**Course Project Report for Undergraduate Students**

**Course Name：** Database Systems

数据库系统(全英)

Internet Application Development

互联网应用开发(全英)

**Course Code：**

**《Final Project》**

**Name**

**Student ID**

**College**

**Major**

**Email**

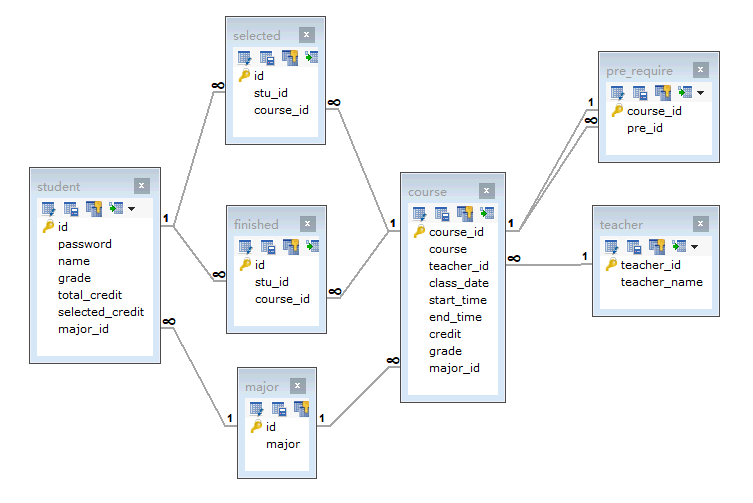
Submit Date：/2021

1. **Project Requirements**
2. *IAD* Requirements: Design and implement a web-based partial application with *Vaadin* (or *SparkJava*) + *MyBatis* on Teaching Administration Management, and mainly implement Select-Courses with necessary helper features (view selectable courses and selected courses, adjust selected courses etc.), considering the following constraints: prerequisite courses, credit limit (no more than 21 credits in one semester), and no time conflicts for selected courses.
3. *DBS* Requirements: You are asked to design and implement an DB-based partial application system on Teaching Administration Management, and mainly implement Select-Courses with necessary helper features (view selectable courses and selected courses, adjust selected courses etc.), considering the following constraints: prerequisite courses, credit limit (no more than 21 credits in one semester), and no time conflicts for selected courses. *No any tools restriction.*
4. **Project Development Tools (Languages)**

In this project, I used *SSM* (that is *Spring*, *SpringMVC* and *MyBatis*) to implement the Select-Course System. For the front-end pages, I used *JSP* (*Java Server Pages*) to implement and communicate with the back-end program.

1. **Database Description**
2. Database Structure

To build the database of the *Subject System*, I created the database called *“studentcource”*. It is composed with the following 7 tables, and their relationship is shown as the following figure.



**Figure 1.** the tables in database *studentcource*

To design a structure of the database, foreign key constraints is used to ensure no invalid relationships of entities. The table *selected* and *finish* represent the selected and finished many-to-many relation between students and courses. And the table *pre\_require* represents the perquisite course relation between courses. The *student* table is to record the basic information of each student, including the unique and primary key *id*, which is the student id, and their password to login the *Subject System*. Also, the *major\_id* is the foreign key that refers the *major* table’s *id*. The *course* table is to record the basic information of each course, where it has two foreign keys *teacher\_id* and *major\_id* to connect the information of teachers who teach the course and what major is this course for.

1. Triggers of the Database



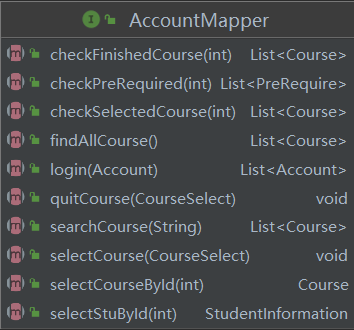
**Figure 2.** the triggers of the database

To regulate the course selecting process, triggers are used to enforce integrity constraints. I created 2 triggers for updating the *selected\_credit* after choosing and quitting the course for each student. At first, I designed the third trigger to detect the invalid process of choosing. However, it cannot transmit the invalided type (such as, time conflict, no enough credit …) to the backend program. Hence, it was abandoned.

