README

Tali Mahar

January 2024

I began by reading the task and summarizing the main design points of the project and the fundamental components (High Level) to outline the system main components.
(I did it first in my notebook with a real pen and later copied to this file ;))

Later I covered the API endpoints by listing the "client" calls that need to be served by the backend, covering the entire File Sharing app operations performed by the users: admin operations, and user action calls.

From there I started designing the code by mapping the microservices to be built.
Each microservice is responsible for a specific area, set of operations, and logic, and gets its dedicated endpoint (*aligned with the API endpoints section*).
I added Audit Logging Service to validate the 10K limit.

From that point, I was ready to "play" with the code.
I built the Python project and environment and configured the Python packages accordingly.
(I am using macOS, iTerm, IntelliJ IDEA 2022.2.2 (Edu))

Later, I described the Technical Specification (one of many approaches..) that can be used as infrastructure for this application and generated a flowchart for the "create new safe-zone" flow using this spec.
(I used claudeAI to create a prompt based on the architecture, and copy it in the eraser ("AI diagram")...)

Once I was finished with the document I completed task #2:
Build a User Authentication and Authorization system, aligning with this *comprehensive* design document.

Thank You,
Tali Mahar

High-level system architecture

File Sharing App

Design Points

Users roles

Admin

- Define policies <> locations
- Define app access permissions
- Do not have access to files (security)

User

- Add, Create, Delete, Edit, and Share files
- Allow to do so only on his permitted dedicated zones (“Groups”)
- Default owner of his own files

Security

- Default visibility - Owner of the file
- Owner can share (Access types: view, comment, edit)

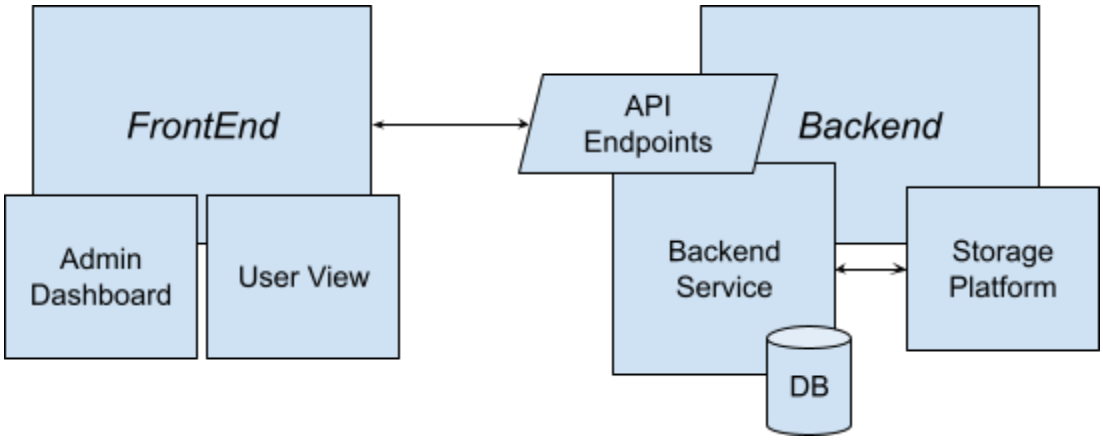
Collaboration

- Internal sharing (co-workers, in the same domain)
- External sharing - Managed by the Admin in a dedicated Safe Zone

Scale

- 10K actions (Add, Create, Delete, Edit, Share) per day

System main components



FrontEnd

Frontend framework split between:
Admin Dashboard where the Admin can manage users, policies, safe zones, and more.
User standard web frot application - file repository view where he can add, create, delete, edit, and share files with others.

BackEnd

The Backend includes:
The actual backend service (the code) is responsible for serving clients' calls (via APIs), authentication and authorization, operations, and logic. The service includes several microservices and works against
The Storage platform (physical or virtual) where the actual files are being stored.
The service stores metadata (files, policies, user credentials, and permissions..) in dedicated DBs.

API Endpoints

User Authentication

POST /api/auth/login

Logs an existing registered user in by validating username and password. On successful login, generates a Token that is returned to identify authenticated future requests from this user.

JSON Request:

```
{
 "username": "user123",
 "password": "secretpass"
}
```

JSON Response:

```
{
 "token": "generated_token"
}
```

POST /api/auth/logout

Logs out the current logged-in user by clearing the token. This invalidates the token on the server to prevent further access to it.

JSON Request:

```
{
 "token": "user_token"
}
```

JSON Response: Success or failure response

User Management

GET /api/users

Returns list of all registered user profiles (usable by admins).

