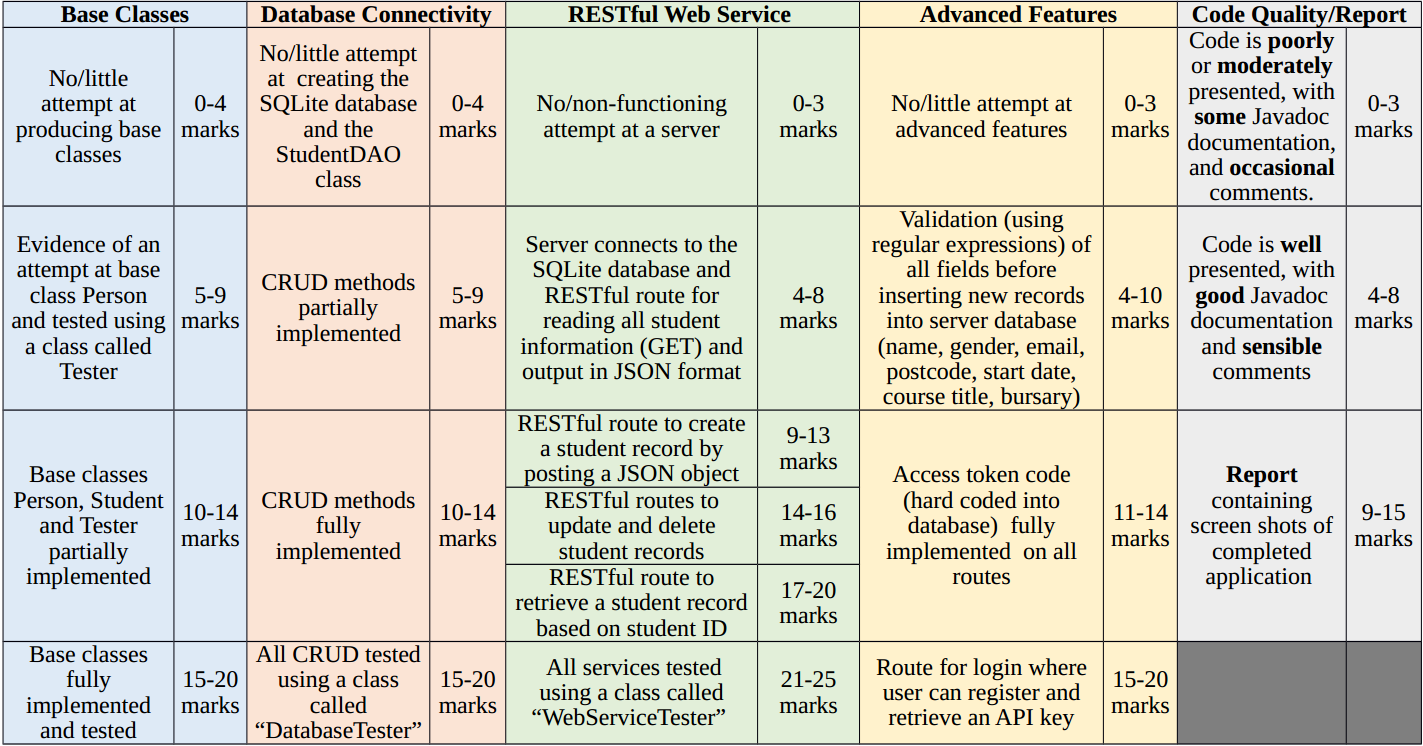
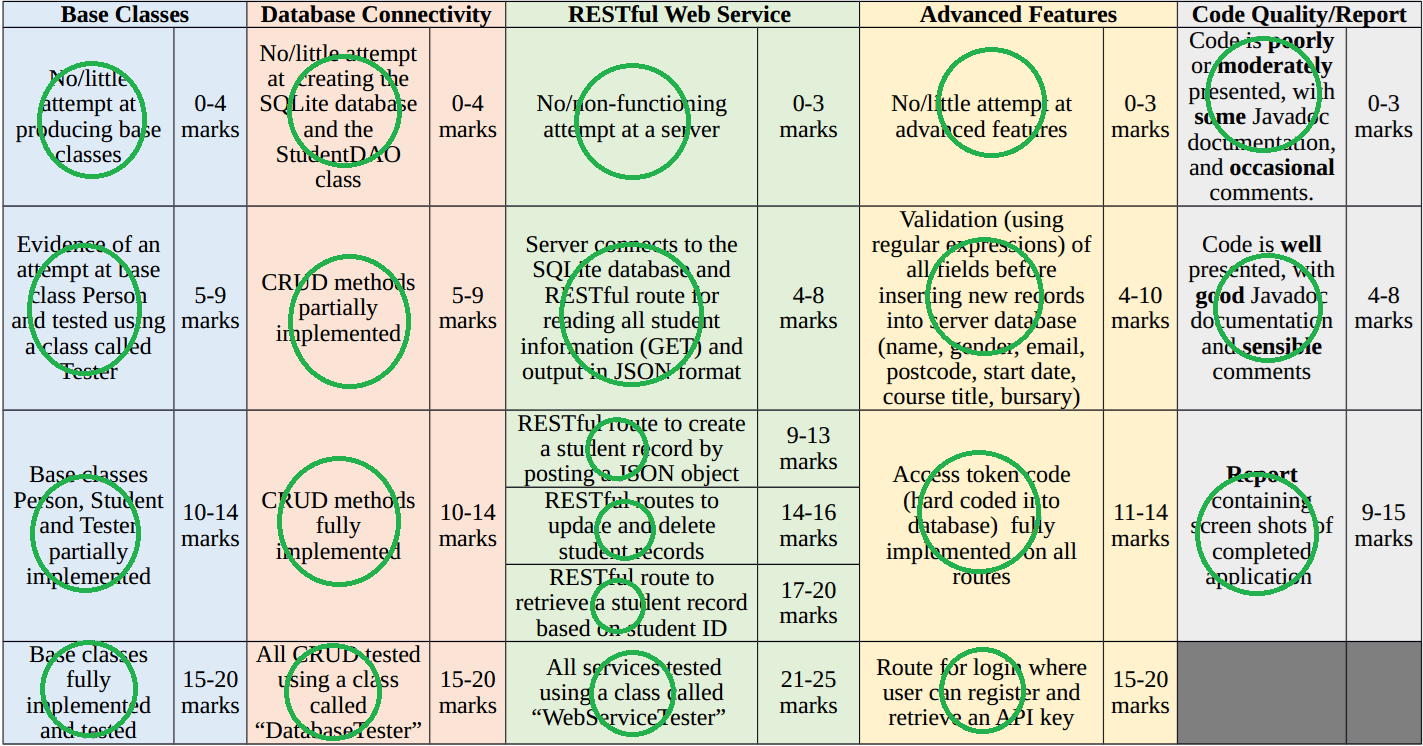
**Piotr Zadka 14056838**

Project report for Java Assignment

„Student RESTful Web Service Coursework”



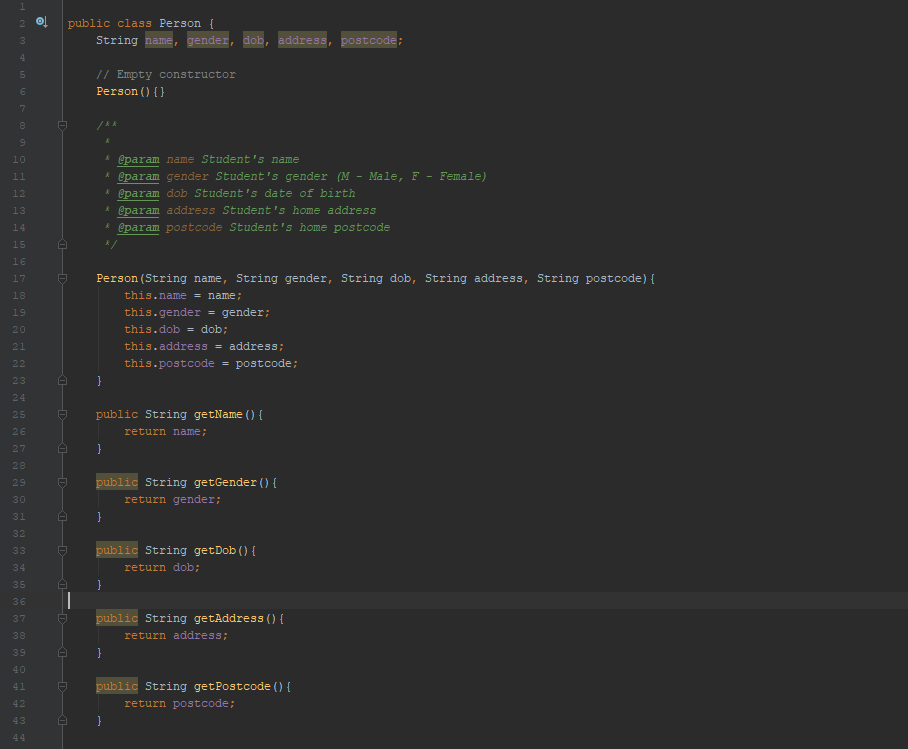
This report is to finalise entire Java project by providing screenshots and narrative explaining which features have been implemented and how they work. Above you can find mark scheme with requirements for the project. Below you can find same mark scheme, with circled in green, features that has been implemented in this project successfully. I will split in each main section end demonstrate functionality of each requirement.



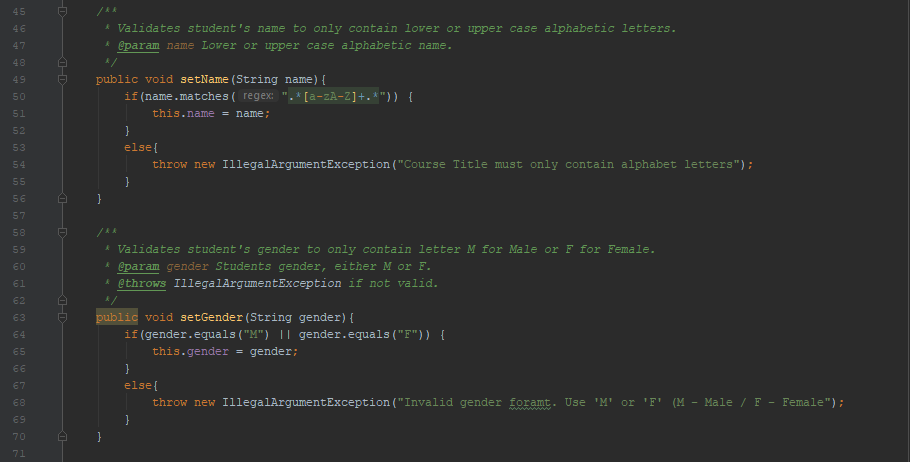
**Base Classes**

Person class:

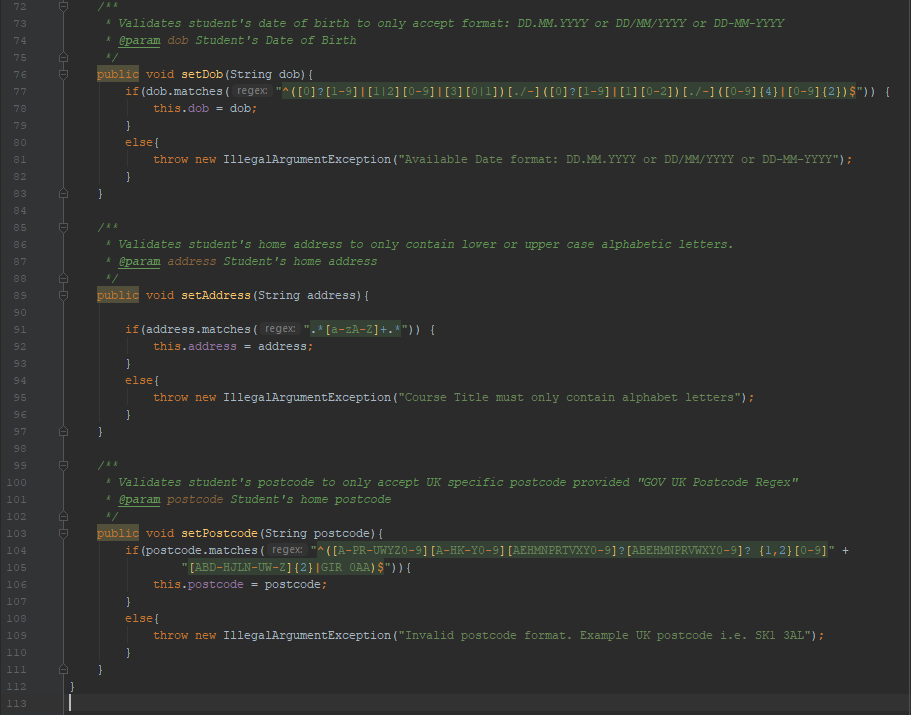
Below you can find entire Person class split in few screenshots with brief explanation what you can find on it.



Person class with empty constructor, constructor, and all getters.



