COMP2411 Report

Group members:

Fan Chun Yin

1. TOC
   1. How it works
   2. How to use it
   3. Presentation
   4. How to compile & reproduce everything
   5. Challenges
2. How it works

This application requires a database connection to a certain compatible system. It will be presented with a simple UI and an intuitive way of using it. The main database for development is upon Oracle Database 12c, although in theory it can run anywhere if we modify the migration SQL to not use Oracle dialects and force the use of ANSI SQL instead.

The application is based on Gradle & Kotlin, with Spring Boot as the foundation of software development toolkit. JavaFX is also used as the main GUI development with another dependency called TornadoFX as the basis of the Rapid UI design DSL.

Kotlin is a language that targets the Oracle Java platform and is also a dependency of TornadoFX. It has numerous improvements compared to Java as a language, such as automatic generation of Beans properties and extension methods.

We also assumed that since the report doesn’t mention Java as a language as requirement, and mentioned only JDBC, so any platform-compatible language to JVM that can run JDBC such as Jython, Scala & Kotlin will also work.

Spring is a very popular Java framework, that most of the internet services providers such as Google, Alibaba & Meituan, etc. Spring Boot is an all-in-one solution that packs away the burden of configuring Spring and it just works. Spring itself provides an Inversion-Of-Control (IoC) framework, which lays the foundation for Dependency Injection (DI) to work. DI is a programming technique, that instead of actively looking for a global instance, we just declare what we wanted and set the object on-the-fly, i.e. an injection of an instance to another instance. We also had intensive use of DI in the UI program to inject database repository and services.

Hibernate is a Java Object-Relational Mapping (ORM) framework that instead of writing raw SQL queries, we can represent it using Java classes & Java annotations, and fetch/save it from a central repository. This makes database manipulation & persistence much easier, and we can apply the OOP techniques we learnt in the University through other courses, such as adding a polymorphic relation. We use Hibernate as the only way of handling data requests in the program.

Flyway is a Java database migration tool that scans for any possible revision upgrade/downgrade of your database and apply them if necessary. This is used to upgrade the generations of our database schema that has come a long way of 9 generations. We use it to also review the need of our database and enrich it if some functionality is lacking.

TornadoFX is a Kotlin based JavaFX framework that provides a simple way of programming UI application in a Model-View-Controller (MVC) style without using traditional FXML in JavaFX. It has model validation out of the box, and it also works with our Spring Boot to inject services to the view.

IntellIJ IDEA by Jetbrains is an Integrated Development Environment (IDE) that makes developing Java-based program in a much faster pace. It will mark your syntax errors, format your program and allow you to run debugger and set breakpoints. It is also used to inspect our database schema & the data it stores, so we can find some demands.

Together they are the architecture of our UI program.

**REMINDER: Use and Only Use Java 8 For This Project, Newer Versions of Java Such As Java 11/13 Will Not Work Due To Removed JavaFX Dependency**

**Beware of this version incompatibility issue. In the mean while please check if your Java version is 8 by “java -version”**

**In the mean while you can use this command to force running on Java 8, e.g.:**

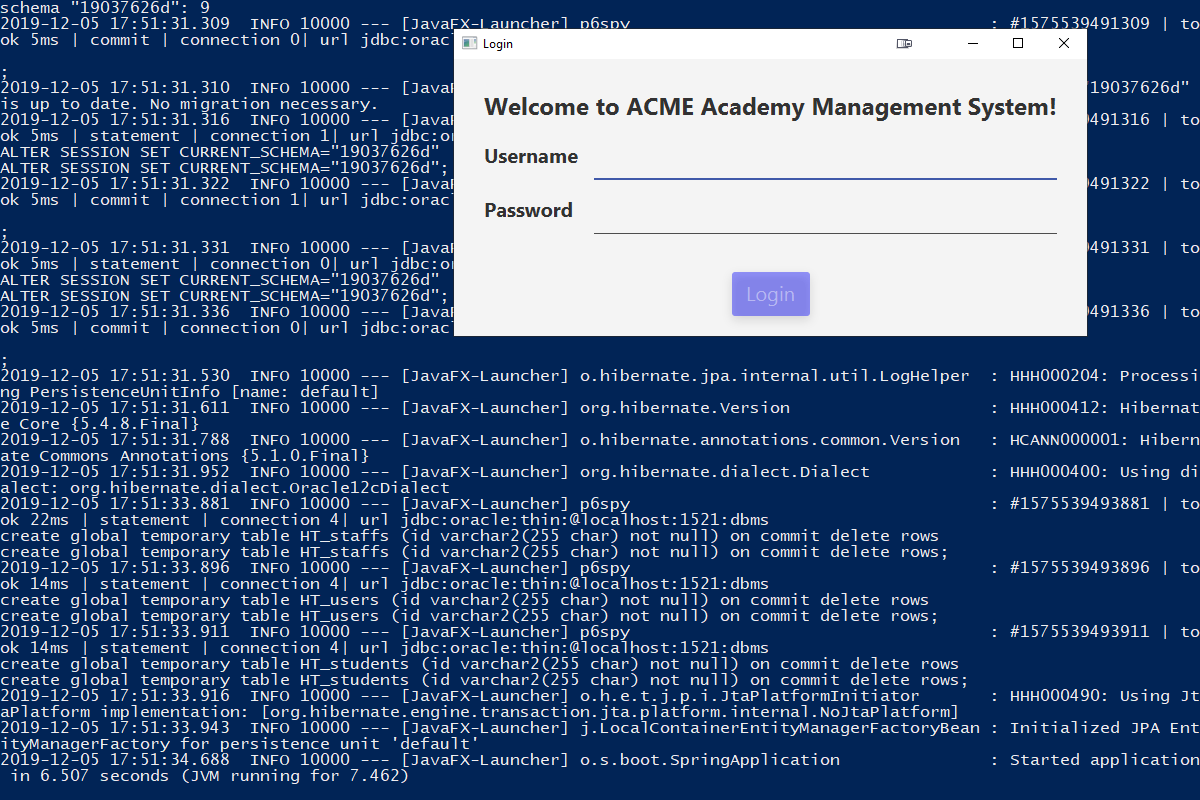
**`$env:PATH="C:\Program Files\Java\jdk1.8.0\_231\bin;$env:PATH"; java -version`**

1. How to use it

The way of using the program is simple, first, you need to be able to contact the target database, in this example we are using an intranet Oracle DB 12c provided by Department of Computing of PolyU. Therefore, to initiate an external connection of the database from the internet, we need to start a proxy connection to the PolyU servers first, a script file called “launch\_reverse\_proxy.bat” had some example codes of doing so.

Assuming we had already built the program and the JAR file, we can start it by “java -jar COMP2411\_Assignment2.jar”. The JAR file should be generated in “<project root directory>/build/libs”.

Then, run “java -jar COMP2411\_Assignment2.jar” to start the application. You should see some verbose logging data, indicating the bootstrap of the application successful. You should be able to see a beautiful UI screen.



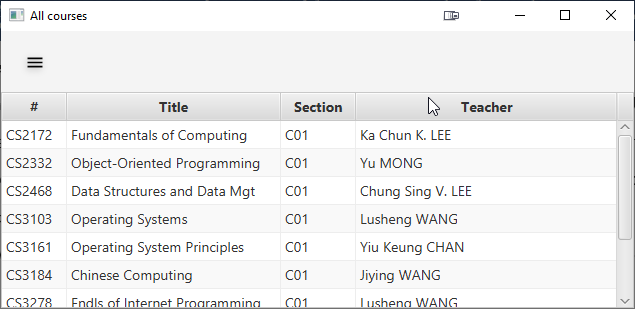
Make sure your connection string is correct. You can also change the connection information via the following three environment variables: DB\_PASS, DB\_CONN\_URL and DB\_USER. Here we have used the database of one of our own teammates. If you happen to want to reproduce to another database, you will need to change the previously mentioned three variables. The migration should run automatically.

You will be able to login as a student or a staff. By default, to login as a student account, you can use ID 50000001 to 50000099, the password is the same as the ID; to login as a staff account, you can use T0000001 to T0000010, also the password is the same as the ID.

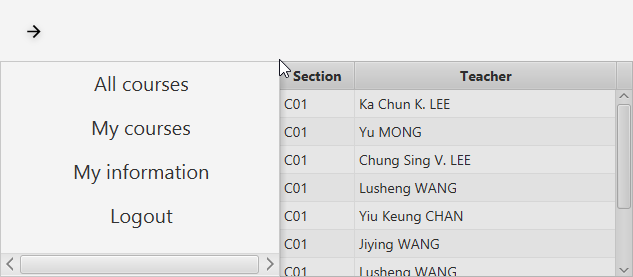
This password however can be changed by either the logged-in student or any staff and is protected by a sophisticated format called BCrypt. If a student happens to lose the password, it cannot be recovered via plaintext and a staff must be involved to reset it.