1. **Backend Implementation of Subject System**

In this project, I used *SSM (Spring, SpringMVC, MyBatis)* framework set. At first, I designed several domains to represent different entity classes. The *Account* class, which is to record the *student id* and the corresponding *password* for login issue. The *Course* class it to record the basic information of each course. And the *CourseSelect* class is to record the *student id* and the corresponding *course id* that this student wants to choose. The *PreRequire* class records the *course id* and its *perquisite course id*. And the final class *StudentInformation* is to record the detailed information of each student.

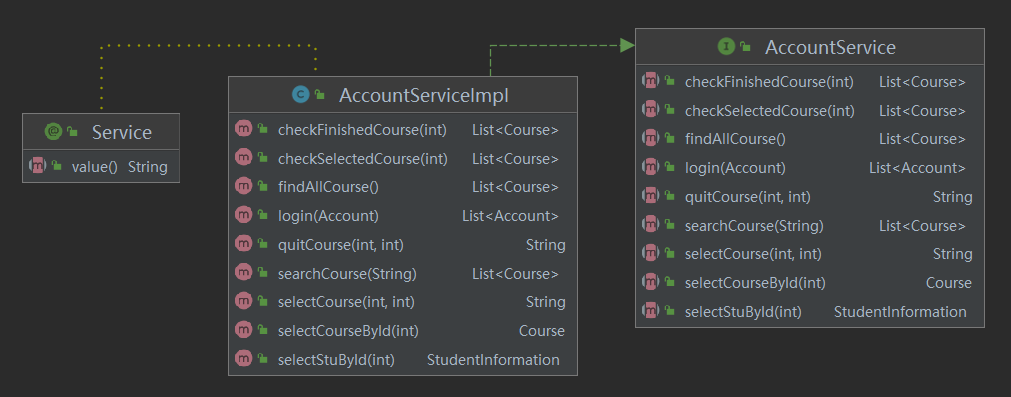
In this project, I used *MyBatis* *Mapper* to connect and operate on the database. And the following *AccountMapper* shows the methods to access and operate the database.



**Figure 3.** the methods in *AccountMapper*

*Spring* is a framework based on the ideology of *IOC* (Inversion of Control) and *AOP* (Aspect Oriented Programming). In my project, the most helpful idea is *AOP*. Based on the core business function and peripheral function aspects of the Subject System, the system can be sectioned into several independent services for a clearer structure.

Based on the *AccountMapper* mapping to the database and providing basic operations on database, the *Service* of *Spring* is designed to provide services for the *Subject System*. By connecting *MyBatis Mapper* with *Spring Service*, the interaction of the program with the database is achieved. The following figure shows the methods of the *AccountService*, which is for services of the *Subject System*.

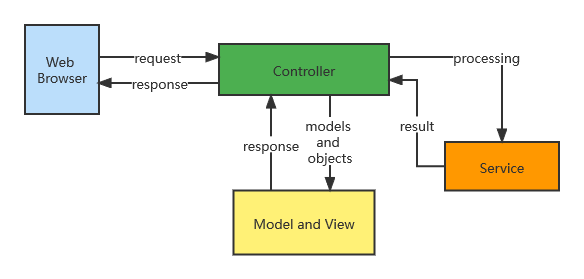


**Figure 4.** the services provided by *AccountService*

As the above figure shows, the *AccountServiceImpl* is to implement the services. As we can see, the *login* service is to check if the input *student id* and its corresponding *password* is correct. If it is, return this *Account* to the *AccountController* (we will introduce later). The *searchCourse* service is to search the courses in the *course* table with the providing *String*. To achieve the fuzzy search, the corresponding *SQL* used *LIKE “%keyword%”* to search for the courses whose name includes the *keyword*. And the service *selectCourseById* and *selectStuById* use the *course id* and *student id* to find the corresponding course information and student information.

