CS 487 Team Project Paper

*Campus Rent System*

Author name： Jingtian Li

Harry Green

Liam Walsh

Vanya Yorgova

Contact Info： jli170@hawk.iit.edu

School： Illinois Institute of Technology

Department： Computer Science

Director： Dennis Hood

Date Completed： 11/ 21/ 2015

Content

[ABSTRACT 3](file:///C:\Users\Administrator\Desktop\论文\1035004559_吴彤_个人生活信息管理系统设计与实现.doc#_Toc386639375)

[Conclusion](file:///C:\Users\Administrator\Desktop\论文\1035004559_吴彤_个人生活信息管理系统设计与实现.doc#_Toc386639422) 19

# ABSTRACT

This team project is to create a program that assists the campus community and to make students’ study experience easier and more efficient. The logics of the project have been elicited into a prototype which specifically focuses on the business transactions rather than the page design.

Idea behind the project are to let students add their idle books and tools to the website and people who need those items can view what are available on the website so they can buy or borrow those secondhand items with a very reasonable price.

The logics of the project contain customer registration, user login and logout, adding idle items, view item lists, place orders and make a transaction.

System security, data access, maintainability, usability, user privilege have been considered. The whole project is highly coupling and is capable to add any new functions inside without altering the current code.

An analysis, design, coding, implementation and testing process cycle has been adopted to realize this project. However, design and coding are bundled very tightly and an agile methodology is adopted in the overall building step.

**Key Words**: Business Logics; Software Development Life-cycle; Engineering Design Lifecycle

# Introduction

## Project Lifecycle

The project is completed through an iteration-based process and it goes over the cycle for several times. Iteration of the process has its own focus in terms of design, building, testing and maintaining.

The implementation step is much more flexible and is applied an agile methodology. The fundamental design of the coding is to separate the entire project to several layers with individual functionality. The purpose of this design is to keep the maintainability feasible so it is easy to expand the project and add new functions without altering the current code.

## Project Environment

This project is designed under a jdk 1.7 environment and it requires a java complier with at most a 1.7 version.

The coding IDE is MyEclipse 11; database is MySQL; server is Tomcat.

All pages have been tested on both IE and Google Chrome browsers.

# Prototype Design

## MVC Structure

The structure can be illustrated with the following graph:

Web Layer

Service Layer

DAO Layer

DAO Interface

JSP

Javabean

Database

DAO

Service

Servlet

Service Interface

User’s Browser

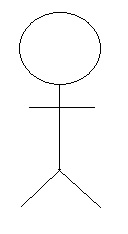
Figure 1. MVC Structure

(The Program is built using javaEE, thus the graph above uses java terminology)

A Model-View-Controller (MVC) structure is utilized in this prototype. The *Model* stands for data encryption classes, *View* stands for visible interfaces and *Controller* represents logical control classes that handle the business logics (including user identity verification, data transfer, transaction logic, etc.)

The project utilizes *factory mode* to separate the program logic to three layers without code connection. The web layer deals with information verification and data transfer; the service layer handle business logics; the DAO layer controls data transaction with the database. These three layers are connected with interfaces which have those classes information stored in a configuration file. So it is easy for the program to expand and to maintain if it needs to in the future.

## Use case diagram



New users

Figure 2. system use case diagram

Users are divided into two categories: regular users and administrators.

Below are the privileges that belong to each user group:

* Communal functions:

Registration

Login

Logoff

Remember login status

View item list

* Functions limited to regular users:

Add Items

Place orders

Delete orders

* Functions limited to administrators:

Download item list.

## Database Design

User

Id (PK)

Username

Password

Nickname

Email

Role

State

Activecode

updatetime

Order

Order\_Id (PK)

Money

Receiverinfo

Paystate

Ordertime

User\_Id (FK)

OrderItem

Produc\_Id (PK, FK)

Order\_Id (PK, FK)

buynum

Product

Id (PK)

Name

Price

Category

Pnum

Imgurl

description

(PK: Primary Key; FK: Foreign Key)

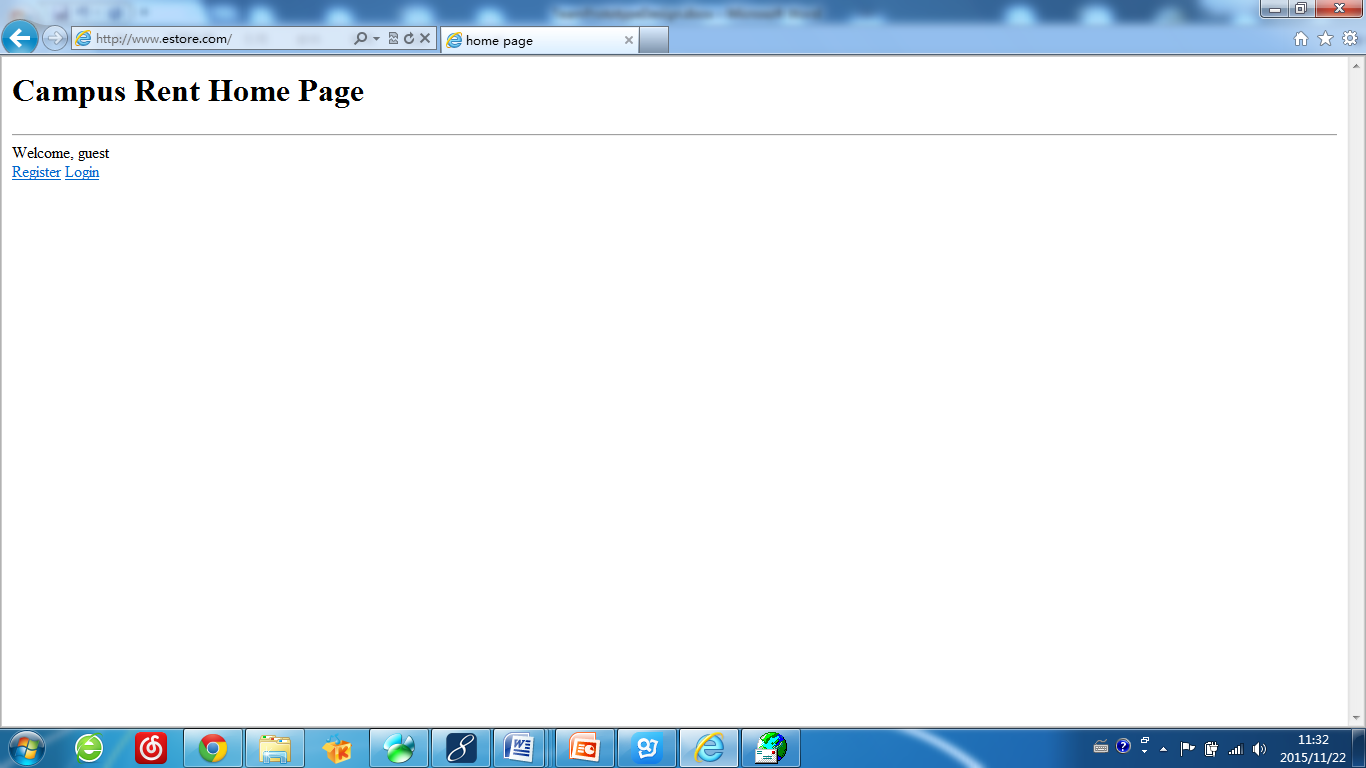
A user can place multiple orders. An order may contain many different products and a product maybe included in several orders. A many-to-many relationship needs to be transferred into two one-to-many relationships, thus, an *OrderItem* table is created.

