forkie

A file repository web app, with a CLI

Group 9

Jakab Zeller, Joe Rourke, Ayman Najah, Tom Potter

System Description

For our assignment, we were allocated the file repository system. The system has two forms of interface, with the intuitive web app or the convenient command-line client. The premise of the system revolves around a remote file repository where users can upload and share their files with groups of other users. A file may have many unique versions, which can each have individual metadata. Users with access may also comment on the files, viewable by other users and are "marked" appropriately when read by all. There are levels of permissions in the system which give access to different actions and pages, such as the admin's exclusive access to the archived files (where files not updated for a year are automatically transferred to). Other features include the ability to email group members and generate reports on statistics about files or groups.

Our Implementation

What we included

From analysing the initial client requests and requirements specification, we aimed to include as much as possible in the final solution. We met up as a group and worked through the requirements, mapping them to GitHub issue enhancements, which could then be assigned amongst our group members. Due to our thorough examination, I believe we've met a majority of the requirements in our final solution, and many of the included acceptance tests in the requirements modelling document were achieved when testing the system.

What we changed

The main changes of our implementation versus the provided documentation are detailed below:

- In our final system, the archive is simply a page displaying files with versions that have been marked as archived; in the original documents it was insinuated that the archive would be a completely different filestore - we did not have the resources, without greatly risking performance and usability, to implement this feature fully.
- In the UML sequence diagram for commenting, it suggests that users are given a confirmation popup before they insert a comment onto a file, this is a feature which our web application and CLI doesn't possess.
- The only major feature our system doesn't implement is the ability for group leaders and admins to alter the permissions of files in bulk. Although as this feature was classed as optional in the requirements document, it's exclusion doesn't impact the functionality of our system for the majority of scenarios. Given more time, we would have definitely been able to include this feature.

What we added

While the design process isn't used for detailing system requirements and platform choices, we feel our final system is unique in it's multi-interface capabilities. We also added the ability for users to send bulk emails to the other members of their groups, allowing us to make more of the application more reusable. Lastly, the reports feature wasn't explicitly meant to produce reports on files - although our system does allow for this.

What we used

The stack for our system is based entirely on Python, allowing us to keep it consistent amongst the different interfaces. We used packages such as Flask, SQLAlchemy, and docopts to help us speed up development. On the web app client side, JavaScript and JQuery are used to add some extra functionality to the page - with Bootstrap, albeit with some modifications, used for the UI. We used the B2 service provided by BackBlaze for file storage as we thought this was a more elegant and secure solution rather than storing the files directly on the Heroku web server. To communicate with the BackBlaze buckets we used the b2sdk python package which is just a wrapper for the B2 cloud storage REST API.

To send emails we used a SaaS platform called SendGrid which has a very usable Python interface, however problems arise when trying to send emails locally - so any email related functionality only works on the Heroku hosted version of our web app, which you are free to browse.

Our Code and Organisation

Our file structure

When planning the file structure, we examined dependency and decomposition views of the design document to give us an idea of how to effectively group our program's components, and make them easily accessible from one another. This lead to the following file structure.

- *app.py* Required here to run the Heroku Flask server
- forkie_runner.py Wrapper for running forkie CLI from root directory
- **client** Package which is called by the command line interface
- **src**
    - **db** This folder contained all the definitions of our SQLAlchemy DB tables
    - **api** Where the API blueprints were stored for different components of our API

- **comments**
- **files** Contains endpoint routes as well as backblaze interface class
- **emails**
- **reports**
- **user**
- **signin**
- **signup**
- **metadata**
- **groups**

To keep our code tidy we used Flask's blueprints architecture model, which allows you to split up routes into various subcategories, with their own url prefixes and encapsulated resources. Therefore, for each subfolder in **api** there exists a routes.py (and some have a utils.py for modular utility functions used within several API endpoints throughout the system) which defines the different endpoints for that API component; these blueprints are then linked into the main Flask server in *app.py*.

Another way we mapped the requirements and designs to our implementation was through the database design. Our database is structured in third normal form, meaning that there are no repeated or multiple values, with linking tables and primary/foreign keys. Using SQLAlchemy we were able to map these tables to "Models", which are simply objects which you can perform SQL operations on in a more abstracted manner than just writing the pure SQL code.

