# Chapter 1 Introduction

#### Objective

The objective of College Information Management System is to allow the administrator of any organi- zation the ability to edit and find out the personal details of a student and allows the student to keep up to date his profile. It’ll also facilitate keeping all the records of students, such as their id, name, mailing address, phone number, DOB etc. So all the information about a student will be available in afew seconds. Overall, it’ll make Student Information an easier job for the administrator and the student of any organization.

The main purpose of this project is to illustrate the requirements of the project College Informa- tion Management System and is intended to help any organization to maintain and manage personal data. It is a comprehensive project developed from the ground up to fulfill the needs of colleges as they guide their students. This integrated information management system connects daily operations in the college environment ranging from Attendance management to communicational means among students and teachers. This reduces data error and ensures that information is always up-to-date throughoutthe college. It provides a single source of data repository for streamlining your processes and for all reporting purposes. It has a simple user interface and is intuitive. This insures that the users spend less time in learning the system and hence, increase their productivity. Efficient security features provide data privacy and hence, increase their productivity.

As we know that, a college consists of different departments, such as course departments, fees management, library, event management etc. Nowadays applications and uses of information technologies is increased as compared to before, each of these individual departments has its own computer system to do their own functionalities. By having one main system they can interact with each other from their respected system by having valid user id and password.

##### Purpose

The purpose is to design software for college database which contains up to date or accurate information of the college. That should improve efficiency and flexibility of college record management and to provide a common and or simple platform for everyone to access the student’s information. College Automation System consists of different modulessuch as student, faculty, admin etc. Our main purpose is to create a software which will manage the working of these different modules. The interconnectivity among modules reduces the time to perform different operational task.

##### Scope

College management is becoming a very essential component in education in this modern day age. With the help of College Automation System we can gather all the useful information needed to the management in few clicks. The College ERP system now computerizes all the details that are maintained manually. Once the details are fed into the system or computer there is no need for various persons to deal with separate sections.

##### Introduction to problem domain

As we know that, a college consists of different departments, such as course departments, fees manage- ment, library, event management etc. Nowadays applications and uses of information technologies is increased as compared to before, each of these individual departments has its own computer system to do their own functionalities. By having one main system they can interact with each other from their respected system by having valid user id and password.

**Chapter 2**

**Design**

Various Design concepts and processes were applied to this project. Following concepts like separation of concerns, the software is divided into individual modules that are functionally independent and incorporates information hiding. The software is divided into 3 modules which are students, teachers and administrators. We shall look at each module in detail.

##### Entity Relationship Diagram

An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define its properties.

By defining the entities, their attributes, and showing the relationships between them, an ER diagram illustrates the logical structure of databases. ER diagrams are used tosketch out the design of a database.

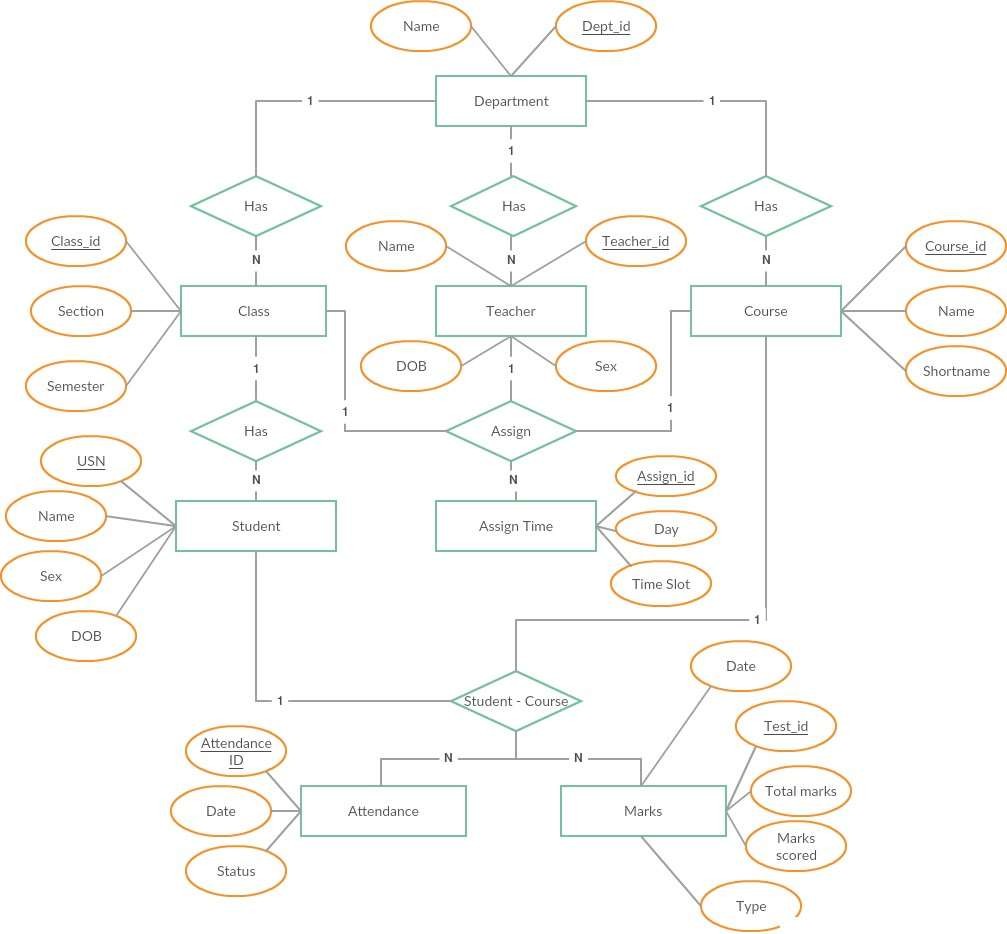


Figure 3: Entity Relationship diagram of college ERP

* 1. **Data Flow Diagram**

A use case diagram at its simplest is a representation of a user’s interaction with thesystem that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses.

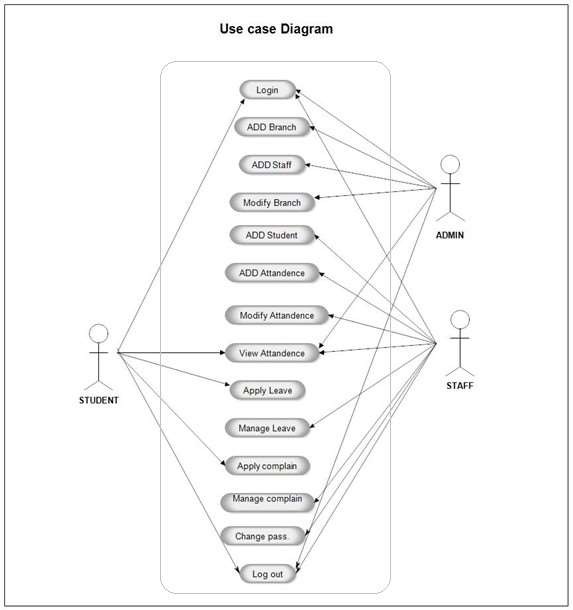


Figure 1: Use Case Diagram of college ERP

#### Class Diagram

The class diagram states the different classes involved in the software. For each class,a set of attributes and method are included. The relationship between the classes arealso specified. For example, the teacher class has the attributes Id, name, phone no, address and methods such as marking attendance, declaring marks and preparing report cards. Each instance of the teacher class belongs to a department. This is specified bythe relationship between Teacher and Department classes.

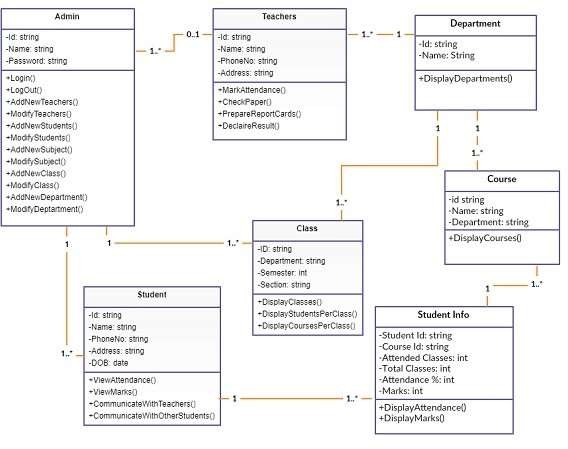


Figure 2: Class diagram of college ERP

#### Sequence

Each student belongs to a class identified by semester and section. Each class belongs to a department and are assigned a set of courses. Therefore, these courses are common to all students of that class. The students are given a unique username and password to login. Each of them will have a different view. These views are described below.

