Development Notes

Giuseppe Barillari, Connall Shurey

June 7, 2023

Contents

1 Overview

We are developing 2 client applications and 1 server.

1.1 Clients

1. Android application, Fauna Finder
2. (Sys)Admin application, Icarius

1.2 Server

1. Penelope

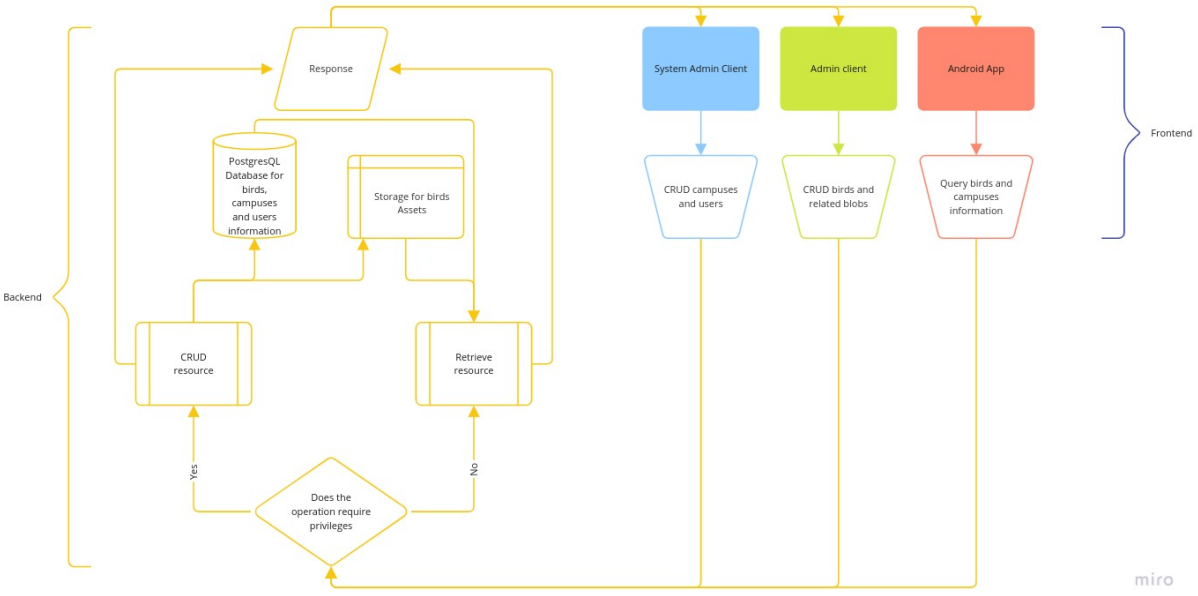
2 Penelope

Penelope is a Java SpringBoot server. It relies on an underlying PostgresQL database instance.

2.1 Development Environment

1. VSCode
   (a) Extension Pack for Java
   (b) Gradle for Java
   (c) Jupyter
   (d) Spring Boot Dashboard
   (e) Spring Boot Tools
   (f) XML (by Red Hat)
2. OpenJDK 17.0.5
3. psql (should come with PostgresQL installation, you're free to use whatever other tool you prefer e.g. pgAdmin4)

2.2 Operation diagram



2.3 Authentication

All CRUD requests must be authenticated. To authenticate a request it must include the following header:

```
Credentials: username=password=timestamp
```

Where:

- `username`: is the user's username.
- `password`: is the user's password in clear text (more on security later).
- `timestamp`: is the ISO timestamp that registers the moment in time the request was sent out of the client.

The `Credentials` key of the header should be in clear text. The `username=password=timestamp` value of the header should be RSA encrypted, not to expose user's credentials.

The `timestamp` effectively acts as a randomiser, as the encrypted string will always be different (ISO timestamp should include milliseconds). The `timestamp` field is checked server-side to ensure no malicious activity is being performed. If the `timestamp` field in the header differs by more than 60 seconds from the time at which the request reaches the server, the server discards the request. This ensures that an attacker can't capture the value of the header once and re-use it with malicious intent.

To RSA encrypt the value of the header, the server provides a public key at the endpoint `/key`. The endpoint returns an empty response with a `Key` header that contains the base64 encoded value of the public key.