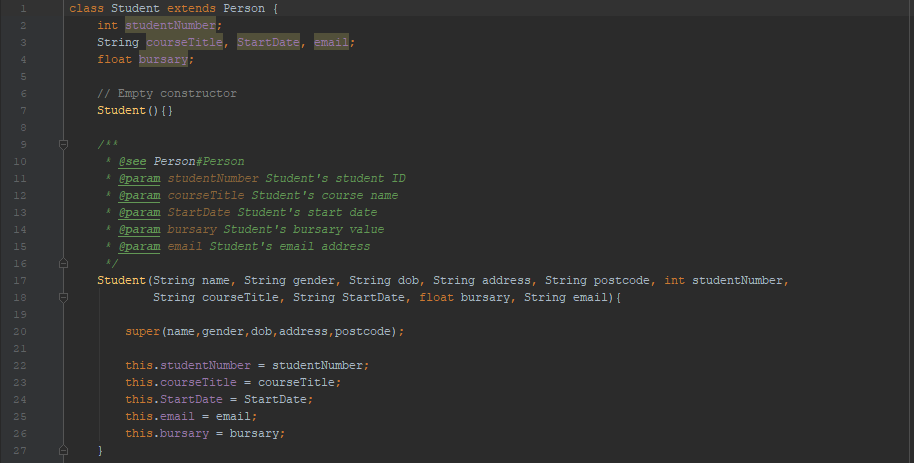
Person class with set Name, set Gender.



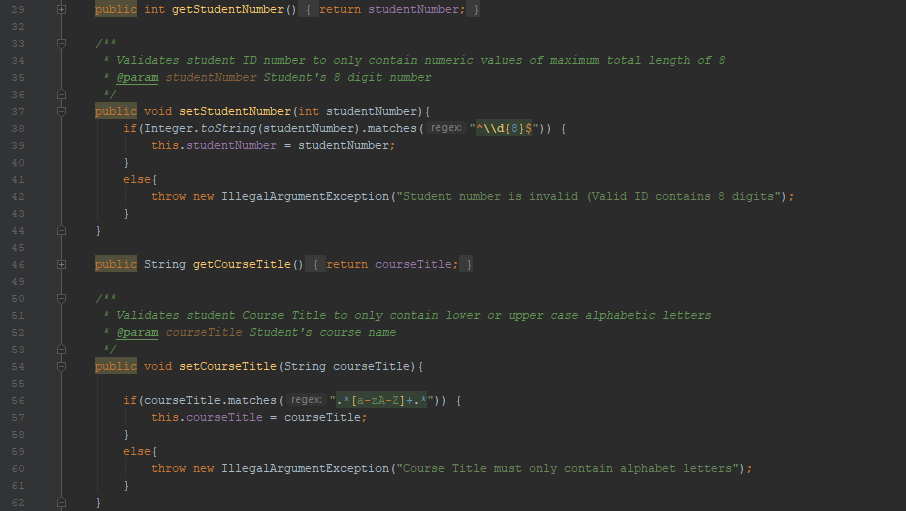
Person class with set Dob, set Address, set Postcode.

Student class:

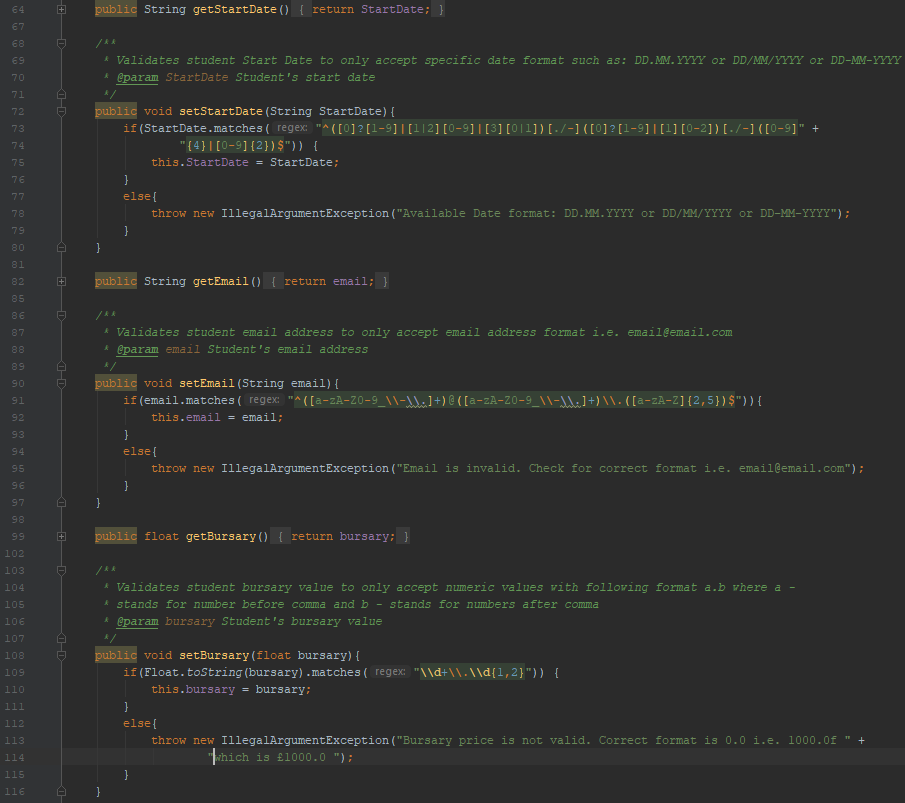
Below you can find entire Student class split in few screenshots with brief explanation what you can find on it.



Student class that extends Person class with empty constructor and regular constructor.



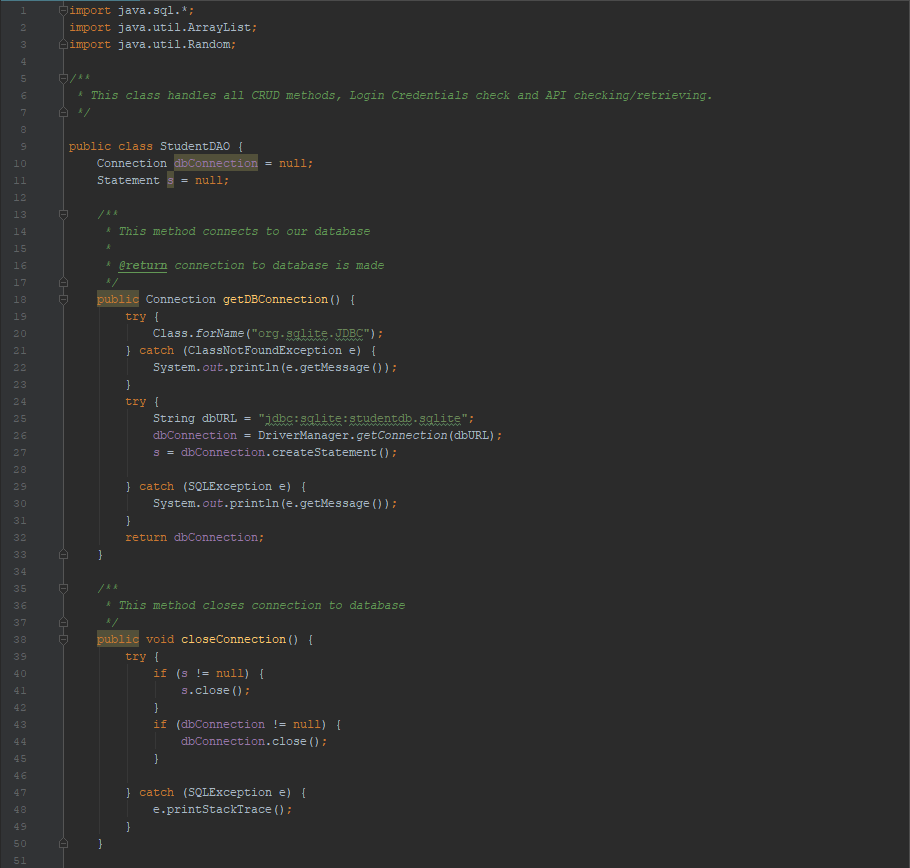
Student class with getters and setters for Student Number and Course Title.



Student class with getters and setters for Start Date, Email and Bursary.

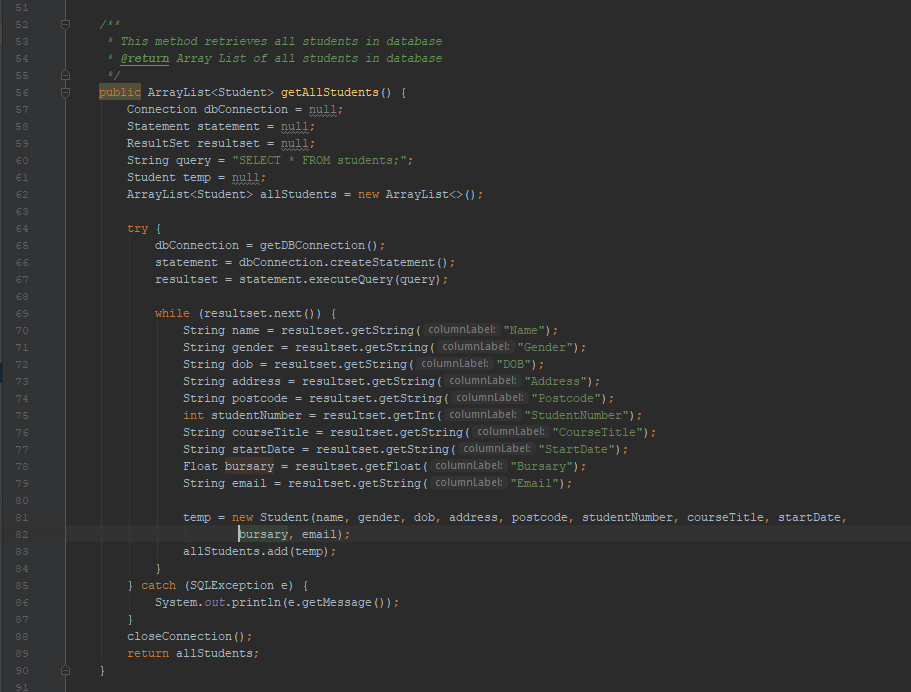
Student DAO class:

Below you can find entire Student Database Access Object Class split in few screenshots with brief explanation what you can find on it.



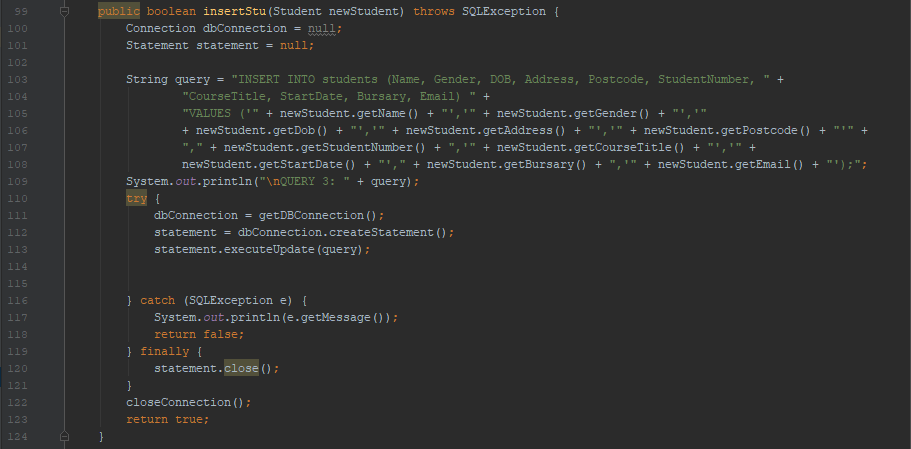
Student DAO getDBConnection, closeConnection methods

These are essentially methods to make or close connection to our database.



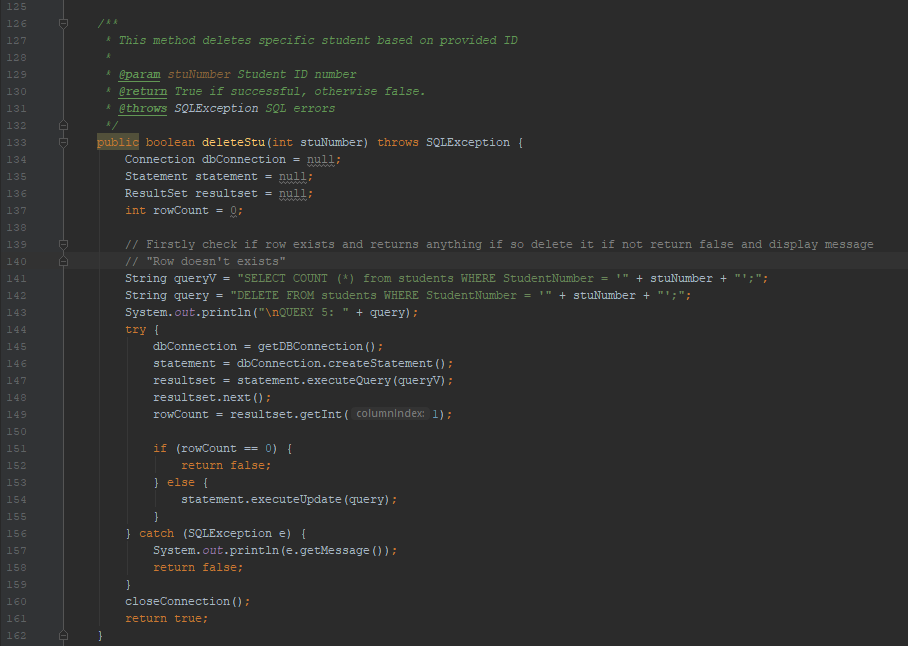
Student DAO getAllStudents method

This method is to retrieve all students from our database and return them as an array list of all students.



