**WareHouseFileArchive Details**

**Case Study 19: Warehouse File Archiver**

**Roles: Admin, Staff**

**Features:**

**Categorized file uploads**

**Download archives**

**Version control for files**

**API Technical Specification Document**

**1. Overview**

**\* API Type: RESTful API**

**\* Protocol: HTTPS (TLS 1.2+)(optional)**

**\* Content Type: application/json**

**\* Authentication: JWT (Role-Based Authorization)**

**\* Rate Limiting: 1000 requests/hour per user (configurable)**

**\* Versioning: URI-based (/api/v1/)**

**\* Documentation: Swagger/OpenAPI 3.0**

**2. Functional Requirements**

**The API must support the following high-level features:**

**\* User Authentication (Login, Logout, Token Refresh)**

**\* Role-Based Access Control**

**\* CRUD operations on core entities**

**\* File Upload support (e.g., profile picture, attachments)**

**\* Search, Filter, Sort, and Pagination**

**\* Audit Logging of all data changes(Selection based on requirement)**

**\* Optional: Real-time update notifications (SignalR/WebSockets)**

**3. Security Requirements**

**\* JWT Bearer Authentication with Access + Refresh Tokens**

**\* Password hashing using BCrypt**

**\* Authorization based on user roles (e.g., Admin, Editor, Viewer)**

**\* Input validation & sanitization**

**\* HTTPS enforced across environments(Optional)**

**\* CORS configuration for frontend clients**

**\* Role-based policy enforcement in controllers(Based on requirement)**

**4. API Endpoints**

**4.1 Authentication**

**Method Endpoint Description**

**POST /api/v1/auth/login Authenticate user and return token**

**POST /api/v1/auth/refresh Refresh expired access token**

**POST /api/v1/auth/logout Invalidate current token**

**GET /api/v1/auth/me Get current user details**

**4.2 User Management**

**Method Endpoint Description**

**GET /api/v1/users List users (with pagination)**

**GET /api/v1/users/{id} Get user by ID**

**POST /api/v1/users Create new user**

**PUT /api/v1/users/{id} Update user details**

**DELETE /api/v1/users/{id} Delete user**

**4.3 Entity Management (Generic example)**

**Method Endpoint Description**

**GET /api/v1/items List items with filters**

**GET /api/v1/items/{id} Get item by ID**

**POST /api/v1/items Create new item**

**PUT /api/v1/items/{id} Update item**

**DELETE /api/v1/items/{id} Delete item**

**4.4 File Upload**

**Method Endpoint Description**

**POST /api/v1/files/upload Upload a file (multipart/form-data)**

**GET /api/v1/files/{filename} Download a file**

**5. Request & Response Standards**

**\* Request Headers:**

**\* Authorization: Bearer <access\_token>**

**\* Content-Type: application/json**

**\* Standard Response Format:**

**json**

**CopyEdit**

**{**

**"success": true,**

**"message": "Item fetched successfully",**

**"data": { /\* object or array \*/ },**

**"errors": null**

**}**

**\* Error Response Format:**

**json**

**CopyEdit**

**{**

**"success": false,**

**"message": "Validation failed",**

**"data": null,**

**"errors": {**

**"field": ["Error message"]**

**}**

**}**

**6. Pagination Standard**

**\* Supported via query parameters:**

**\* page (default: 1)**

**\* pageSize (default: 10, max: 100)**

**http**

**CopyEdit**

**GET /api/v1/items?page=2&pageSize=25**

**\* Response example:**

**json**

**CopyEdit**

**{**

**"data": [...],**

**"pagination": {**

**"totalRecords": 120,**

**"page": 2,**

**"pageSize": 25,**

**"totalPages": 5**

**}**

**}**

**7. Database Guidelines**

**\* Use GUID or long as primary key**

**\* Soft delete support using IsDeleted flag**

**\* Audit columns: CreatedAt, CreatedBy, UpdatedAt, UpdatedBy**

**8. Logging & Monitoring**

**\* Use structured logging (e.g., Serilog)**

**\* Track all API calls with timestamps, user IDs, endpoint names**

**\* Log errors with stack traces**

**\* Enable Application Performance Monitoring (APM)**

**9. Testing**

**\* Unit Testing for services and controllers (xUnit/NUnit)**

**\* Integration Testing with in-memory DB**

**\* Postman Collection for all endpoints**

**\* Contract testing if third-party services are integrated**

1. **Corporate Document Management**

| **Item (Category)** | **Example Archive Files** | **File Category** |
| --- | --- | --- |
| HR | LeavePolicy.pdf, CodeOfConduct.docx | Policies |
| Finance | Q1Report.xlsx, AnnualBudget.pdf | Reports, Invoice |
| IT | ServerSetupManual.pdf, APIKeys.txt | Manuals, DesignDocs |
| Legal | NDA.docx, VendorAgreement.pdf | Notices, Manuals |