The server will try and decrypt the value of the `Credentials` header by using the matching private key to the exposed public key. It will then check `username`, `password`, `timestamp` validity and whether the user can CRUD the requested resource. If the server cannot decrypt the value of the header, the request is not authenticated and is rejected. The request is also rejected if `username` and `password` don't match or if the user does not have CRUD access to the requested resource.

2.4 Storage structure

2.4.1 Assets

Images, Videos and Audio files are stored locally on the machine for development purposes only. In production, the application should use a dedicated storage service (such as AWS S3 buckets) not to pollute the application's file system with assets.

The assets are retrieved using the FileSystemStorageService class. The class implements the StorageService interface so that, when moving to production, a class tailored to communicating with the dedicated storage service can be implemented.

The StorageService interface currently accommodates for a function to load a resource from the database. This should probably be moved somewhere else.

Assets are cached upon request. Caches are evicted when a bird's data is updated.

2.4.2 XMLs

XML files are generated upon request and cached. On data changes in the database, any XML reflecting said data will have its cache entry evicted. The XML will be cached again in the next generation (request).

2.5 TLS

In production, the server enforces TLS connections via a certificate provided by Let's Encrypt. The certificate is only valid for the following domain: `penelope.barillari.me` (I personally own the `barillari.me` domain, if we were a real company, we would have bought an appropriate domain).

2.6 Running Penelope Locally

The quickest (and easiest) way to run Penelope is via the provided Docker containers. Alternatively, you will need to have a PostgreSQL database instance running to which you need complete access. For marking purposes, I provided a special configuration that allows you to run the server without any PostgreSQL instance. Furthermore, using the marking configuration, TLS is disabled by default within Penelope.

2.7 Running Penelope for marking

Prior to hand-in, I introduced a new configuration file: `application-marking.properties`. This configuration allows markers to run Penelope without a PostgreSQL instance. Instead of using PostgreSQL, it uses the H2 **in-memory** database. As this is an in-memory database, changes to the database are not persistent, i.e. when you stop/restart the server any data you stored on the database will be wiped. Markers still need to set a storage path (storage is persistent, so file uploads will be safe after stopping/restarting the server):

- You must change the `penelope.storage.base-folder` field in

  `src/main/resources/application-marking.properties`

  to a path you have access to. Example values:

  - Linux: `/home/joe2k01/Documents/sweng/penelope_storage`
  - Windows: `C:\Users\joe2k01\Documents\penelope_storage`

Finally, from the root of the source project, run the command:
Windows:

```
.\gradlew.bat bootRun
```

UNIX

```
./gradlew bootRun
```

Alternatively, you can use the run button on the top right in VSCode.

2.8 Running Penelope for team-members/developers

2.8.1 Disabling TLS before executing

As the TLS certificate is only valid on the `penelope.barillari.me` domain, you will need to disable TLS. To disable TLS, locate the source file:

`src/main/java/sweng/penelope/PenelopeApplication.java`

and comment out the following line of code (line 26):

```
@PropertySource("classpath:application-ssl.properties")
```

2.8.2 Running with Docker

To run the server via Docker you need to execute the following command from within the `docker/` folder at the root of Penelope's source project:

`docker compose up -d --build`

Note that you need to be part of the docker group in UNIX systems or to be an Administrator on Windows.

2.8.3 Running without Docker

You **should not** run Penelope outside of the Docker containers. However, here's what you'd need:

- A PostgreSQL instance you can reach.
- The ProstgreSQL instance must have a database named `cbg`.
- The PostgreSQL instance must have a user `penelope` with password `longboi`.
- The `penelope` user within the PostgreSQL instance must have full administrative access to the `cbg` database.
- You must change the `spring.datasource.url` field in

  `src/main/resources/application.properties`

  to match the URL to your PostgreSQL instance (if running locally it should be

  `jdbc:postgresql://localhost:5432/cbg`

  , assuming you did not change the port).
- You must change the `penelope.storage.base-folder` field in

  `src/main/resources/application.properties`

  to a path you have access to. Example values:
  - Linux: `/home/joe2k01/Documents/sweng/penelope_storage`
  - Windows: `C:\Users\joe2k01\Documents\penelope_storage`

Finally, from the root of the source project, run the command:
Windows:

`.\gradlew.bat bootRun`

UNIX

`./gradlew bootRun`

Alternatively, you can use the run button on the top right in VSCode.