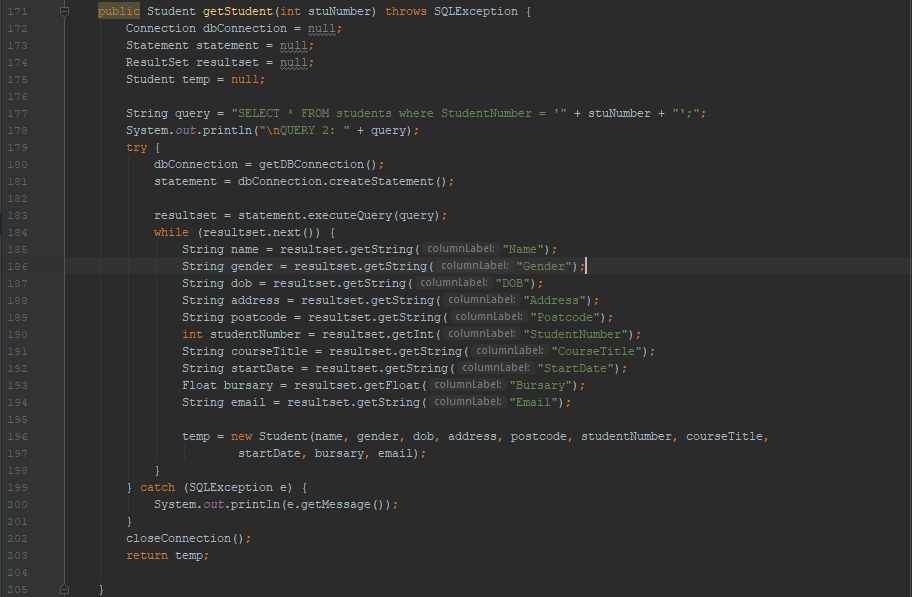
Student DAO insertStu method

This method handles inserting new student object into database and return true/false if fails/success to do so.



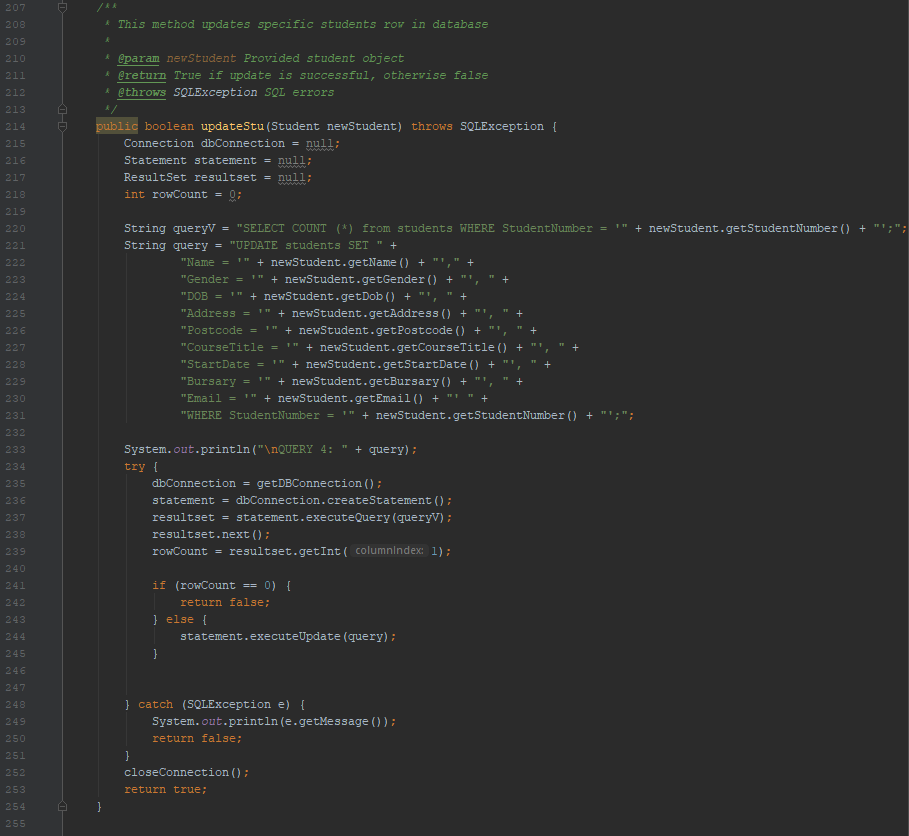
Student DAO deleteStu method.

This method handles student number and delete is from database. Returns true if deletion was successful or false if unsuccessful.



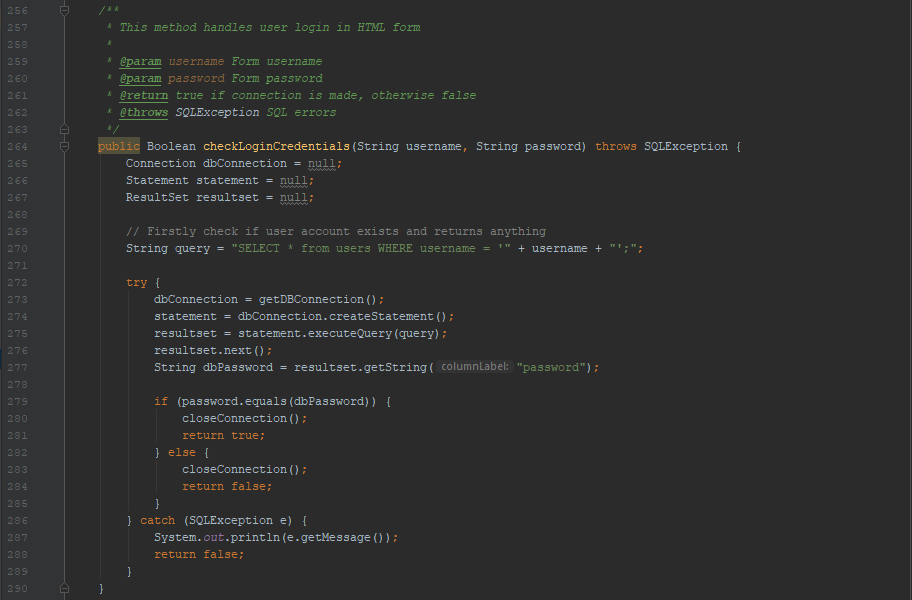
Student DAO getStudent

This method handles student number and retrieves specific student by that ID number. Return a student object with all student information inside.



Student DAO updateStu

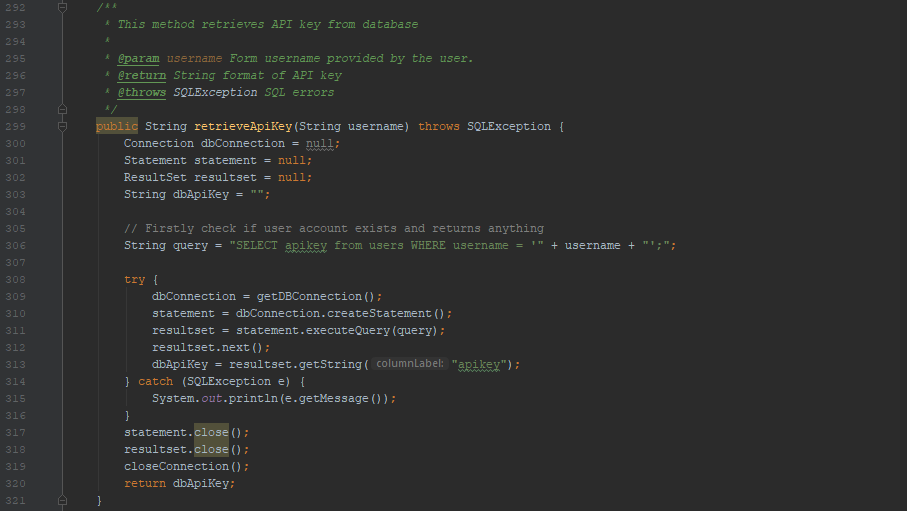
This method handles student object and update specific student by overwriting entire object into database with the information included in that student object.



Student DAO checkLoginCredentials

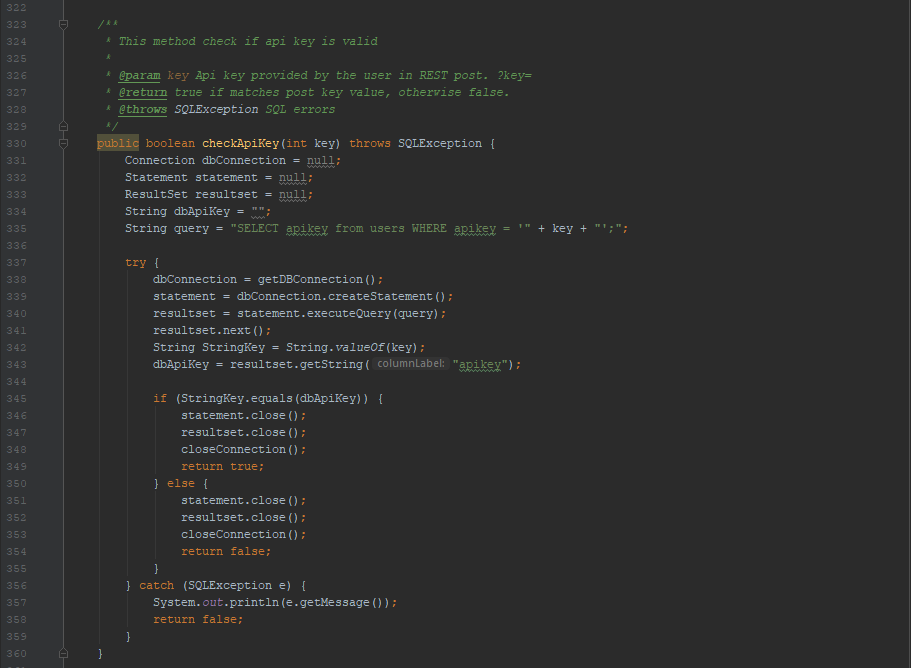
This method checks if login details provided in html form are correct.

To login and retrieve api key please use <http://localhost:8000/login>



Student DAO retrieveApiKey

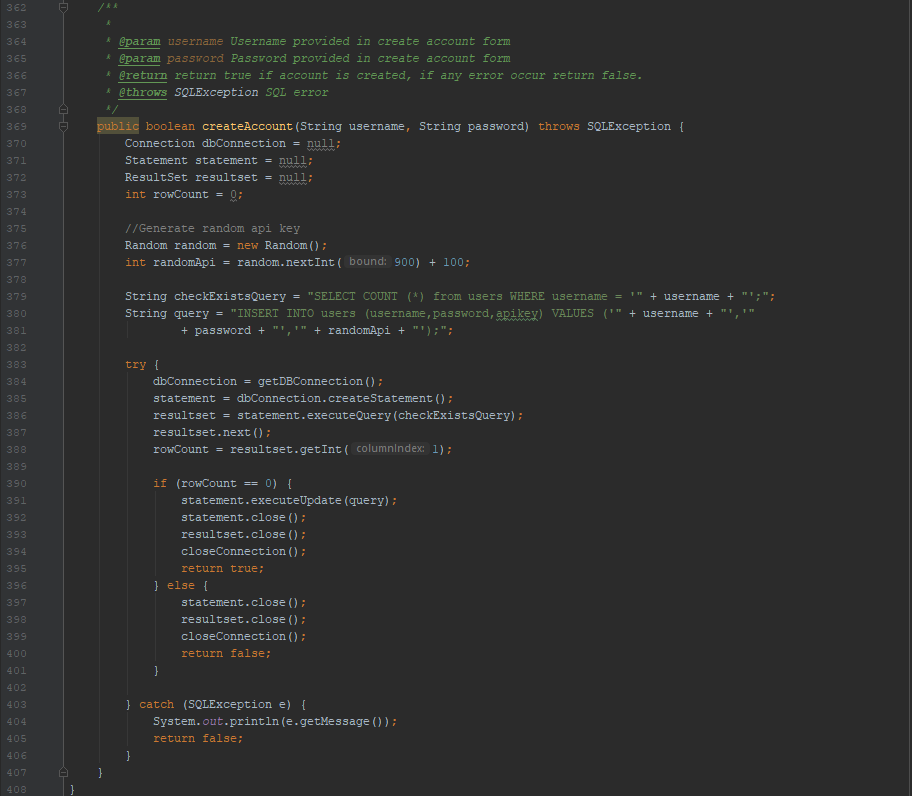
This method check provided username (provided in the login form above) exists in the database and if so retrieves and displays the api key for specific user.



Student DAO checkApiKey

This method checks if provided api key (in post method in REST client) is correct with the one stored in database.

In example user provides “?parameter1&parameter2&**key=123**” this method checks if the key=123 is equal to the one found in the database. If so returns true if no returns false.