1. Presentation

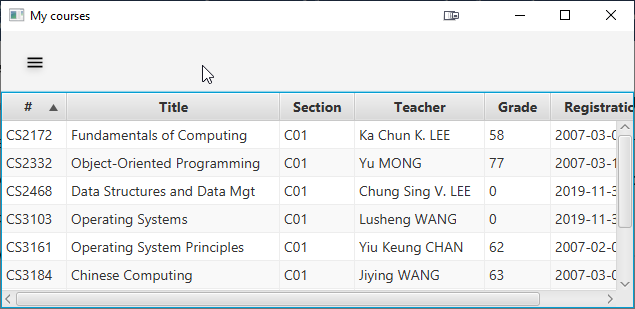
**Student view**



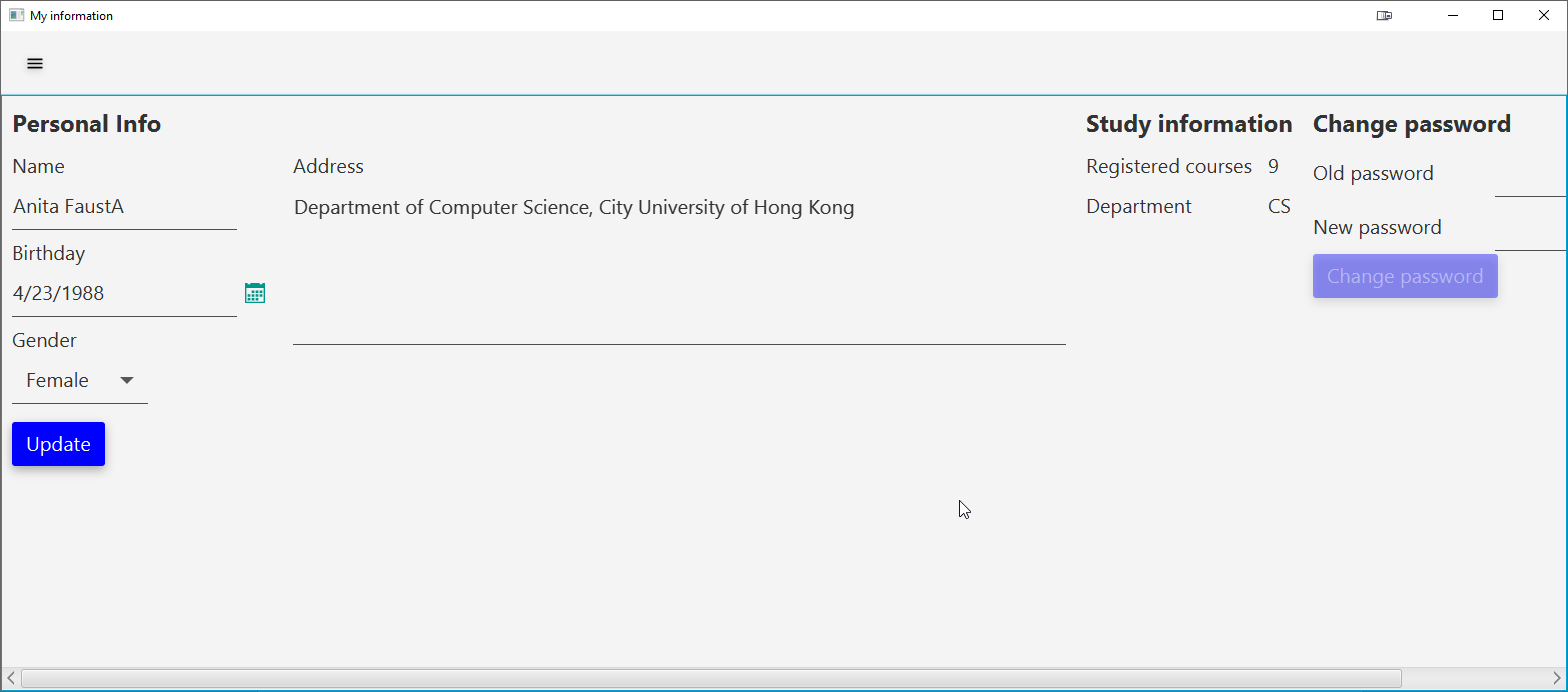
You can see all the courses



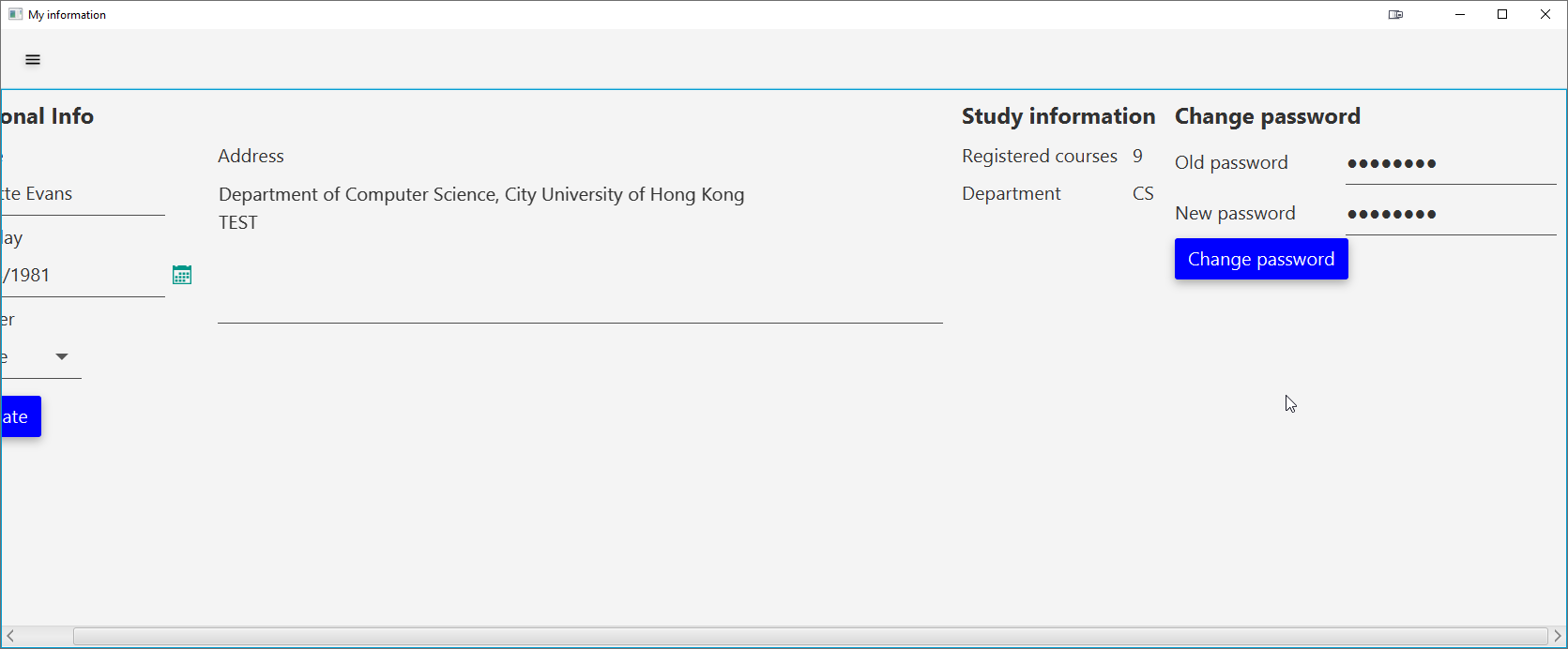
You can access a navigation menu



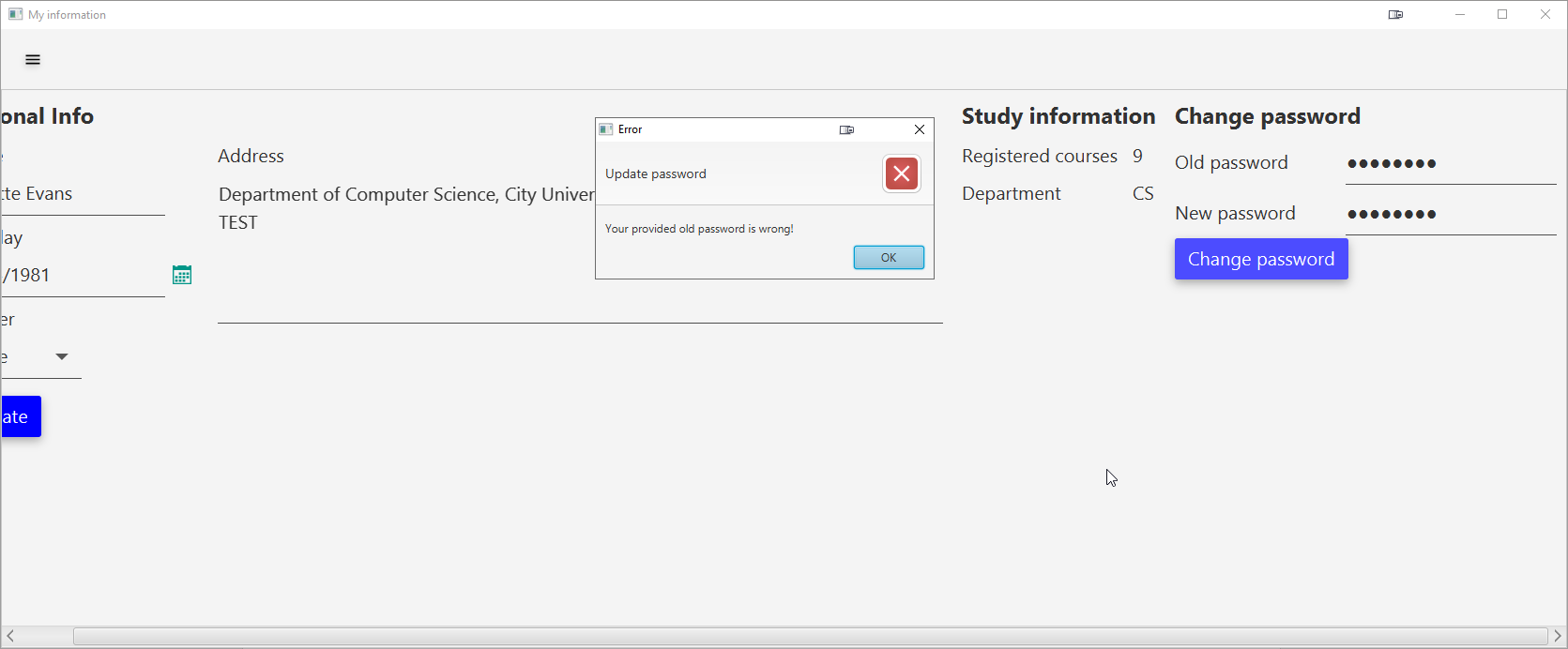
You can also see the courses your registered.



