In the third project assignment, you will use the database design/implementation and some of the SQL queries you created in the first and second project assignments. You will design an application with a graphical user interface (GUI) that will meet the following requirements:

- You have to implement a Desktop application in Java or Python. For Java use Swing (taught in BPC-PC2T) or JavaFX, for Python, it is up to you. If you already know some web-based frameworks Spring/Jakarta EE/Django/Flask and arbitrary JavaScript library/framework you can also use them.
- The application must be compilable and runnable from command-line (use Maven or Gradle – check seminar project), e.g., to compile/build `mvn clean package` and to run `java -jar my-bds-app.jar`.
- The database will contain the user passwords in a hash form (hashed using Argon2, PBKDF2 or any other recommended hashing algorithm).
- The application will have a sign-in window with username/password authentication (if a user enters the wrong credentials, the application will pop up "Username or password is not valid"). This sign-in window won't be any fake login. It must be really validating the username + password from the database (consider also loading user roles – in enterprise systems, the roles during the authentication process are saved to the security context and used later for authorization purposes).
- Create a database user role for the application (not a superuser role!) that can sign in to the database and have suitable privileges. Furthermore, create/use a different than "public" schema for your database (e.g., the one from the project assignment 2).
- Create CRUD operations (create, read, update, delete) for at least three entities. Do not forget to have it in the GUI (applies to all operations). For one entity, implement the findAll operation.
- For one entity, provide a detailed view (you have to use JOIN here).
- Implement one operation in GUI that invokes more than one query and runs them all in one transaction. If something fails, roll back that transaction.
- Implement at least three windows showing different entities within the GUI. For example, three GUI components show a list of entities such as persons, addresses, and contacts.
- Create a possibility to filter data about any selected entity (e.g., find persons by family name).
- Create a dummy table in the database and create a GUI window where you can simulate the SQL injection attacks. Simulate both (injection with `drop table` and injection where you retrieve more data than expected (e.g., 1=1)).
  - Explain the necessity of PreparedStatements.
- Log exceptions suitably (e.g., use SLF4J Logback), set logs archiving per day (avoid log-and-throw antipattern). Avoid Log4j2 older versions < 2.17.1 (as was thrown up in 2021 by CVE 2021-45046, CVE 2021-45105, and CVE-2021-4483).
- Create a script that will back up your database every midnight (*NOTE: you can use* `pg_dump`). In practice, the backups might be performed using daily backups plus continuous backups of transactions and logs using wal-archiving, these backups would be then backed to the Cloud Object Storage (e.g., Amazon S3).

- Generate a self-signed certificate for your database/application (use `openssl`) so the communication between the application and database is encrypted.
  - The SSL configuration in the PostgreSQL is performed within `postgresql.conf` file (see: `https://www.postgresql.org/docs/current/ssl-tcp.html`) and `pg_hba.conf` (see: `https://www.postgresql.org/docs/current/auth-pg-hba-conf.html`).
  - From the perspective of Java, the certificate is imported into Java `cacerts` file via build-in utility `keytool` (see: `https://docs.oracle.com/en/java/javase/11/tools/keytool.html`).
  - **Hint:** Application to database connection string then can look similar to that (in some systems the & needs to be replaced with &):
    `jdbc:postgresql://127.0.0.1:5432/bds-db?currentSchema=bds&ssl&sslmode=verify-ca`
    - `currentSchema=bds` means that you won't need to put `bds` prefix before each table in your SQLs in the code.
    - For certificate verifications, check: `https://www.postgresql.org/docs/current/libpq-ssl.html`.
  - **Hint:** Cooperate with your classmates to set up certificates for the database and application.
- Externalize the application configurations, so that database and other essential settings (e.g., any user/app credentials) are not bundled within the code or the `.jar` file. For instance, use `.properties` or `.yaml` file.
  - **Hint:** check the template project.
- **Important:** Create the GitLab project repository named "bds-project-assignment-3" inside your student space (`https://gitlab.com/but-courses/bpc-bds/student-projects/2023/<YOUR_SPACE>/bds-project-assignment-3`). Furthermore, upload all the files that are essential to run your project. It would be good to include the documents from the first project assignment (the diagrams, etc.) to provide a context for your project.
  - Do not forget to add `.gitignore` file that will ignore the target folder and other unnecessary files (e.g., IDE configs).
  - Create a README.md file that will describe how the application can be built and run and what is the goal of that application. For example, inspire from `https://github.com/patrickfav/bcrypt` but you do not need to be so detailed.
  - Your GitLab repository should have at least 5 commits.
  - Create a document (`.pdf`) that will describe what the user can do in your application and what is the goal of your application (at least 3x A4).
  - Add a LICENSE file with a suitable license (e.g., consider MIT or Apache 2).
  - Create/Or generate the document that lists all the external libraries that your project is including together with their license (e.g., use `mvn project-info-reports:dependencies`). For example, check `https://gitlab.com/but-courses/bpc-bds/java-db-training/-/tree/master/licenses`. It is necessary for you or your company to sell software to check the licenses of used dependencies. Before you consider the usage of an external library, check its license first!
- **Optional:** Implement Transparent Data Encryption (TDE) for PostgreSQL. This type of encryption is also known as the encryption of data at rest.
- **Optional:** Generate SSH key-pair and configure your GitLab account to be able to use SSH. Set the remote-url origin of your project to use SSH.

- **Optional:** Use the arbitrary procedure to retrieve any data and show them in the GUI.
- **Optional:** Implement the JUnit tests for selected database queries (for these tests, use an in-memory database, e.g., H2 (with the same schema as your original database), where you will insert relevant mock data).
- **Optional:** Dockerize your project. Develop a `Dockerfile` and provide instructions on how to build and run your project in a Docker container.
- **Optional:** Integrate GitLab CI (`.gitlab-yi.cml`) for your project to check the build/test phase.

This project assignment is performed by each student solely.

This project assignment is for 6 points.

- **Target date: 31th of December (inclusive)**. If anyone would like to submit and defend the project earlier (before Christmas) let me know via email.
- The final project defences will be held at the beginning of January 2024.
- 1 for satisfying all the requirements (if any is not met, the number of points is set to 0).
- 0-5 based on the quality of the project and the individuality of your work (if your solution is just the copy-paste of the template project, your work will be evaluated with low points).

Note: To run `mvn` command from your terminal it is essential to download and configure Maven and Java in the system environment variables (to ease the Maven configuration you can benefit from Maven Wrapper (see: `https://maven.apache.org/wrapper/`)).