# Function implementation and realization

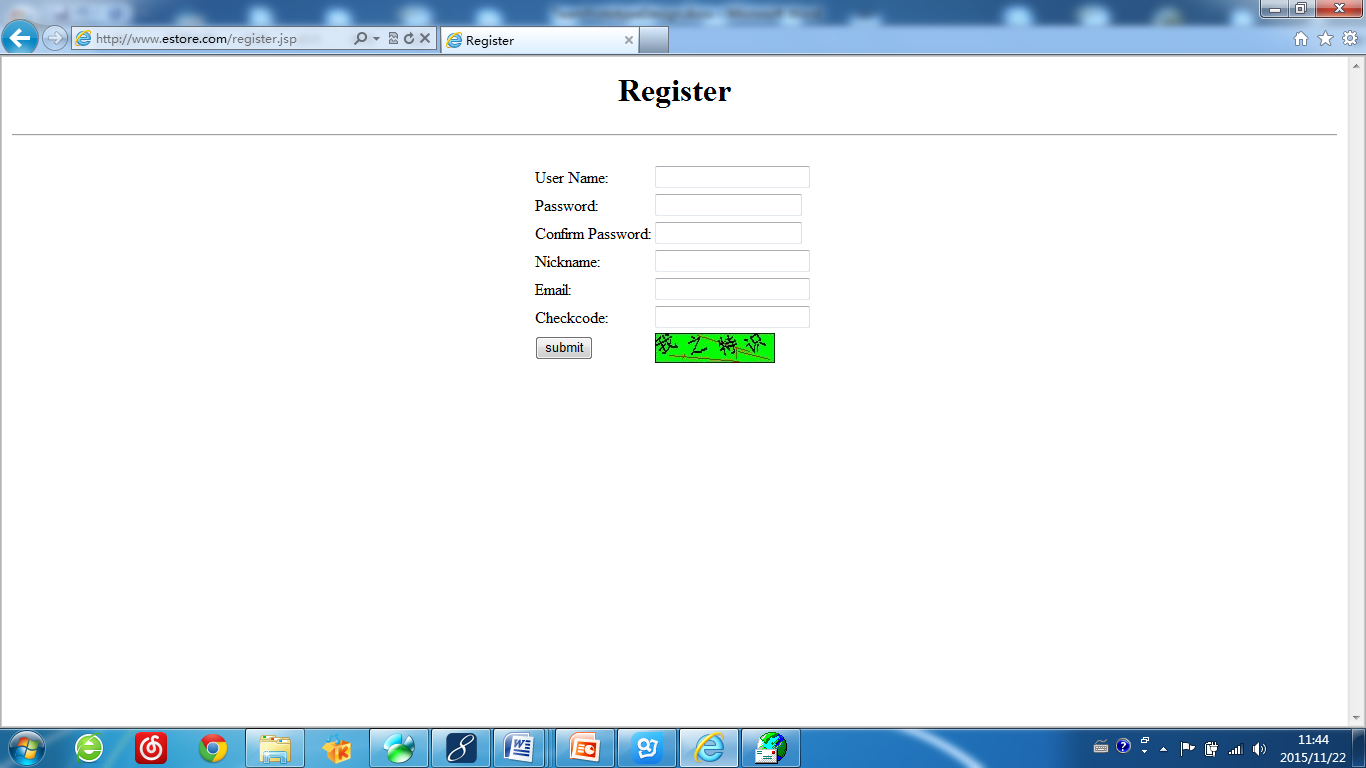
This section presents every function of the project and gives a whole process from registration to place an order.

Performance analysis, data security, usability and user categories control will be analyzed with the process demonstration.