##### Student information

Each student can view only their own personal information. This includes their personal details like name, phone no, address etc. Also, they can view the courses they are enrolled in and the attendance, marks of each of those.

##### Attendance information

Attendance for each course will be displayed. This includes the number of attended classes and the attendance percentage. If the attendance percentage if below a specified threshold, say 75%, It will be marked in red otherwise it be in green. There will also be a day wise attendance view for each course which shows the dateand status. This will be presented in a calender format.

##### Marks information

There will be 5 events and 1 semester end examination for each course. The marks for each of these will be provided in the ERP system.

##### Notifications and events

This section is common to all students. Notification are messages from the admin such as declaration of holidays, test time-table etc. The events and their details are specified here.

#### Flow Chart

Each teacher belongs to a department and are assigned to classes with a course. Teachers will also have a username and password to login. The different views for teachers are described below.

##### Information

The teachers will have access to information regarding the courses and classes they are assigned to. Details of the courses include the credits, the syllabus plan. Details of the class include the department, semester, section and the list of students in

each class. The teacher will also have access to information of students who belong to the same class as as the teacher.

##### Attendance

The teacher has the ability to add and also edit the attendance of each student. For entering the attendance, they will be given the list of students in each class and they can enter the attendance of the whole class on a day to day basis. There will be two radio buttons next to each student name, one for present and the other for absent. There will also be an option for extra classes. Teachers can edit the attendance of each student either for each student individually or for the wholeclass.

##### Marks

The teacher can enter the marks for the 5 events and 1 SEE for each course they are assigned. They also have the ability to edit the marks in case of any changes. Reports such as the report card including all the marks and CGPA of a student can be generated.

#### Administrator

* + - * The administrator will have access to all the information in the different tables in the database. They will access to all the tables in a list form. They will be able to add a entry in any table and also edit them. The design of the view for the admin will provide a modular interface so that querying the tables will be optimized. They will be provided with search and filter features so that they can access data efficiently.

# Chapter 3 Implementation

The college ERP system has three main user classes. These include the students, teachers and ad- ministrator. This section will explain in detail all the features and the working of those for each user class.

#### Student

##### Login

Each student in the college is assigned a unique username and password by the administrator. The user- name is the same as their USN and so is the password. They may change it later according to their wish.

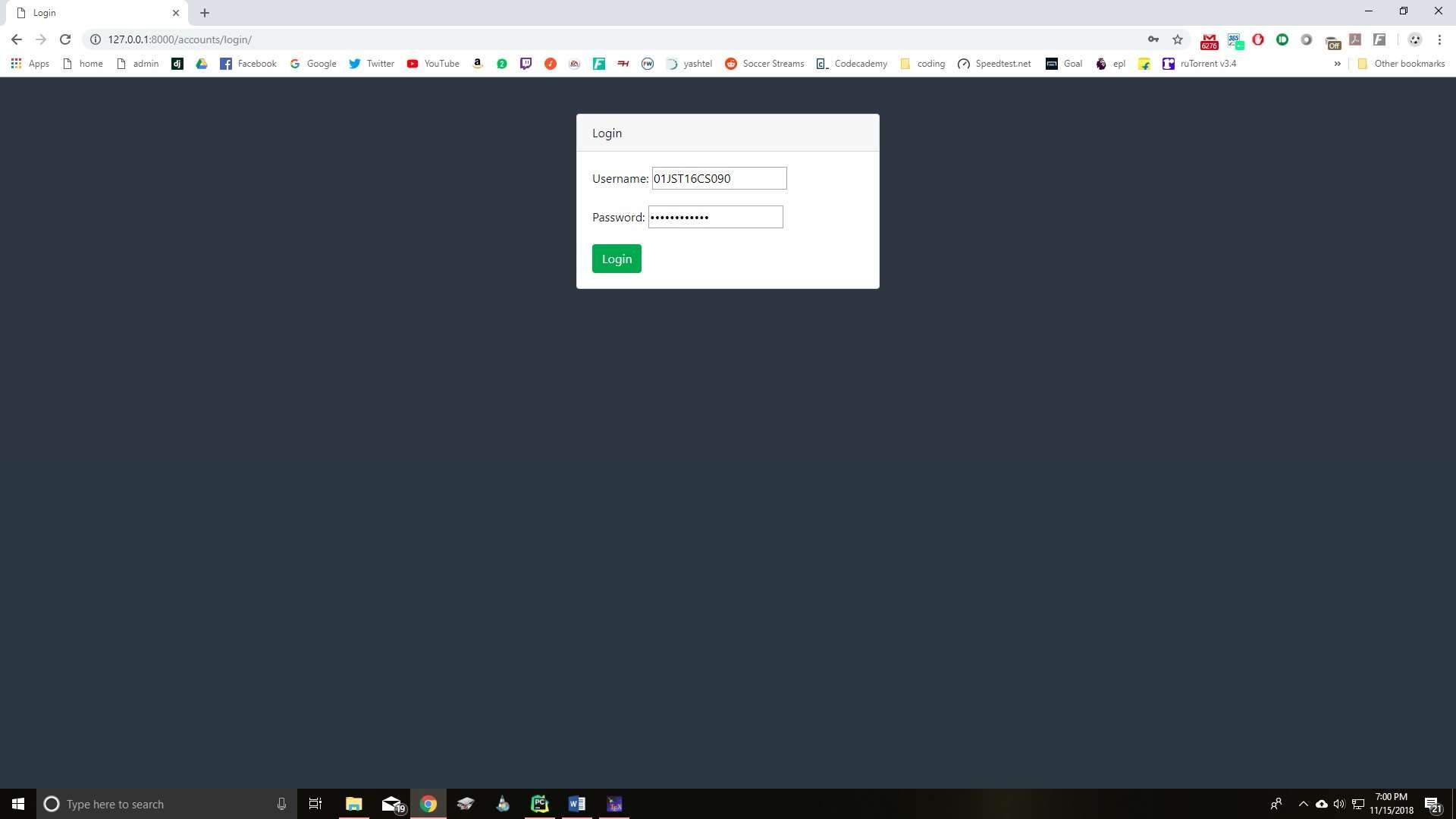


Figure 4.1: Student Login Page

##### Homepage

After successful login, the student is presented a homepage with their main sections, attendance, marks and timetable. In the attendance section the student can view their attendance status which includes the total classes, attended classes and the attendance percentage for each of their courses.

In the marks section, the student can view the marks for each of their courses out of 20 for 3 internal assessments, 2 events. Also, the semester end examination for 100 marks. Lastly, the timetable provides the classes assigned to that student and day and time of each in a tabular form.

