FileFinder Developer Guide

Running FileFinder

Please refer to the Installation Manual for detailed instructions on how to configure environment variables, application secrets, and an AWS S3 bucket for getting started.

Debugging

XDebug can be used for debugging the PHP source code. The following environment variables will need to be set:

```
# XDebug modes to enable
XDEBUG_MODES=
# XDebug log level. 0 is lowest
XDEBUG_LOG_LEVEL=
```

XDEBUG_MODES → Controls which XDebug features are enabled
XDEBUG_LOG_LEVEL → Controls how much information is logged

Recommended values for XDEBUG_MODES -

- *debug* - Enables step debugging
- *develop* - Enables development helpers like *var_dump()*
- *coverage* - Enables code coverage analysis, and is required to be able to generate code coverage reports from PHPUnit
- *profile* - Enables profiling, which can be used to analyze performance bottlenecks

Running automated tests

Note: 'coverage' mode is required in the XDebug configuration to be able to get a coverage report from the automated tests run by PHPUnit

The *phpunit.xml* file bundled with the FileFinder source code defines a test-suite that runs all automated tests written for the FileFinder application.

Running the tests from within the api container -

```
phpunit --configuration ./phpunit.xml --testsuite filefinder
--coverage-text'
```

Running the tests from an external shell -

```
docker exec <container_name> /bin/sh -c './vendor/bin/phpunit
--configuration ./phpunit.xml --testsuite filefinder
--coverage-text'
```

Replace *<container_name>* in the command with the name of the docker container where the unified backend is running.