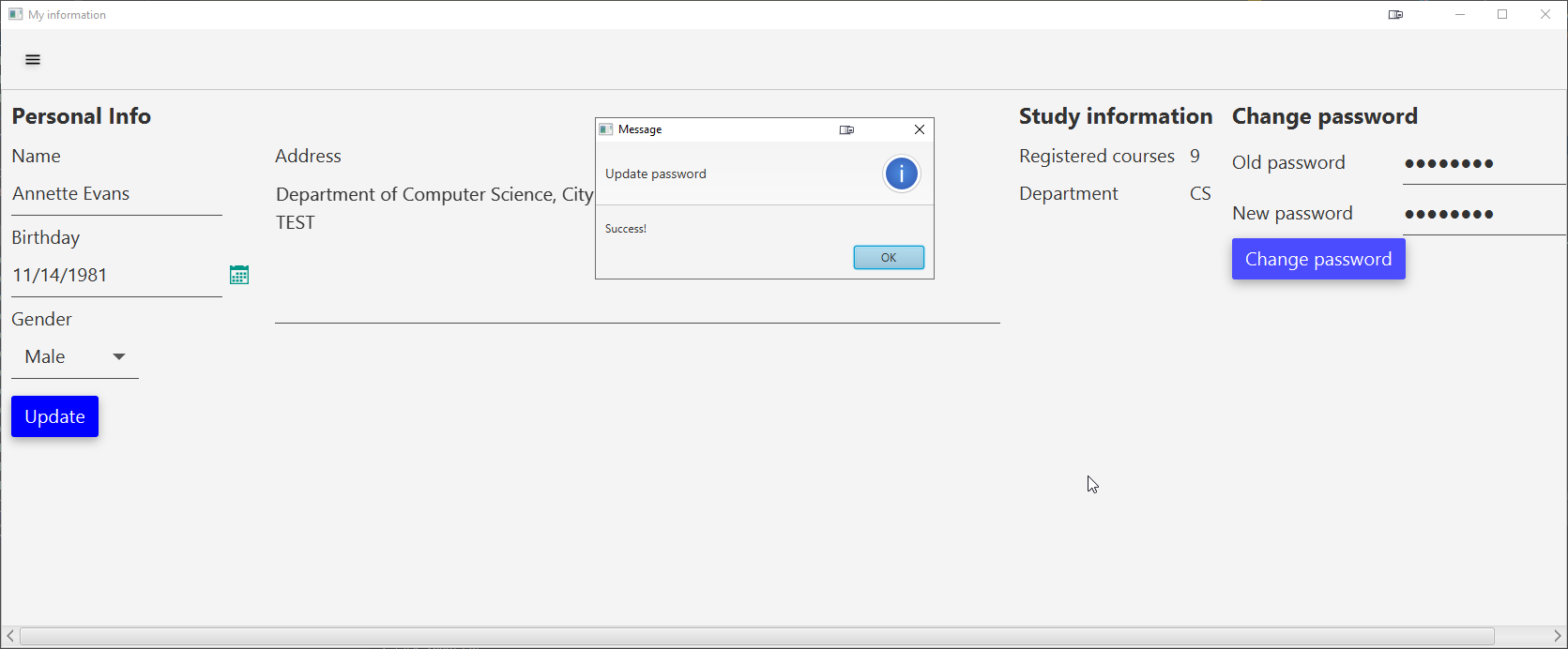
And you can change your own personal information.



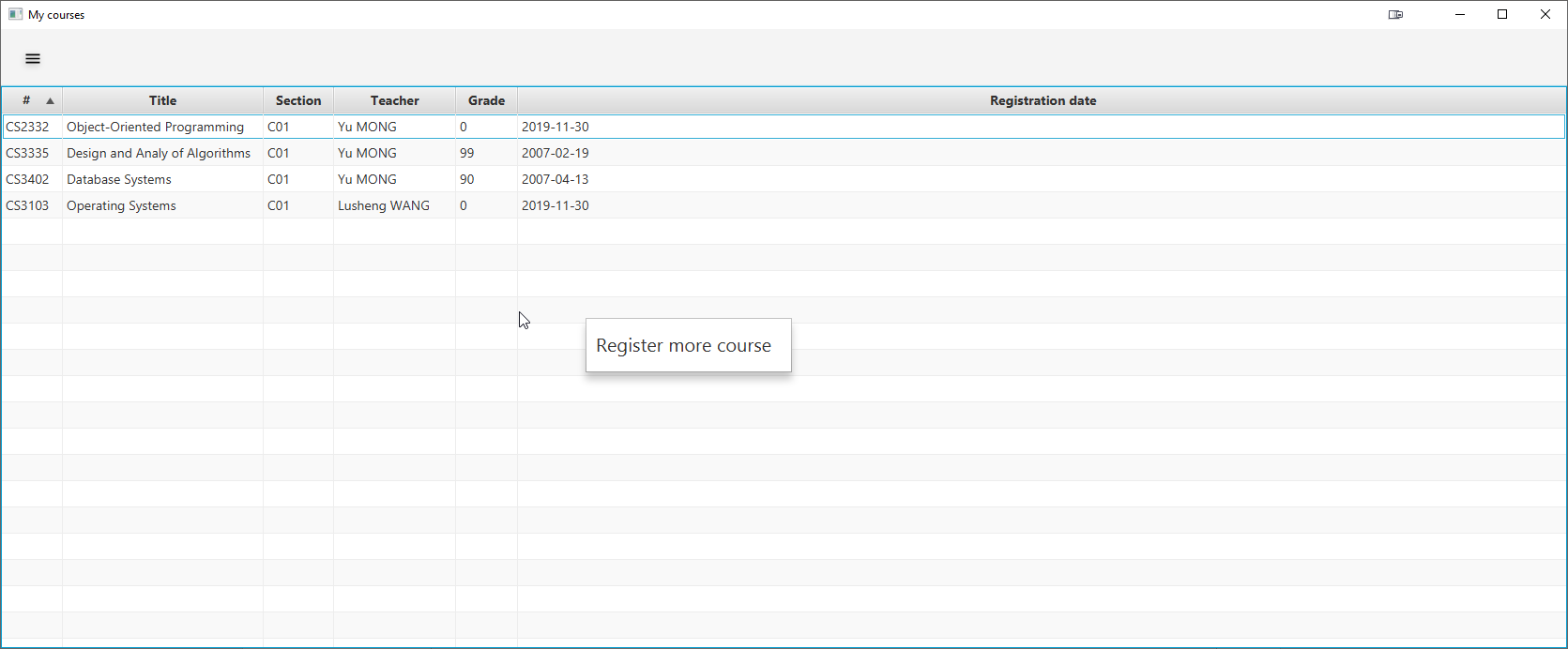
You will need to supply both old and new password to continue.