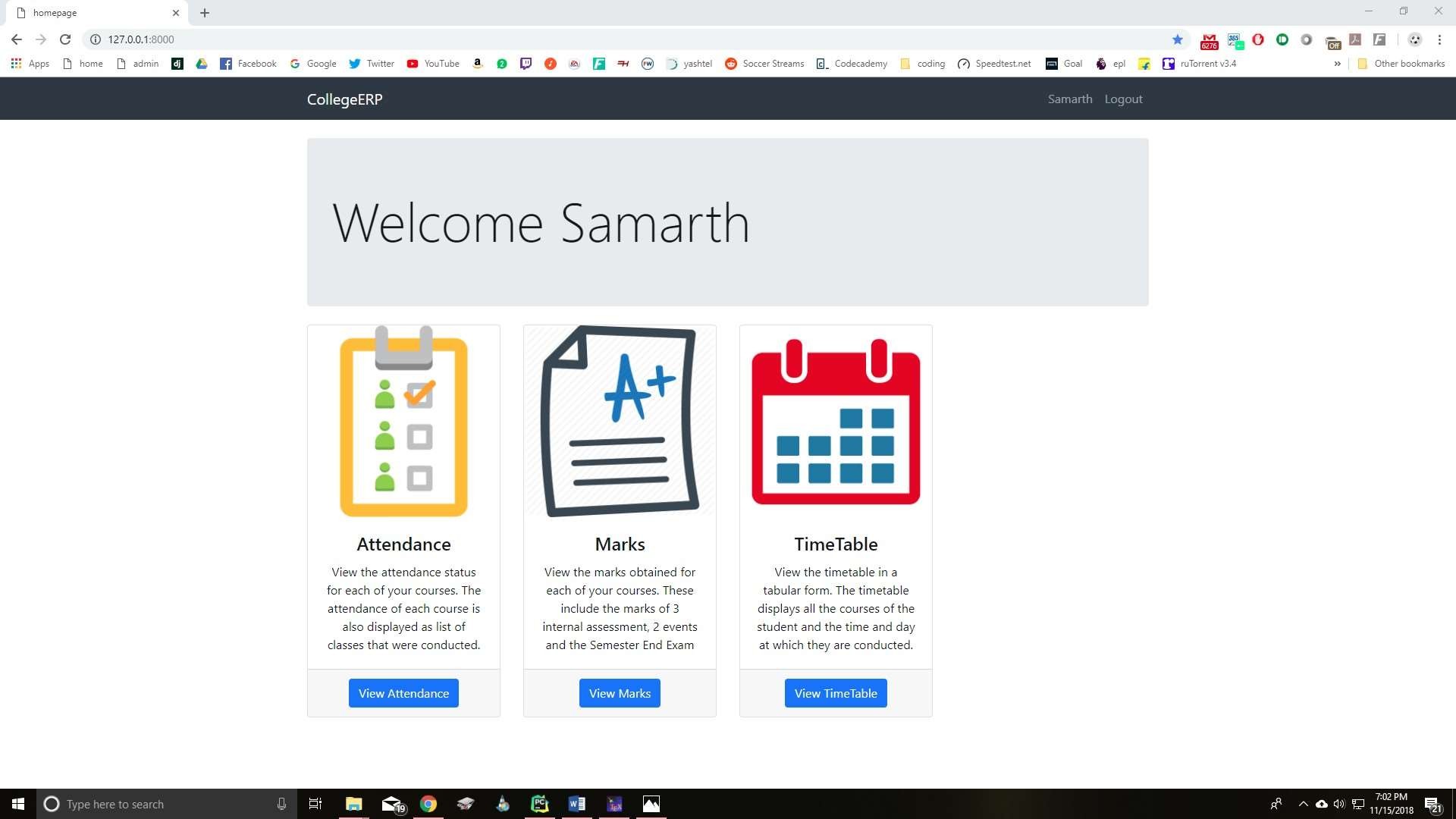


Figure 4.2: Student Home Page

##### Attendance

On the attendance page, there is a list of courses that is dependent on each student. For each course, the course id and name are display along with the attended classes, total classes and the attendance percentage for that particular course. If the attendance percentage is below 75 for any course, it is displayed in red denoting shortage of attendance, otherwise it is green. If there is any shortage, it specifies the number of classes to attend to make up for it. If you click on each course, it takes you tothe attendance detail page.

**Attendance Detail**

This page displays more details for the attendance in each course. For each the course, there is a list of classes conducted and each is marked with the date, day and whether the student was present or absent on that particular date.