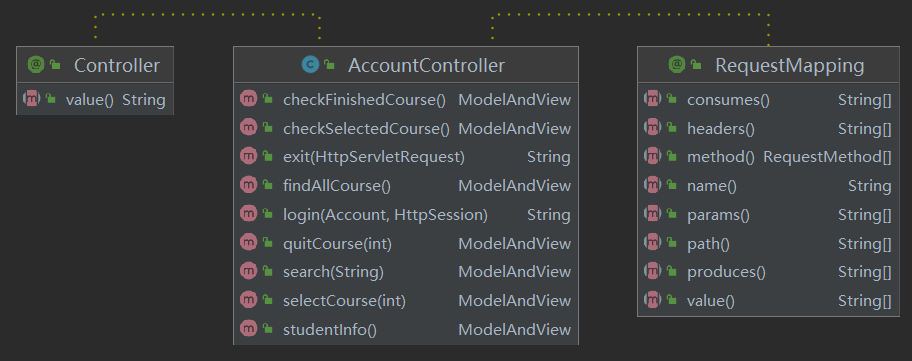
The most important one is the *selectCourse* service. To select a valid course, first, we need to check the conditions of the course that wants to be chosen and courses selected by this student. I compared the *course id* of the course that wants to be chosen with the selected courses for this student, to check whether it is selected. Then, compare the *date,* *start time* and *end time* to check that if they have time confliction. After checking the courses selected, I compare the *grade* and *major* of the student and the those of the course, to check that whether the course is for the *grade* and *major* of this student. Then, I checked whether the course has perquisite course, if it has, go check the finished courses to see if the perquisite course is finished. Besides, for the course you have finished, you are not allowed to choose it again. If all above conditions are satisfied, the course selection is valid.

For *SpringMVC*, in this project, it is used to transmit the web requests to the *SpringMVC Controller*, which is the *DispatcherServlet.* The controller will accept the request, commission the *Spring Service* for processing, get the processing result for reprocessing and return web with the response model and view.



**Figure 5.** the *MVC* architecture

Based on the services of *Spring* provided, the *Controller* of *spring web* is designed to control the *web* behaviors. The following figure displays the methods of *AccountController* with the mapping to the specific behaviors of the *web*.

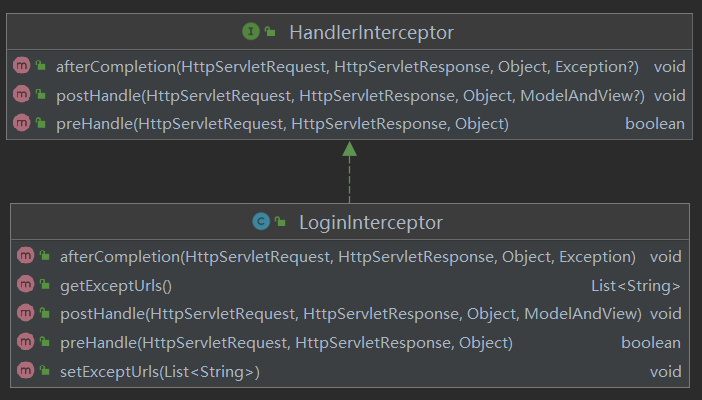


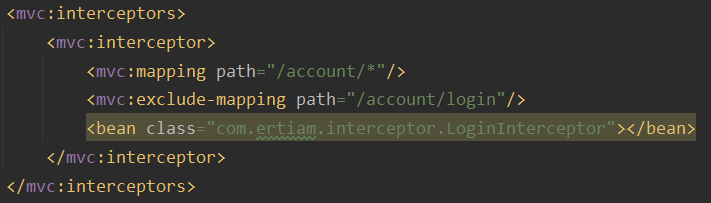
**Figure 6.** the structure of *AccountController*

Each method of the *AccountController* has a unique *RequestMapping*, which maps the *web* action or *URL* to the method. With wiring to the *AccountService*, the *AccountController* is able to ask for the services provided by the *AccountService*.