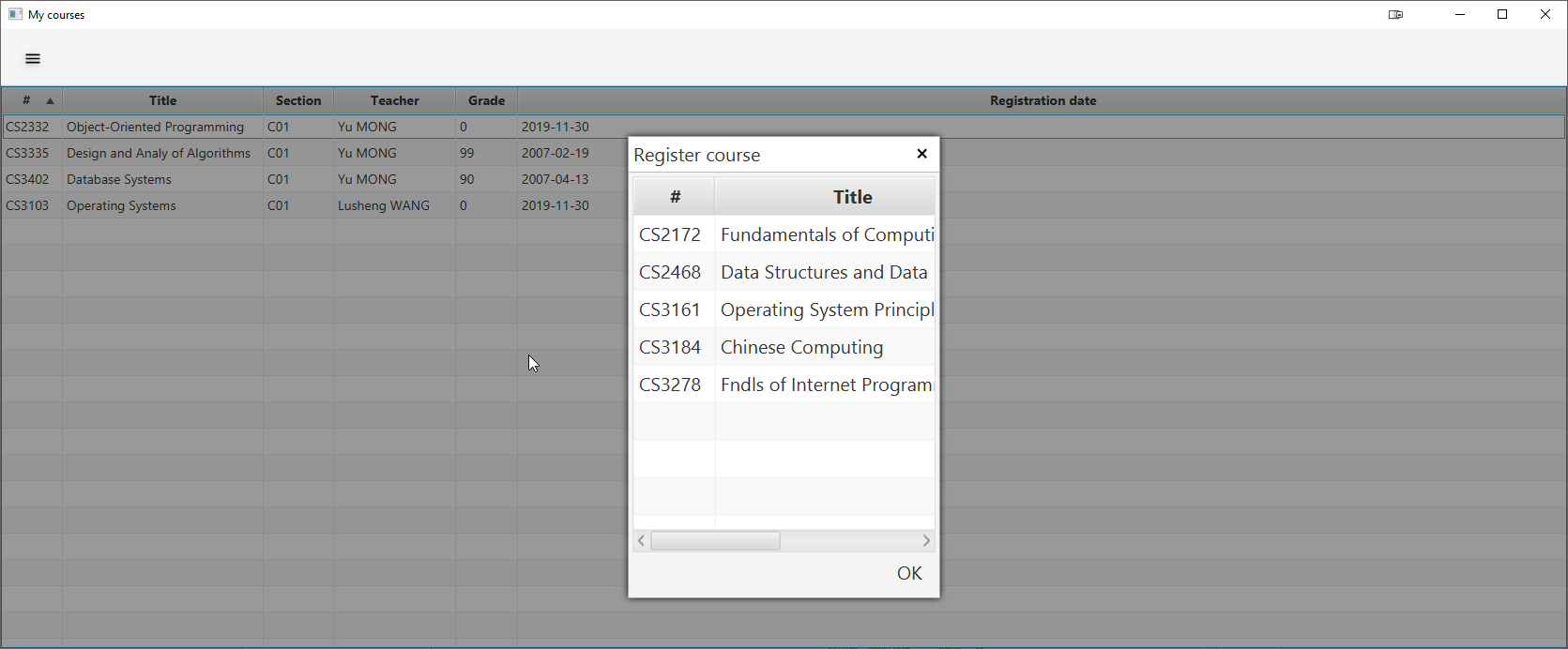
If you have provided a wrong old password, you will not be able to change it in order to guarantee security.



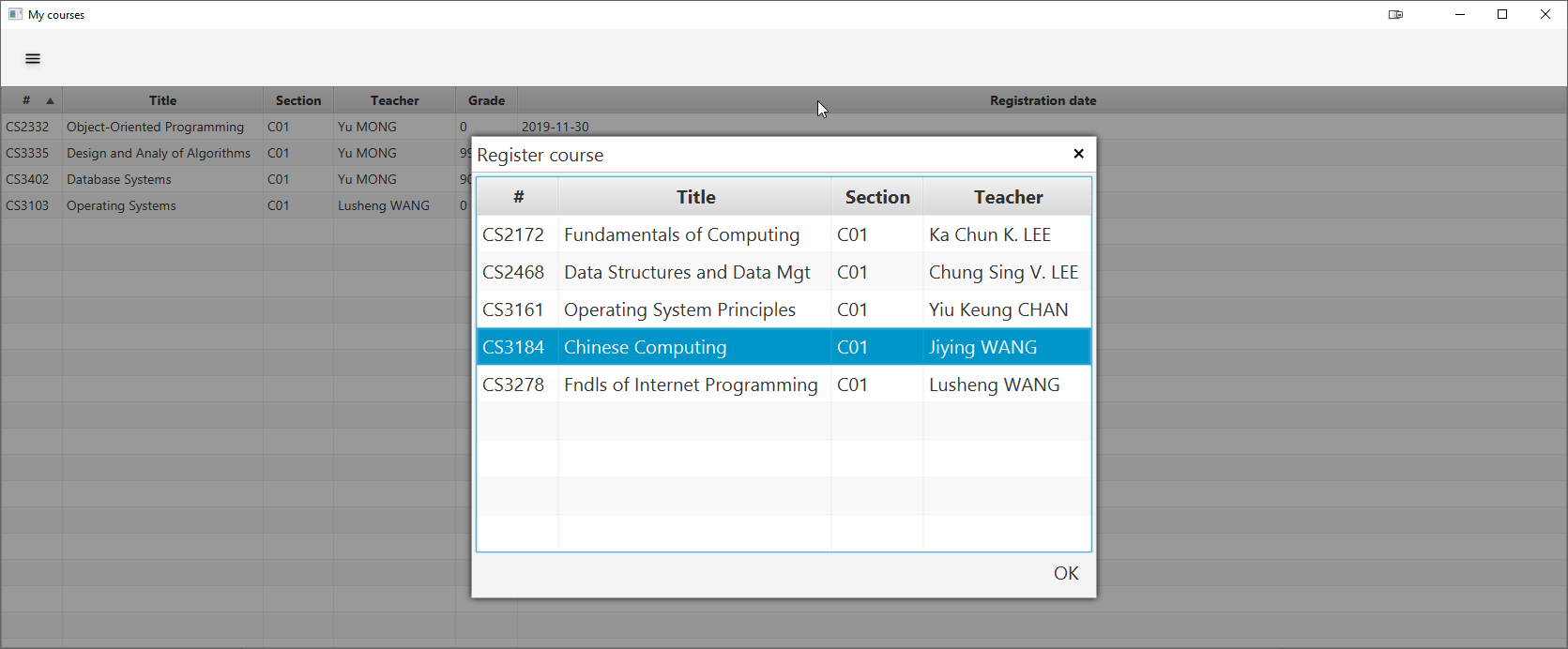
If you have supplied the right password, you will be able to change it successfully.