JSON Request: N/A
JSON Response: Array of User entity object

```
{
  "total": 253,
  "page": 1,
  "perPage": 10,
  "users": [
    {
      "id": 123,
      "name": "John Doe",
      "role": "user",
      "email": "john@email.com",
      "group": ["Marketing"],
      "createdAt": "2020-01-02",
      "updateAt": "2020-01-02"
    },
    {
      "id": 124,
      "name": "Jane Smith",
      "role": "admin",
      "email": "jane@email.com",
      "group": ["Sales"],
      "createdAt": "2020-01-02",
      "updateAt": "2020-01-02"
    },
    {
      "id": 125,
      "name": "Bob Williams",
      "role": "user",
```

```
        "email": "bob@email.com",
        "group": ["HR"],
        "createdAt": "2020-01-02",
        "updateAt": "2020-01-02"
      }
   ]
}
```

GET /api/users/{userId}

Returns specific user details based on userID

JSON Request: N/A
JSON Response: User entity object

```
{
"id": 123,
"name": "John Doe",
"role": "user",
"email": "john@email.com",
"group": ["Marketing"],
"createdAt": "2020-01-02",
"updateAt": "2020-01-02"
}
```

POST /api/users/

Create a new user account. Expect user data like name, email, department (group).

JSON Request:

```
{
"name": "John Doe",
"role": "user",
"email": "john@email.com",
"group": ["Marketing"],
"password": "john@email.com",
}
```

JSON Response: User entity object (with ID, without password)

PUT /api/users/{userId}

Update attributes of an existing user account.
(also being used by the Admin for group assignment)

JSON Request:

```
{
"group": ["Marketing", "Sales"],
}
```

JSON Response: User entity object

DELETE /api/users/{userId}

Delete the user account of a specific user

JSON Request: N/A
JSON Response: User entity object

File (/Folder) Management

GET /api/files

Returns folders and files associated with the logged-in user.
This applies also to the Admin user (privacy)

JSON Request: N/A
JSON Response: Array of all files and folders

```
[
 {
  "resource_id": "123e4567-e89b-12d3-a456-426614174001",
  "name": "colors.pdf",
  "is_directory": false,
  "path": "/documents/brandV2",
  "group": "Marketing",
  "type": "PDF",
  "content": "BASE64_ENCODED_FILE_BYTES",
  }
 },
```

```
 {
  "resource_id": "223e4567-e89b-12d3-a456-426614174002",
  "name": "brandV2",
  "is_directory": true,
  "path": "/documents/",
  "group": "Marketing",
  "type": "Folder",
  "content": null,
 },
]
```

GET /api/files/{resourceId}

Returns the file/folder data with its metadata

JSON Request: N/A

JSON Response: Resource (File/Folder) entity object with metadata

```
{
  "resource_id": "123e4567-e89b-12d3-a456-426614174001",
  "name": "colors.pdf",
  "is_directory": false,
  "path": "/documents/brandV2",
  "group": "Marketing",
  "type": "PDF",
  "content": "BASE64_ENCODED_FILE_BYTES",
  "metadata": {
    "createdAt": "2024-01-23T12:34:56Z",
    "updateAt": "2024-01-23T14:45:30Z",
    "openAt": "2024-01-23T14:45:30Z",
    "owner_user_id": "user123",
    "size": 1024, //bytes
    "permissions": {
      "view": ["user123", "user456"],
      "edit": ["user123"],
      "comment": ["user123"]
    }
  }
}
```

POST /api/files/

Create a new file/folder

JSON Request:

```
{
   "name": "colors.pdf",
   "is_directory": false,
   "path": "/documents/brandV2",
   "group": "Marketing",
   "type": "PDF",
   "content": "BASE64_ENCODED_FILE_BYTES"
 }
```

JSON Response: Resource (File/Folder) entity object with metadata

```
{
   "resource_id": "123e4567-e89b-12d3-a456-426614174001",
   "name": "colors.pdf",
   "is_directory": false,
   "path": "/documents/brandV2",
   "group": "Marketing",
   "type": "PDF",
   "content": "BASE64_ENCODED_FILE_BYTES",
   "metadata": {
     "createdAt": "2024-01-23T12:34:56Z",
     "updateAt": "2024-01-23T14:45:30Z",
     "openAt": "2024-01-23T14:45:30Z",
     "owner_user_id": "user123",
     "size": 1024, //bytes
     "permissions": {
       "view": ["user123", "user456"],
       "edit": ["user123"],
       "comment": ["user123"]
     }
   }
 }
```

PUT /api/files/{resourceId}

Update a file parameter.
Also used for content updates (but not for permissions)

JSON Request:

```
{
  "content": "NEW_BASE64_ENCODED_FILE_BYTES",
 }
```

JSON Response: Resource (File/Folder) entity object with metadata

DELETE /api/files/{resourceId}

Delete file/folder (and the files in it)

JSON Request: N/A
JSON Response: Resource (File/Folder) entity object with metadata

Collaboration Management

POST /api/collaboration/share

Share a file/folder with co-workers or with external organizations.