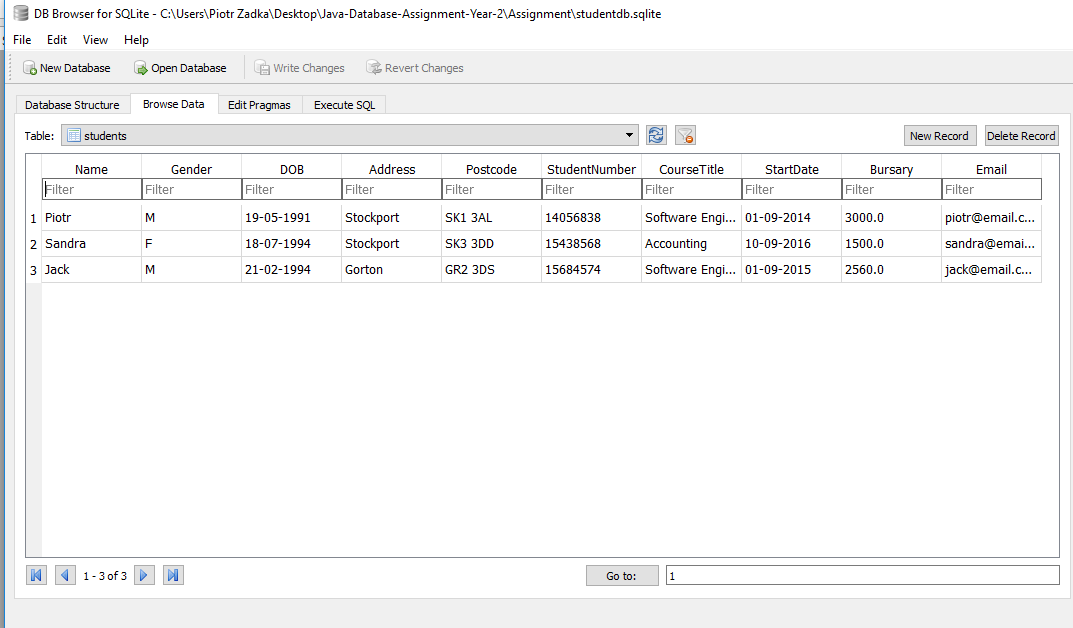


Student DAO createAccount

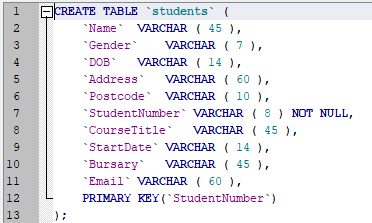
This method handles username and password information provided in HTML form and creates new account for a user by inserting his details into database and assigning him a random 3-digit API key that later can be retrieved by login.

**Database Connectivity**

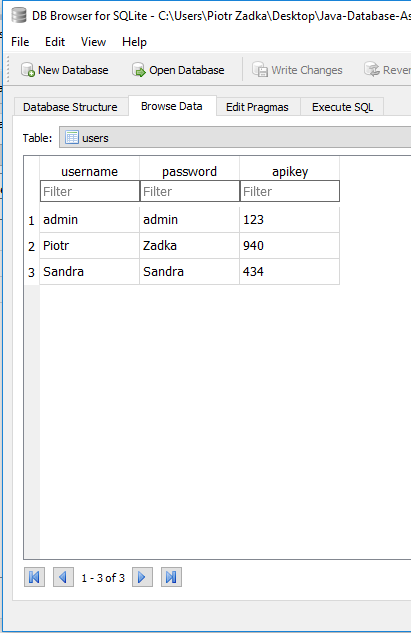
SQLite database:



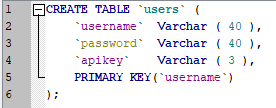
Students table with some data inside that is being modified and tested later in Database Tester class.



Students table schema.



Users table with some data inside that is being used for inserting new accounts, login users and finally retrieving a randomly generated API key.



Users table schema.

Student DAO class:

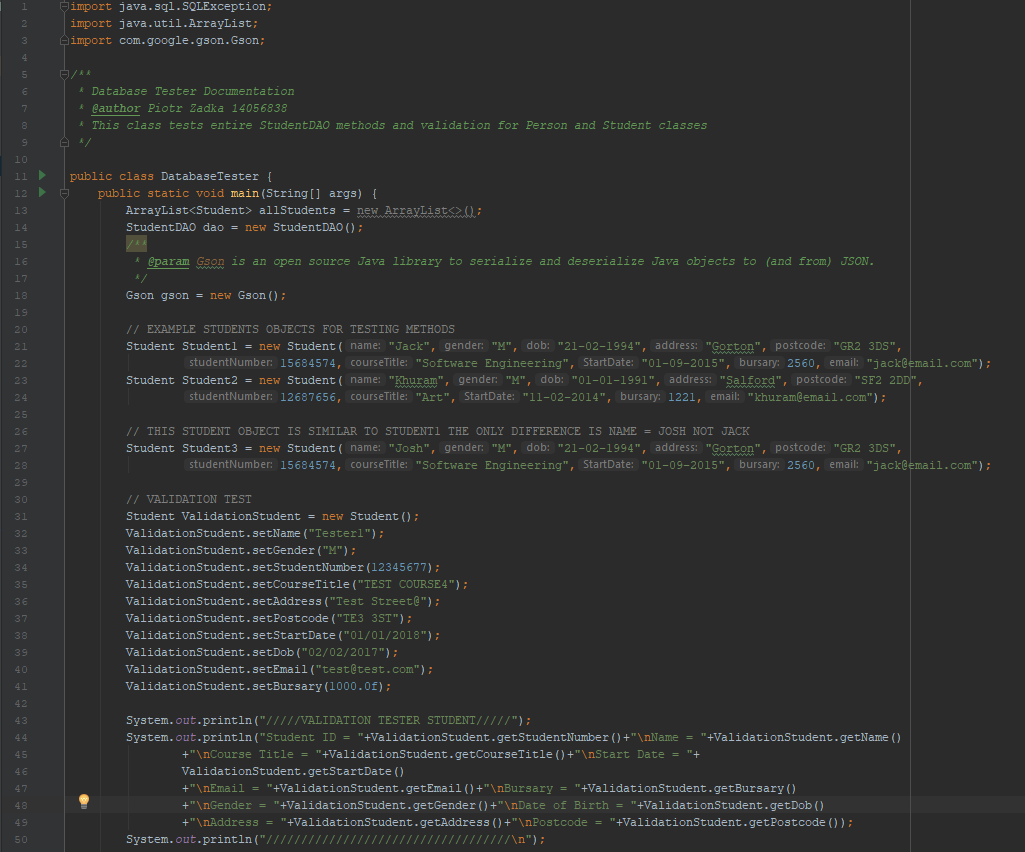
Students DAO class screenshots have been provided in previous section “Base Classes” under Student DAO part.

CRUD methods:

Each CRUD method has been provided in previous section “Base Classes” under Student DAO part. Each CRUD method has been presented, commented within program and briefly explained under each screenshot.

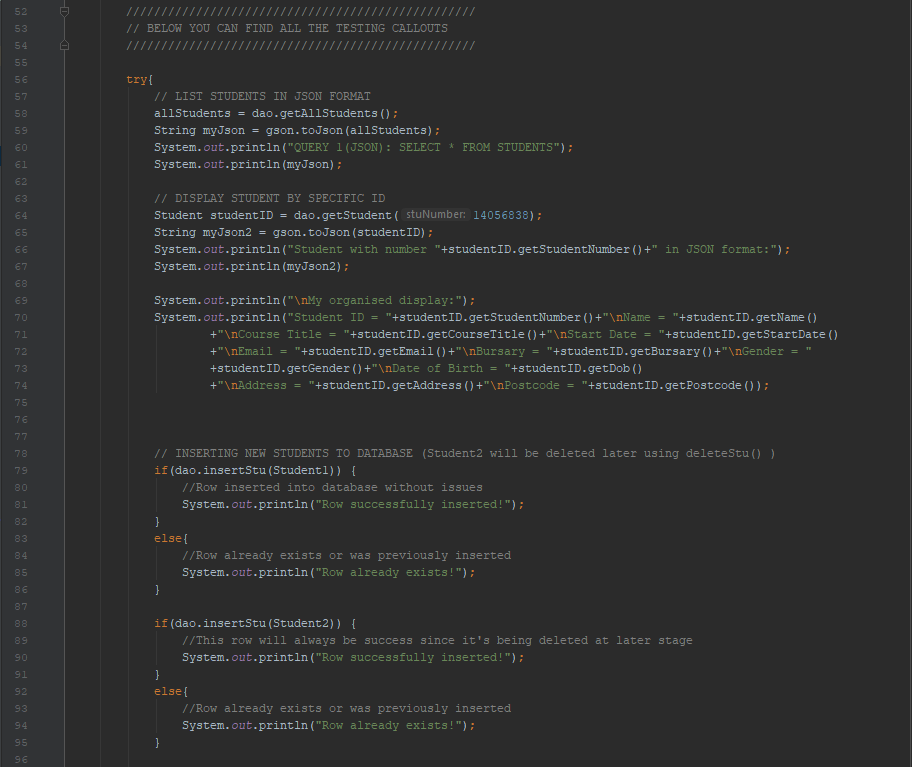
Database Tester:

This class tests each crud method. Additionally, it tests validation on a dummy test Student object.



-Database Tester class, in the screenshot above you can see three Student objects that were used in CRUD testing.

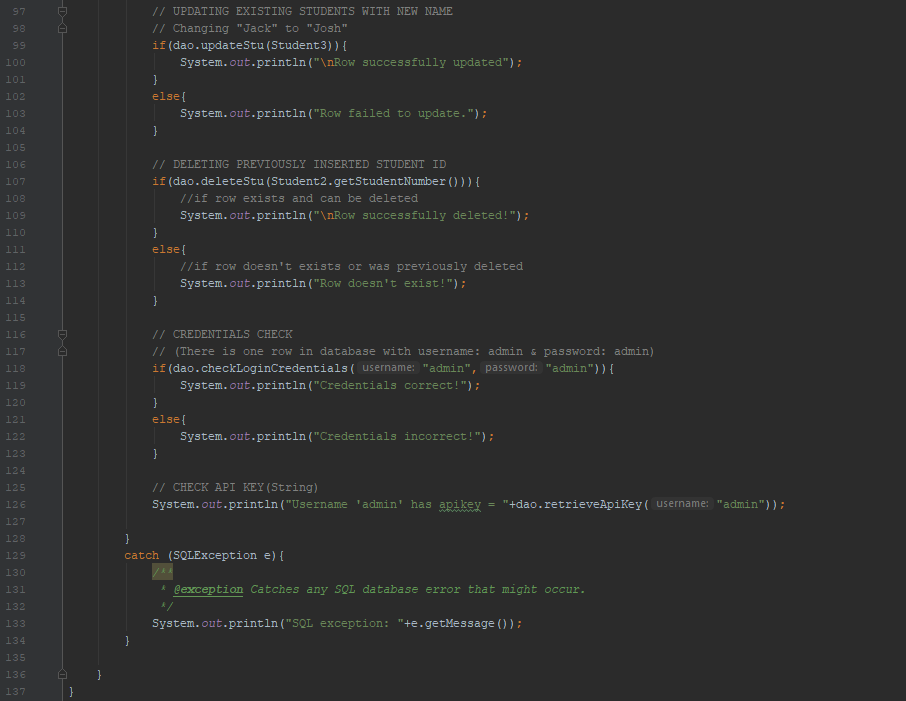
-Below Student Objects you can find a Validation Test which is a part where I create a new Object by inserting “VALID” data and checking for eventual errors.



-Listing all students using getAllStudents() method callout.

-Displaying specific student by ID using getStudent() method callout.

-Inserting new student using Student1 and Student 2 object created previously above.



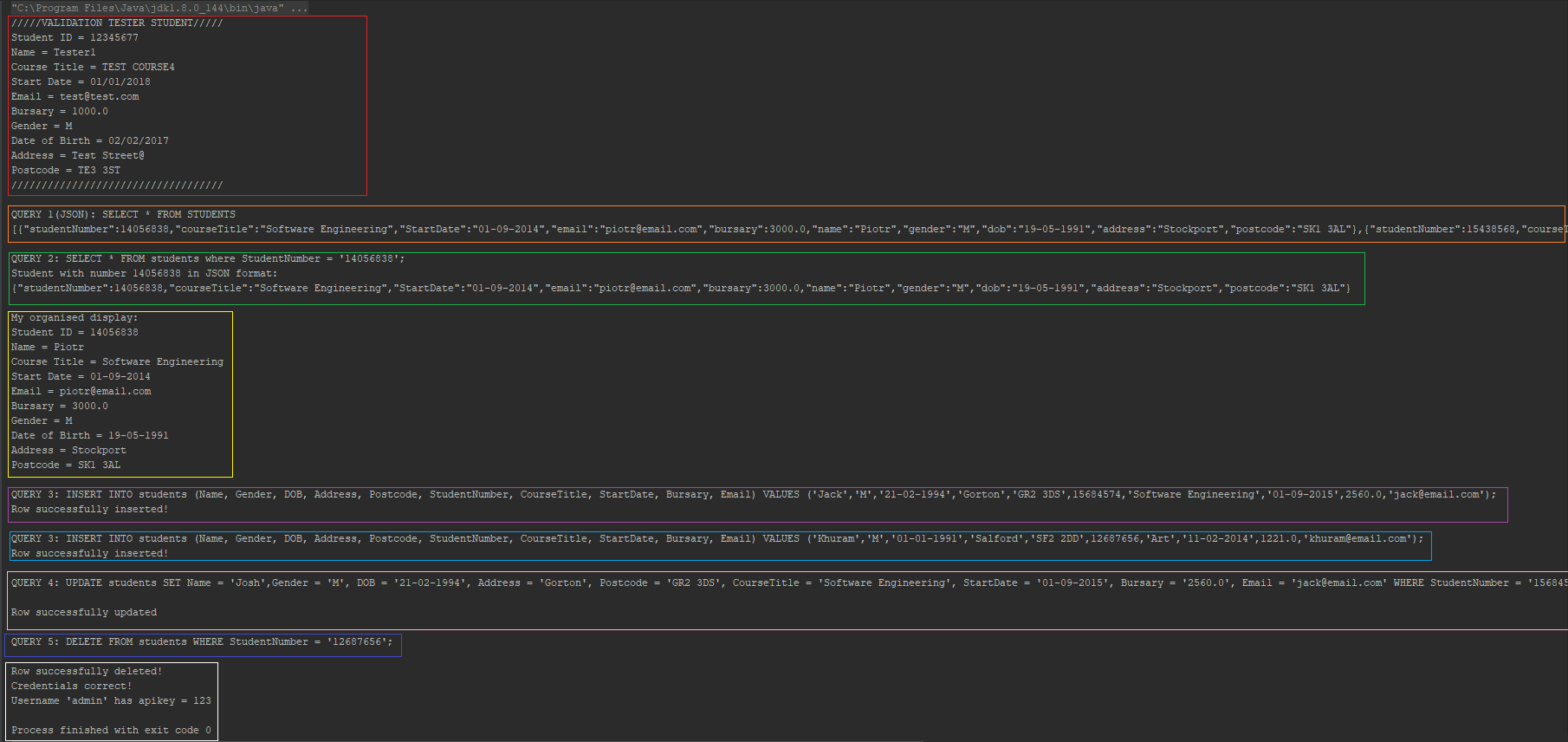
-Updating row in database by handling Student3 object into updateStu() method callout.