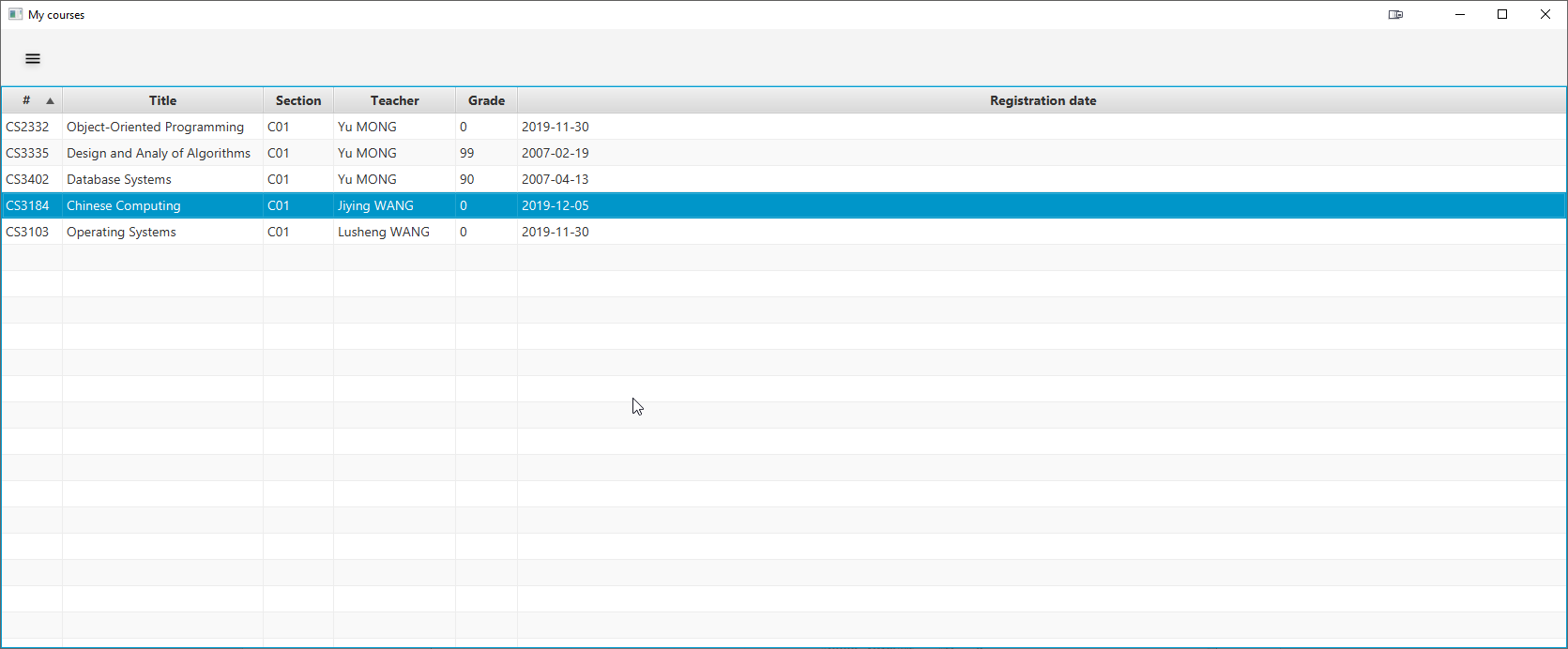


You can also access the register course modal by right-clicking on the table

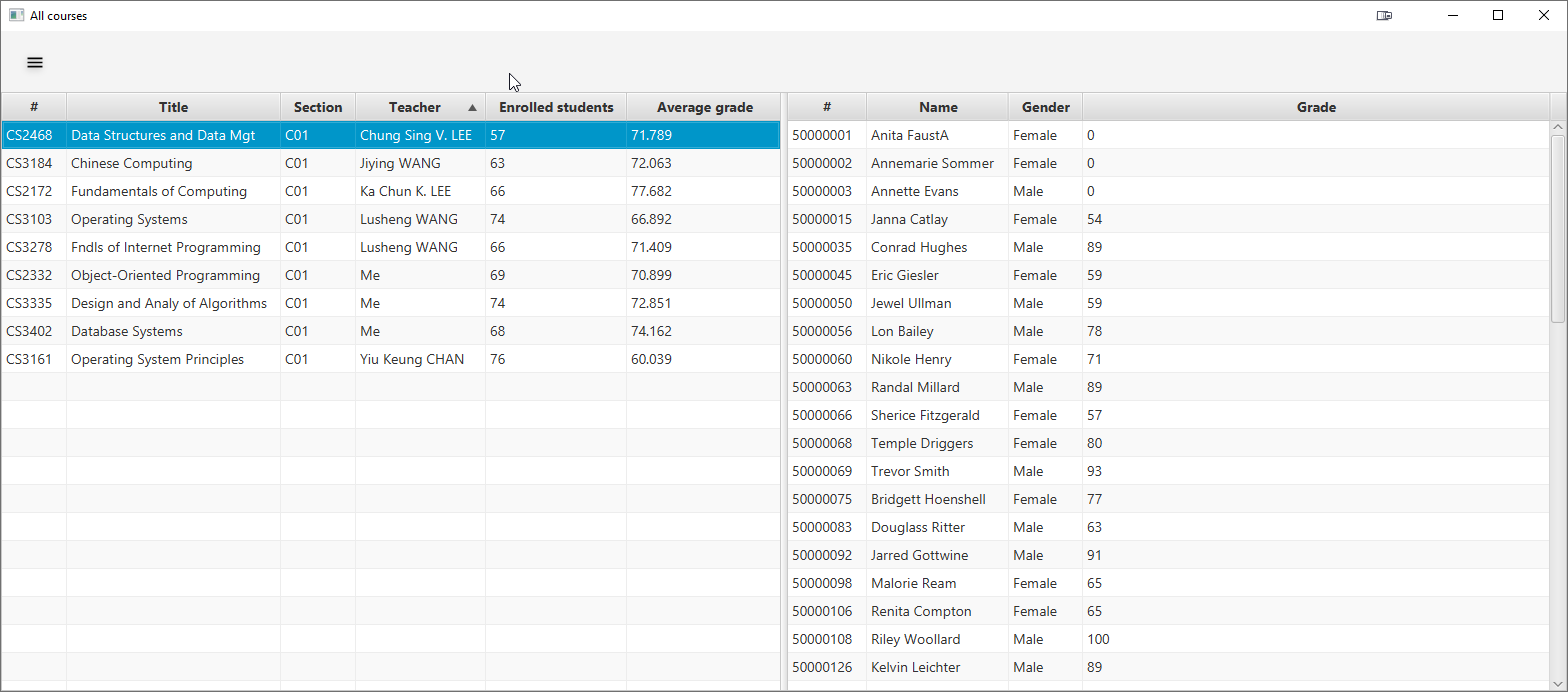


If you click on it, you will be greeted with a simple prompt to select the course you want to register. Only one at a time.

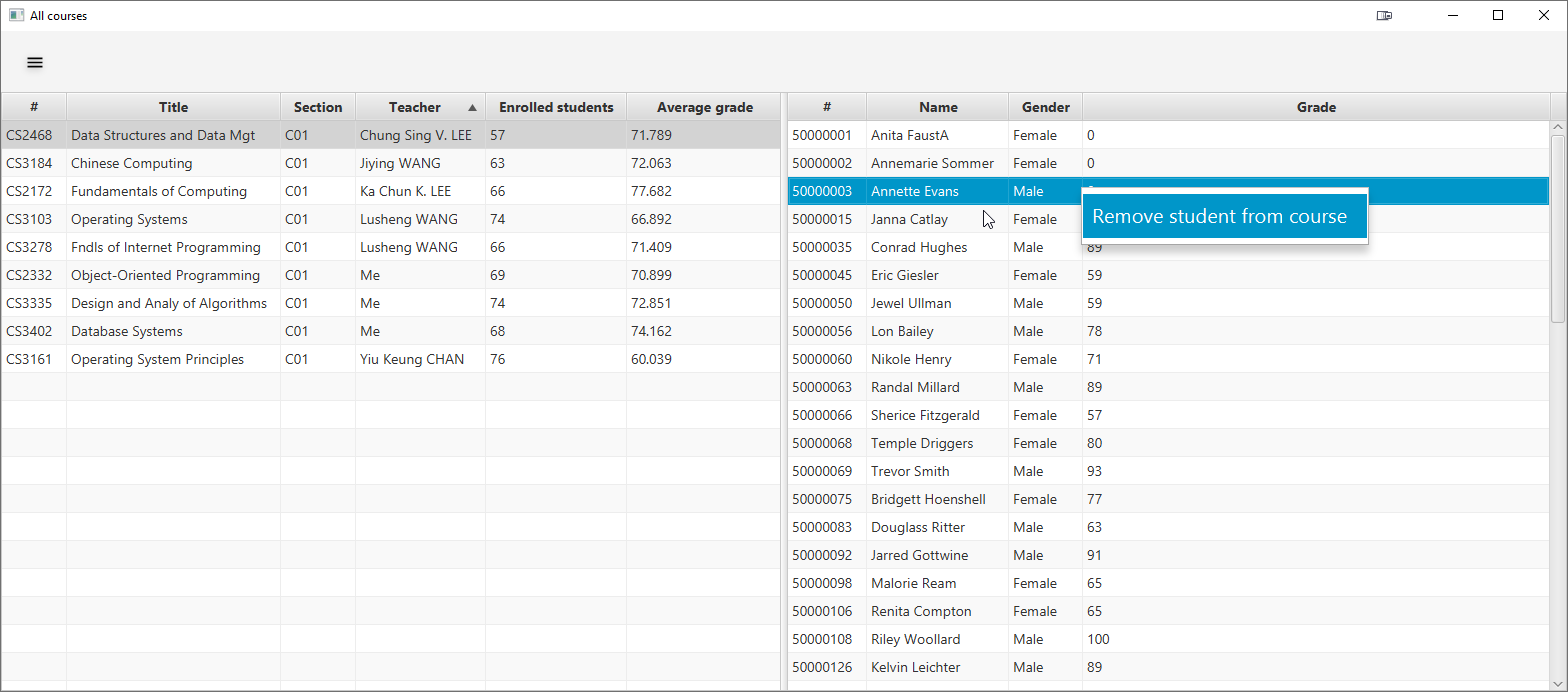


We have successfully registered a course.