The parameters of the methods in *AccountController* are obtained from the *web* side. Besides, for the methods with the return type *ModelAndView*, where *ModelAndView* is the data type provided by *springMVC*. *ModelAndView* is used to store the result data after processing, and the view page that displays the data.

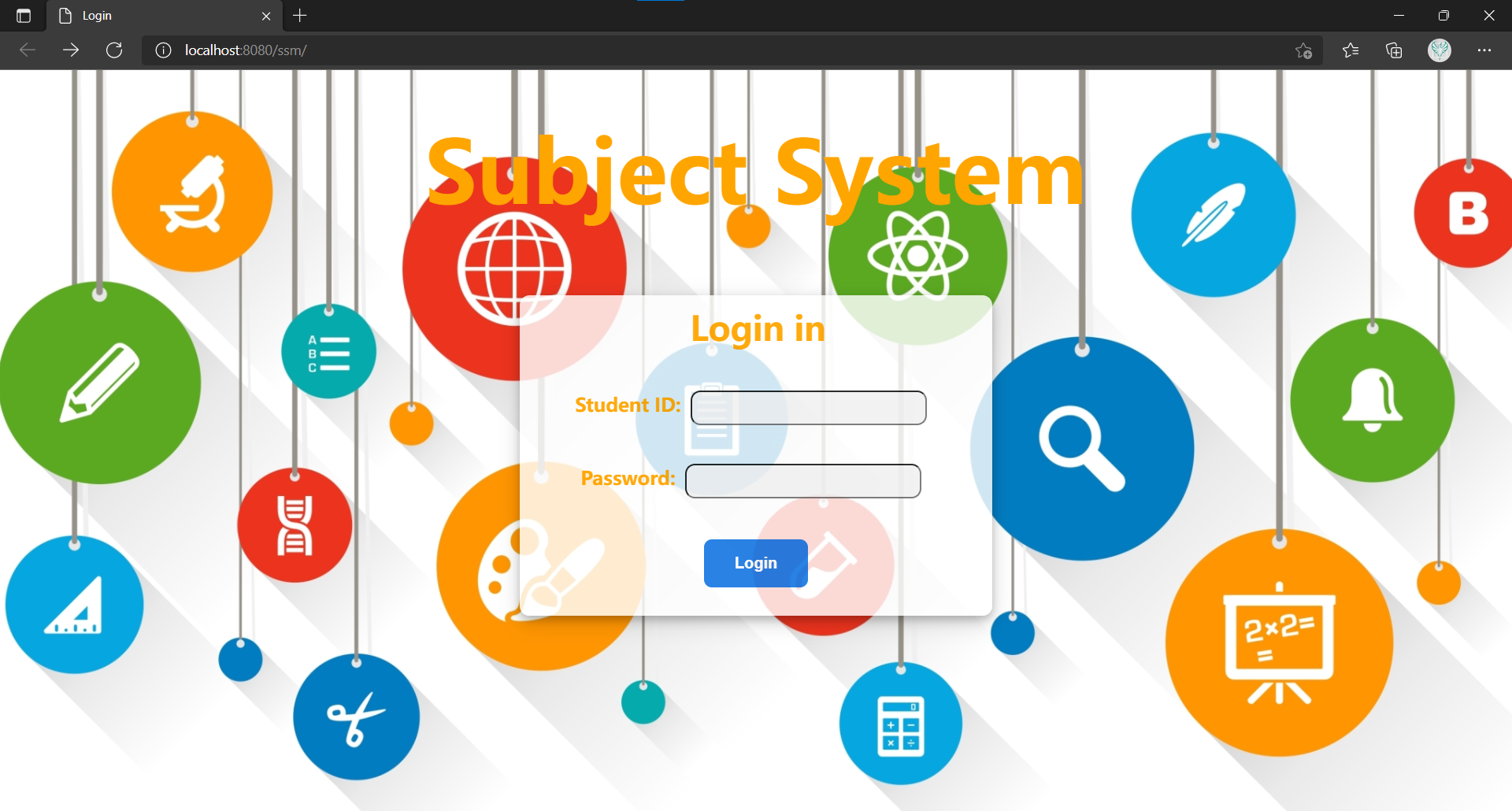
In this project, to intercept the unauthorized access to the *Subject System*. That is to access the system without successful login. I rewritten the *HandlerInterceptor* of *springMVC* to block all *RequestMapping* except the *login mapping* until login successfully. To determine whether the login is successful, the return of the *login* method of *AccountService* is used in *preHandle*. If it had return, that means the login is successful, and the interceptor will no longer intercept other *RequestMapping* in this session. The structure of the *LoginInterceptor* is shown in the following figure.