-Deleting student from database using Student2 object (For convenience, this object was previously inserted into database in insertStu() method and in this part, it’s being removed from the database)

-Checking credentials by providing administrator login details into checkLoginCredentials() method. If method find such user in the database then displays information that Credentials are Correct/Incorrect.

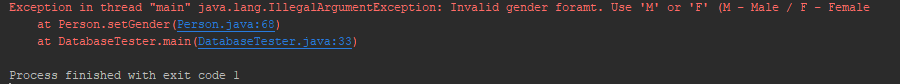
-Checking API key by providing username(in this case Admin) into method retrieveApiKey() and displaying the assigned apikey from database.

Console result by running Database Tester:



Tip: Control + Mouse Scroll Wheel to zoom in. (or simply run DatabaseTester.class in the program)

Red square: Result of adding a new student and validating the provided data. No errors returned means data is valid.



Example of an error that could occur if we would change gender from M/F to G or any other letter/number.

Orange square: Result of listing all students using getAllStudents(). Result is displayed as JSON object format.

Green square: Result of displaying one specific Student by ID (14056838). Result is displayed as JSON object format.

Yellow square: Result of organised display of previously displayed student in regular format.

Purple square: Result of inserting new student1 object into insertStu(). Result successful!

Blue square: Result of inserting new student2 object into insertStu(). Result successful!

Pink square: Result of updating a student row in database by passing a new Student Object with amend information inside. Result successful!

Dark blue square: Result of deleting a student(14056838) from the database using deleteStu(). Result “Row Deleted”

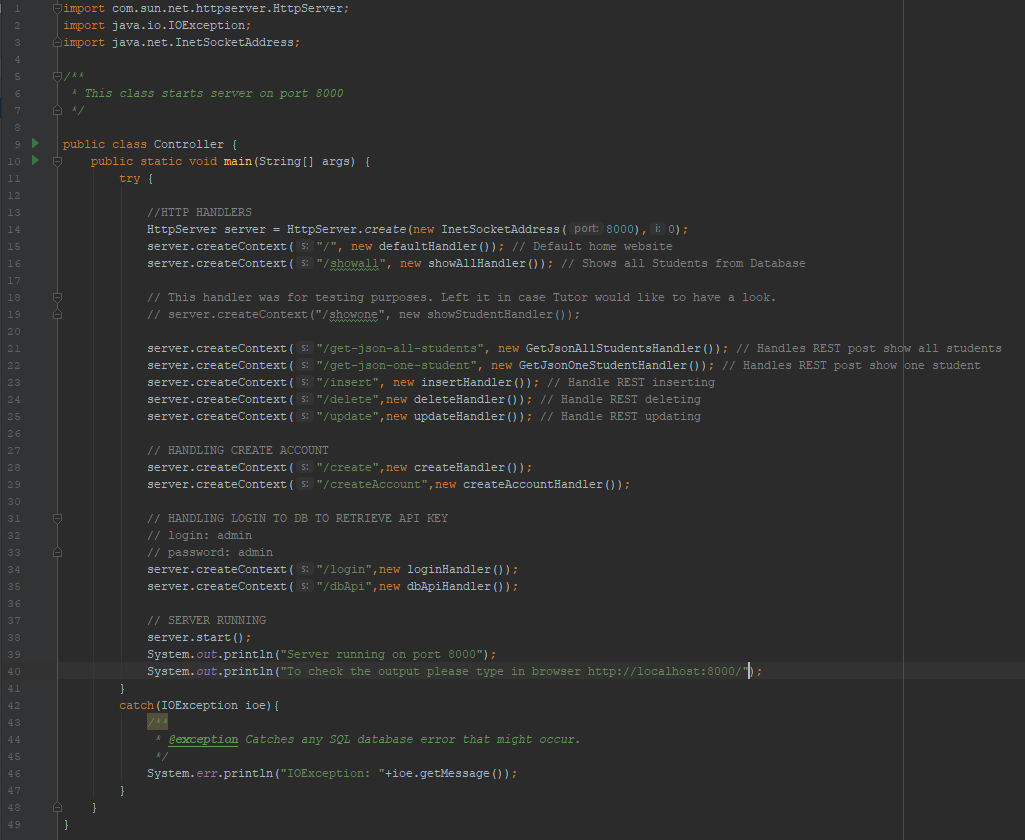
White square:

-Provided credentials correct! (checkLoginCredentials(“admin”,”admin”)

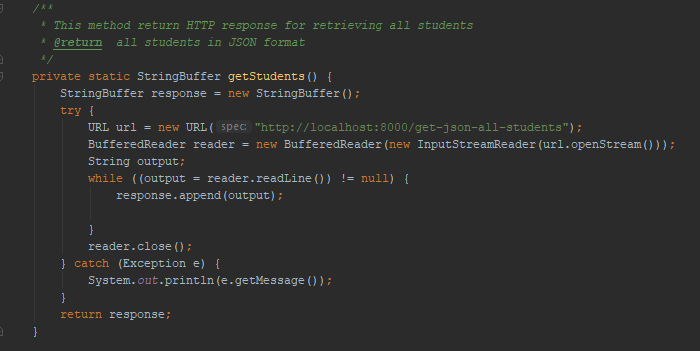
-Retrieved api key for admin which is 123.

**RESTful Web Service**

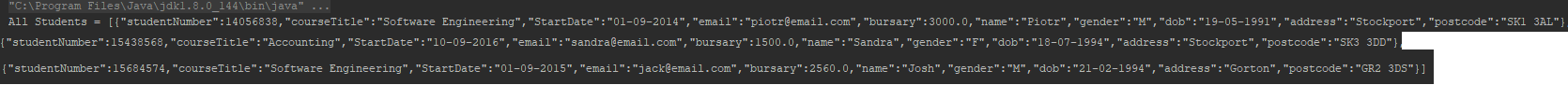
Server connects to the SQLite database and RESTful route for reading all student information (GET) and output in JSON format.



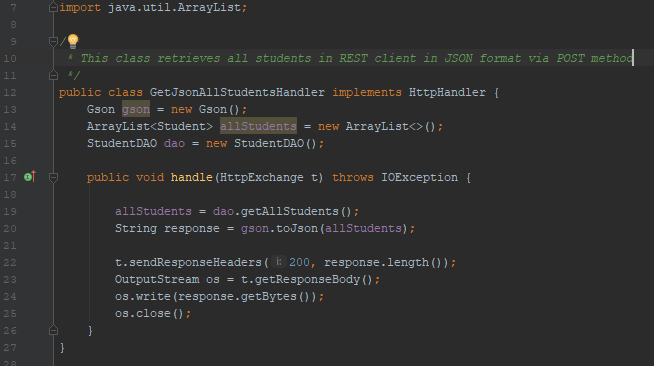
Controller class that connects to database with all context routes including /get-json-all-students to retrieve all students in JSON format.



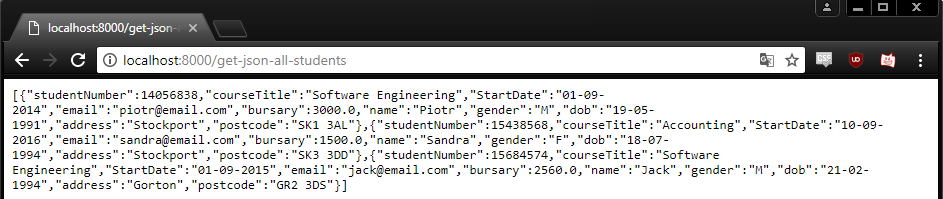
Reading GET information from URL /get-json-all-students



Console output in JSON format

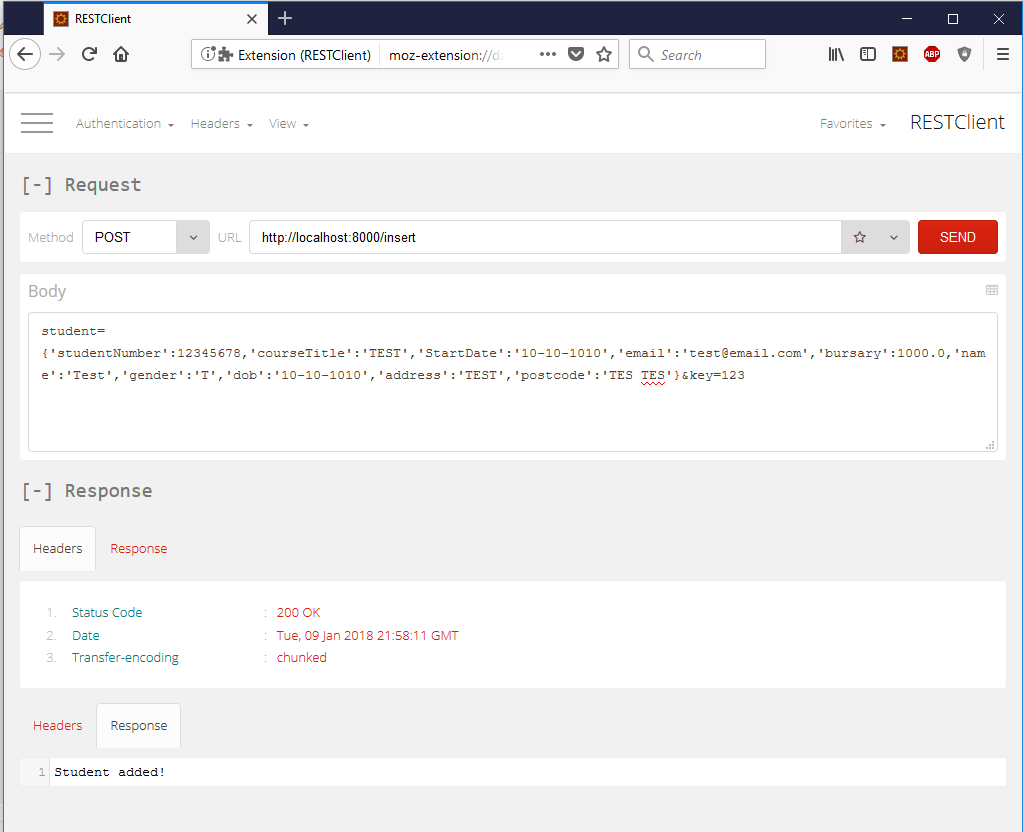


All student’s handler class.



Bonus: Displaying all students in JSON format in browser.

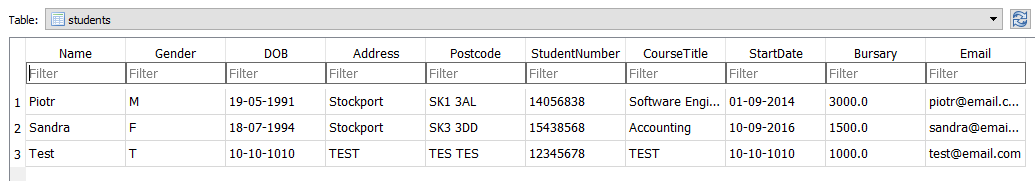
RESTful route to create a student record by posting a JSON object.



RESTful route to insert “TEST” student object using POST method. As we can see the record was successfully inserted and the response was “Student added!”



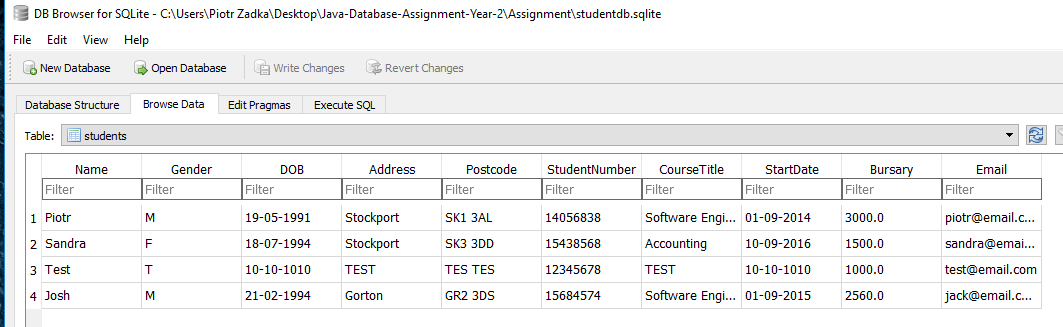
Result in program console after posting above method.