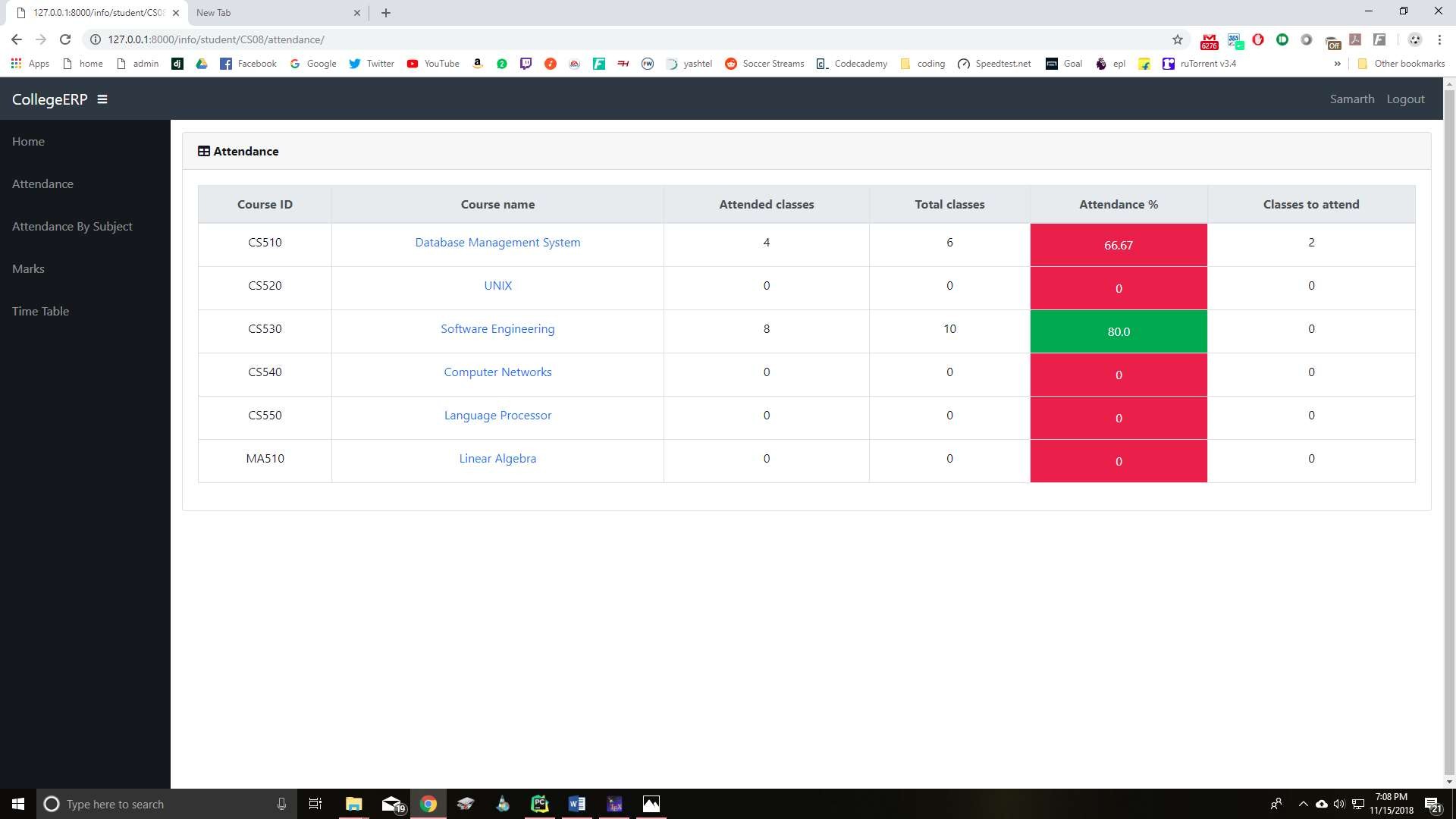


Figure 4.3: Student Attendance Page

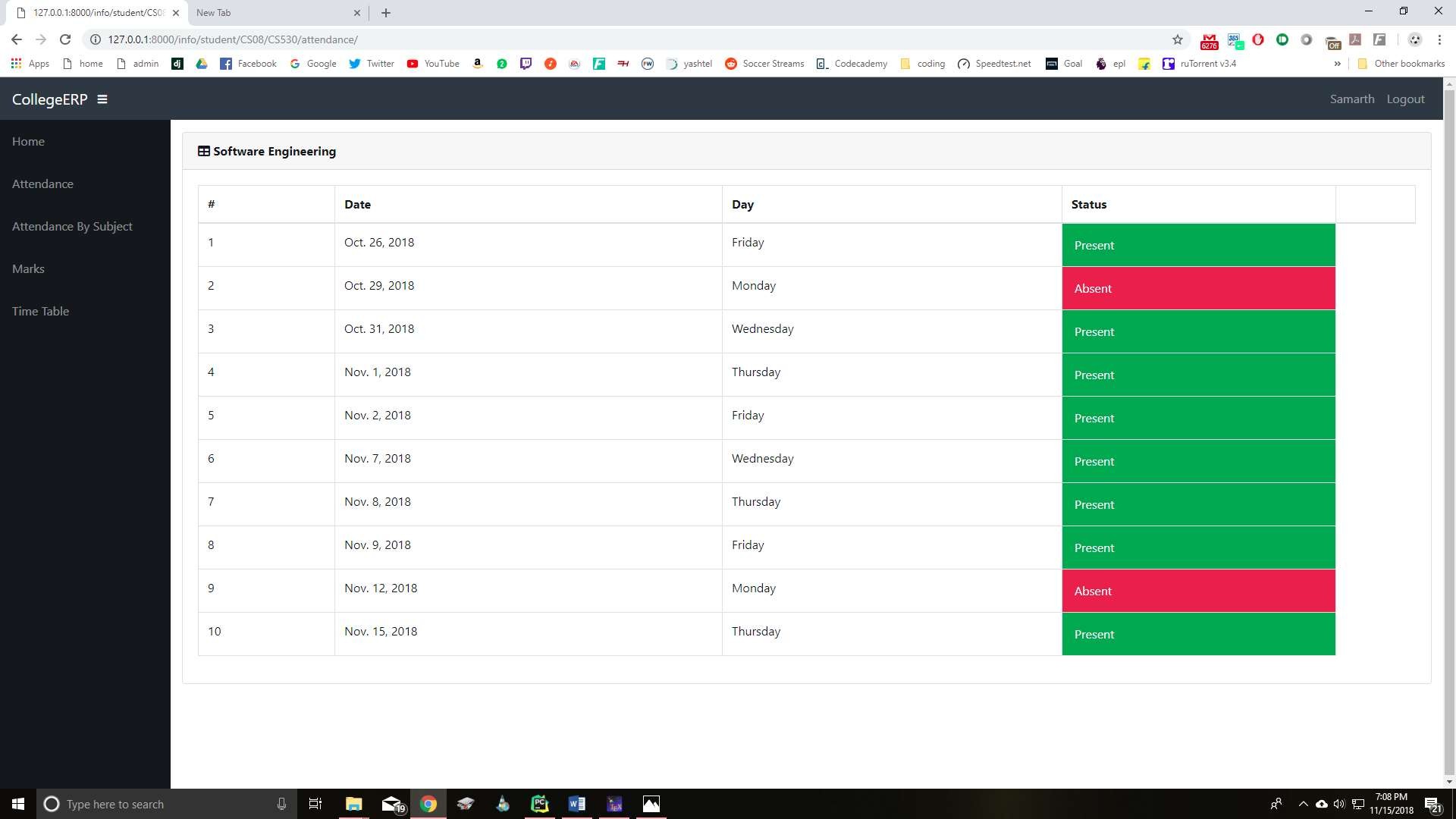


Figure 4.4: Student Attendance Detail Page

##### Marks

The Marks page is a table with an entry for each of their courses. The course id and name are specified along the marks obtained in each of the tests and exams. The tests include 3 internal assessments with marks obtained out of a total of 20, 2 events such as project, assignment, quiz etc., with marks out of

1. Lastly, one semester end exam with marks out of 100.

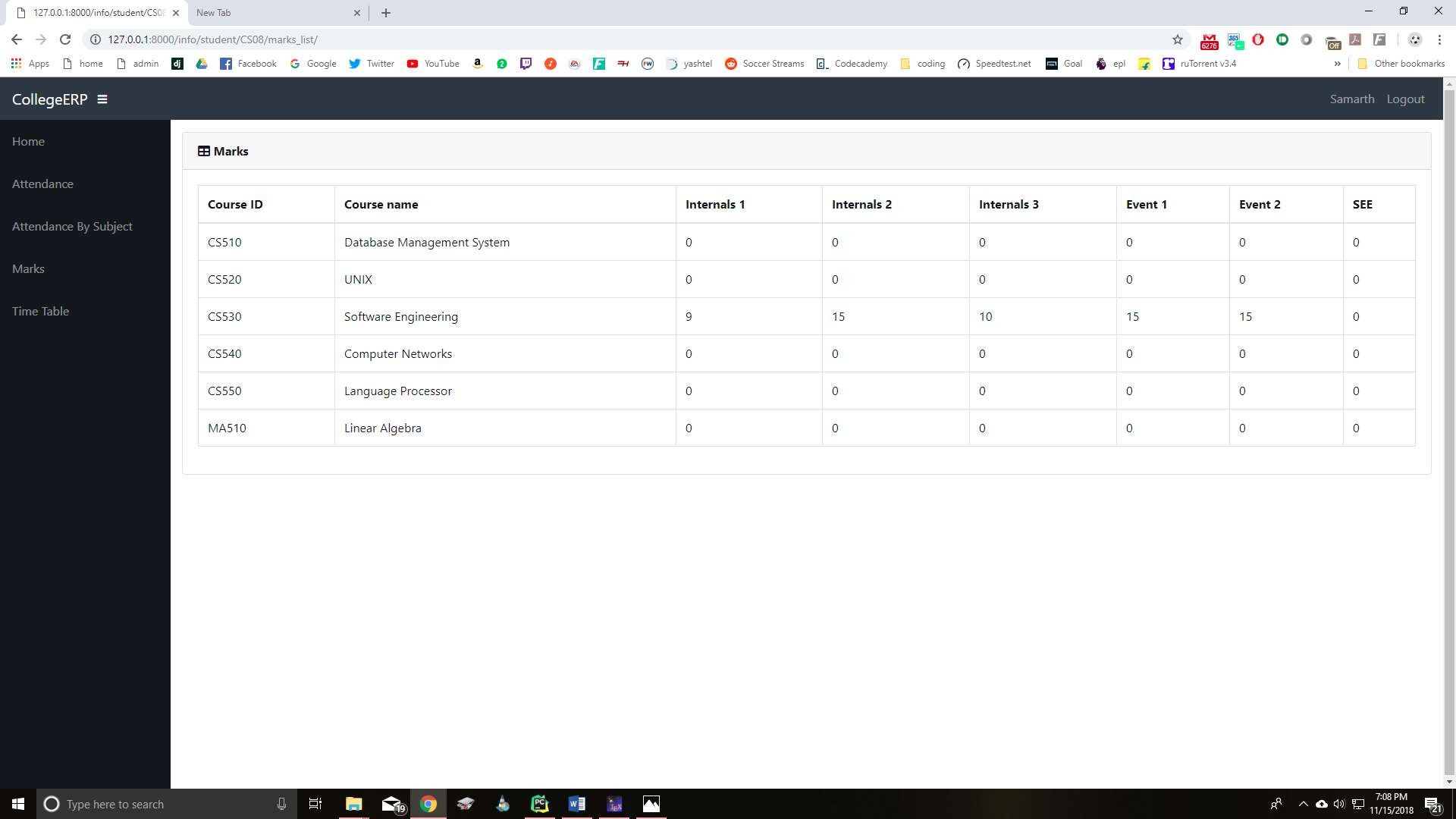


Figure 4.5: Student Marks Page

##### Timetable

This page is a table which lists the day and timings of each of the classes assigned to the student. The row headers are the days of the week and the column headers are the time slots. So, for each day, it specifies the classes in the time slots. The timetable is generated automatically from the assign table, which is a table containing the information of all the teachers assigned to a class with a course and the timings the classes.



Figure 4.6: Student Timetable

#### Teacher

##### Login

Each teacher in the college is assigned a unique username and password by the administrator. The username is their teacher ID and the same for password. The teacher may change the password later.