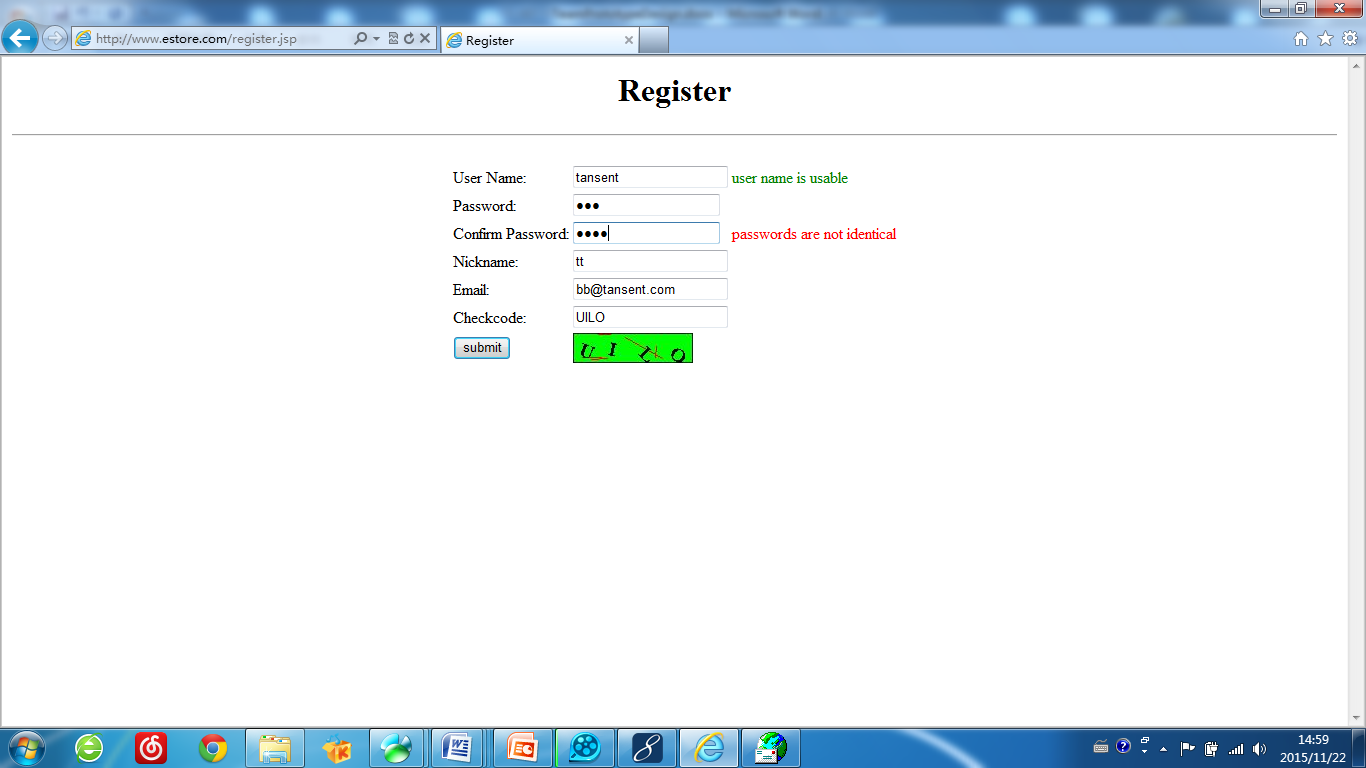
**First time to visit *Campus Rent*:**



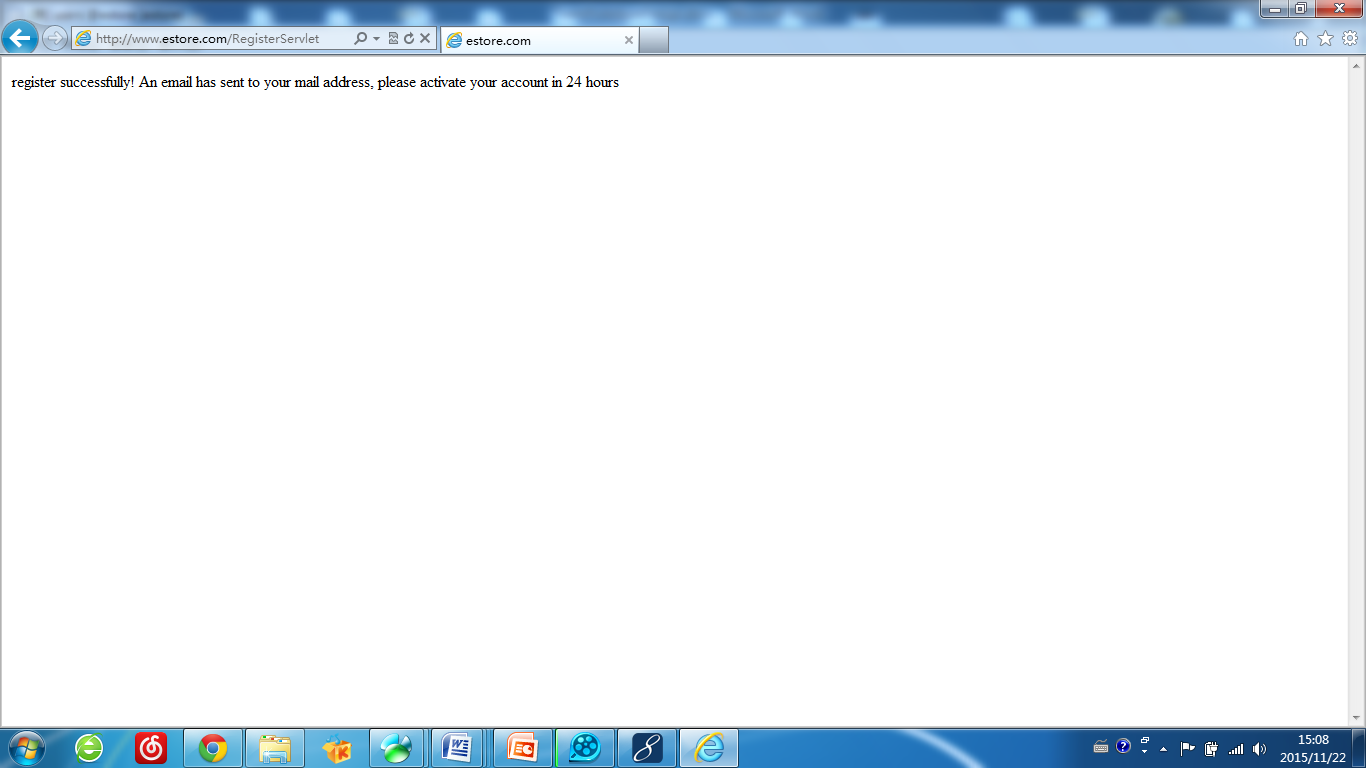
**Register:**



If there is redundant information or incorrect information, for example, existed username or inconsistent password entered.

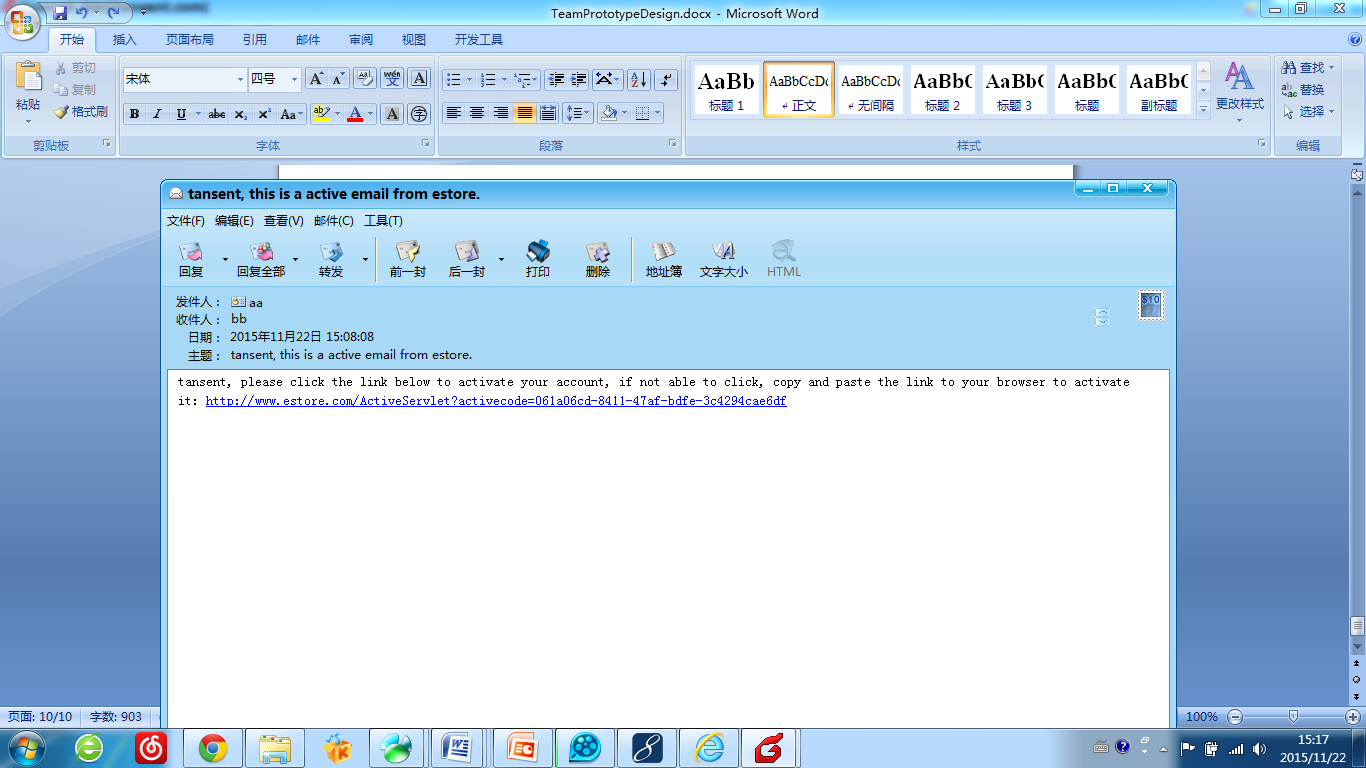


After successfully fill the table, a new record will be inserted into the database and a notification will pop up to show that the registration is completed.

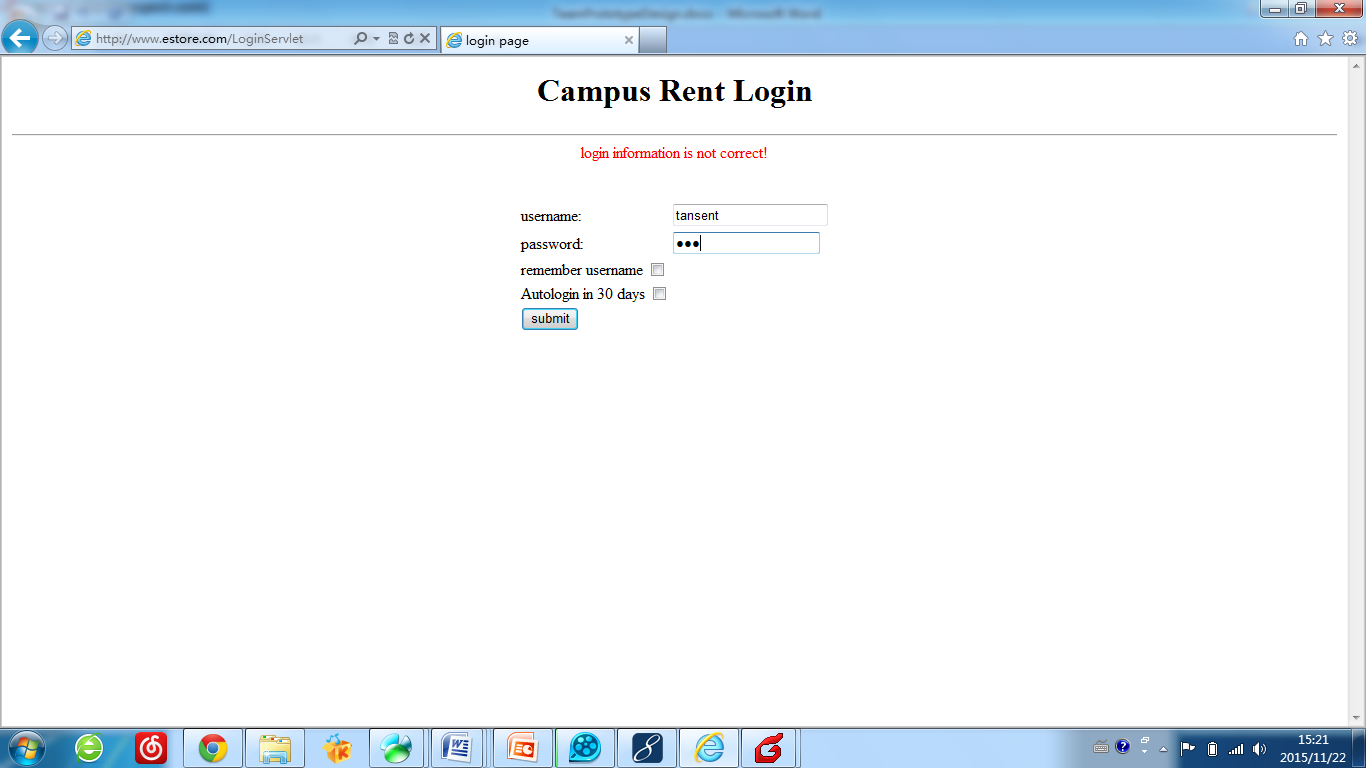


**Activate the email:**

An email will be sent to the email address the user has entered to confirm the registration status. If the user does not activate the link, a hold in database will prevent the user from logging in.