3 Android

The Android client must be able to handle XML files provided by the server and XML files loaded by the user from the local file system. Hence, the client needs to comply with the standard fully.

3.1 Running the client in Android Studio

Running the client for local development implies you already disabled TLS in Penelope. If not, please see 2.6.

Once TLS is disabled in Penelope, from the source root of the project, locate the file:

`app/src/main/res/values/strings.xml`

And change the entry (line 7):

```
<string name="serverURL"
↪  translatable="false">https://penelope.barillari.me:8080/</string>
```

To

```
<string name="serverURL"
↪  translatable="false">http://your.computer.ip.address:8080/</string>
```

Where `your.computer.ip.address` is your computer's local IP address, it should look something like `192.168.0.87` (Although this is very specific to your own network. The example is my computer's IP address provided by a DHCPv4 compliant router, yours will be different. Do not use the example value). **Ensure** the protocol is `http` and not `https`.

In debug configuration (which should be the default one when running from Android Studio) you will not need the next step, jump to the last paragraph. Next, from the source root of the project, locate the file:

`app/src/main/AndroidManifest.xml`

And insert the following line at line 16, without removing anything:

```
android:usesCleartextTraffic="true"
```

This allows the app to communicate with Penelope without TLS handshakes. **This must not be done for production**.

You can now run the app on an Android emulator (provided by Android Studio) or on your own Android device. Click the run button on the top right within Android Studio. Ensure the target is your intended device, and the build target is `app`.

4 Icarius

Icarius is our (Sys)Admin desktop application.

4.1 Operation

Upon the first launch, the application requires credentials to log in as a user. Depending on the permissions of the user, the UI will display different options.

4.1.1 Admin UI

The Admin user interface displays options to CRUD birds to our database, through Penelope.

4.1.2 SysAdmin UI

The SysAdmin user interface displays the same options as the Admin one, plus the option to CRUD users and to CRUD campus locations.

4.2 Actions

Please see the Penelope API documentation for the below actions.

4.2.1 Creating a Bird

In order to register a bird on our database we need to first store its related assets. Therefore, the UI must force the user to upload all the required assets (see Assets) before adding bird information. Upon uploading each asset the application can expect a response containing a URL to retrieve the just uploaded asset. All assets are uploaded through a POST request.
The next step for the user is to input the text information regarding the bird. Finally the application makes a POST request to Penelope to store the data.

4.2.2 Editing a Bird

The UI displays a list of all the birds the current user has access to. The user can then edit any text or asset information through a PATCH request to the server.

4.2.3 Deleting a bird

Deleting a bird happens via a DELETE request. All text and asset information is deleted server side.

4.2.4 Creating a Campus

Creating a campus happens via a POST request. The campus information only requires a name to be supplied by the user.

4.2.5 Editing a campus

Editing a campus happens via a PATCH request, the only campus information that can be edited is the name which is supplied by the user.

4.2.6 Deleting a campus

Deleting a camoys happens via a DELETE request. All text information (database) regarding the campus and all the birds within the campus (text and assets) is deleted server side.

4.3 Running Icarius

4.3.1 Connecting to Penelope

Running the Icarius client for local development implies you already disabled TLS in Penelope. If not, please see 2.6.

Once TLS is disabled in Penelope, from the source root of the project, locate the file:

`src/main/java/icarius/App.java`

And change the field (line 19):

```
public static final String BASE_URL =
↪  "https://penelope.barillari.me:8080";
```

To

```
public static final String BASE_URL =
↪  "http://your.computer.ip.address:8080";
```

Where `your.computer.ip.address` is your computer's local IP address, it should look something like `192.168.0.87` (Although this is very specific to your own network. The example is my computer's IP address provided by a DHCPv4 compliant router, yours will be different. Do not use the example value). **Ensure** the protocol is `http` and not `https`.

4.3.2 Logging in

With Icarius successfully connected to a running Penelope instance, developers can log in with the default System Admin Login:

- Username: `theAdmin`
- Password: `thePassword`

With these credentials, developers should now have access to all of Icarius features