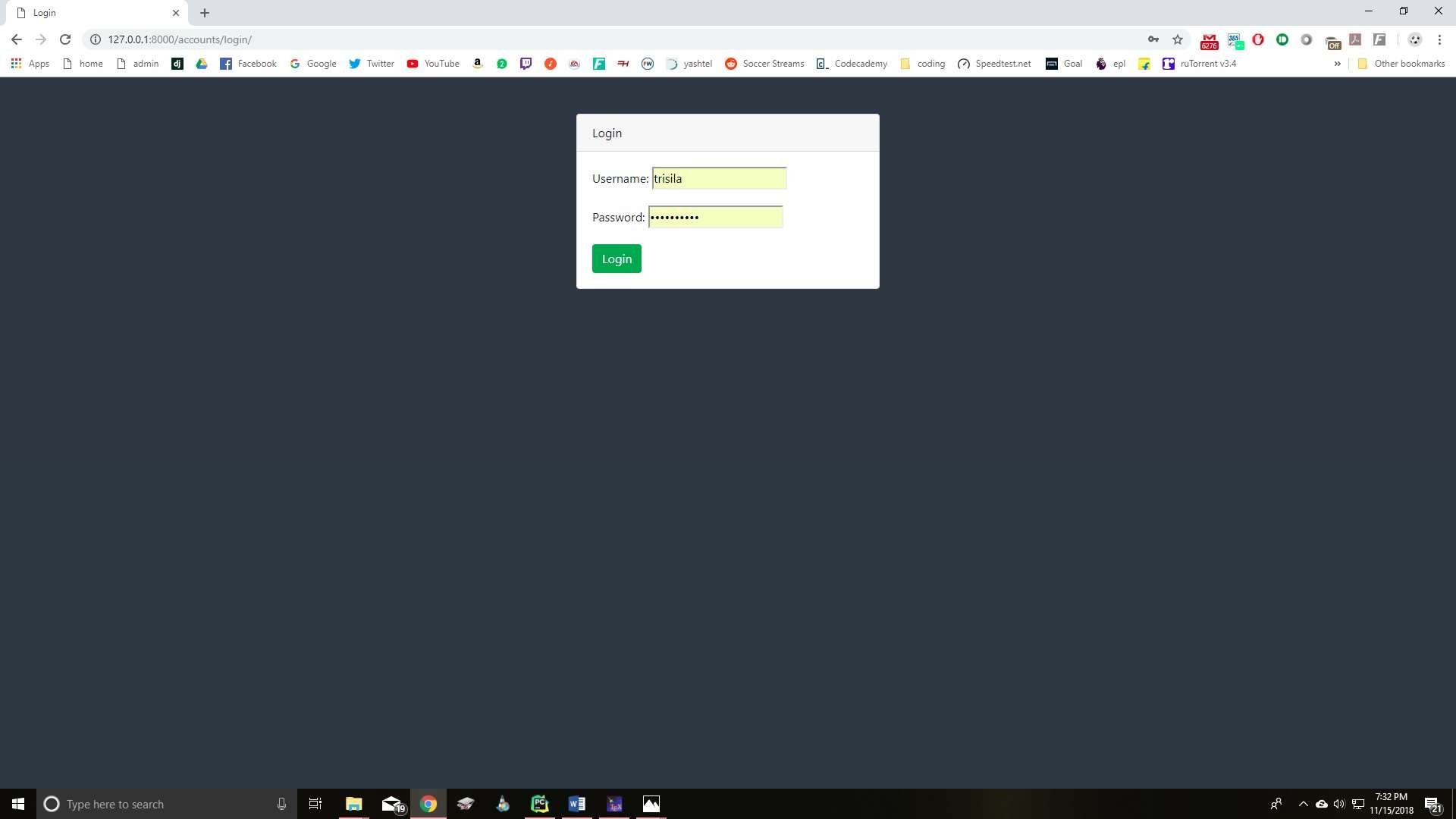


Figure 4.7: Teacher Login

##### Homepage

After successful login, the student is presented a homepage with their main sections, attendance, marks, timetable and reports. In the attendance section, the teacher can enter the attendance of their respective students for the days on which classes were conducted. There is a provision to enter extra classes and view/edit the attendance of each individual student. In the marks section, the teacher may enter the marks for 3 internals, 2 events and 1 SEE for each student. They can also edit each of the entered marks. The timetable provides the classes assigned to the teacher with the day and timings in a tabular form. Lastly, the teacher can generate reports for each of their assigned class.

##### Attendance

There is a list of all the class assigned to teacher. So, for each class there are 3 actions available. They are,

**Enter Attendance**

On this page, the classes scheduled or conducted is listed in the form of a list. Initially, all the scheduled classes will be listed from the start of the semester to the current date. Thus, if there is class scheduled for today, it will automatically appear on top of the list. If the attendance of any day is not marked itwill be red, otherwise green if marked. Classes can also be cancelled which will make that date as yellow. While entering the attendance, the list of students in that class is listed and there are two options next

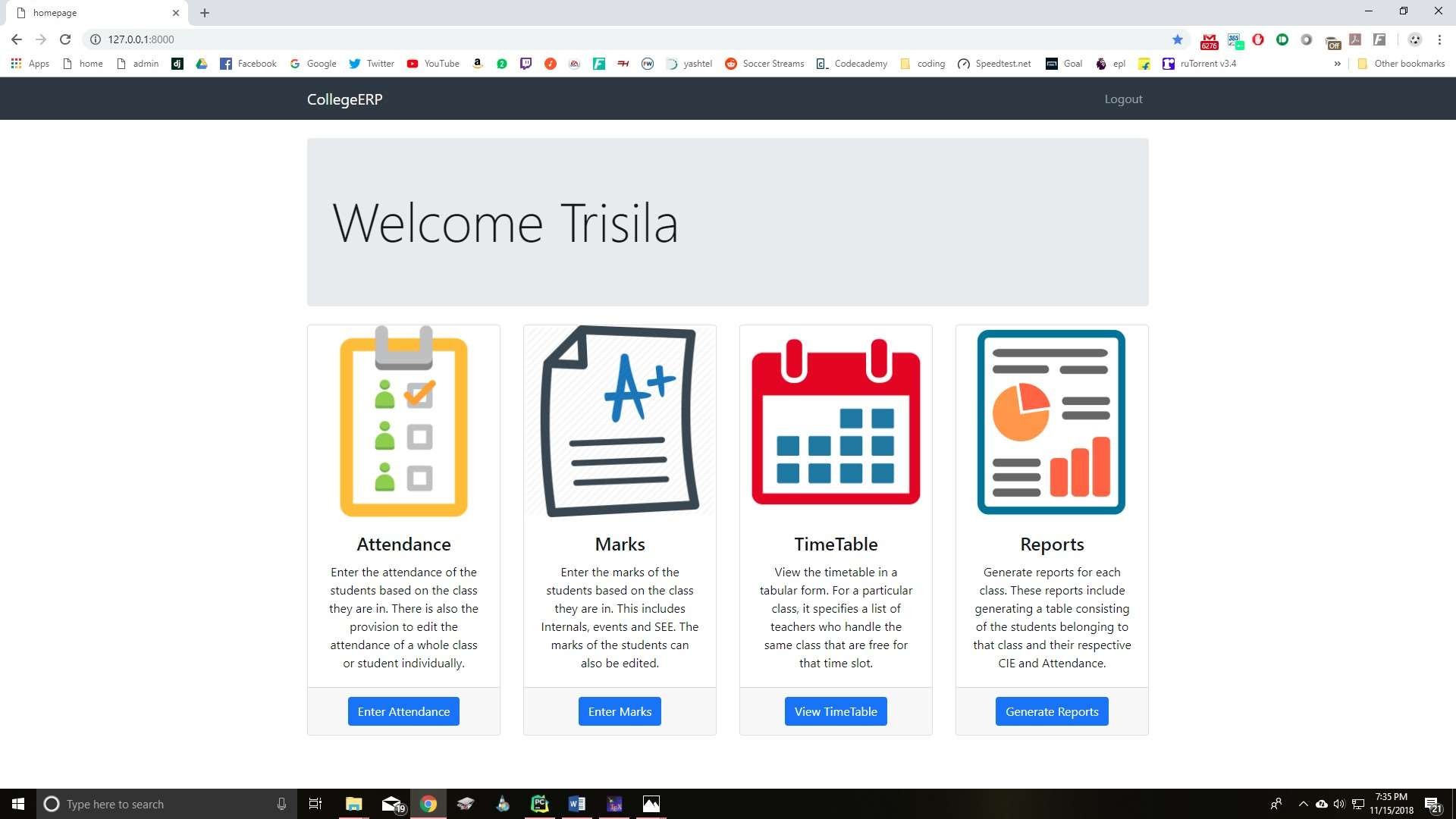


Figure 4.8: Teacher homepage

to each. These options are in the form of a radio button for present and absent. All the buttons are initially marked as present and the teacher just needs to change for the absent students.