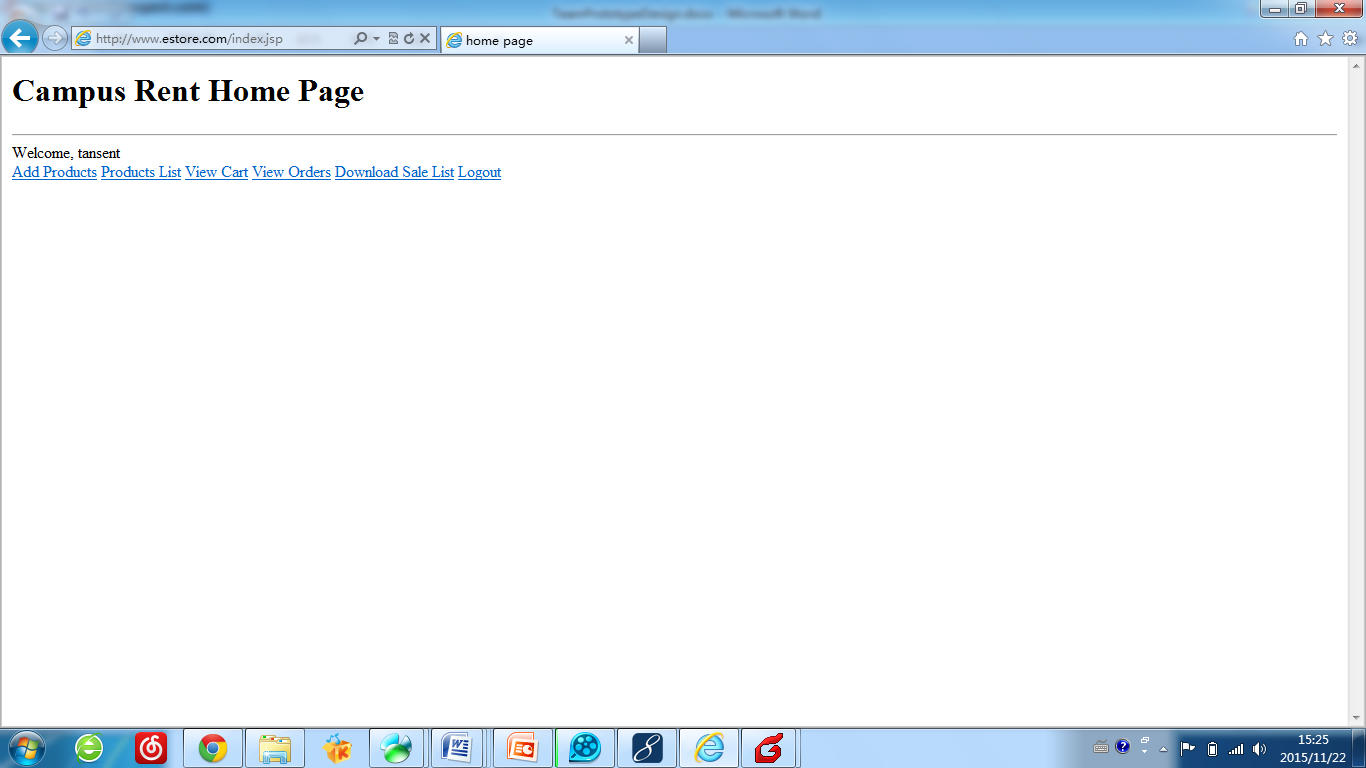


**Login:**



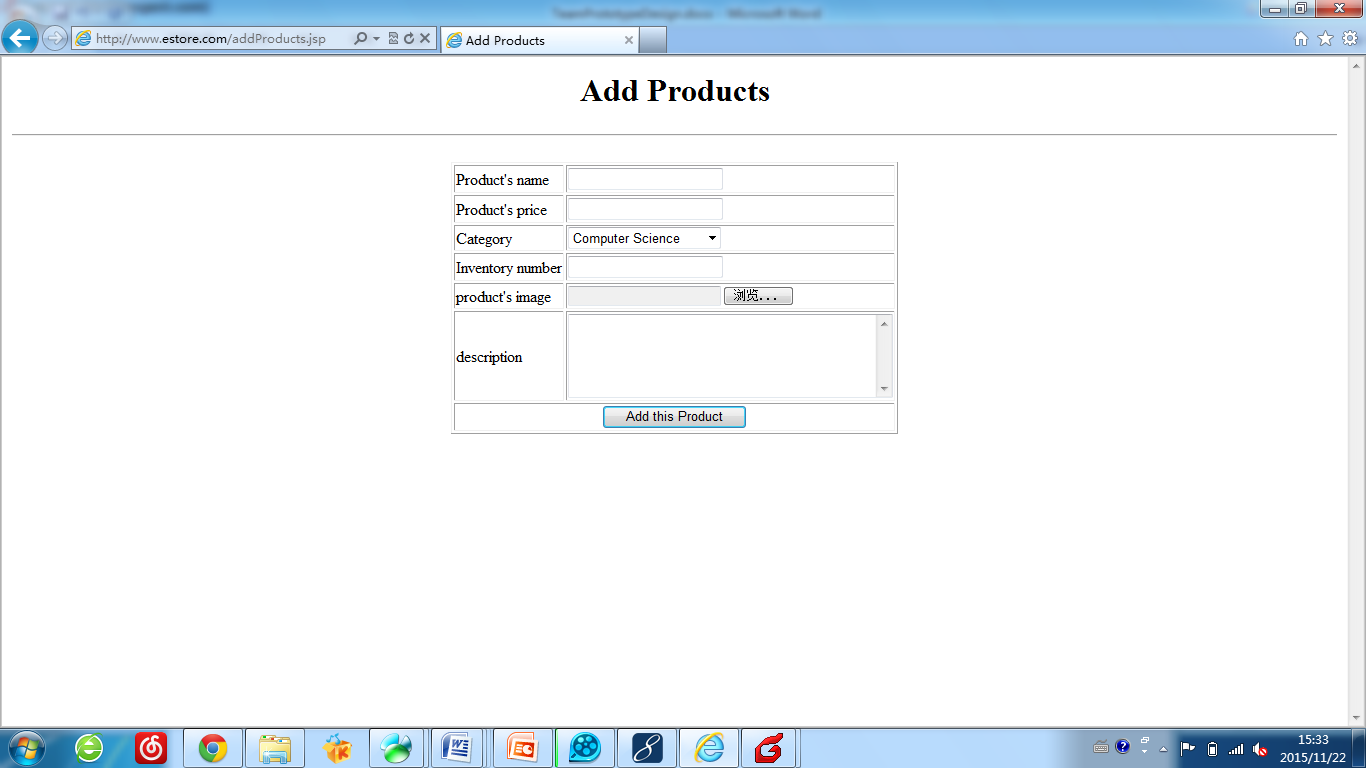
If the user enters information that is not consistent with what has been stored in the database, the system will prompt an error.

**Homepage:**



From this homepage, regular users and administers have different privileges. Regular users can do all the functions above except “Download Sale List” while administrators cannot add products or place orders.