2. **Manufacturing Company**

| **Item (Category)** | **Example Archive Files** | **File Category** |
| --- | --- | --- |
| Machines | LatheManual.pdf, DrillSpecs.xlsx | Manuals |
| QC Reports | BatchInspection.csv, DefectAnalysis.pdf | Reports |
| Safety Procedures | FireDrillProcedure.pdf | SafetyInstructions |

3.  **Government or Legal Archiving**

| **Item (Category)** | **Example Archive Files** | **File Category** |
| --- | --- | --- |
| Council Meetings | MayMeetingMinutes.pdf | MeetingMinutes |
| Budget 2023 | BudgetSummary.pdf, Expenses.xlsx | Invoice |
| Court Orders | Case1234\_FinalOrder.pdf | Notices |

4.  **Healthcare Institution**

| **Item (Category)** | **Example Archive Files** | **File Category** |
| --- | --- | --- |
| Patient ABC | XRayResult.pdf, Prescription1.pdf | Reports, Prescriptions |
| Billing Records | Invoice\_ABC123.pdf | Invoice |
| Clinic Policies | CovidPolicy2023.pdf | Policies |

**5. Software Development Team Repository**

| **Item (Category)** | **Example Archive Files** | **File Category** |
| --- | --- | --- |
| API Documentation | AuthAPI.md, RateLimiting.md | Documentation, Documentation |
| Release v1.0.0 | ReleaseNotes\_v1.pdf, TestCases\_v1.json | ReleaseNotes, TestCases |
| UI Design | Wireframe.png, UX\_Specs.docx | DesignDocs |

**Procedures:**  
  
dotnet ef migrations add addAuthentication --context WareHouseArchiveAuthDbContext

dotnet ef database update --context WareHouseArchiveAuthDbContext

dotnet ef migrations add addWareHouseDb --context WareHouseDbContext

dotnet ef database update --context WareHouseDbContext

**Visit:**  
http://localhost:5239/test.html

**Controller: ItemsController**

**1. Get All Items**

* **HTTP Method**: GET
* **URL**: /api/v1/items
* **Description**: Retrieves all items with pagination support.
* **Authorized Roles**: Admin, Staff
* **Input JSON**: None (uses query params: page, pageSize)
* **Response JSON**:

{

"data": [ { "id": "guid", "name": "string", description”: “string”, "categories": ["string"] } ],

"pagination": {

"totalRecords": 100,

"page": 1,

"pageSize": 10,

"totalPages": 10

}

}

**2. Get Item By ID**

* **HTTP Method**: GET
* **URL**: /api/v1/items/{id}
* **Description**: Get a single item by its ID.
* **Authorized Roles**: Admin, Staff
* **Input JSON**: None
* **Response JSON**:

{

"success": true,

"message": "Item retrieved successfully",

"data": { "id": "guid", "name": "string", description”: “string”, "categories": ["string"] },

"errors": null

}

**3. Create Item**

* **HTTP Method**: POST
* **URL**: /api/v1/items
* **Description**: Create a new item with name and categories.
* **Authorized Roles**: Admin
* **Input JSON**:

{

"name": "string",

“description” : “string”,

"categories": ["category1", "category2"]

}

**Response JSON**:

{

"success": true,

"message": "Item retrieved successfully",

"data": { "id": "guid", "name": "string", “description”: “string” ,"categories": ["string"] },

"errors": null

}

**4. Update Item**

* **HTTP Method**: PUT
* **URL**: /api/v1/items/{id}
* **Description**: Update item name and categories.
* **Authorized Roles**: Admin
* **Input JSON**:

{

"name": "string",

description”: “string”

"categories": ["category1", "category2"]

}

**Response JSON**: Updated item with success.

**5. Delete Item**

* **HTTP Method**: DELETE
* **URL**: /api/v1/items/{id}
* **Description**: Delete an item by ID.
* **Authorized Roles**: Admin
* **Input JSON**: None
* **Response JSON**:

{

"success": true,

"message": "Item deleted successfully",

"data": { "id": "guid", "name": "string" , description”: “string”, “categories": ["category1", "category2"]

},

"errors": null

}

**Controller: FileUploadController**

**1. Upload File**

* **HTTP Method**: POST
* **URL**: /api/v1/files/upload
* **Description**: Uploads a new file to an item and category.
* **Authorized Roles**: Admin
* **Input JSON**: Multipart/form-data

file: (binary)

category: string

itemId: guid

* **Response JSON**:

{

"success": true,

"message": "File uploaded successfully",

"data": {

"id": "guid",

"fileName": "file.pdf",

"fileExtension": ".pdf",

"fileSizeInBytes": 123456,

"filePath": "/files/...pdf",

"versionNumber": 1,

"category": "string",

"itemId": "guid"

},

"errors": null

}

**2. Download File**

* **HTTP Method**: GET
* **URL**: /api/v1/files/{id}/download
* **Description**: Downloads a file by ID.
* **Authorized Roles**: Admin, Staff
* **Input JSON**: None
* **Response JSON**: Binary file stream.

**3. Get All Versions of a File**

* **HTTP Method**: GET
* **URL**: /api/v1/files/{itemId}/versions
* **Description**: Returns all versions of a file for an item.
* **Authorized Roles**: Admin, Staff
* **Input JSON**: None
* **Response JSON**:

{

"data": [

{ "versionNumber": 1, "filePath": "string", "createdAt": "timestamp" },

...