JSON Request:

```
{
 "resource_id": "123e4567-e89b-12d3-a456-426614174001",
 "share_with": ["user456", "external_org@example.com"],
 "access_type": "edit"
}
```

JSON Response: Success or failure response

POST /api/collaboration/safe-zone

Define a new safe zone with the permitted external organizations

JSON Request:

```
{
 "safe_zone": "/shared/marketing",
 "external_domains": ["example1.com", "example2.com"]
}
```

JSON Response: Success or failure response

Microservices

Auth

/api/auth/

Responsible for User authentication and authorization, Issuing and validating tokens to allow access to protected resources (files, folders).

User Management

/api/user/

Managing user accounts and profile data, CRUD operations on user entities, Search/retrieve users
Also being used by the Admin for user groups (“zone”) assignments

File Management

/api/file/

Responsible for handling file and folder operations within the organization.
Focuses on creating, adding, deleting, and editing files and folders in the shared storage.

Collaboration

/api/collaboration/

Facilitates collaboration features, including sharing documents within the organization and with external organizations. Enforces access policies defined by the Admin. Allows Admin to define "safe zones" for cross-organization sharing.

Audit Logging Service

Records user actions for auditing purposes. Helps track the 10k daily actions.

Technical Specification

FrontEnd

1. Web application built using a modern frontend framework (e.g., React, Angular, or Vue.js).
2. Software as a Service (SaaS) refers to a software distribution model where applications are hosted by a third-party provider and made available to customers over the Internet.
3. Communicates with the backend through APIs.
4. 2 modes - User view, Admin panel (allow only for Admin users)
5. Login with OAuth 2 (workers) and local login (external users)
   OAuth 2.0 is a standard protocol for authorization, allowing users to log in using third-party services. Local login likely refers to traditional username/password authentication.

6. Handles user authentication with JWT (JSON Web Token) tokens

BackEnd

1. The backend will be implemented using Python and hosted on AWS.
2. The backend service will include several microservices, each responsible for specific functionalities.
3. API gateway (Common choices are Kong, Tyk, Amazon API Gateway) acts as an entry point for API requests, routes requests to the appropriate microservices

Database

1. MySQL for Relational Data (Primarily utilized for user-related operations and audit logging)
2. DynamoDB for NoSQL Data (Handles flexible, unstructured data like files and collaboration-related information)

Storage

1. Can be physical (or virtual).
2. Storing actual folder and file contents
3. *Example:* A scalable cloud storage solution (e.g., AWS S3, Google Cloud Storage)