**Figure 7.** the structure and intercepted recourse of *LoginInterceptor*

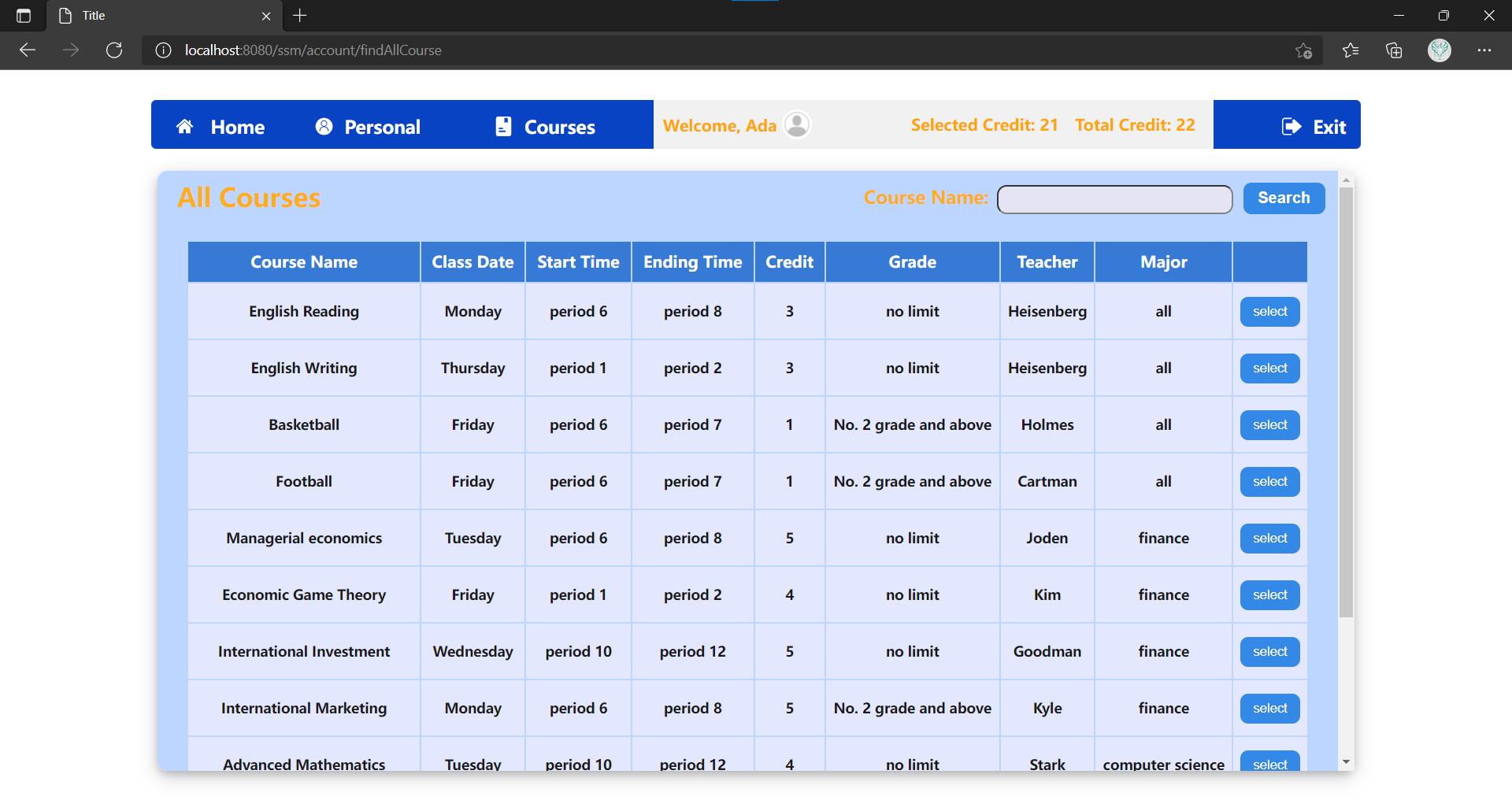
1. **Web Implementation**
2. Login page



**Figure 8.** the Login page

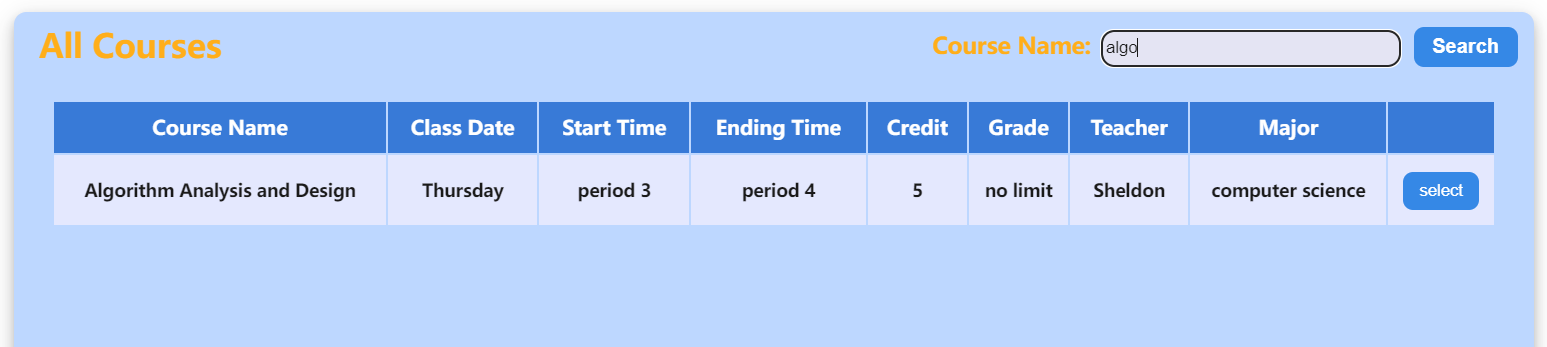
When accessing the system, the student user will get to the login page, students are required to input their *student id* and their *password* into the *form*, and submit it. The *form* is bound with the *action* called *login* of the *AccountContorller*. After clicking the *Login* button, the *id* and *password* will be *post* to the *AccountController* and be processed. If the *id* or *password* is not correct, the system will return back to the login page again.