]

}

**Controller: UserController**

**1. Get All Users**

* **HTTP Method**: GET
* **URL**: /api/v1/users
* **Description**: List all registered users with pagination.
* **Authorized Roles**: Admin
* **Input JSON**: None (query params: page, pageSize)
* **Response JSON**:

{

"data": [ { "id": "guid", "username": "string", "roles": ["Admin"] } ],

"pagination": { "totalRecords": 10, "page": 1, "pageSize": 10, "totalPages": 1 }

}

**2. Get User by ID**

* **HTTP Method**: GET
* **URL**: /api/v1/users/{id}
* **Description**: Fetch user details by ID.
* **Authorized Roles**: Admin
* **Input JSON**: None
* **Response JSON**: Standard user DTO.

{

"data": [ { "id": "guid", "username": "string", "roles": ["Admin"] } ]

}

**3. Create User**

* **HTTP Method**: POST
* **URL**: /api/v1/users
* **Description**: Add a new user to the system.
* **Authorized Roles**: Admin
* **Input JSON**:

{

"username": "string",

"email": "string",

"password": "string",

"roles": ["Admin", "Staff"]

}

**4. Update User**

* **HTTP Method**: PUT
* **URL**: /api/v1/users/{id}
* **Description**: Update user details like email, roles.
* **Authorized Roles**: Admin
* **Input JSON**:

{

"email": "string",

"roles": ["Admin"]

}

**5. Delete User**

* **HTTP Method**: DELETE
* **URL**: /api/v1/users/{id}
* **Description**: Delete user by ID.
* **Authorized Roles**: Admin
* **Input JSON**: None

**Controller: AuthController**

**1. Register a New User**

* **HTTP Method**: POST
* **URL**: /api/v1/auth/register
* **Description**: Registers a new user with one or more roles (Admin or Staff).
* **Authorized Roles**: Anonymous
* **Input JSON**:

{

"username": "admin@example.com",

"password": "StrongPassword123!",

"roles": ["Admin", "Staff"]

}

* **Response JSON (Success)**:

{

"success": true,

"message": "User was registered! Please login.",

"data": {},

"errors": null

}

* **Response JSON (Invalid Role)**:

{

"success": false,

"message": "Validation failed",

"data": null,

"errors": {

"Roles": ["Invalid role. Only 'Admin' or 'Staff' are allowed."]

}

}

**2. Login**

* **HTTP Method**: POST
* **URL**: /api/v1/auth/login
* **Description**: Authenticates a user and returns JWT + refresh token.
* **Authorized Roles**: Anonymous
* **Input JSON**:

{

"username": "admin@example.com",

"password": "StrongPassword123!"

}

* **Response JSON (Success)**:

{

"success": true,

"message": "Login successful",

"data": {

"jwtToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",

"refreshToken": "random-refresh-token"

},

"errors": null

}

* **Response JSON (Failure)**:

"Username or Password is incorrect"

**3. Logout**

* **HTTP Method**: POST
* **URL**: /api/v1/auth/logout
* **Description**: Logs out the current user by clearing their refresh token.
* **Authorized Roles**: Admin, Staff
* **Input JSON**: None
* **Response JSON**:

{

"success": true,

"message": "Successfully logged out. Refresh token cleared.",

"data": null,

"errors": null

}

**4. Refresh Token**

* **HTTP Method**: POST
* **URL**: /api/v1/auth/refresh
* **Description**: Reissues a new JWT + refresh token pair if the previous access token has expired and the refresh token is valid.
* **Authorized Roles**: Anonymous
* **Input JSON**:

{

"accessToken": "expired.jwt.token",

"refreshToken": "valid-refresh-token"

}

* **Response JSON (Success)**:

{

"success": true,

"message": "Token refreshed successfully",

"data": {

"jwtToken": "new.jwt.token",

"refreshToken": "new-refresh-token"

},

"errors": null

}

* **Response JSON (Invalid Token)**:

{

"success": false,

"message": "Invalid access token",

"data": null,

"errors": {

"Token": ["Could not extract principal"]

}

}

* **Response JSON (Expired Refresh Token)**:

{

"success": false,

"message": "Invalid or expired refresh token",

"data": null,

"errors": {

"Token": ["Invalid or expired refresh token"]

}

}

**5. Get Current User Info**

* **HTTP Method**: GET
* **URL**: /api/v1/auth/me
* **Description**: Returns currently logged-in user's profile and roles.
* **Authorized Roles**: Admin, Staff
* **Input JSON**: None
* **Response JSON (Success)**:

{

"success": true,

"message": "User fetched successfully",

"data": {

"id": "guid",

"userName": "admin",

"email": "admin@example.com",

"roles": ["Admin"]

},

"errors": null

}

* **Response JSON (Failure)**:

{

"success": false,

"message": "User not found",

"data": null,

"errors": {

"User": ["User not found"]

}

}