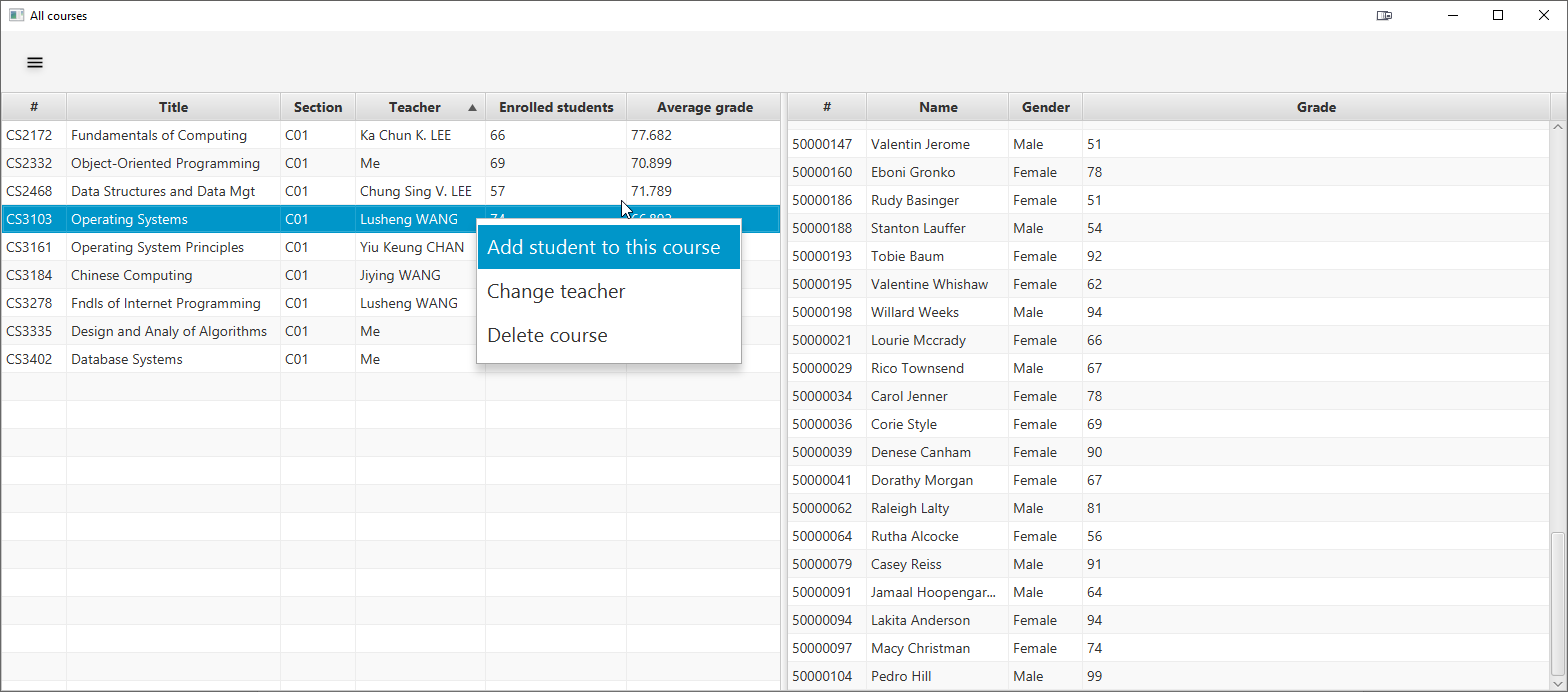
**Staff view**



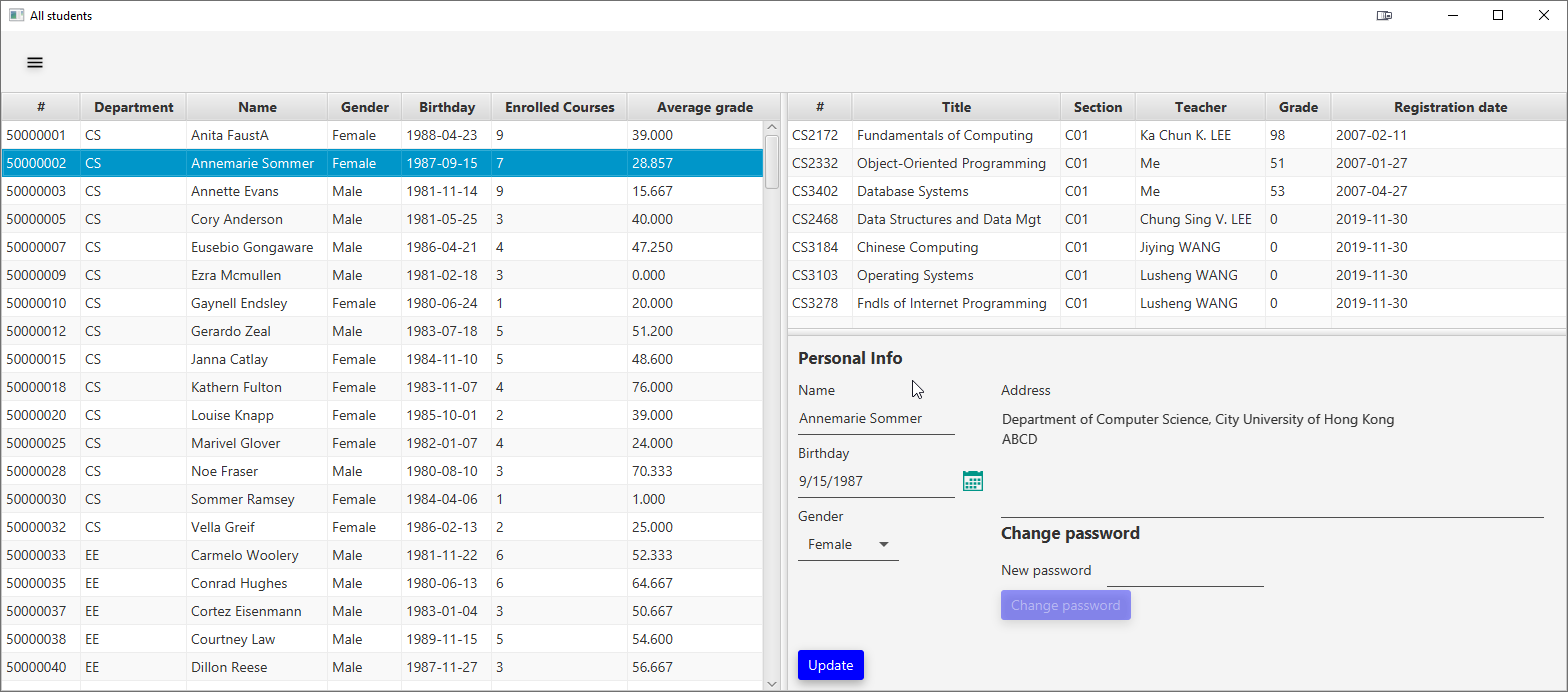
Here we have a display of all courses and the students associated with it.



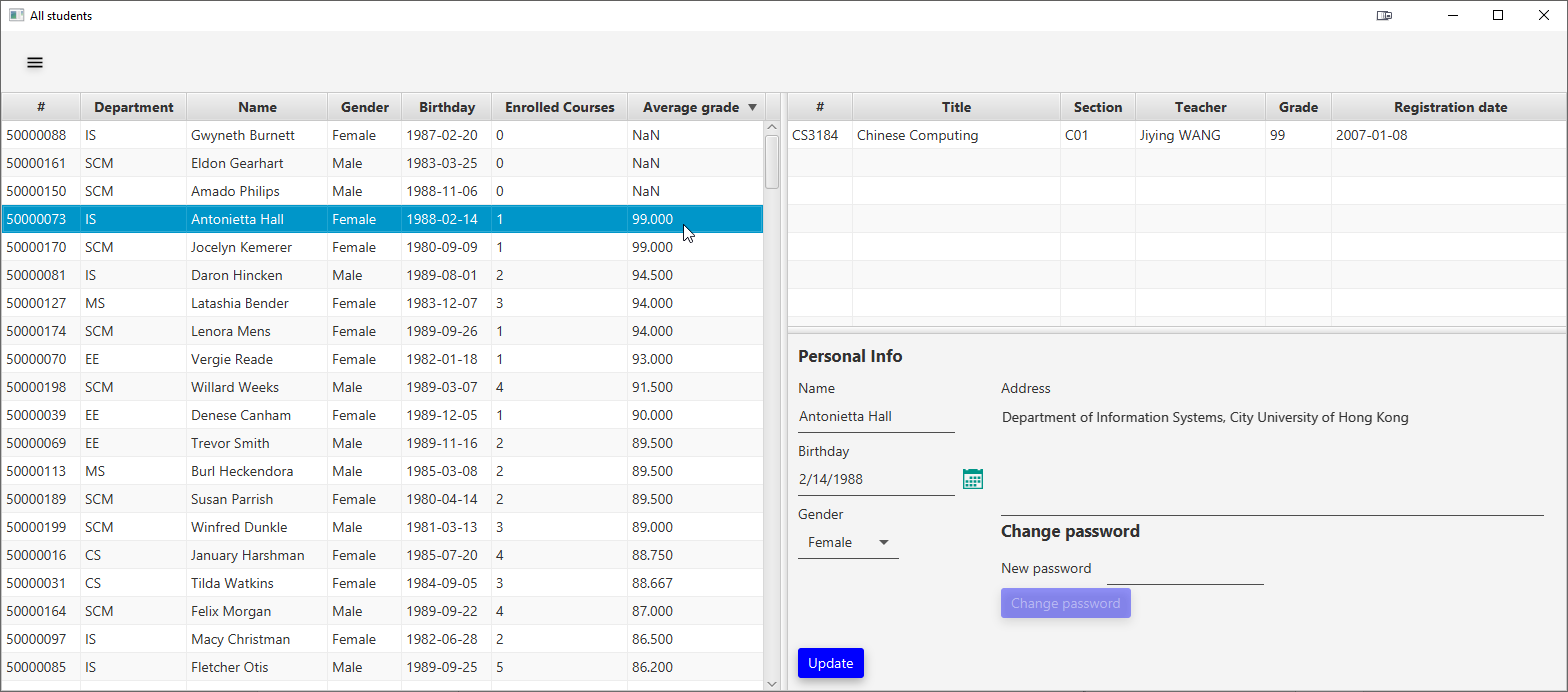
You can remove a student from the course by accessing the context menu.



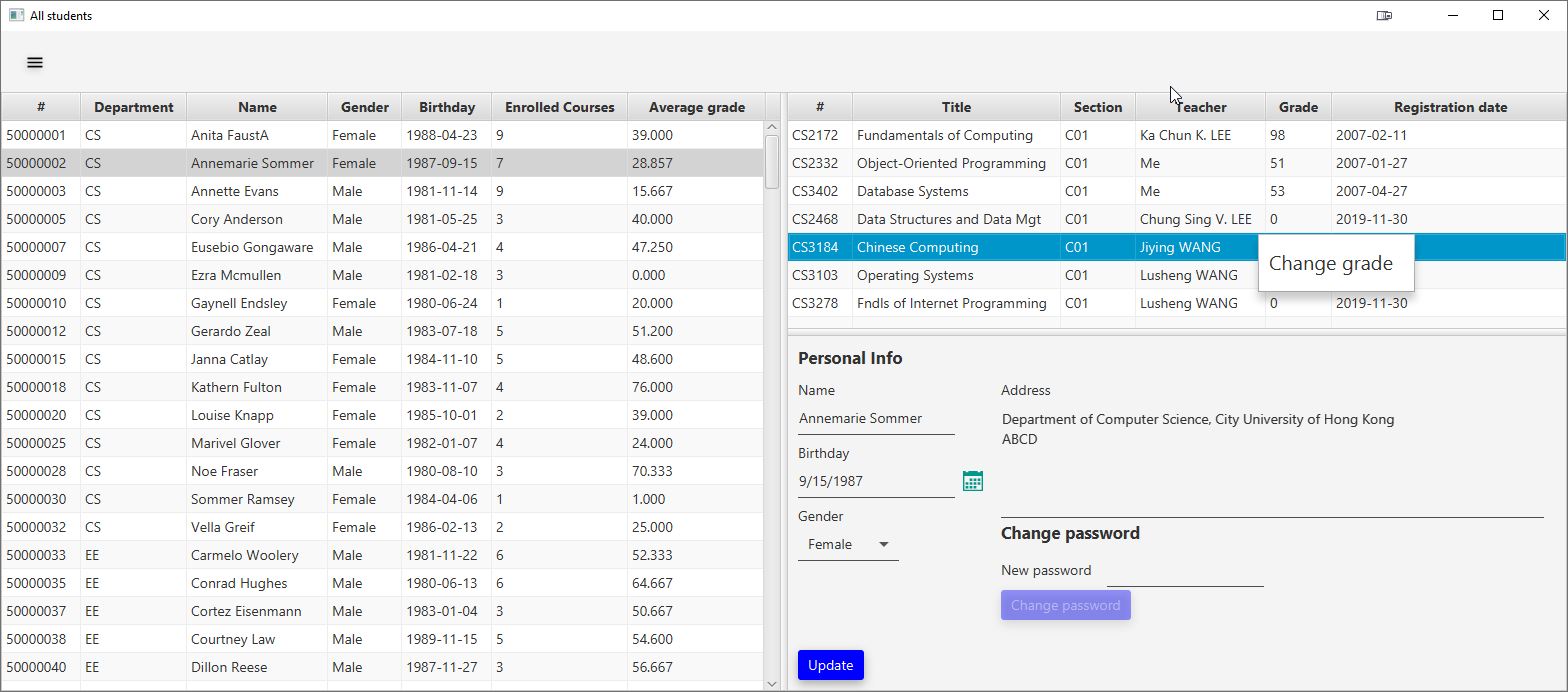
You can add more students, change the staff in charge or even remove the course entirely through the context menu (Associated enrolment entities will also be deleted).