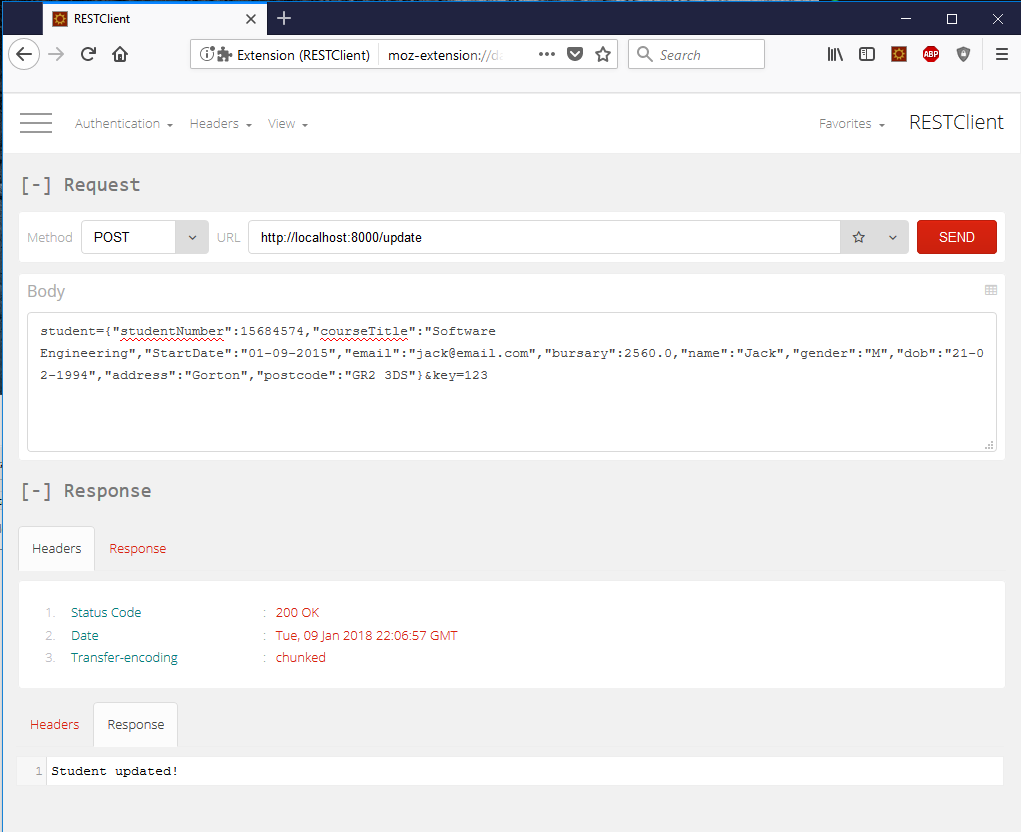
Current result of Students table (Test student added)

RESTful route to update and delete student records

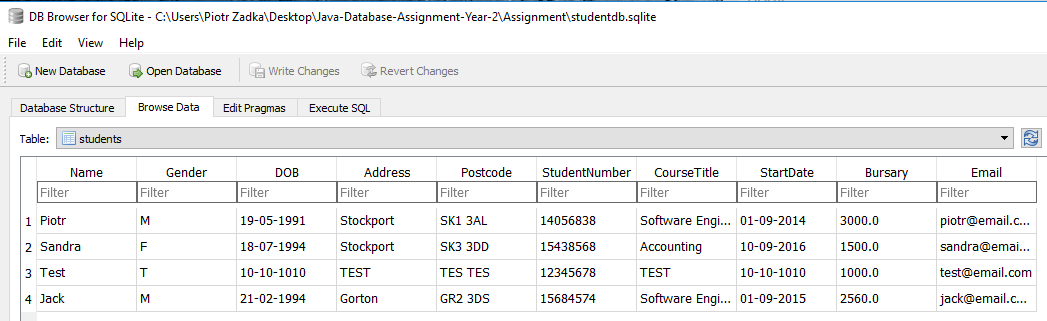
UPDATE:



Students table before executing POST method in REST Client (Name Josh in row 4 visible)

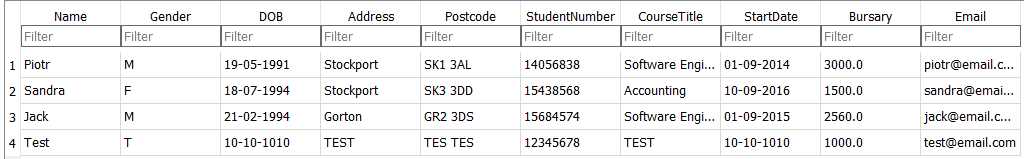


Student object passed using POST method and updating name from “Josh” to “Jack”. The response is successful and with message “Student updated!”

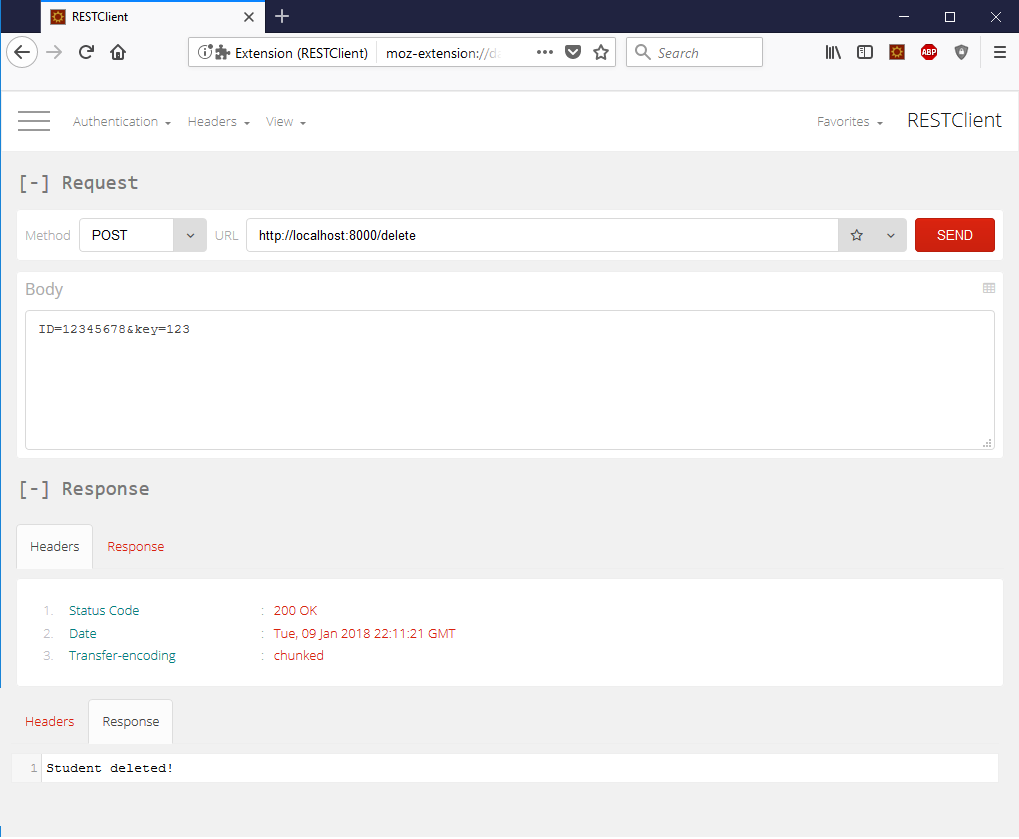


Students table after executing POST method (Name Jack in row 4 visible)

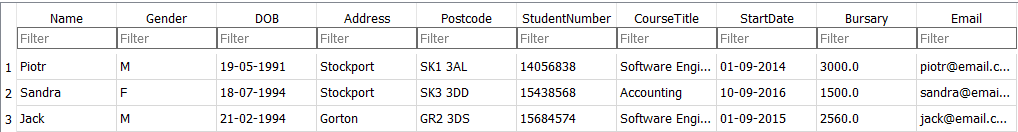
DELETE:



Student table before executing POST method (Row 4 with Test student exists)

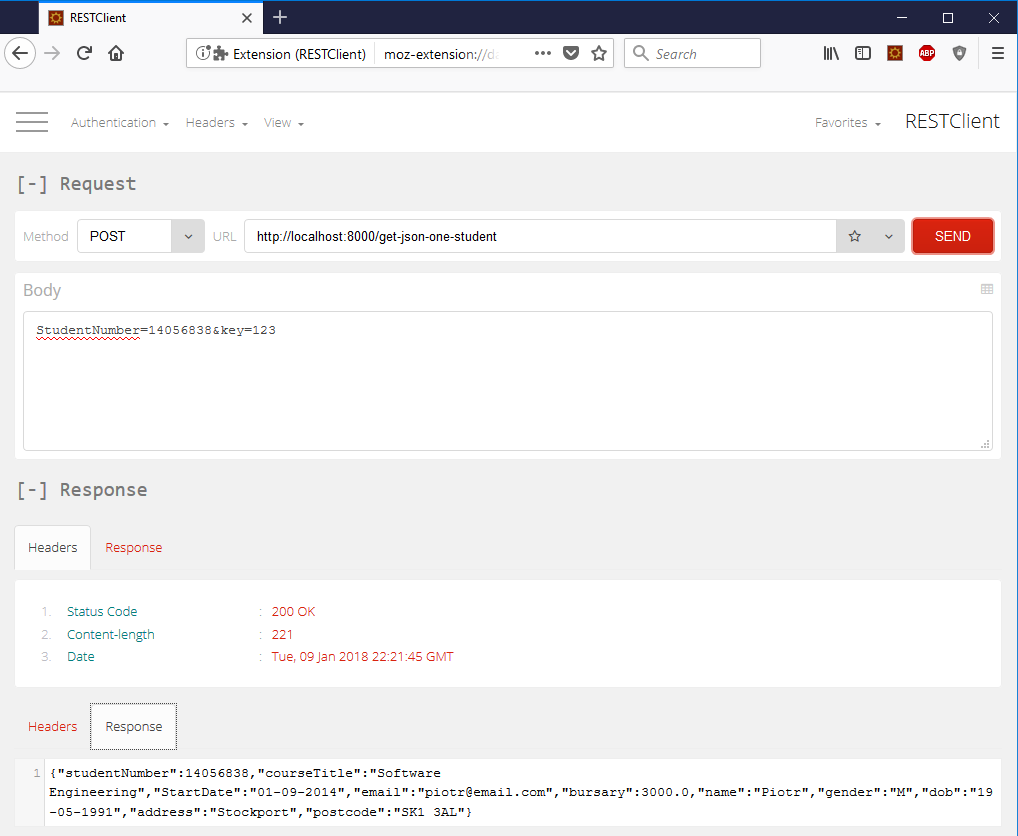


Student ID passed using POST method and deleting student with ID = 12345678. Request was successful with displayed message “Student deleted!”



Student with ID = 12345678 successfully deleted from database

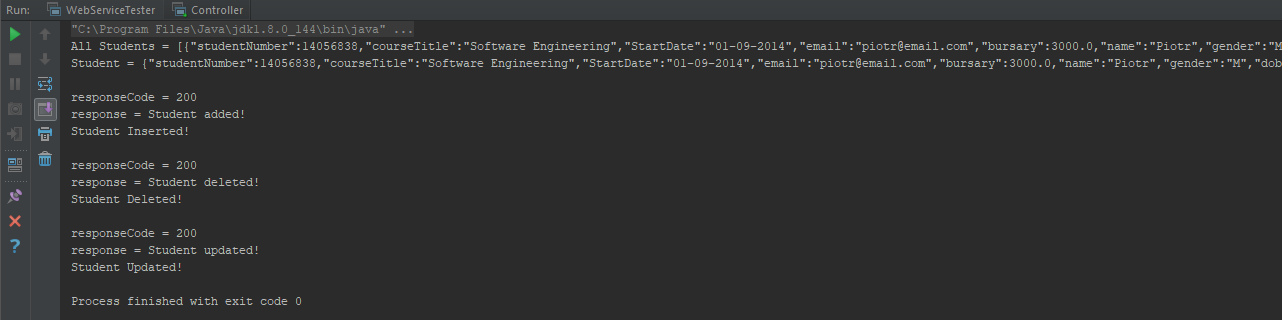
RESTful route to retrieve a student record based on student ID



Student Number POST’ed to /get-json-one-student.

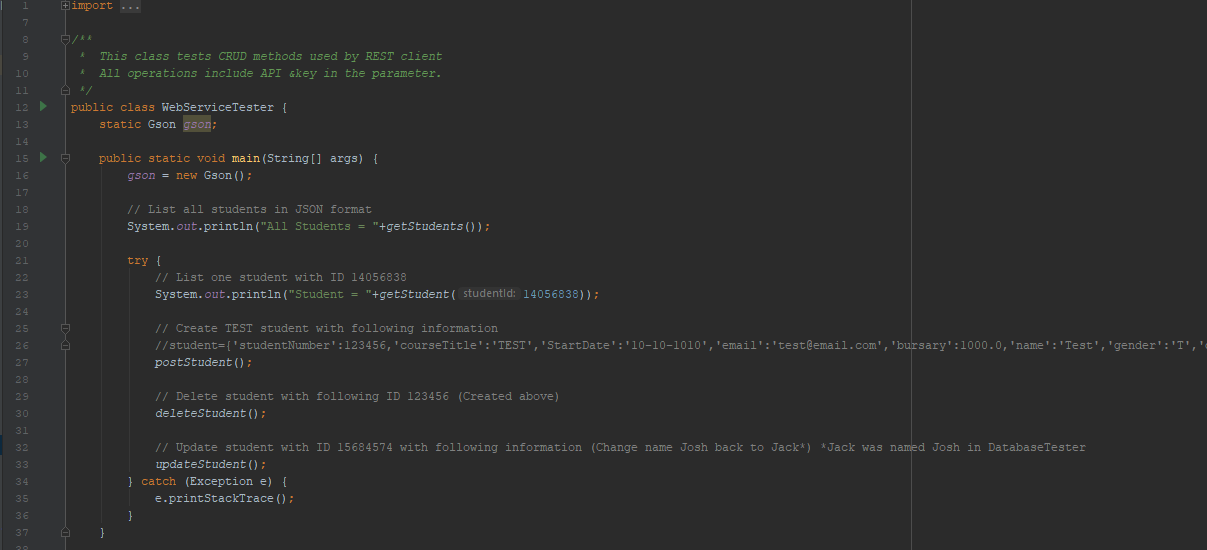
Result is successful and response is posted in JSON format.