**Add Products:**

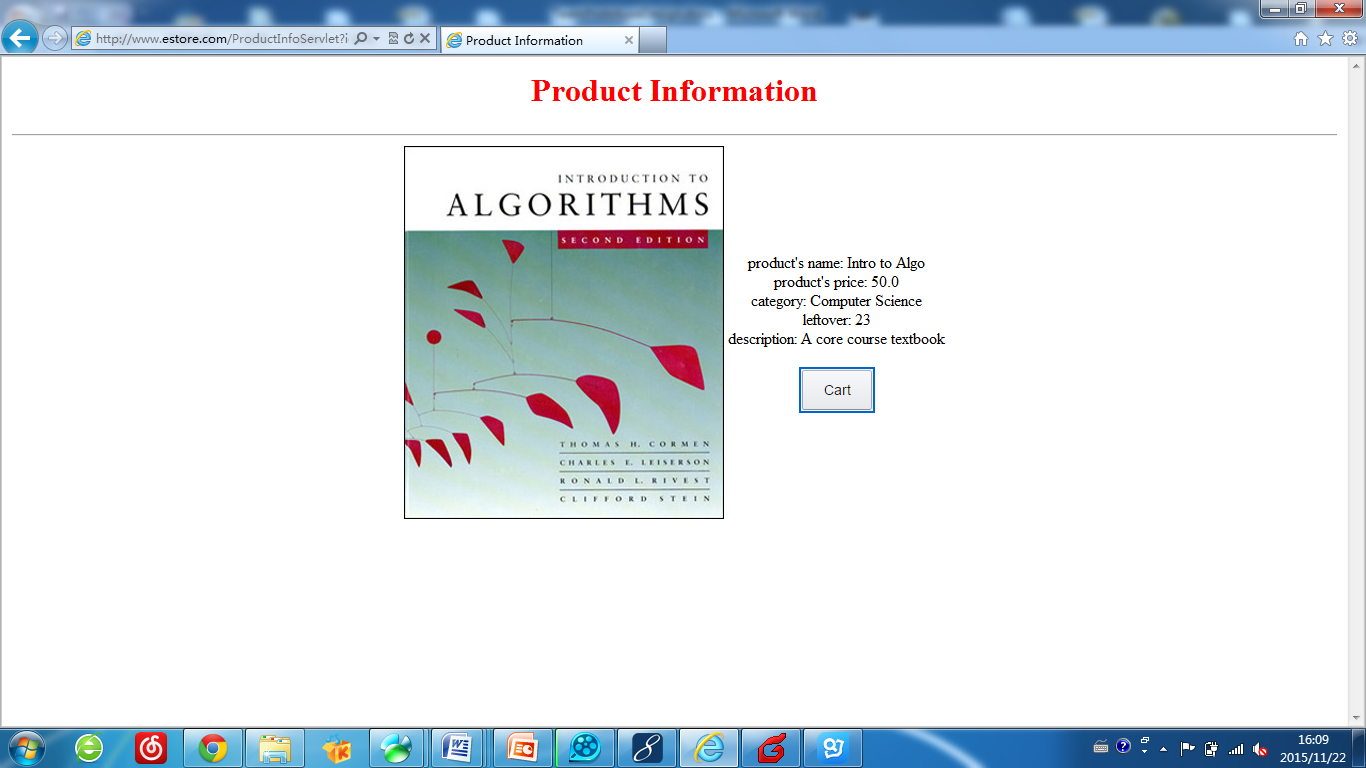


Any student can add their idle items to the system. If an identical item has already existed in the database, only inventory number of the product will change.

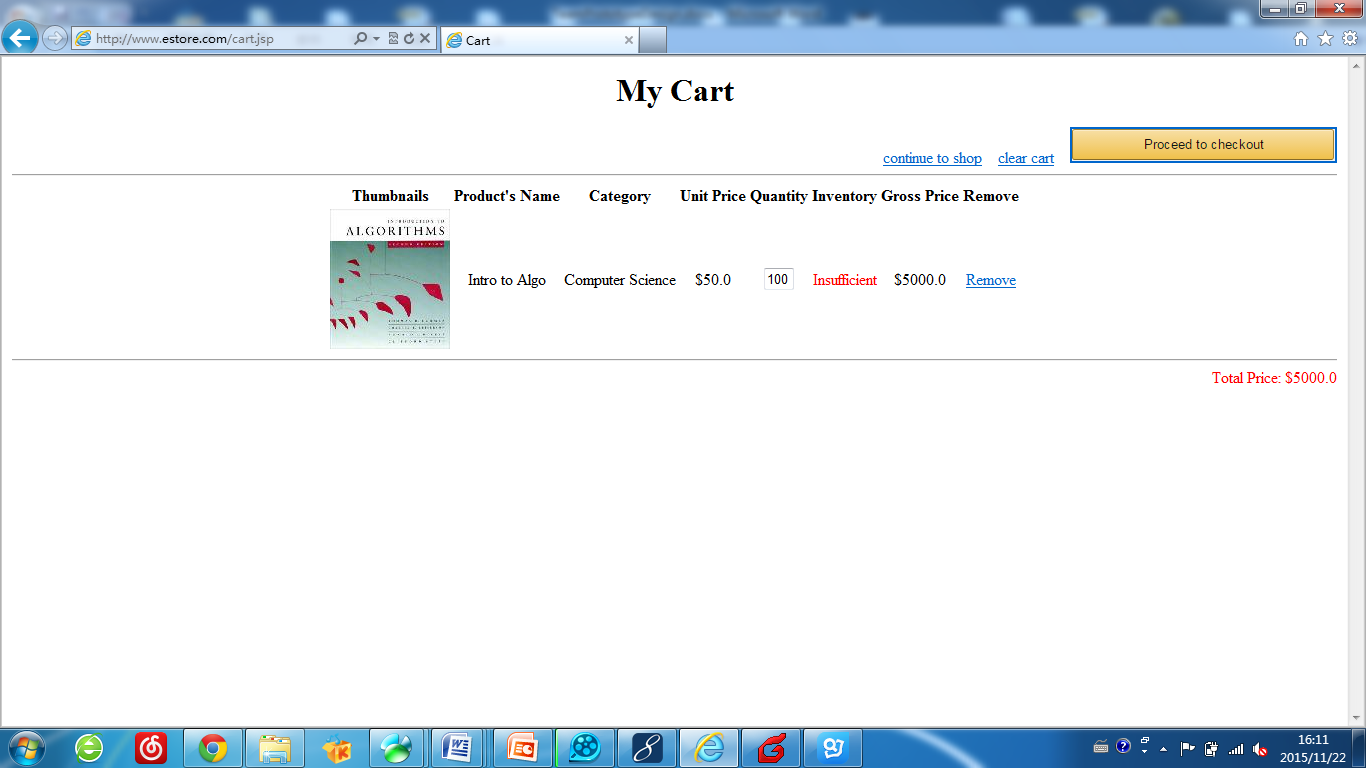
**Product Lists:**



Click the thumbnail will lead the user to a comprehensive product introduction page.