1. Home (All courses) page



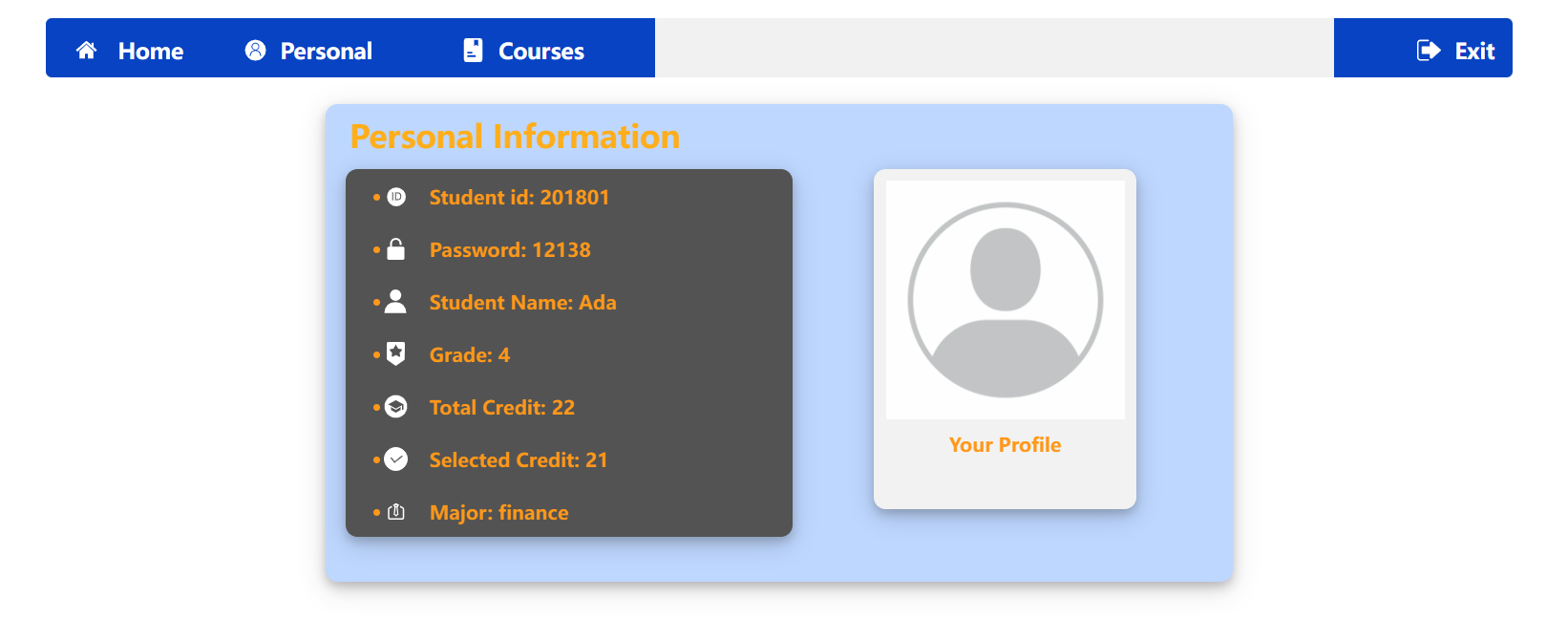
**Figure 9.** the Home (all courses) page

This page’s *URL* is bound with the *findAllCourse* method of the *AccountController.* As the above figure shows, the basic personal information will be shown on the navigation bar, and all courses with their basic information will be shown. At the last column, the *select* button is provided, which is bound with the *selectCourse* method. The system will try to select this course if all conditions are satisfied. There is a search function provided for users to have accurate or fuzzy search for course name as the following figure shows.