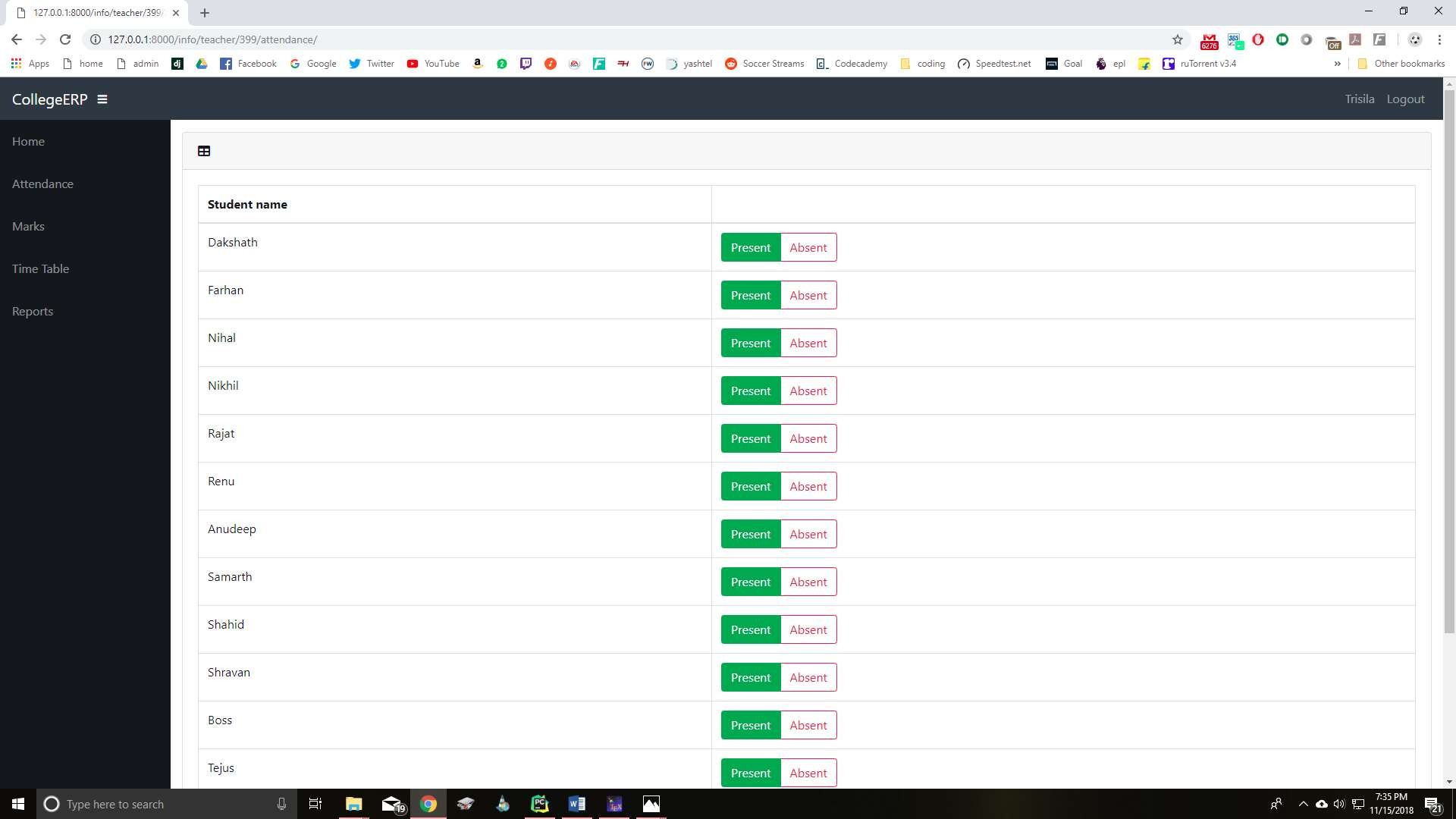


Figure 4.9: Entering attendance

**Edit Attendance**

After entering attendance, the teacher can also edit it. It is similar to screen for entering attendance, only the entered attendance is saved and display. The teacher can change the appropriate attendance and save it.

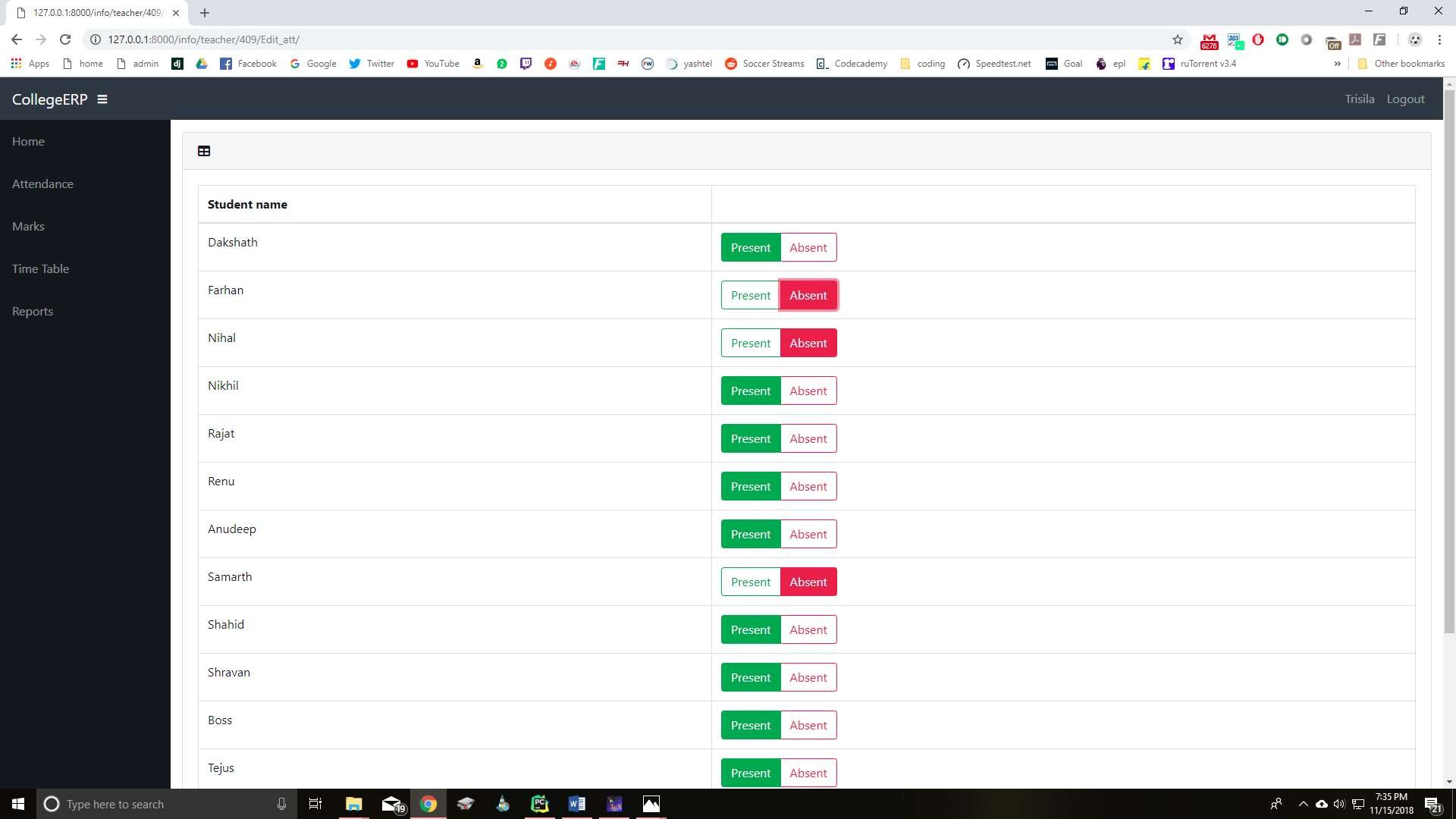


Figure 4.10: Editing attendance

**Extra Class**

If a teacher has taken a class other than at the scheduled timings, they may enter the attendance for that as well. While entering the extra class, the teacher just needs to specify the date it was conducted and enter the attendance of each of the students. After submitting extra class, it will appear in the listof conducted classes and thus, it can be edited.

**Student Attendance**

For each assigned class, the teacher can view the attendance status of the list of students. The number of attended classes, total number of classes conducted and the attendance percentage is displayed. If the attendance percentage of any of the students is below 75, it will be displayed in red. Thus, the teacher may easily find the list of students not eligible to take a test.

**Student Attendance Details**

The teacher can view the attendance detail of all their assigned students individually. That is, for all the conducted classes, it will display whether that student was present or absent. The teacher can also edit the attendance of each student individually by changing the attendance status for each conducted class.