All services tested using a class called “WebServiceTester”

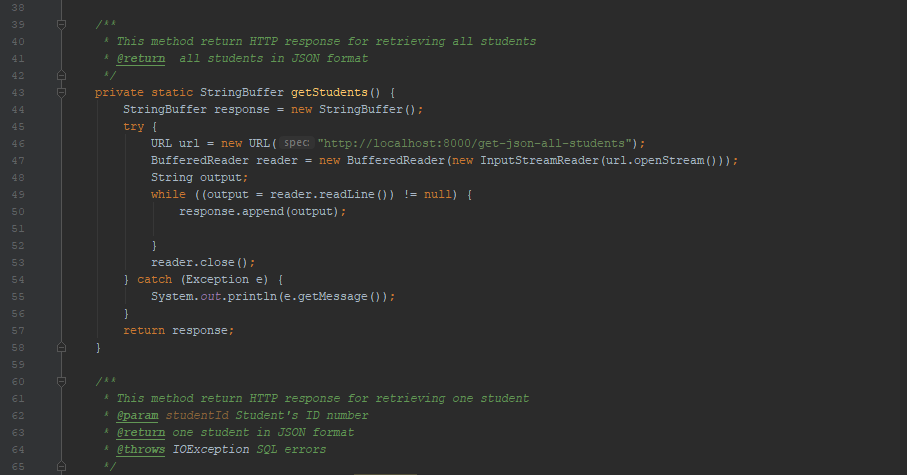


Result of running WebServiceTester.class

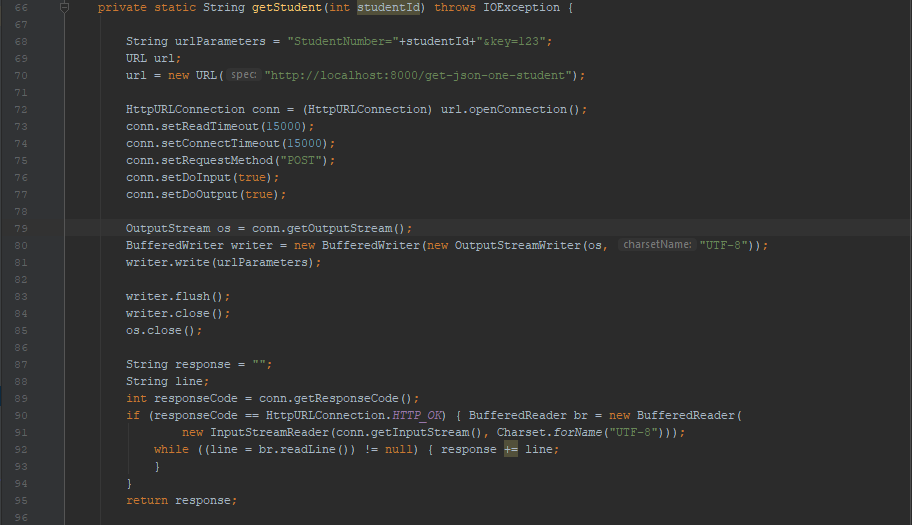
Below you can find screenshots for all methods that it contains.



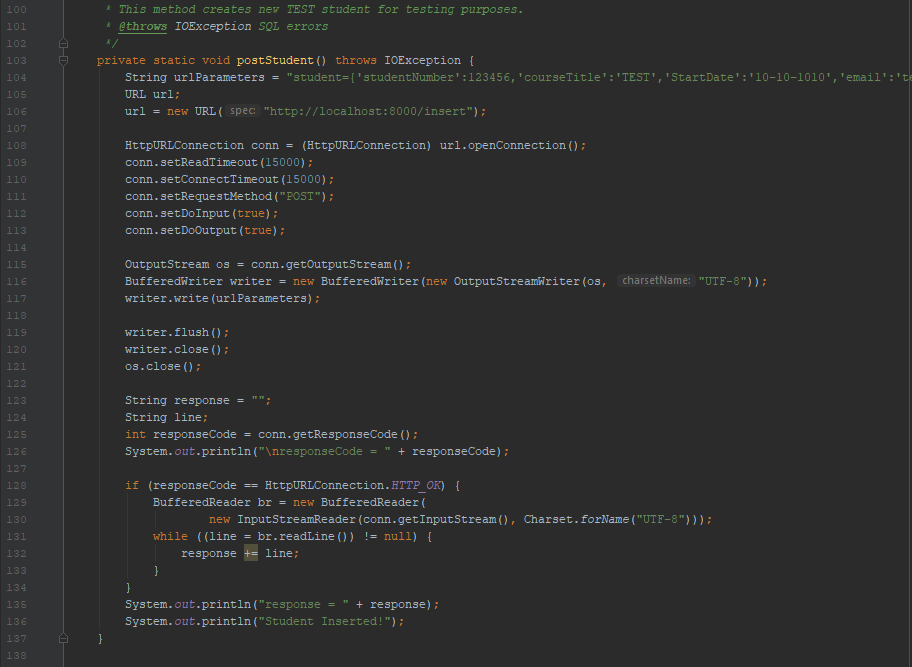
WebServiceTester with getStudents(), getStudent(), postStudent(), deleteStudent(), updateStudent()



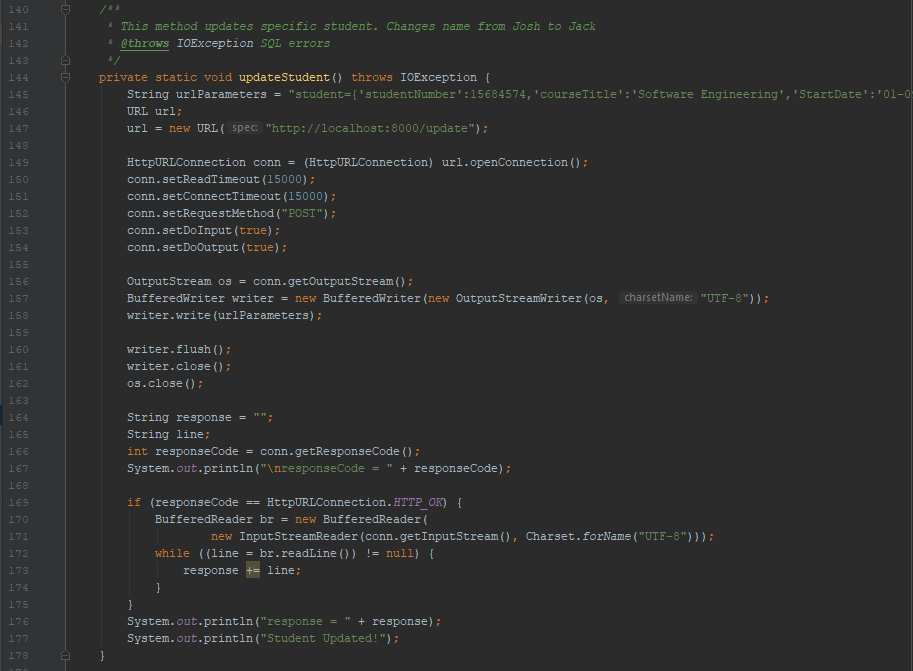
getStudents() method



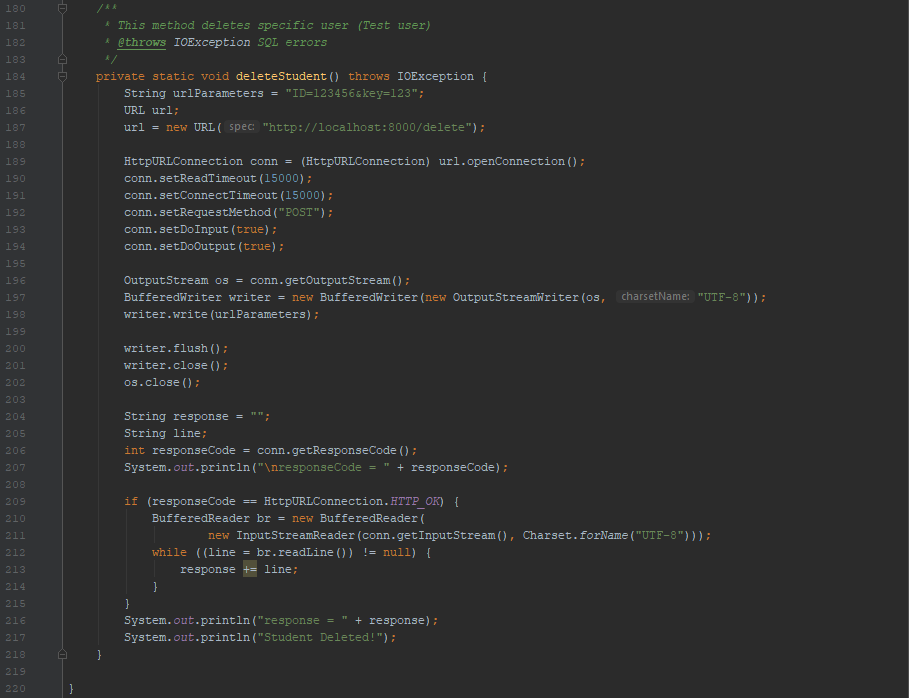
getStudent() method



postStudent() method

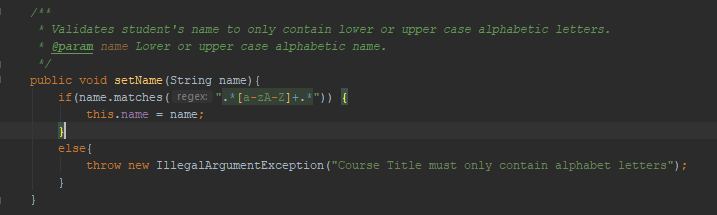


updateStudent() method

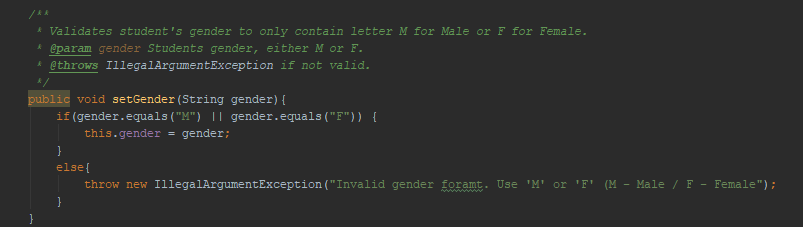


deleteStudent() method

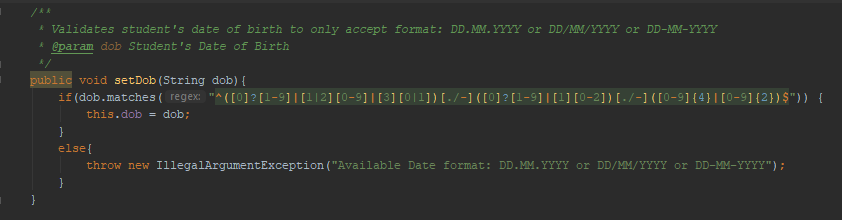
Validation



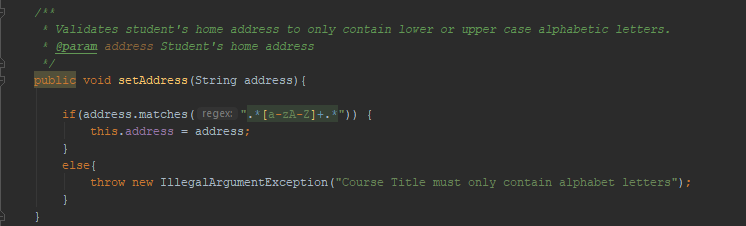
Validation for Name to only contain lower or upper-case alphabet letters



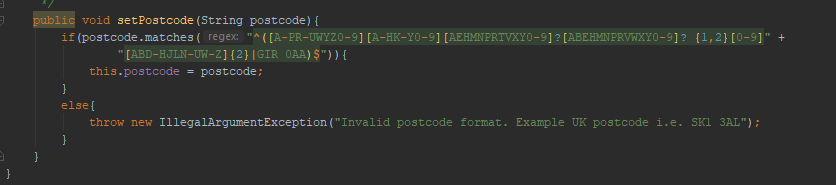
Validation for Gender to only accept letters M – Male or F – Female



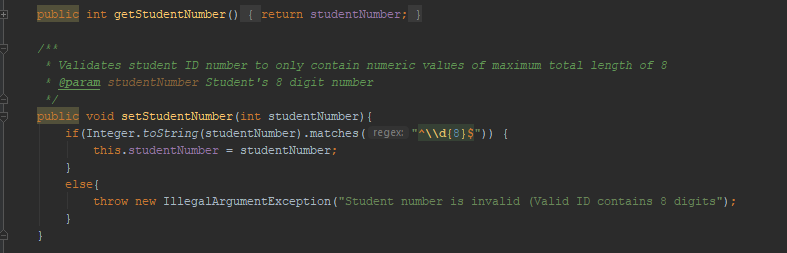
Validation for Date of Birth to only accept specific format visible above



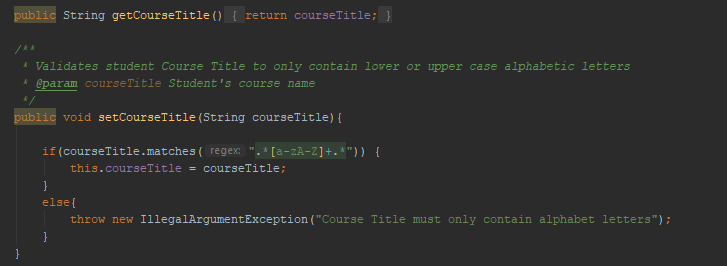
Validation for Address to only accept lower or upper-case alphabet letters.