Approach for testing

Due to our development process using vertical slicing, where different team members developed different functionality concurrently before moving onto another section of the implementation, our system was tested thoroughly upon completion of each slice - this included testing directly and indirectly affected units of the slice. During the stage in which we were developing the file upload slice for example, we tested with various file sizes and found that due to Heroku's limited file resources, the size of the files our system accepts is likely around < 100MB, and this was discovered due to the extreme test scenarios we enacted. When our system was nearing completion we ensured that we made thorough use of all the application's functionality, and attempted to run all the possible acceptance tests, in order to prevent any mishaps during our presentation and when used by any client's in the future.

Installation Instructions

Forkie web server setup

For now you can only set up the server to run on localhost but the latest version of forkie will be deployed on *https://file-rep0.herokuapp.com/*.

How to use `installforkie.sh` script

To install the localhost and run on localhost you will need the scripts inside the `bin` directory. There are two scripts inside: `forkielocal.sh` and `installforkie.sh` (make sure scripts are executable). If you do not have the forkie source files downloaded from github already you can download the required files using the `installforkie.sh` script (usage below). Otherwise continue onto the section describing the use of `forkielocal.sh`.

```
./path/to/scripts/installforkie.sh [-o <output> | -h]
```

This script will first check if you have all the required command line tools installed to complete the installation. It will then check whether the specified output folder (`-o`) already contains an installation. Then it will download a tarball containing the latest commit of forkie from the master branch of the github repository using the github API. It will then extract, cleanup and finally ask if the user wants to `pip install` the requirements.txt.

How to run a forkie repository on localhost using `forkielocal.sh`

To start the forkie repository on localhost run the `forkielocal.sh` script (does not take any extra arguments). This will export all the environment variables required to run the forkie repository onto your local machine, check if the heroku Procfile exists in the current directory and runs the `heroku local web` command which starts hosting the forkie repository on *http://0.0.0.0:5000*.

Forkie CLI setup

To install the forkie CLI you need the `pip3` command and you can simply use (only tested on Linux and macOS):

```
pip3 install forkieCLI
```

If that does not work you can also run the `forkie_runner.py` script inside the source code directory, or inside the output location specified if using `installforkie.sh`. This script takes exactly the same subcommands, options and arguments as the forkie CLI command described in the user manual below (see the user manual on how to get started with the forkie CLI).

Forkie CLI missing dependencies

If you are missing python dependencies to run the CLI you can use pip to install the `requirements_cli.txt` file (also found in the source code directory) by using the command: `pip3 install -r requirements_cli.txt`. Dependencies should be installed automatically if using pip to install forkieCLI but will be needed if using `forkie_runner.py`.

User manual

Forkie website

To use the forkie website you must first create an account. When first visiting the website click the "Want to create an account?" link to setup a new account. Once that is done it will take you back to the login page where you can sign in.

Dashboard

- Once you sign in you are taken to the dashboard. On the top left there is a drop down for the groups that you are in (see *Group Dashboard*). You can also add a new group by selecting the "New Group" button.
- On the top right (from left to right) there is the new file and logout buttons (see *New File*)
- On the left hand side of the page there is the bulk comment button and notifications tile. The bulk comment button allows the user to comment on one or more files which they have access to. Once a bulk comment has been made it shows up in the notifications tile of all the users who also have access to that file.
- In the middle there is the files tile which shows the files (see *File*) that you have created/updated.

Group Dashboard

- When clicking on a group from the drop down menu in the main dashboard it will take you to the *Group Dashboard*.
- The "Back to dashboard" button on the top left will take you back to the main dashboard
- On the left the "Members" tile shows all the members in the current group
- In the middle the "Group Files" tile shows all the files (see *File*) in the current group
- On the right there are the "Group Actions" (and "Admin Actions" if you are the group leader).

  Group Actions
  - "New file" (see *New File*)
  - "Email Group" lets you email all members of the group
  - "Leave Group" will ask for permission and then leave the group if granted

  Admin Actions (these can only be accessed by the group leader)
  - "Add Member" opens dialog asking for a user's email (they must be registered on the current forkie repository).