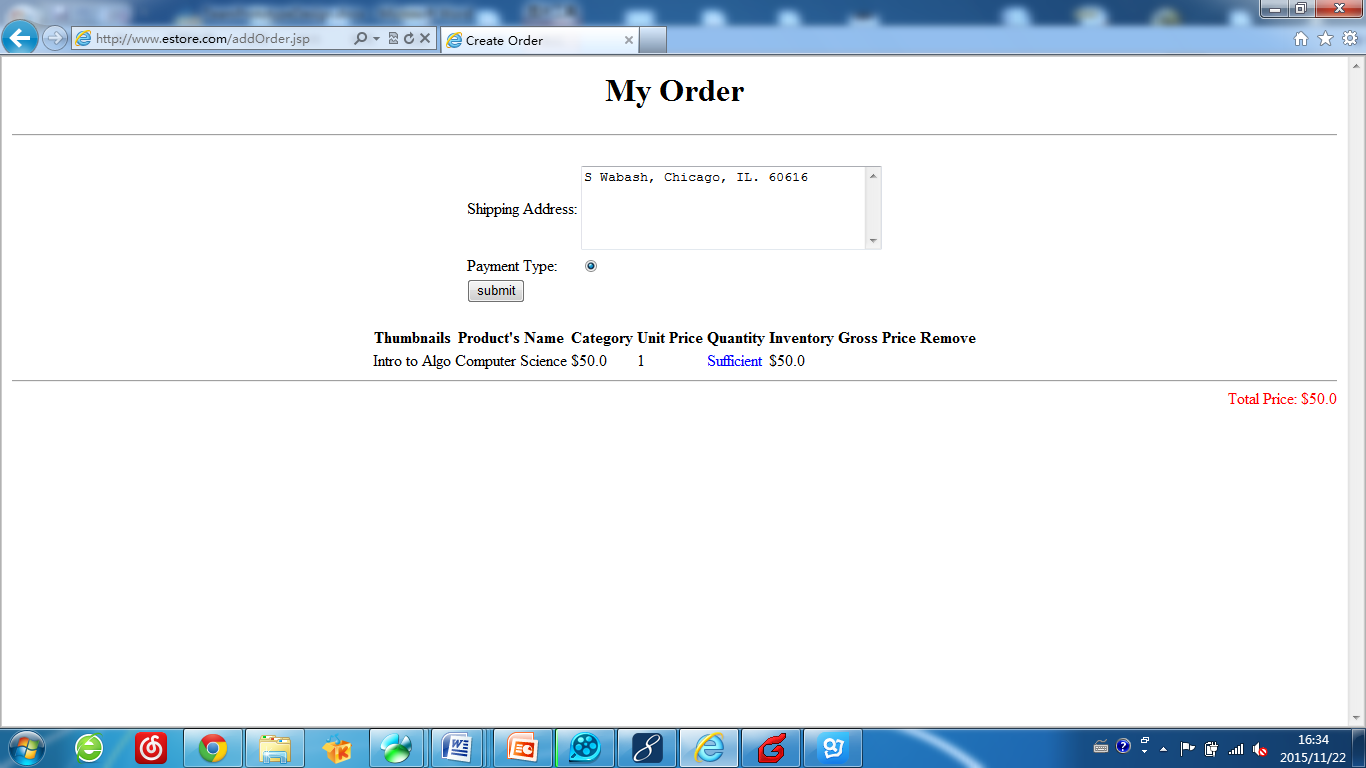


**Add items to cart:**

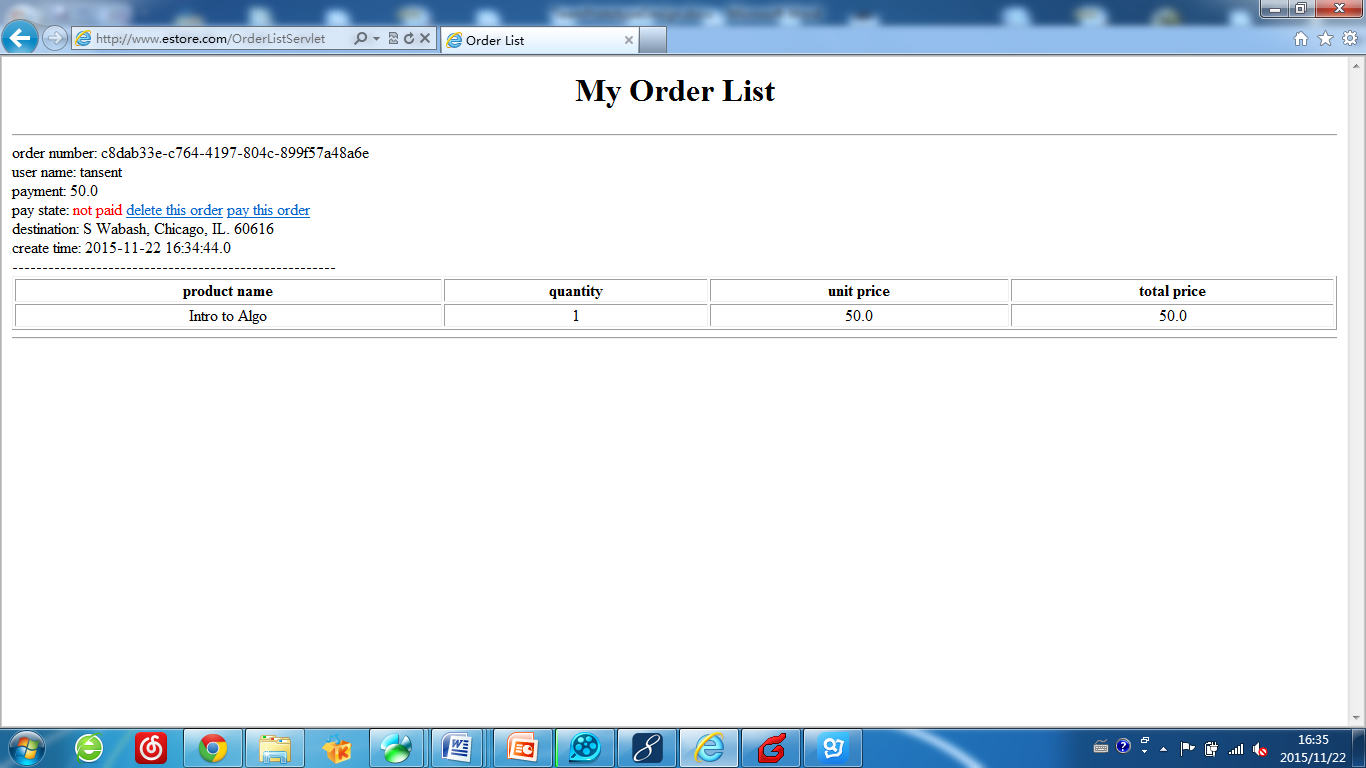


When entering items to the user’s cart, the system will verify the quantity user has entered. If the quantity is greater than the number in the database, the system should inhibit this transaction.