Database Design

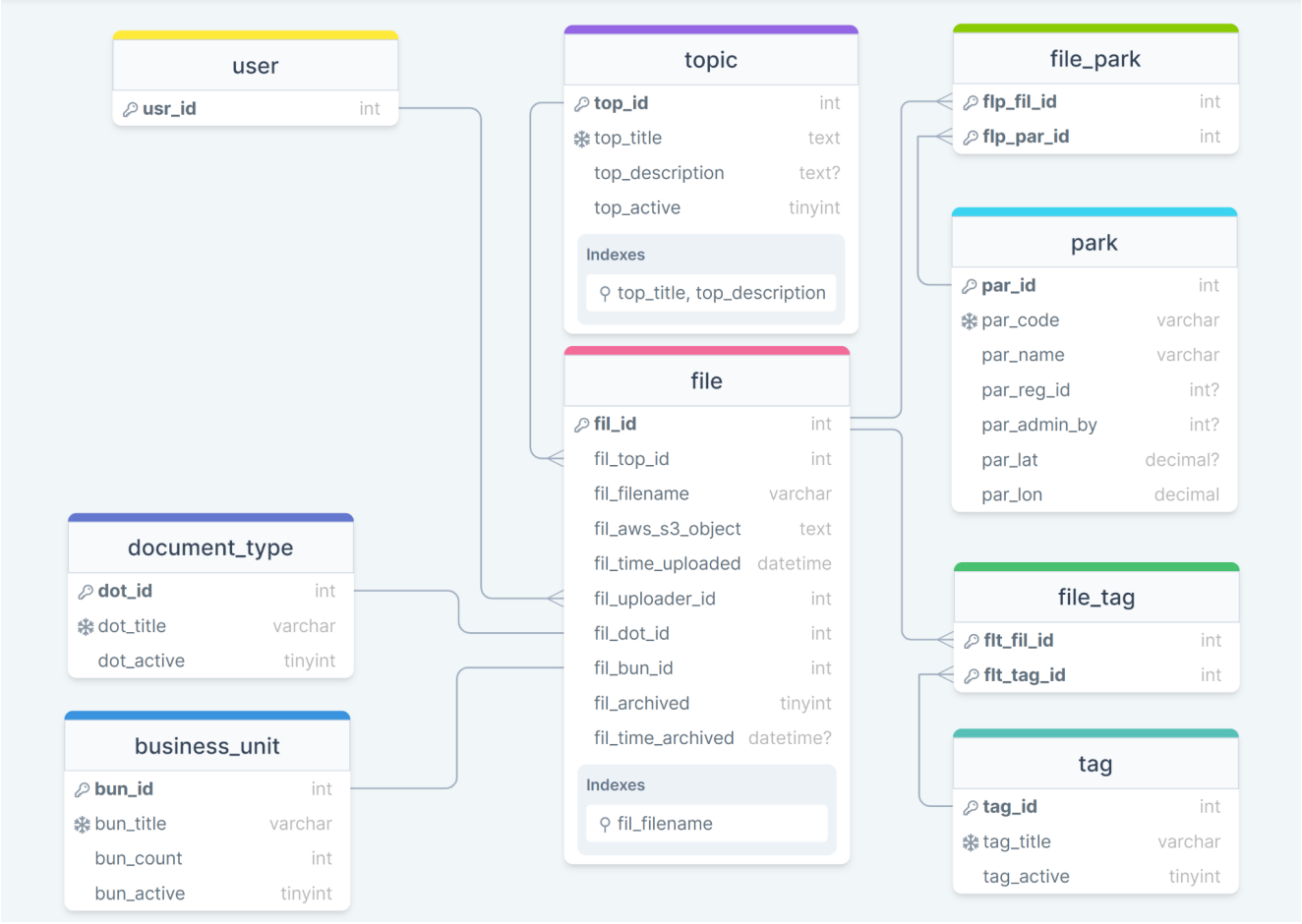


Figure 1

The core functionality of FileFinder is supported by the database schema shown in *Figure 1*. Each table has an *active* field that denotes the usage of that database entry in the FileFinder application.

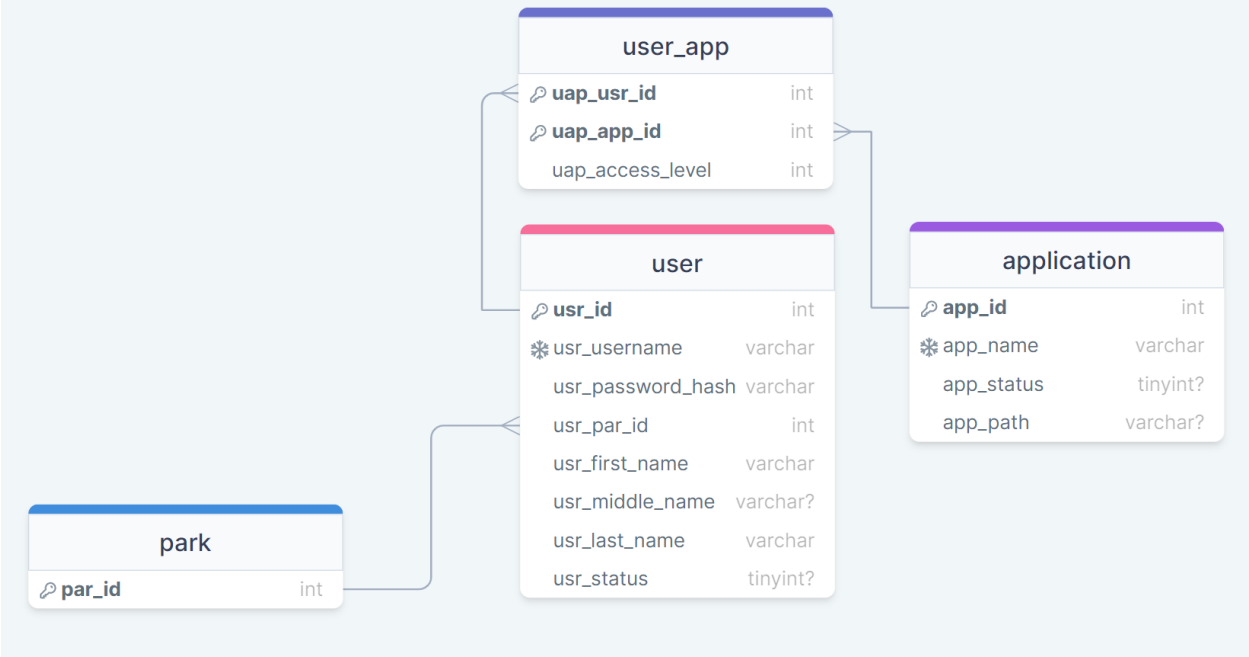
When an entry in the tables for the Business Unit, Document Type and Tag attributes is set as inactive, those attributes do not show up on the frontend as selectable options when a file is being uploaded. The options do show up as selectable choices when searches are being performed. Thus, no new uploads can be associated with inactive attributes, but files already uploaded with the inactive attributes are still searchable.