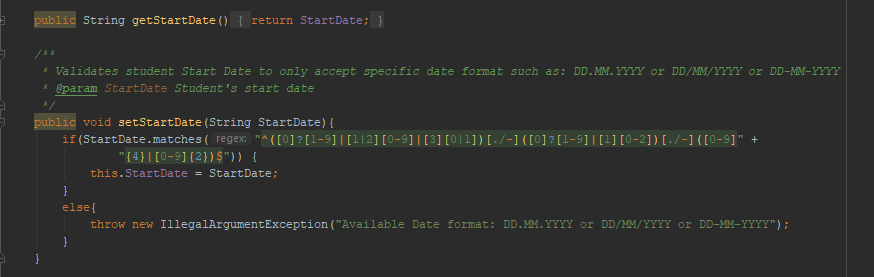
Validation for postcode to only accept format provided by UK government.



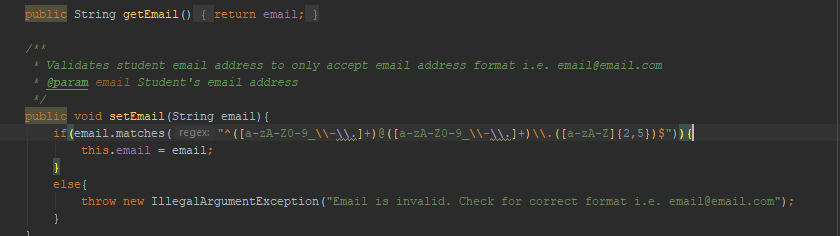
Validation for Student Number to only accept 8 digit numbers



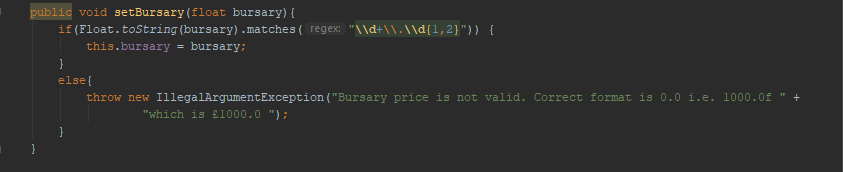
Validation for Course Title to only accept lower or upper-case alphabet letters.



Validation for Start Date to only accept specific date format visible above



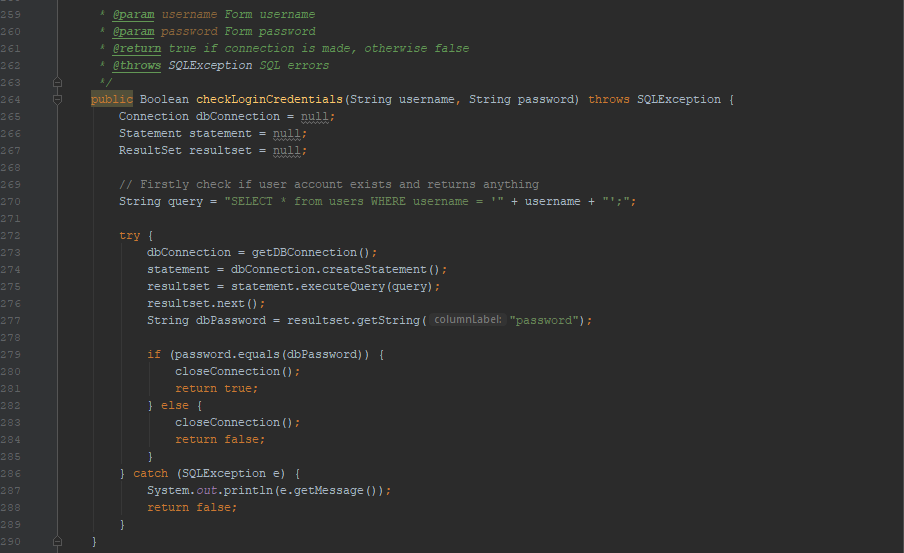
Validation for email address to only accept correct format i.e. [email@email.com](mailto:email@email.com)



Validation for bursary to only accept specific value format such as 1000.0 or 1234.65 (with no maximum letters before dot and only 2 after dot)

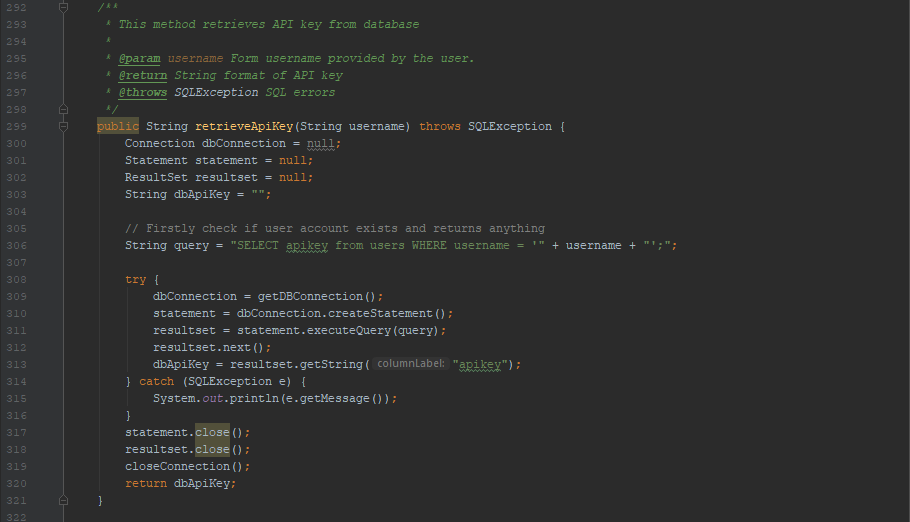
Route for login where user can register and retrieve an API key

As visible on previous screenshots each RESTful POST request had extra parameter **?key=123** which was a API key for admin account. Each user can create their own account and retrieve a randomly generated api key from database. All screenshots explaining the process are below.



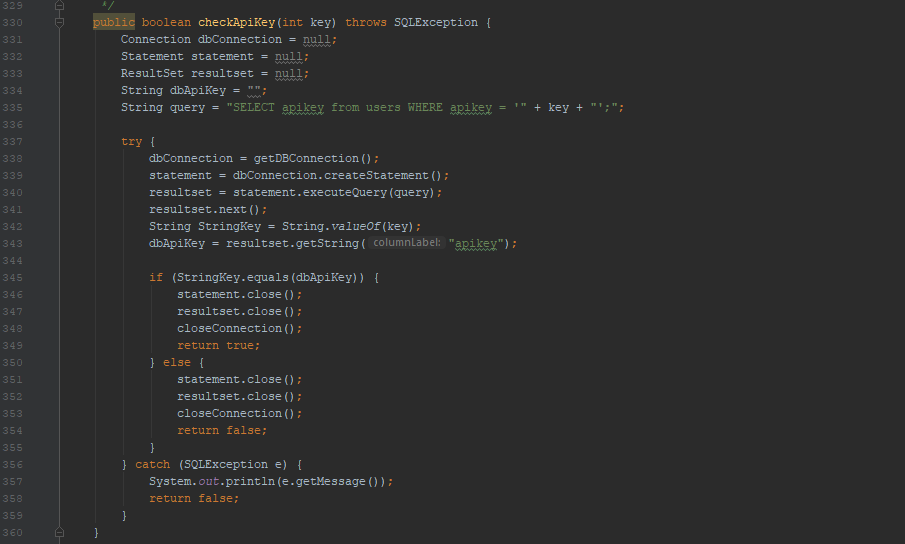
checkLoginCredentials() method in StudentDAO.class

This method checks if provided username/password is correct to connect to database and retrieve API key



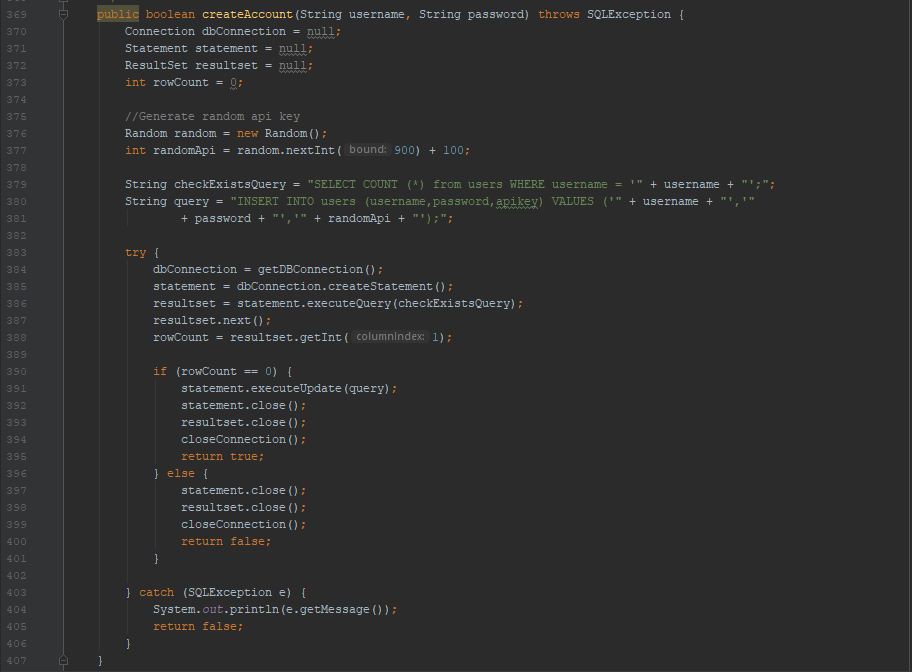
cretrieveApiKey() method in StudentDAO.class

This method retrieves api key based on provided username in HTML form



checkApiKey() method in StudentDAO.class

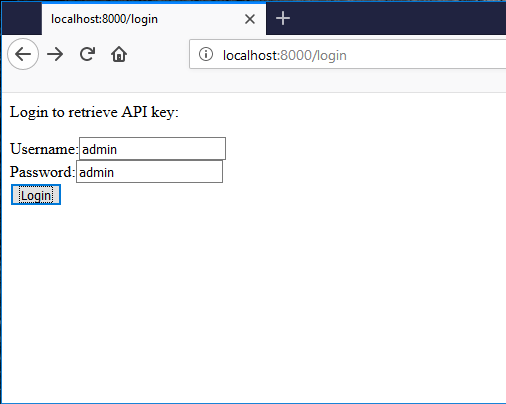
This method checks if provided apikey is correct with the one that is in database



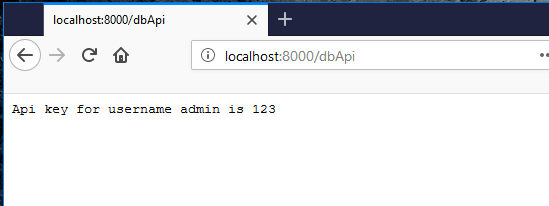
createAccount() method in StudentDAO.class

This method passes values provided in HTML form to create new user account.

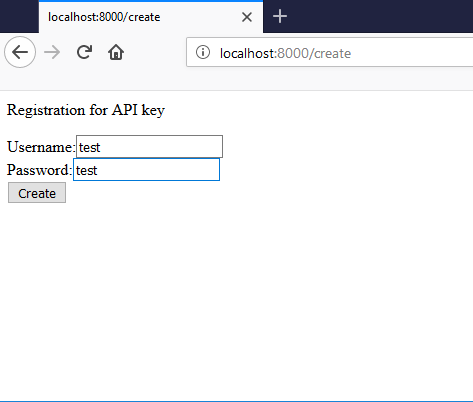
Before adding new user method checks if there is already a user with such username.



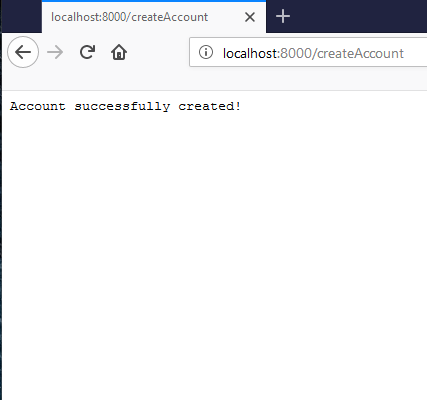
Browser Login form



Key for user “admin” retrieved



New user creation showcases



Account successfully created!

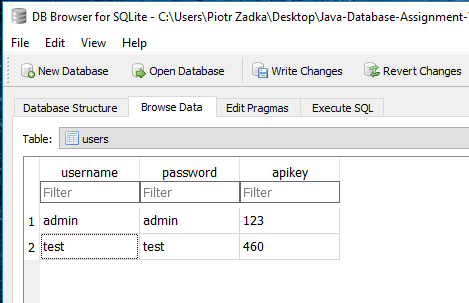


Table after executing previous screenshots.