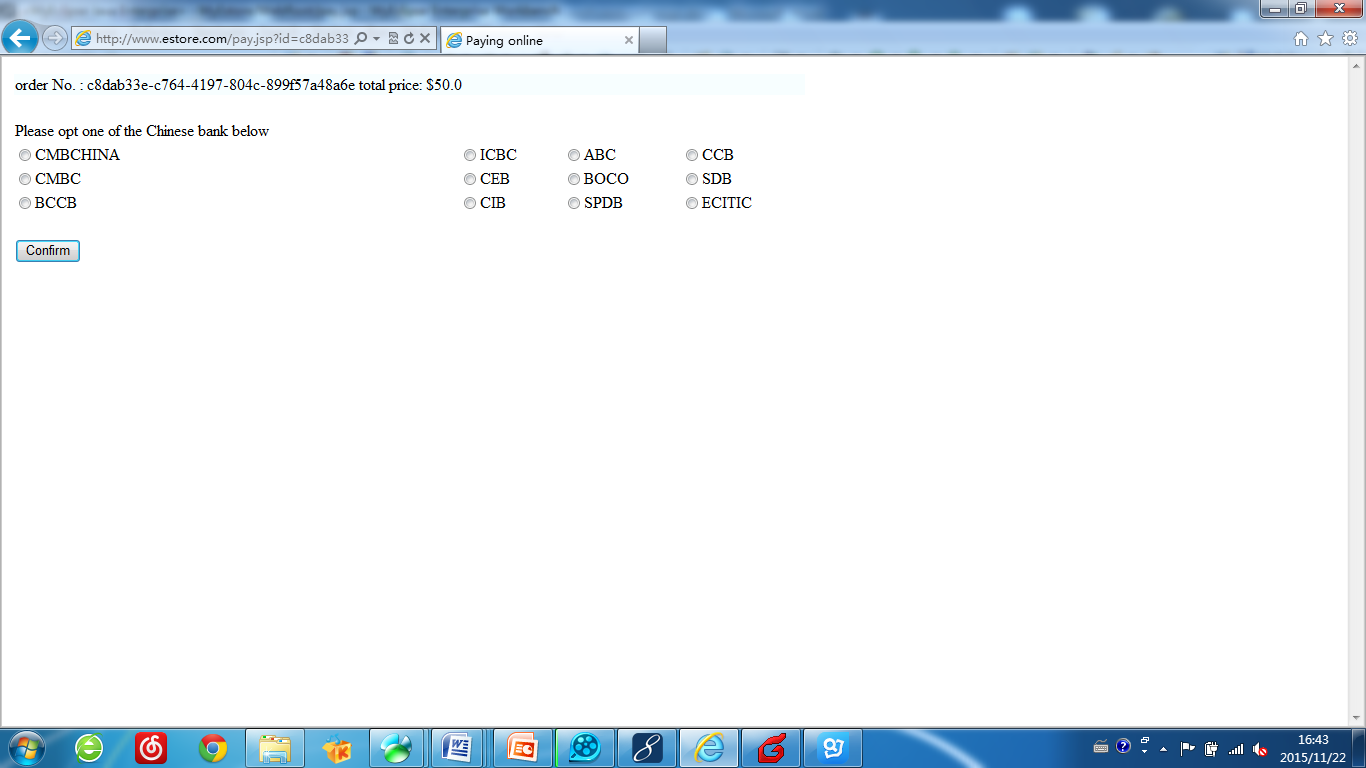
**Create Order:**



**View Orders:**

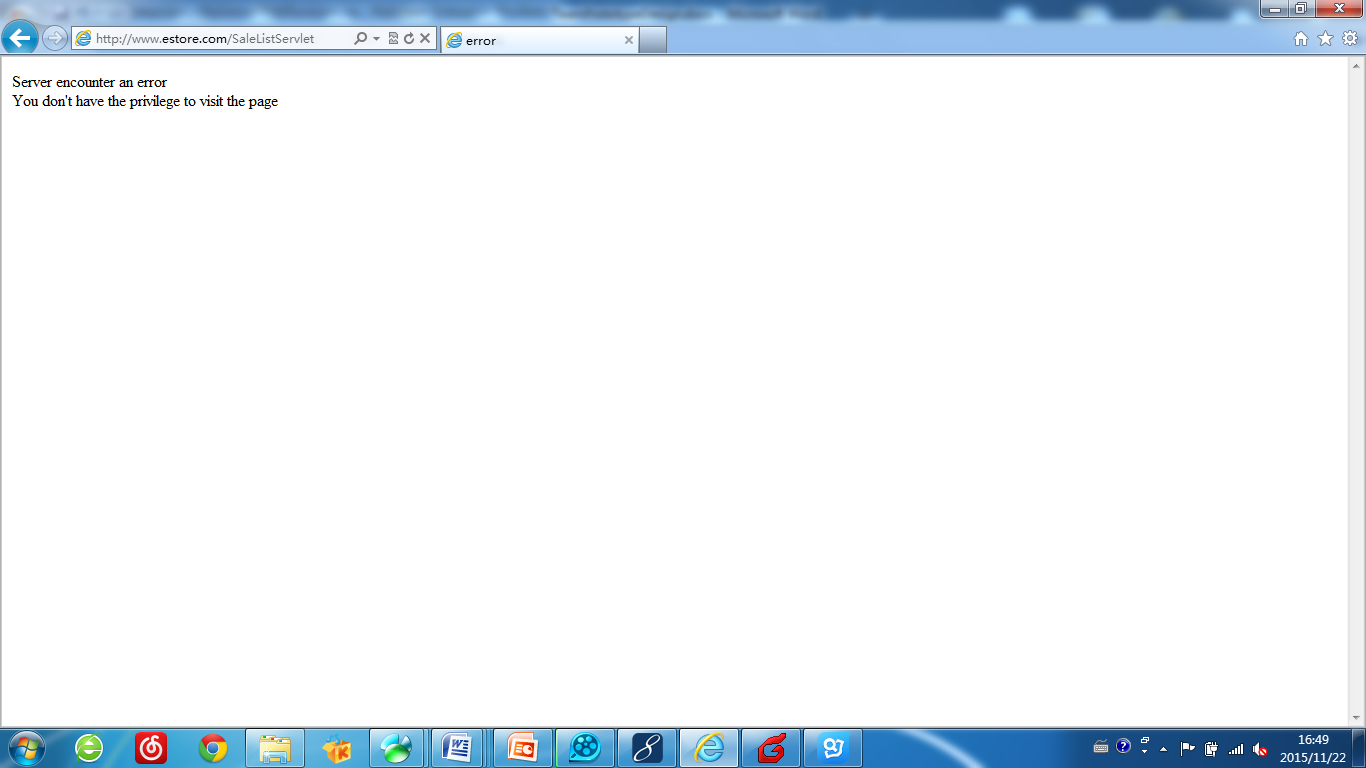


**Pay this order:**



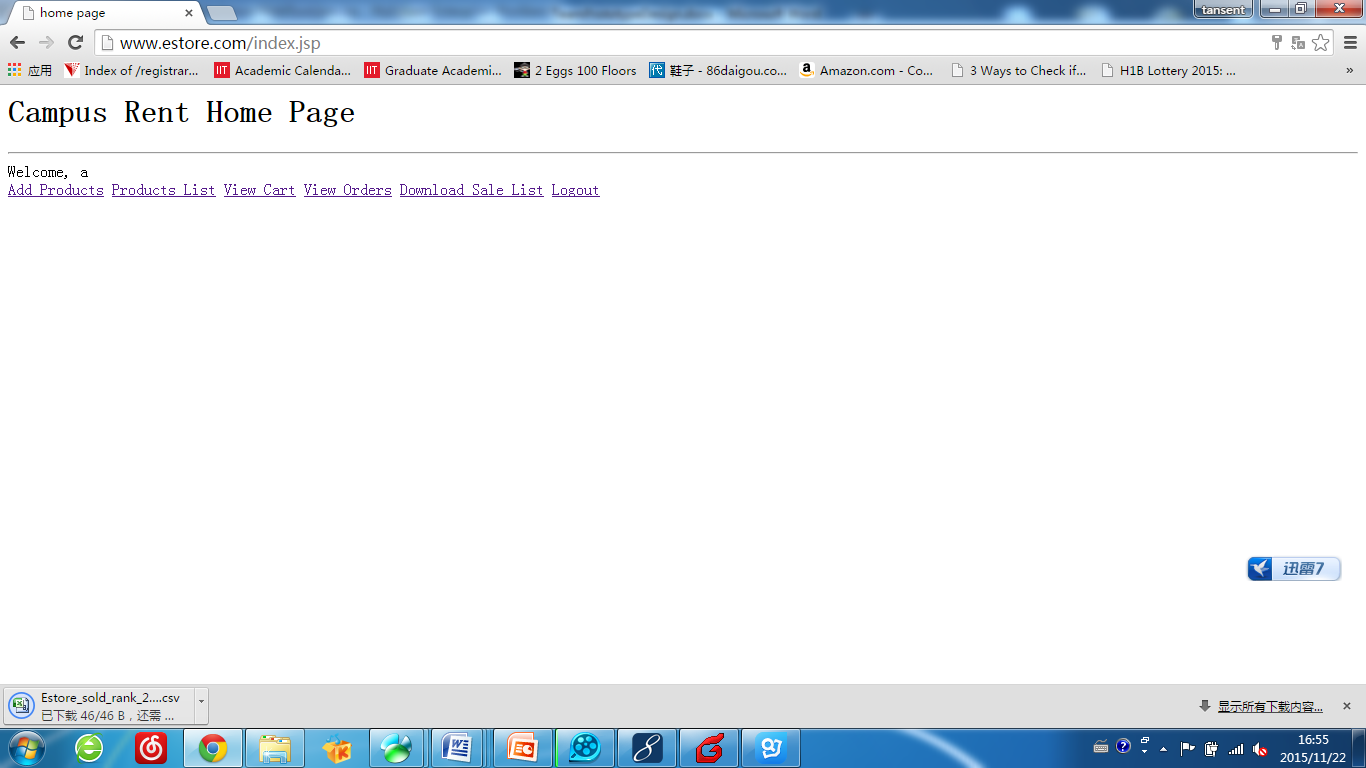
These banks’ payment methods have been encrypted into an interface. After the realization of the payment interface, there is no need to realize each individual bank’s payment interface.

**Download Sale List:**



If a regular user try to visit a page beyond his/her privileges, the system will prompt a hint says “no privilege”.