The table below describes the usage of each database table.

Table Name	Functionality	Foreign References
file	• Stores metadata (through data fields and foreign references) for a single file uploaded to FileFinder • *fil_aws_s3_object* is the AWS S3 object name for the actual contents of the file that are uploaded to an AWS S3 bucket	• ID for the topic under which the file is uploaded • ID for the user that uploaded the file • Business Unit ID • Document Type ID
topic	• Stores topic title and description	
business_unit	• Stores Business Unit title • *count* field is unused → It was meant to be used for the *Most Accessed Categories* functionality defined in the stretch goals, but the functionality was not implemented	
document_type	• Stores Document Type title	
tag	• Stores Tag Title	
park	• Stores all park metadata • *par_admin_by* is a self-reference to the park table to denote 'parent' Parks in the management hierarchy	• Self-reference for indicating 'parent' park

file_park	• Join table used for many-to-many relationships between the *file* table and the *park* table	• Primary key is a composition of File ID and Park ID
file_tag	• Join table used for many-to-many relationships between the *file* table and the *tag* table	• Primary key is a composition of File ID and Tag ID
user	• Stores user metadata along with the username and the hashed password • This is a separate user table that is independently used by the FileFinder application *NOTE: Check section on Users & Access Levels below*	

Database Design - Users & Access Levels



Two tables - the *application* table, and the *user_app* table were created to support user access levels within FileFinder. The *application* table is used to maintain the names and usage status of each application within the new unified backend. The *user_app* table is

used as a many-to-many join table between *user* and *application*, and stores the access level for the user in the form of an integer, with the convention being that a higher number indicates more permissions available to the user.

AWS S3 Usage Pattern

- AWS S3 object naming - The ramsey/uuid library is used to generate a v4 UUID, that is appended to the front of the filename that is being uploaded, delimited by a hyphen
    - Example - If a user is uploading a file named '*instructions.pdf*', the name for the corresponding object in AWS S3 would look like this - '*52e8616a-b3a6-40bb-a29e-e3e7f5f937de-instructions.pdf*'

- File Uploads - Any files being uploaded are sent to the backend from where it is uploaded to the configured S3 bucket.
    - Pre-signed POST URLs were explored as an alternative to sending the files to the backend, but the idea was scrapped due to the problems it would pose with error-handling

- File Downloads - Pre-signed URLs are used for downloading files from the S3 bucket. When a user tries to download a file, a request is sent from the frontend with the S3 object name for the file. The backend uses this name to generate a pre-signed URL that is alive for 120 seconds. This URL is sent back to the frontend, where it is set as the window location, thus downloading the file without the need for any further user input

REST API Documentation

Payload Definitions/Examples

The JSON structures defined below will be referred to in the API documentation table

Park Payload

```
{
    "id": 57,
    "code": "CLNE",
    "name": "Cliffs of the Neuse State Park",
    "region": 12,
    "latitude": 35.2354,
    "longitude": -77.8932,
```

```
    "adminBy": 42
}
```

Business Unit Payload

```
{
    "id": 17,
    "title": "Human Resources",
    "count": 114
}
```

Business Unit Request

```
{
    "title": "Accounting"
}
```

Tag Payload

```
{
    "id": 33,
    "title": "Inventory"
}
```

Tag Request

```
{
    "title": "Equipment"
}
```

Document Type Payload

```
{
    "id": 2,
    "title": "Policies & Directives",
}
```