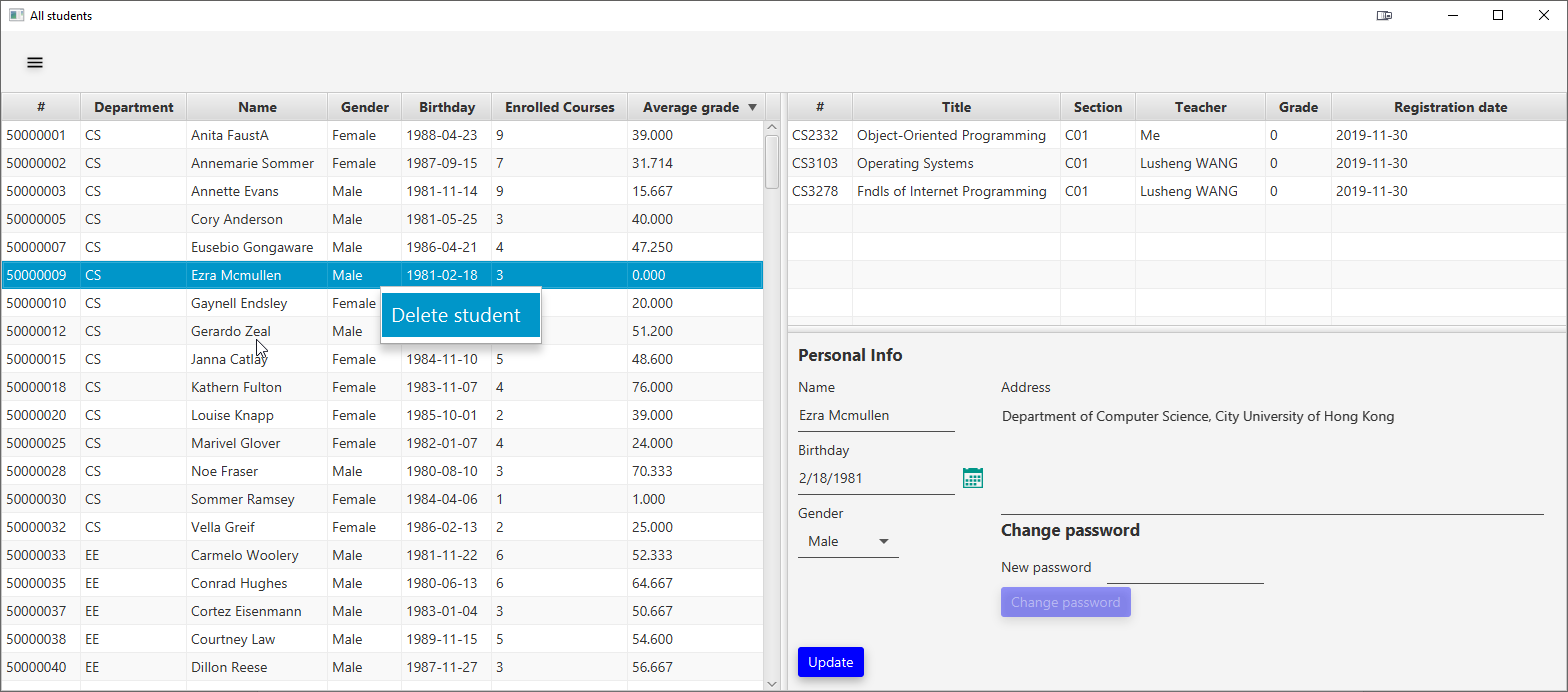


You can select the student row and see their registered course, or to modify their information, e.g. password.

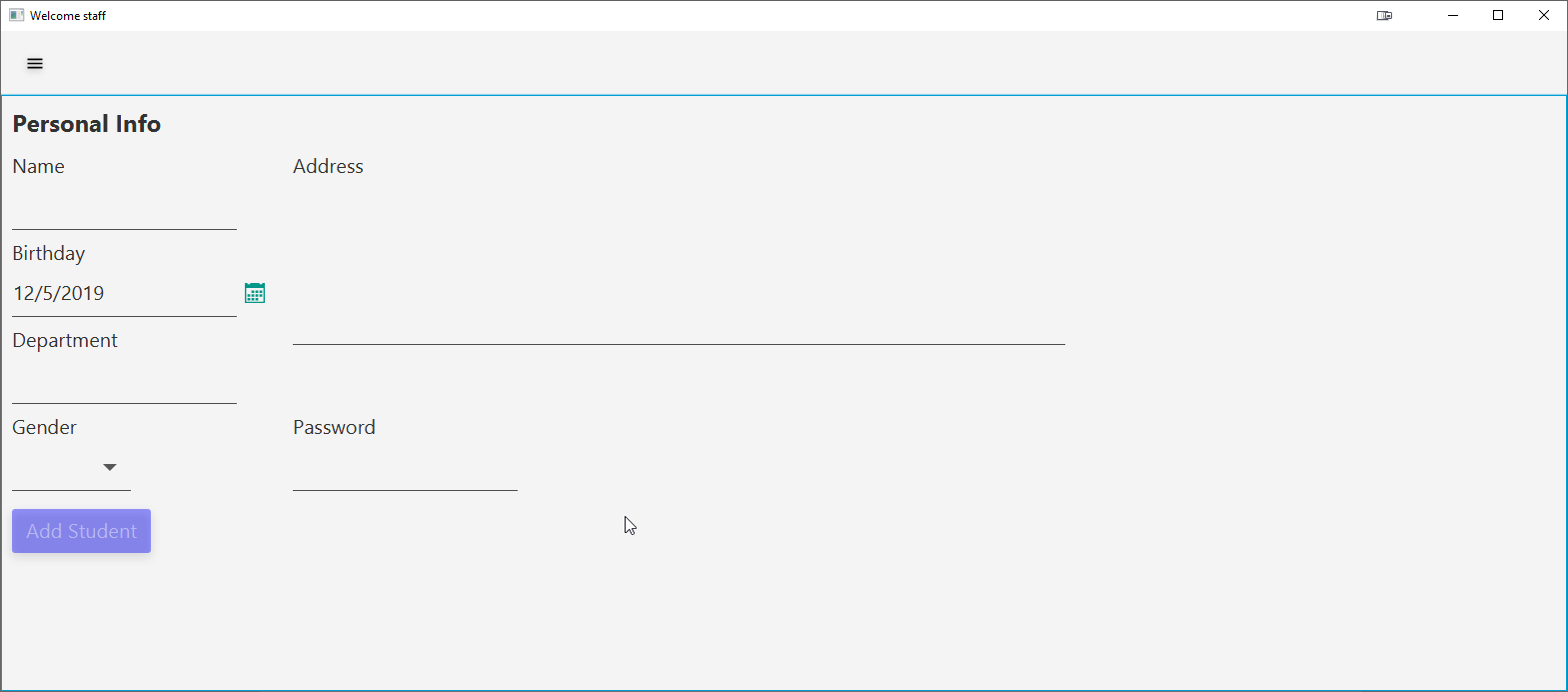


You can also sort the table by clicking the column headers.

You can change the grade of a student through the context menu.