Infrastructure

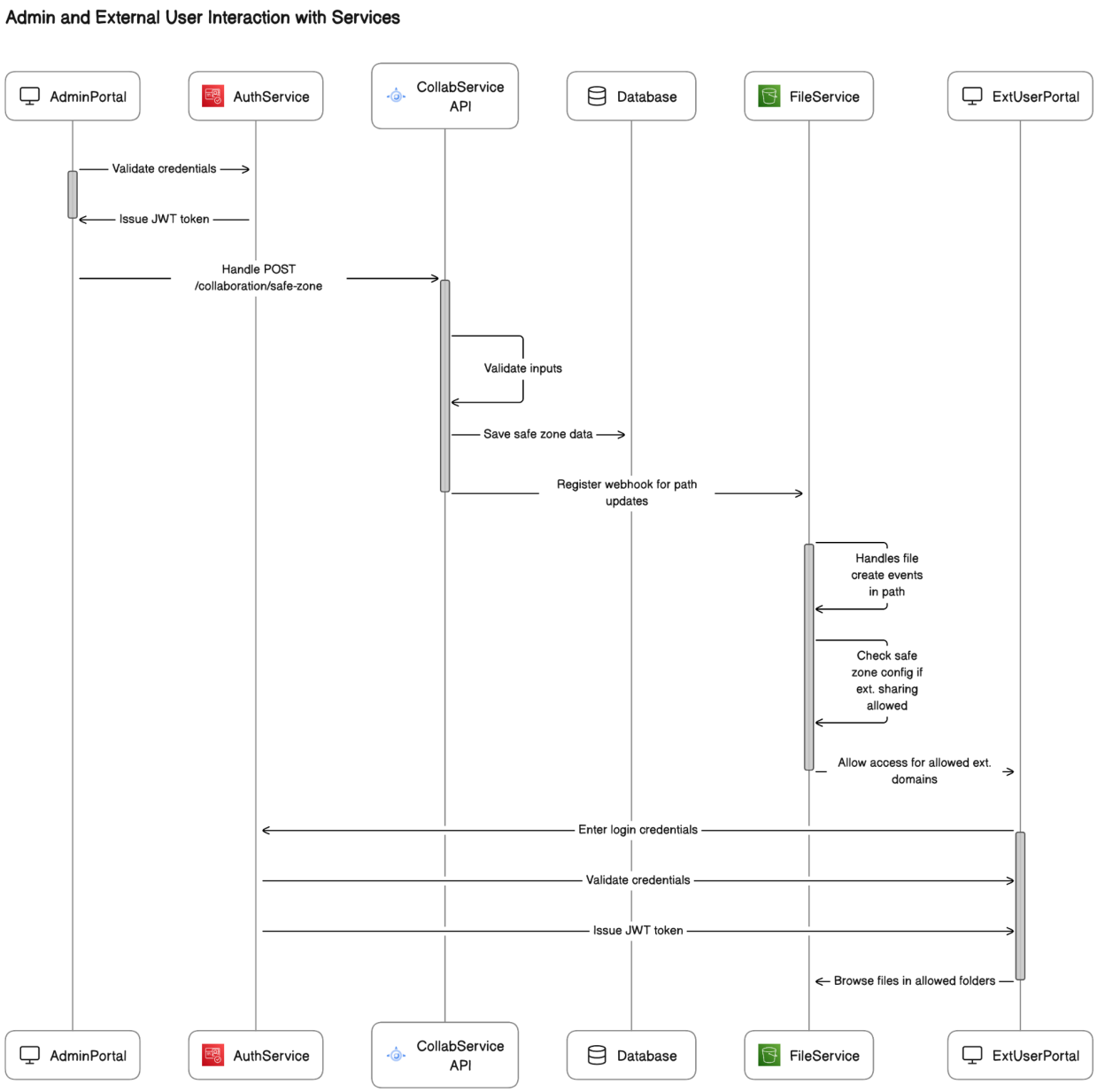
1. Cloud Provider: AWS (Amazon Web Services).
2. Compute Service: Amazon EC2 instances for hosting backend services.
3. Containerization: Docker for containerizing microservices.
4. Container Orchestration: AWS Fargate for managing containers without manual intervention.
5. Load Balancing: Elastic Load Balancer (ELB) for distributing incoming application traffic.
6. Scalability: Auto Scaling is configured to adjust the number of running instances based on demand.

Version Control

1. Utilizes GitHub as a version control and collaborative platform for hosting and managing code repositories.
2. Supports branching, merging, issue tracking, pull requests, and collaboration features.

Flowchart

Flow: Safe zone creation



key steps:

Admin Portal

1. Admin opens browser and navigates to admin URL
2. Enters admin credentials and logs in
3. Authentication service validates credentials and issues JWT token
4. Clicks to add a new safe zone

5. Fills out safe zone form details
6. Submits the form to create the safe zone

Collaboration Service API

7. Handles POST request to /collaboration/safe-zone endpoint
8. Validates the input data
9. Saves the safe zone data in the database
10. Registers a webhook for file path updates

File Service

11. Handles file create events in the shared pat
12. Checks if external sharing is allowed as per the safe zone config
13. Allows access for matching external domains

External User Portal

14. External user opens browser and navigates to login page
15. Enters credentials and logs in
16. Can browse files in the allowed shared folders

User Authentication and Authorization System

GitHub

Public project
HTTPS: https://github.com/talimahar/FileSharing.git
SSH: git@github.com:talimahar/FileSharing.git

```
(venv) talimahar@Talis-MacBook-Pro FileSharing % flask routes --all-methods

Endpoint             Methods             Rule
-------------------  ------------------  -----------------------
auth.login           OPTIONS, POST       /api/auth/login
auth.logout          OPTIONS, POST       /api/auth/logout
static               GET, HEAD, OPTIONS  /static/<path:filename>
user_mgmt.get_users  GET, HEAD, OPTIONS  /api/users
```

```
Flask==3.0.1
Flask_JWT_Extended==4.6.0
flask_sqlalchemy==3.1.1
Werkzeug==3.0.1
./requirements.txt (END)
```