    - “Remove Member” removes a member from the chosen group from a drop down (this can be done from any group).
    - “Rename Group” renames group
    - “Generate Report” creates a PDF file of all the groups info
    - “Delete Group” will ask for permission and then delete the group

New File

- When clicking on the New File button on the *Dashboard* or the *Group Dashboard*, the new page file is displayed.
- The “Choose File” button opens the OS specific file selector.
- The user can select the group to upload the file to from the available groups in the drop down menu (this only contains the groups that the user is a member of).
- The comment field is where the user would type a comment that will be sent to all the members in the group
- The green upload file icon button will then upload the selected file to the repository. After this is done you will be redirected to the file’s page (see *File*)

File

- Every file has its own page which can be accessed by clicking on a file inside the “Files” tile.
- The file page contains the “Groups with Access” tile that is a list of all the groups that can also see and update this file. When these groups are clicked on they will redirect the user to the group’s *Group Dashboard* (if they are in that group).
- The “File Comments” tile contains the comments relating to that file. Each comment can be marked as read or deleted.
- “File Versions” contains all the versions of the file that exist. If a specific version is clicked on it will take you to a page with more information (see *File Version*).
- “File Actions”, can be used to create a new version of the file or add a comment. When creating a new version a dialog is shown where the user can choose the new file version and add a version title which is displayed in the “File Versions” tile.
- “Admin Actions” are displayed when the user is a group leader or an admin.
    - “Add Group” adds a group to the groups who can access the file
    - “Revoke Group Access” revokes access from a group who can access the file
    - “Generate report” generates a PDF report that can be downloaded with the information of the file including metadata
    - “Delete file” removes all file versions from the repository

File Versions

- Every file version has a page associated with it. This page contains the version’s metadata and other actions
- You can download the file version you are currently on by clicking the “Download File” button
- The “Version Actions” tile contains actions to do with the file version’s metadata
    - “Added Metadata” allows you to add any title and value to the metadata table
    - “Edit Metadata” allows you to edit the title metadata and the any other metadata that users have added

- "Delete Version" will ask for permission and delete the file if granted

Forkie CLI

The forkie CLI was made to **accompany** the forkie website not replace it.

Main flow

- `forkie login <repo>` will "log you into" the repository with the `<repo>` URL (full URL) and create a `.forkie` directory inside the current directory as well as a directory for the repo including a binary file (which contains the users cookie) and a JSON file which contains the b2 backblaze keys used to access the forkie bucket.
    - E.g. If you wanted to log into the repository hosted on your localhost (`0.0.0.0`) on port `5000` then the command would be:

```
forkie login http://0.0.0.0:5000
```

- The login subcommand will ask the user if they want to create an account if no account is found on the web server
- You can then use any of the commands listed below…

The subcommands

The subcommands require you to be in the same directory as the .forkie directory. You can view all usages by using the `--help` option...

```
forkie make [-v | --verbose] [(-m <message>)] (<file>)...
```

- Start tracking new file in a repository
- Will search through all the available repository cookies inside the `.forkie` directory and prompt the user as to which they want to add to and then prompt which group to add to
- If an identical file is found then forkie will prompt the user to either start tracking the new file or to use the update subcommand instead
- The ellipsis denotes that more than one file can be specified (with optional name)
- If no `<message>` argument is given then the default editor will be opened
- Options:
    - `-m --message`: The message/description of the file.
    - `-v --verbose`: Display info about the inner workings

```
forkie update [-v | --verbose] [(-m <message>)] (<file>)...
```

- Create revision to file or multiple files
- The update subcommand works in a similar way to the make subcommand
- Will first find every file that matches the filename of `<file>` in every repository in .forkie directory

- The user then gets to pick which repository to update. If one of the files isn't contained inside the chosen repository then the user can either continue with the ones that or abort
- If the user uploads a version that matches the current version inside the chosen repository then uploading will be aborted and the user is prompted
- Options:
    - `-m --message`: The message/description of the file.
    - `-v --verbose`: Display info about the inner workings

```
forkie find (-a | -n <name> [(-p <group>)]) [-vd] [(-c <comment>) [-f |
--force]]
```

- Find a file/list files
- The `-a` option returns all files in every repository in `.forkie`
- If the `-n` is specified then it will search all repos by filename. The optional `-p <group>` is used to also search by group name
- You can download any of the returned files by adding the `-d` option
- Bulk commenting can be done by adding the `-c <comment>` option and argument. Force (`-f`) can be used not ask for permission
- Options:
    - `-a`: All files
    - `-d --download`: Show download context
    - `-n`: Search by name
    - `-p`: Search by group name
    - `-v --verbose`: Verbose
    - `-c --comment`: Bulk comment
    - `-f --force`: Force/Don't ask for permission first

```
forkie group (-V [--peeps] [<email>...] | (--add | --rm | --change) (-p
<group>) [<email>]) [-vf]
```