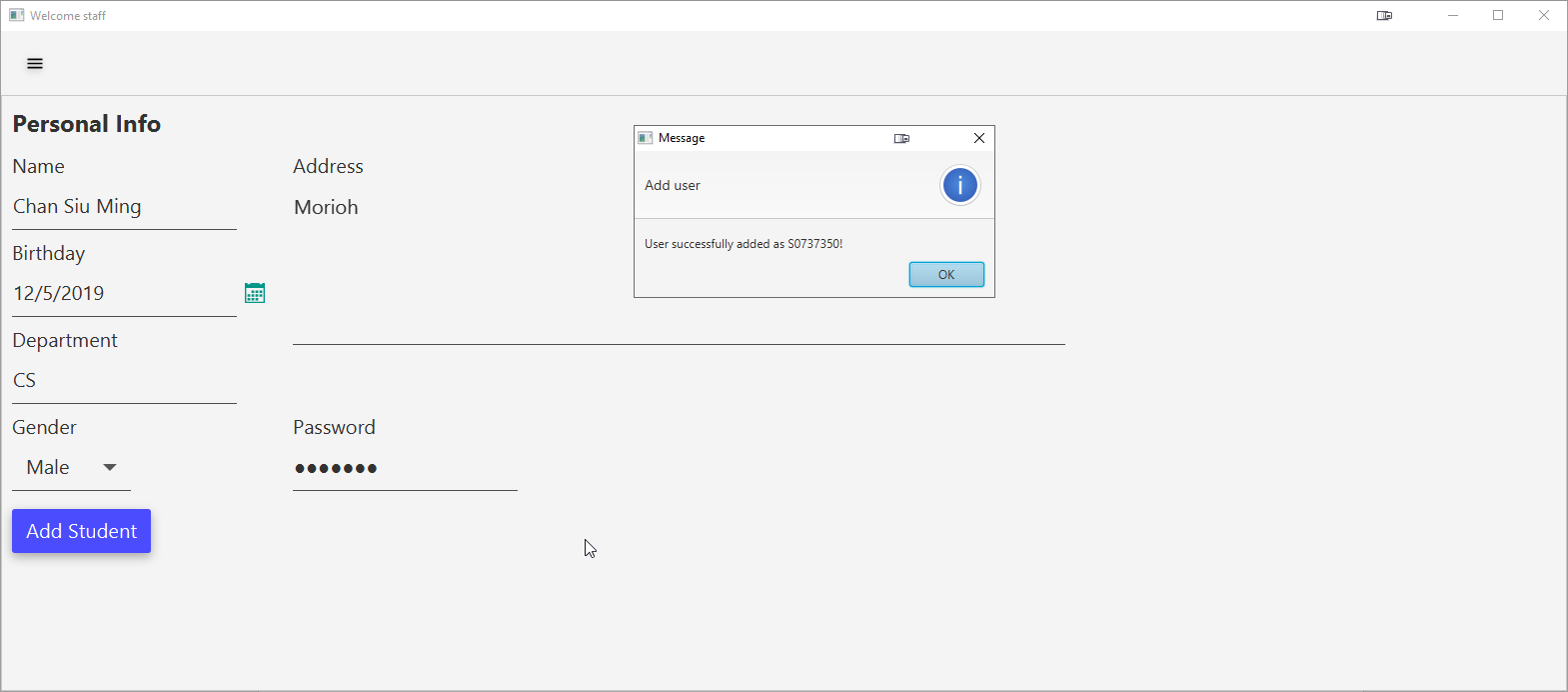
Or even remove a student from existence.

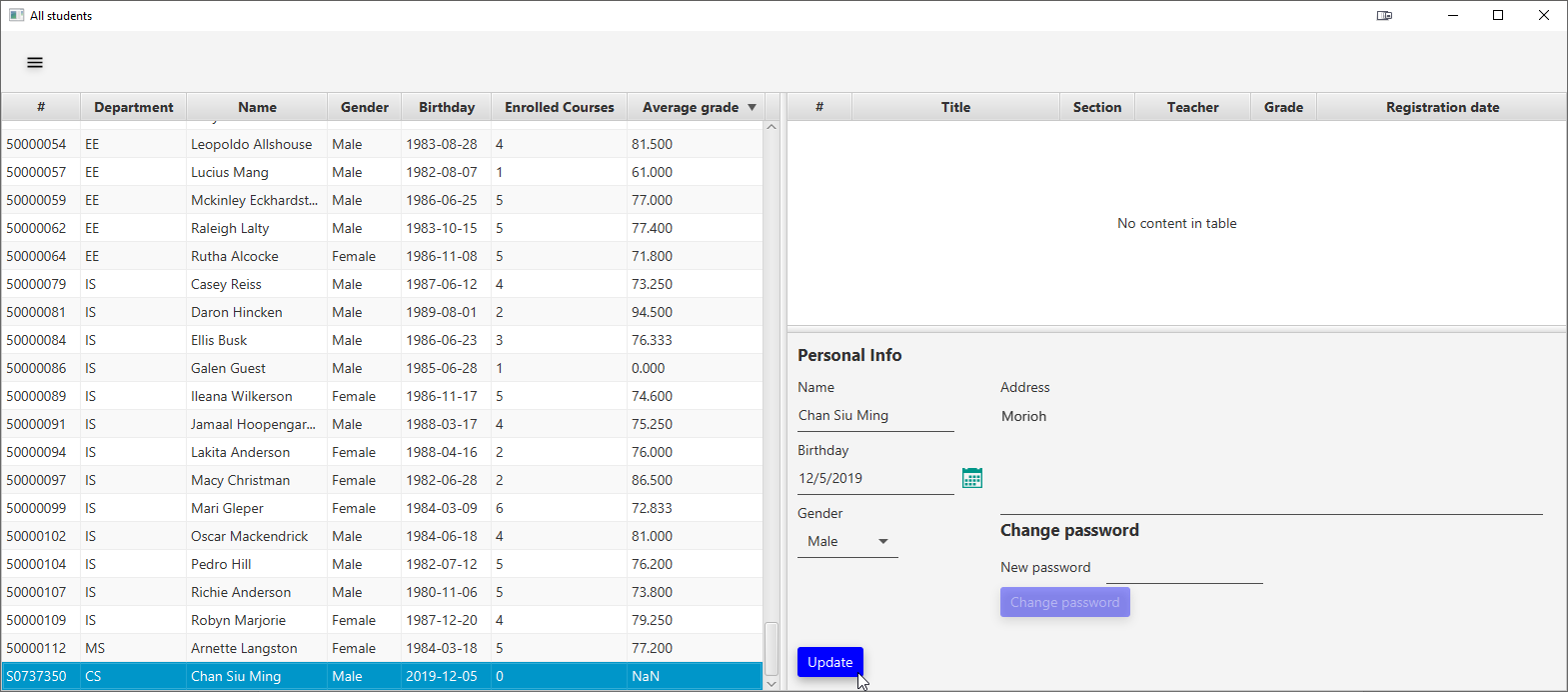


You also have a template to generate a student. However, the ID is randomly assigned.



We assigned random data. You can see the button is now activated. Only if you have filled in all information the button will activate.





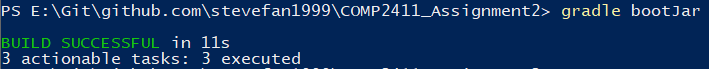
We have successfully added a student and is ready to give the credential to the people involved.

1. How to compile and reproduce everything

First, as we have already mentioned, Java 8 is a must, as there is no forward compatibility. Second, you will need to set up Gradle. Gradle is a Java build tool which is like Apache Maven but better. Thirdly, a CI option to compile the JAR through GitLab CI is available. Continuous Integration (CI) is like an automated bot that does all the hard work for you like in a pipeline. You will have to download the JAR file from the artifacts the pipeline generated in `build/libs`.

If you want to build a jar manually on the local system instead, you might not have to install Gradle, as there is a wrapper available to use. Simply run `gradle bootJar` and Gradle will download all the dependencies, compile all Kotlin files to Java, and let Spring Boot package the dependencies.

If you do everything right, you will be greeted with this.



The JAR file will be generated in `<project root directory>/build/libs` as we previously mentioned.

1. Challenges

The obvious challenge is that Hibernate does not really fit with already existing tables constructed from SQL, as some table entries are hardly representable in Hibernate, e.g. composite key that requires an embedded entity to deal with. Also, there is an entity synchronization problem, as our persistent entity is not shared in only one places, you may have to access them in a list of one-to-many relations or vice versa, but if you update them somewhere in a thread, the changes are not reflected to other threads immediately.

Second, we have some problem in designing the UI. Although TornadoFX is convenient to use, the way it spawns a view is still too classical due to its ties with JavaFX, so we cannot have modern Model-View-Presenter (MVP), component-based designs like React/Vue we see in modern web development. There is also a data synchronization problem in the JavaFX properties that tries to provide Model-View-ViewModel (MVVM) bindings, but is also mixed with observer pattern, which makes it very confusing to use and follow.

Thirdly, we cannot debug the database efficiently due to maximum connection problem. We are greeted with the following error:

[61000][2391] ORA-02391: exceeded simultaneous SESSIONS\_PER\_USER limit.

That we cannot see if the data is really modified in the database on the fly, and we can only do that until the main program closed, i.e. all database connection held released.

Fourthly, we cannot use the database efficiently due to the infamous N+1 problem in ORM tools. N+1 problem is that we query records one by one in a one-to-many or many-to-many situation due to lazy loading. This creates a huge performance problem, so we are forced to use eager loading. Eager loading loads all relational data at once instead of one by one at the cost of memory.

Lastly, it is also very hard to convert the entities into real object-oriented classes, even if we have visualized our UML classes internally. For example, we don’t know what relation should be converted to use Collection<?>, a one-dimension collection, and what relation should be converted to use Map<?, ?>, an associative collection. We believe it is due to the difference between set theory and type/category theory and is beyond our control.

End