**Figure 10.** the result of the fuzzy search

1. Personal Information page

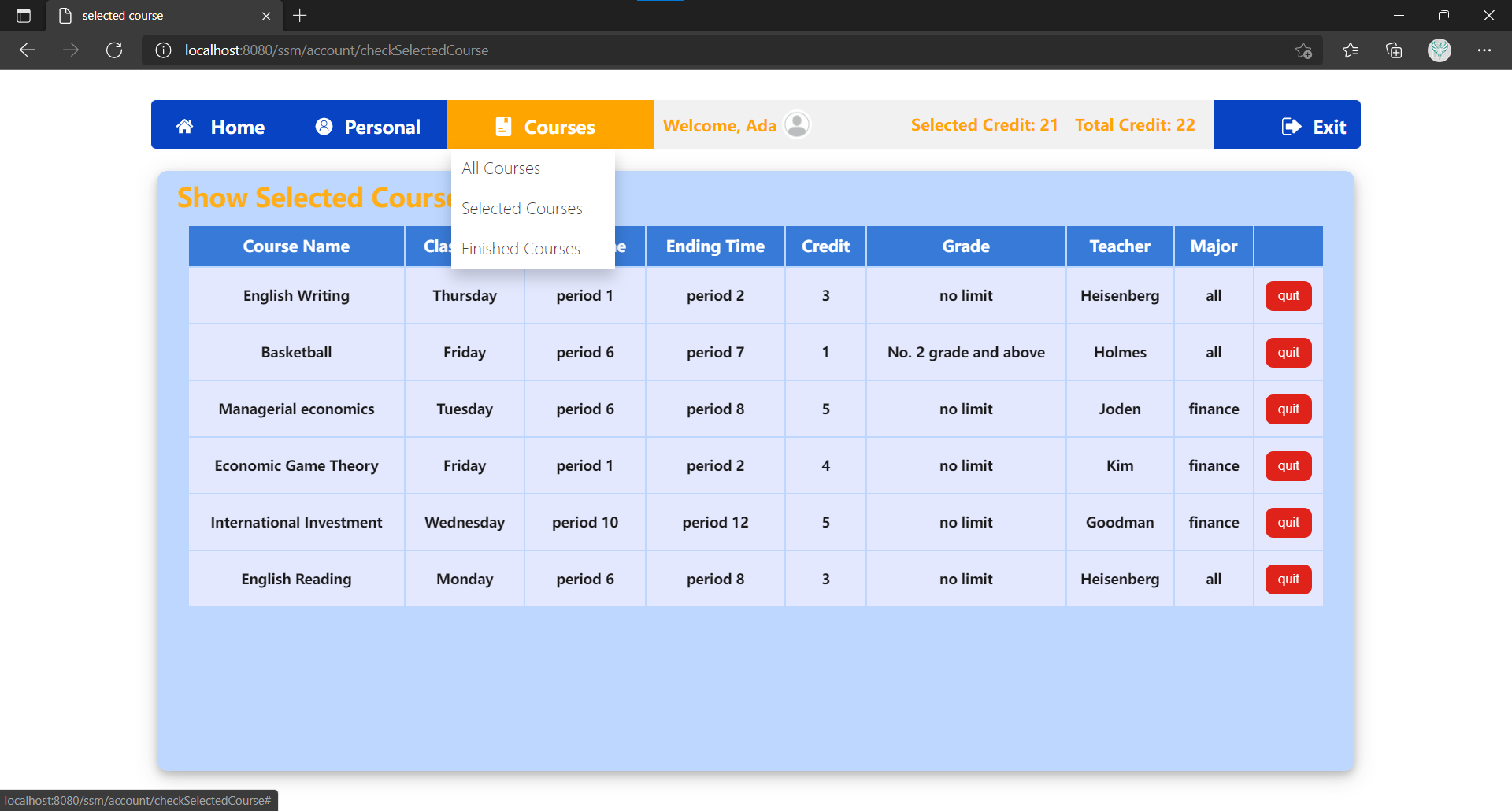


**Figure 11.** the personal information page

As the above figure shows, this page shows the detailed personal information. This page’s *URL* is bound with the method *studentInfo* method of the *AccountController.*

1. Selected Courses page

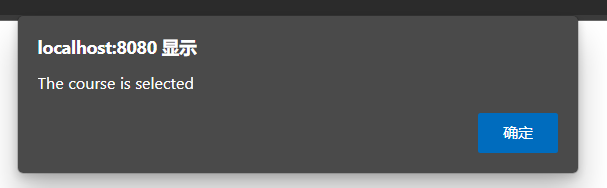
After the user *select* a course from the homepage, or user click the button on the navigation bar, the selected courses page will be shown.



**Figure 12.** the selected courses page

As the above figure shows, the user is able to click *Finished Courses* to check the finished courses. Also, the user is able to quit the courses that has been chosen. When clicking it, the action is bound to the *quitCourse* method of the *AccountController*. After that, it will return back this page again. If the user clicks the *Exit* button of the navigation bar. The session will be invalidated, and return to the login page. The user has to login again to access the system.

When coming into the selected courses page after the user *select* or *quit* a course, it will pop up an alert to inform the user about the state of selecting course or of quitting course. For example, as the following figure shows.



**Figure 13.** the inform of state of selection