queries MongoDB for the file metadata
4. Server increments the download count
5. Server updates the 'lastAccessed' timestamp
6. Server locates the physical file in the 'uploads/' directory
7. Server sends the file to the user's browser

4. Key Components

4.1 Server Components

4.1.1 Server Entry Point (index.js)

The main server file initializes the Express application, connects to MongoDB, and sets up routes and middleware.

Key responsibilities:

- Environment configuration
- Database connection
- Middleware setup
- Route configuration
- File serving
- Error handling

4.1.2 File Model (models/File.js)

Defines the schema for file documents in MongoDB.

Fields include:

- filename: The name of the file on the server
- originalName: The original name of the uploaded file
- path: The file path in the server's file system
- size: The file size in bytes
- type: The file type category (document, image, etc.)
- extension: The file extension
- uuid: A unique identifier for accessing the file
- downloadCount: Number of times the file has been downloaded
- uploadDate: When the file was uploaded
- lastAccessed: When the file was last accessed

4.1.3 File Routes (routes/files.js)

Handles API endpoints for file operations.

Endpoints include:

- POST /api/files/upload: Uploads a new file
- GET /api/files/:uuid: Gets file metadata by UUID
- GET /api/files/download/:uuid: Downloads a file by UUID

4.2 Client Components

4.2.1 Main App Component (App.js)

The root React component that sets up routing and renders the main layout.

4.2.2 Upload Page Component

Handles file selection and upload, displays progress, and shows the sharing interface after upload.

4.2.3 File Details Page

Displays file information and sharing options for a specific file.

5. Database Schema

5.1 File Collection ('detail')

```
{
  "_id": ObjectId,
  "filename": String,
  "originalName": String,
  "path": String,
```

```
  "size": Number,
  "type": String,
  "extension": String,
  "uuid": String,
  "downloadCount": Number,
  "uploadDate": Date,
  "lastAccessed": Date,
  "createdAt": Date,
  "updatedAt": Date
}
```

Copy

Insert

6. API Endpoints

6.1 File Upload

- **Endpoint**: POST /api/files/upload
- **Description**: Uploads a file to the server
- **Request**: Multipart form data with 'file' field

- **Response**: JSON with file URL and UUID
- **Example Response**
  :
  ```
  {
    "file": "http://localhost:8000/files/abc123",
    "uuid": "abc123",
    "fileName": "document.pdf",
    "fileSize": 1024,
    "fileType": "document"
  }
  ```

  Copy

  Insert

6.2 Get File Metadata

- **Endpoint**: GET /api/files/:uuid
- **Description**: Gets metadata for a specific file
- **Response**: JSON with file details
- **Example Response**
  :

```
{
  "uuid": "abc123",
  "filename": "document.pdf",
  "size": 1024,
  "type": "document",
  "extension": "pdf",
  "downloadCount": 5,
  "uploadDate": "2023-09-15T12:34:56.789Z",
  "downloadLink": "http://localhost:8000/files/abc123"
}
```


Copy

Insert

6.3 Download File

- **Endpoint**: GET /files/:uuid
- **Description**: Downloads a file by UUID
- **Response**: File download

7. Security Considerations

- File size limits (100MB) to prevent abuse
- File type validation
- No user authentication (public file sharing)
- No file expiration (files remain available indefinitely)

8. Deployment Considerations

- Environment variables for configuration
- Static file serving for the React app in production
- MongoDB Atlas for database hosting
- File storage on the server's file system

9. Future Enhancements

- User authentication
- File expiration
- Password protection for files
- Email notifications
- File preview
- Multiple file upload

10. Conclusion

This file sharing application provides a simple, efficient way to share files with others. The MERN stack architecture allows for scalability and maintainability, while the separation of concerns between client and server ensures a clean codebase.