- Subcommand to handle everything to do with groups
- Four main options:
    - View: view all groups (in all repositories inside `.forkie`), or just peeps by using `--peeps`, and filter by email (only can view the groups that the user is a part of)
    - Add: if only `-p` is specified then it will create a group with name `<group>`. If `-p` and `<email>` is specified then it will add the user of the given email and add them to the group of `<group>` (only if you are the group leader).
    - Remove: remove group by just specifying `-p` or a remove a user from the `<group>` by also specifying the user's `<email>` (only if you are the group leader)
    - Change: rename the group by just specifying `-p` or move a user to another group by also specifying the user's `<email>` (only if you are the group leader of both groups)
- All of these, except view, all require permission (y/n) to complete the chosen action
- Options:

    - `--peeps`: View people
    - `--add`: Add person/people to a group
    - `--rm`: Remove person/people from a group
    - `--change`: Move person/people to another group

```
forkie report (-p <group> | <email>) [(-o <file>)] [-v | --verbose]
```

- For generating and viewing reports about groups/users
- Generate a PDF report containing information on:
    - On a group using `-p`
    - On an individual user by specifying `<email>`
- Specify the local output path of the PDF by using `-o`

```
forkie login (<repo>) [-v | --verbose]
```

- "logs into" the web server to authenticate the users identity. Then will create a binary file containing the cookie that will authenticate the user in the web server
- Used to login to the repo with the URL of `<repo>`
- Will suggest the user to signup if their credentials aren't found on the repository

How we worked as a group

Overview

Initially working as a group we had started a WhatsApp group chat. We discussed meeting up and talking about how we should begin planning the development of the project. Despite one of our team members not being able to make major changes during lab sessions due to not having git, we were able to plan work. We met several times throughout the course of development, and we communicated through the groupchat when we could.

We had utilised an in-built function within the GitHub interface which functioned similarly to Trello and we had different objectives with differing priorities. Once tasks were completed or altered we would be notified and the task would be complete, allowing the developer to move on to the next task they were assigned.

All members of the group were required to use technologies unknown to them beforehand, so it was a learning curve for everyone. We used the README document in the Git repository in order to put examples on how to use the different modules in our system, in order to assist the other members of our group.

Contribution Table

Name	***Contributions***
Joe	For this assignment, I organised the group meetings (including the initial planning meeting where we began listing out the required enhancements to the system), and created the WhatsApp group we used to communicate in. Within our final solution I had the task of developing our only major horizontal slice, the database tables, which were challenging as I was new to PostgreSQL and SQLAlchemy. I wrote a majority of the API endpoints (namely those within comments, metadata, file upload, authentication, and groups), using Flask's blueprint architecture, this involved having to examine the design and requirements documents to ensure I included as much of the functionality as possible. The web app UI was also my responsibility, ensuring the system met as many of Nielsen's usability heuristics as possible to make it clean and usable, and that the appropriate data was requested for each route in the web app. For this documentation, I wrote the system description, and the description of our implementation sections.
Jakab	I was responsible for the forkie CLI and some of the web server endpoints such as the file query endpoints along with most of the backblaze interface class. For the CLI I found it hard to implement the requirements in a way that seemed to flow the best in terms of user experience. I had to modify some of the endpoints so that they worked better with the CLI and implement some new endpoints such as the endpoint to move a member to a different group. I also wrote the install and run scripts as well as the file compare module which was ultimately replaced by the backblaze interface. For the documentation I wrote the user manual and installation sections.
Ayman	I was responsible for developing add and remove group members, archiving the file, and developed the generate report function. I helped coordinate meet ups with team members, and suggested planning of splitting the work equally. I completed the "How we worked as a group" section. I edited minor template elements.
Tom	I mainly worked on API functions related to metadata and determining whether a comment had been read.

Contribution Percentages

	Joe	***Jakab***	***Ayman***	***Tom***
Planning	25%	25%	25%	25%
Setup	50%	30%	10%	10%

Development	45%	40%	10%	5%
Coordination	30%	20%	40%	10%
Testing	25%	25%	25%	25%
Signed by:	JR	JZ	AN	TP