Document Type Request

```
{
    "title": "Instruction Manuals",
}
```

Topic Payload

```
{
    "id":90,
    "title":"Trails Upgrades Summer 2022",
    "description":"All documents related to planned upgrades during Summer
2022"
}
```

Topic Payload Full

```
{
    "id":90,
    "title":"Trails Upgrades Summer 2022",
    "description":"All documents related to planned upgrades during Summer
2022"
    “files”: [File Payload]
}
```

User Payload

```
{
    “id”: 121,
    “firstName”: “George”,
    “lastName”: “Russell
}
```

File Payload

```
{
    "filename":"clne_budget.pdf",
    "aws_s3_object_name":"DWA218Dawe201HL21-clne_budget.pdf",
    “topic”: Topic Payload,
    “documentType”: Document Type Payload,
    “businessUnit”: Business Unit Payload,
    “tags”: [Tag Payload],
    “parks”: [Park Payload],
    “Uploader”: User Payload,
    “timeUploaded”: “2022-05-18 15:31:09”
}
```

Search Result Payload

```
{
    “files”: [File Payload],
    “topics”: [Topic Payload]
}
```

File Upload Sub-Model

```
{
    “filename”: “contacts.pdf”,
    “fileContent”: binary string,
    “businessUnitId”: 10,
    “documentTypeId”: 11,
    “tags”: [12, 14, 19],
    “parks”: [32, 51, 33]
}
```

File Upload Model Request

```
{
    “topic”: {
```

```
        "title": "New Hire Orientation",
        "description": "All documents required to get new hires up to speed"
    },
    "uploaderId": 120,
    "files": [File Upload Sub-Model]
}
```

Method	Route	Description	Request Body	Response Body	Possible Response Codes
GET	/parks	Get all parks	N/A	Array of Park Payload	200, 500
GET	/businessunits	Get all business units / all business units with a given active status	N/A	Array of Business Unit Payload	200, 500
POST	/businessunits	Add a new business unit to the database	Business Unit Request	Business Unit Payload (newly created resource)	200, 500
PATCH	/businessunits/{id}/count	Increment usage count for given business unit	N/A	Business Unit Payload (updated)	200, 404, 500
PATCH	/businessunits/{id}/active	Switch usage status for a given business unit	N/A	Message containing the updated usage status	200, 404, 500
GET	/tags	Get all tags / all tags with a given active status	N/A	Array of Tag Payload	200, 500
PATCH	/tags/{id}/active	Switch usage status for given tag	N/A	Message containing the updated usage status	200, 404, 500
POST	/tags	Add a new tag to the database	Tag Request	Tag Payload (newly created resource)	200, 409, 500
GET	/documenttypes	Get all document	N/A	Array of Document	200, 500

		types / all document types with a given active status		Type Payload	
POST	/documenttypes	Add a new document type to the database	Document Type Request	Document Type Payload (newly created resource)	200, 409, 500
PATCH	/documenttypes/{id}/status	Switch active status for a given document type	N/A	Document Type Payload (updated)	200, 404, 500
GET	/search	Search for files	N/A	Search Result	200, 500
GET	/files/s3url/{objectName}	Get presigned URL for downloading the specified file	N/A	Array with one element (the presigned URL)	200, 404, 500
POST	/upload	Upload Files	File Upload Model Request	Array of File Payload (newly created resources)	200, 500
PATCH	/files/{id}/archive	Archive / unarchive a specified file	N/A	Message with updated archival status	200, 404, 500
GET	/topic/{id}	Get topic information and all files under the specified topic	N/A	Topic Payload Full	200, 404, 500
PATCH	/topic/{id}/edit	Edit topic title / description	Topic Payload	Topic Payload (updated)	200, 404, 500
DELETE	/files/{id}	Delete file with the given id	N/A	Message indicating success / failure	200, 500