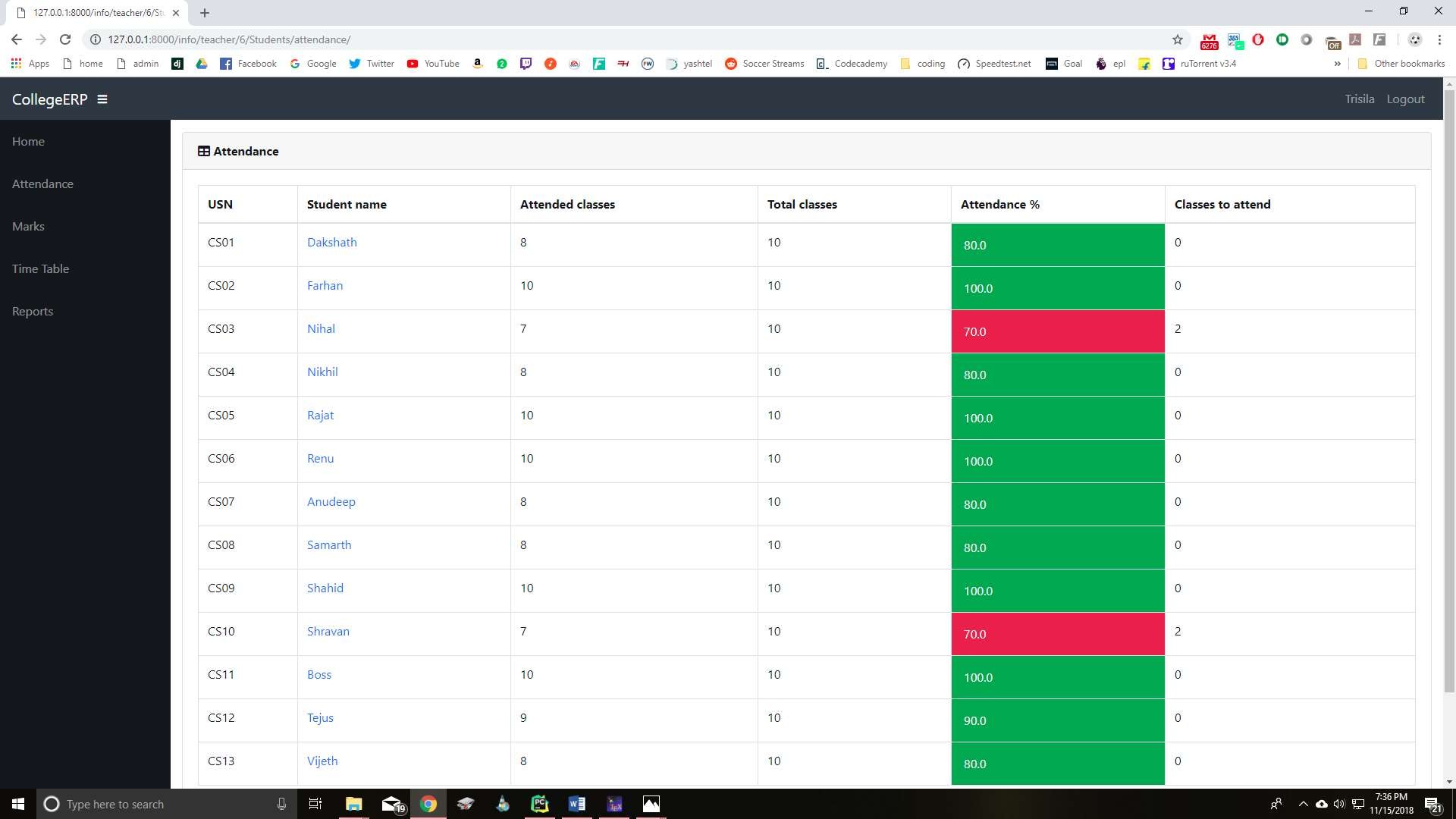


Figure 4.11: Attendance of students in a class

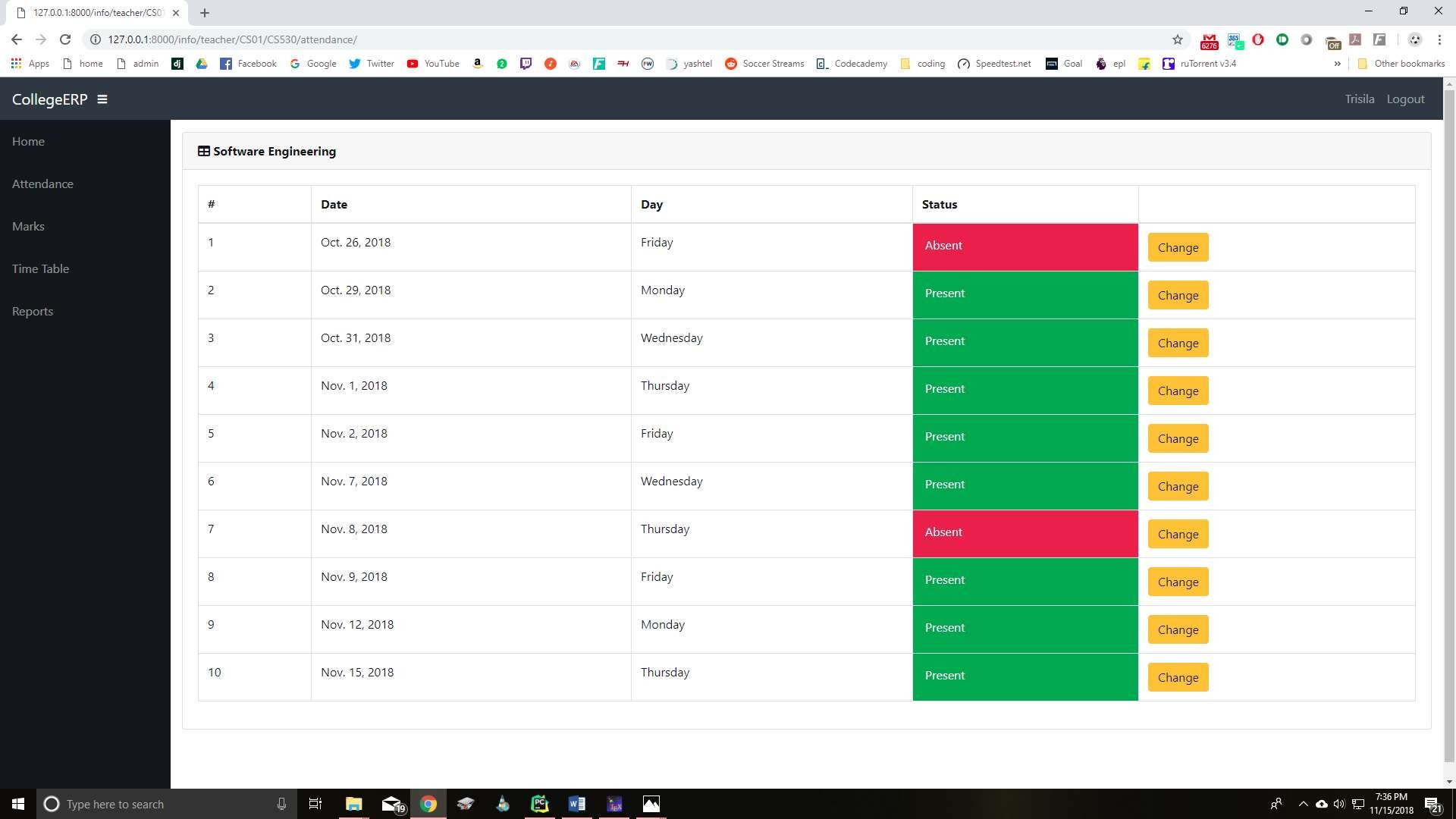


Figure 4.12: Attendance details of an individual student

##### Marks

On this page, the list of classes assigned to the teacher are displayed along with two actions for each class. These actions are,

**Enter Marks**

On this page, the teacher can enter the marks for 3 internal assessments, 2 events and one semester end exam. Initially all of them are marked red to denote that the marks have not been entered yet. Oncethe marks for a test is entered, it turns green. While entering the marks for a particular test, the listof students in that class is listed and marks can be entered for all of them and submitted. Once, themarks are submitted, the students can view their respective marks. Incase if there is a need to change the marks of any student, it is possible to edit the marks.