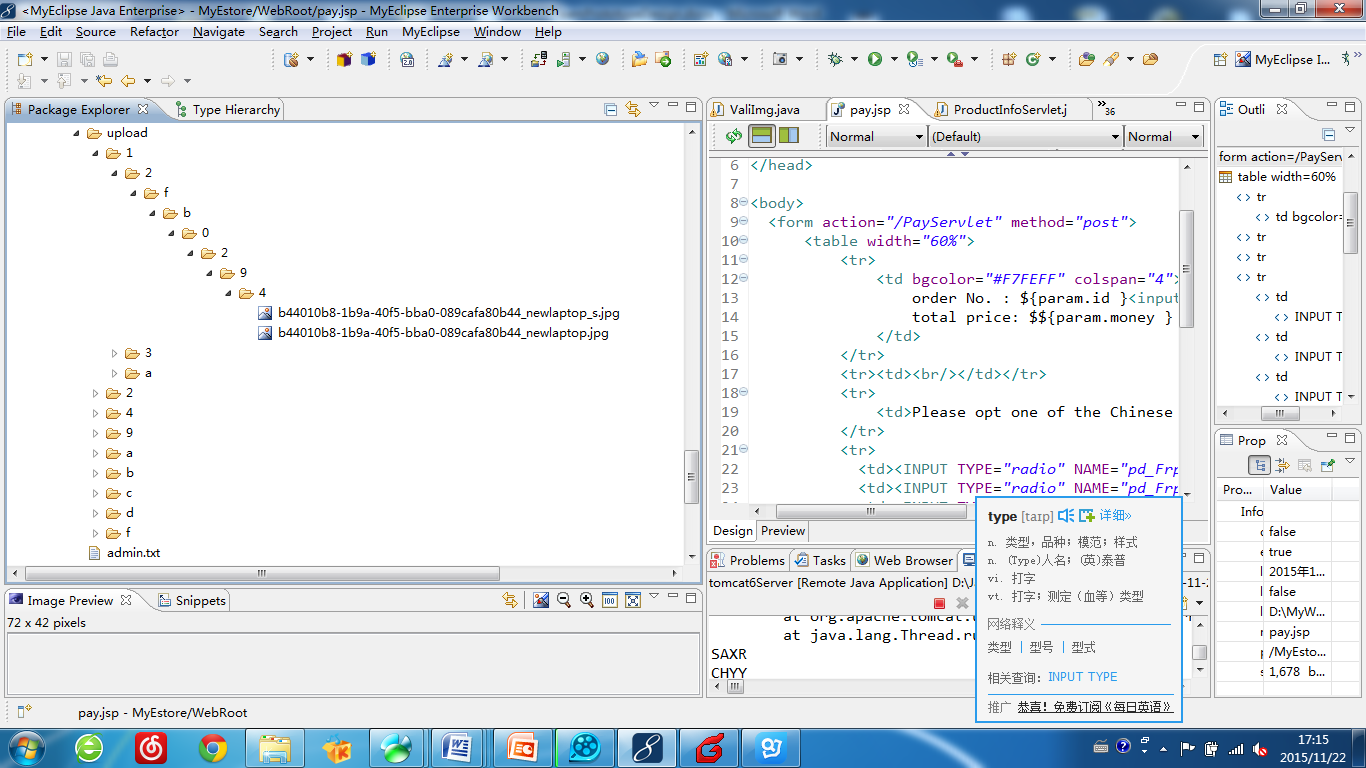
An administrator is able to download the sale list:



## Search Performance Improvement

If all items’ images are stored in a single folder, the system will encounter lags when the quantity of the images becomes large.

In this project, big files like images are stored randomly in different layers of files based on their names. In this way, the pressure of searching a file will reduce significantly and the time used for searching will be unnoticeable.



## Data Security

Before data transfers from front pages to the back end, a verification using javascript will check if the data is legitimate. If not, it will prompt some information to notify users to modify their information. For example, these are the logic code to check if passwords and emails are entered in a correct format.

//check if two passwords are identical

**var** psw1 = document.getElementsByName("password")[0].value;

**var** psw2 = document.getElementsByName("password2")[0].value;

**if**(psw1 != psw2){ //not equies()

document.getElementById("password2\_msg").innerHTML "<font color='red'>passwords are not identical</font>";

canSub = **false**;

}

//check if the email format is correct

//^\w+@\w+(\.\w+)+$ xxx@xxx.xxx.xxx

**var** email = document.getElementsByName("email")[0].value;

**if**(email !=**null** && email !="" &&!/^\w+@\w+(\.\w+)+$/.test(email)){

document.getElementById("email\_msg").innerHTML = "<font color='red'>format of email is not correct</font>";

canSub = **false**;

}

However, only placing a checking barrier in front pages is not sufficient, because front pages’ code can be exposed to users. Users may take advantage of the code to see the logic to circumvent the verification.

Backend verification should also be built:

Corresponding verification code for checking passwords’ and emails’ format:

**if**(form.getPassword()==**null** || form.getPassword().trim().equals("")){

token = **false**;

form.getError().put("password", "password can not be empty!");

}

**if**(form.getCpassword()==**null** || form.getCpassword().trim().equals("") || (!form.getPassword().equals(form.getCpassword()))){

token = **false**;

form.getError().put("Cpassword", "passwords do not match!");

}

**if**(form.getEmail()==**null** || form.getEmail().trim().equals("")){

token=**false**;

form.getError().put("email", "email can not be empty!");

}**else**{

**if**(!form.getEmail().matches("^([a-z0-9A-Z]+[-|\_|\\.]?)+[a-z0-9A-Z]@([a-z0-9A-Z]+(-[a-z0-9A-Z]+)?\\.)+[a-zA-Z]{2,}$")){

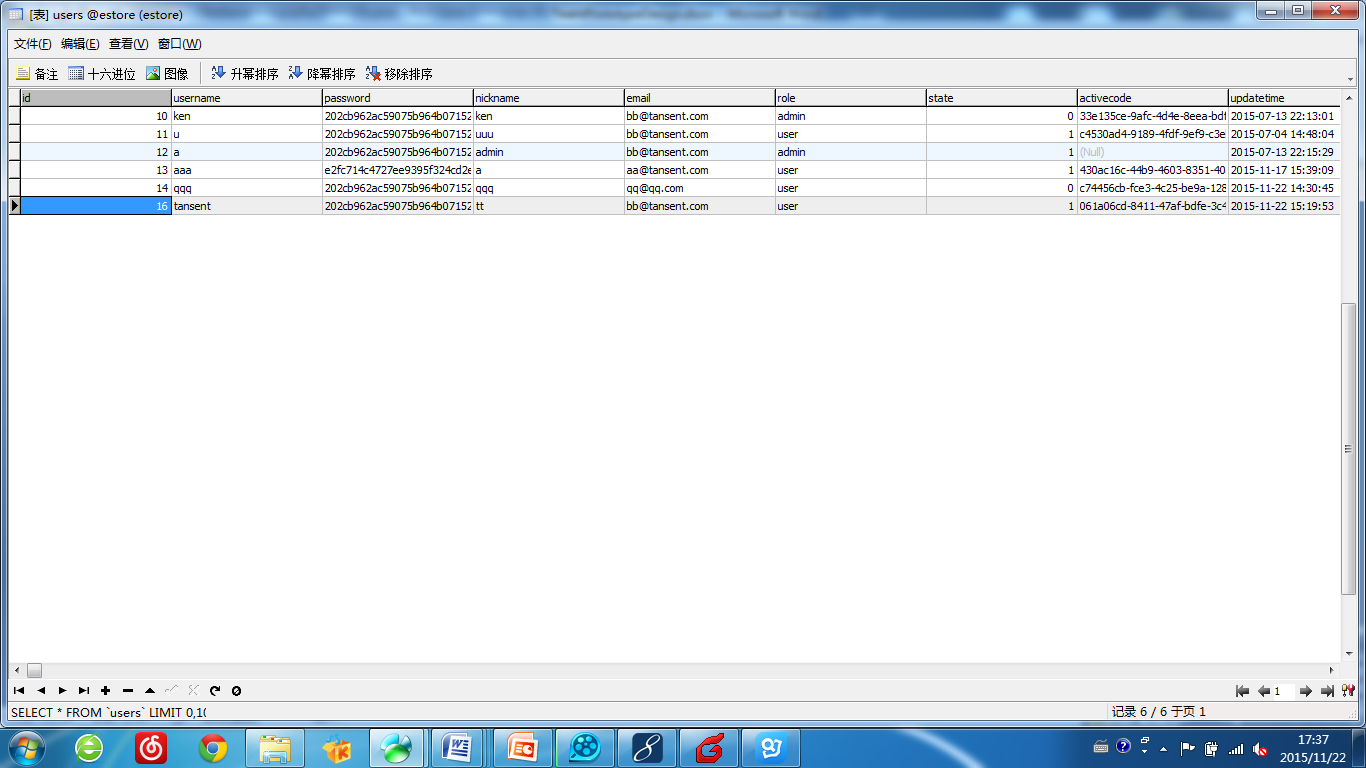
token=**false**;

form.getError().put("email", "email's format is incorrect!");

}

}

Private data such as passwords are encrypted even administrators who have access to the database do not understand what the passwords are. (The only method for a user who forget his/her password is to create a new password)



## Maintainability

The *factory mode* is used to segregate layers with different logic focus. For example, if the database of the project changes, thus SQL statements change as well. The places where require a change of code is only in DAO layer.

With Reflection technique, the system returns a class according to the current very name of the class:

**public** <T **extends** Dao> T getDao(Class<T> clazz){

String className = *prop*.getProperty(clazz.getSimpleName());

**try** {

**return** (T) Class.*forName*(className).newInstance(); //When execute Class.forName, java will look for all the libs to check if they are correct

} **catch** (Exception e) {

e.printStackTrace();

**throw** **new** RuntimeException(e);

}

}

Classes are reflected to a configuration file which controls the connection relationships for classes in different layers that are logical relevant.