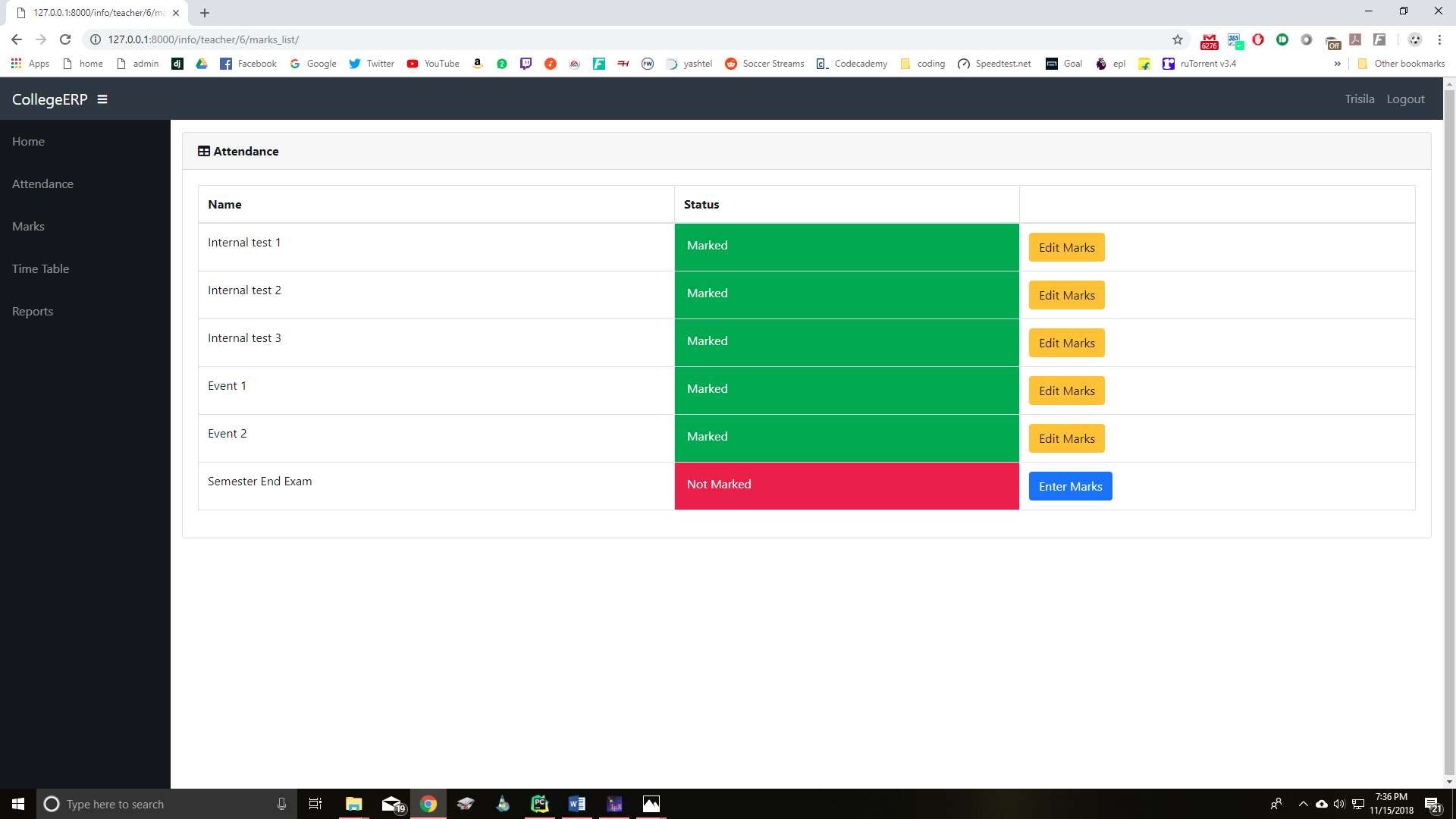


Figure 4.13: Entering marks

**Edit Marks**

Marks for a test can be edited. While editing, the list of students in that class is displayed along with already entered marks. The marks to be updated can be changed and submitted. The students can view this change immediately.

**Student Marks**

For each assigned class, the teacher has access to the list of students and the marks they obtained in all the tests. This is displayed in a tabular form.

##### Timetable

This page is a table which lists the day and timings of each of the classes assigned to the teacher. The row headers are the days of the week and the column headers are the time slots. So, for each day, it specifies the classes in the time slots. The timetable is generated automatically from the assign table, which is a table containing the information of all the teachers assigned to a class with a course and the timings the classes.

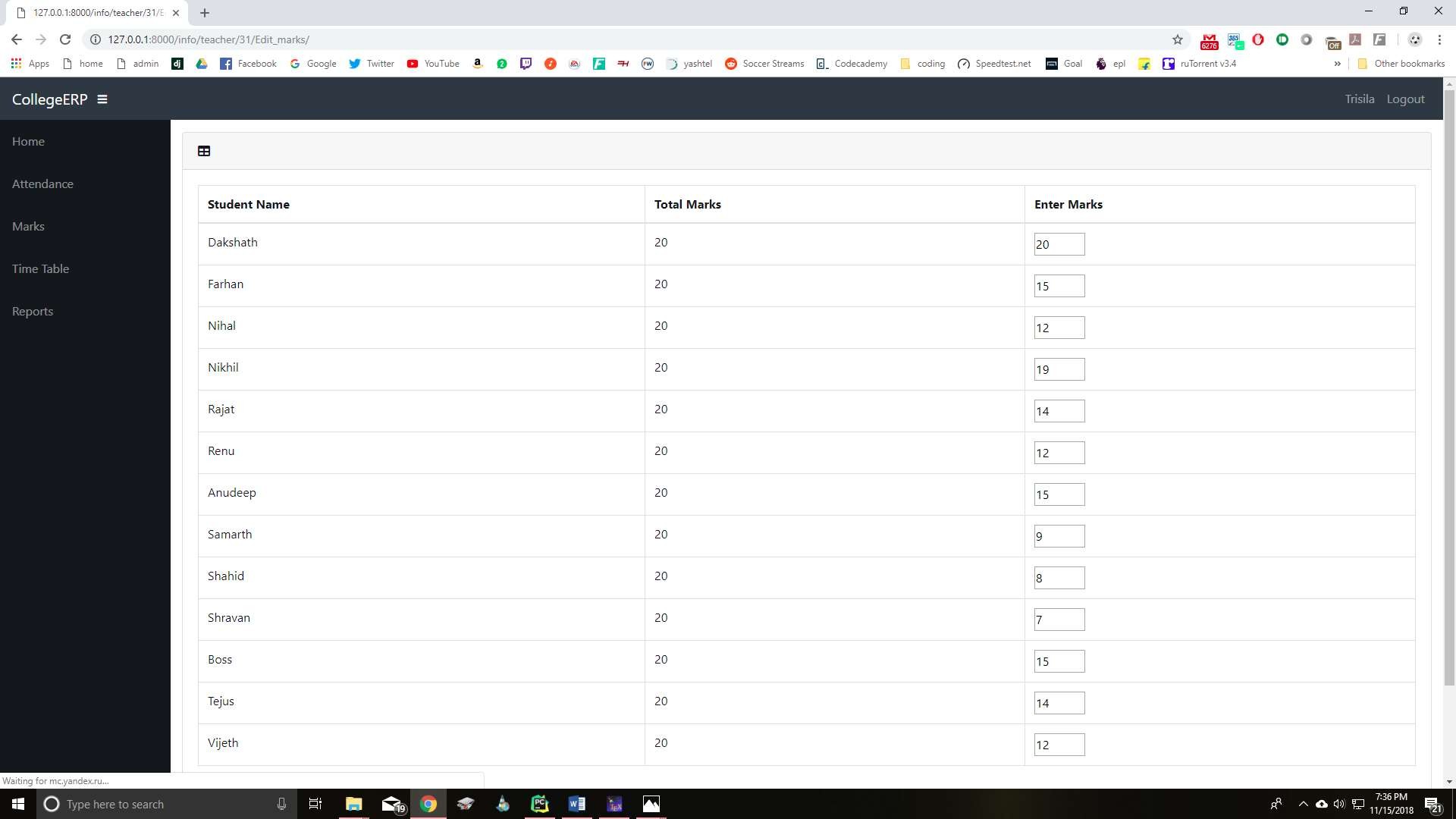


Figure 4.14: Editing marks

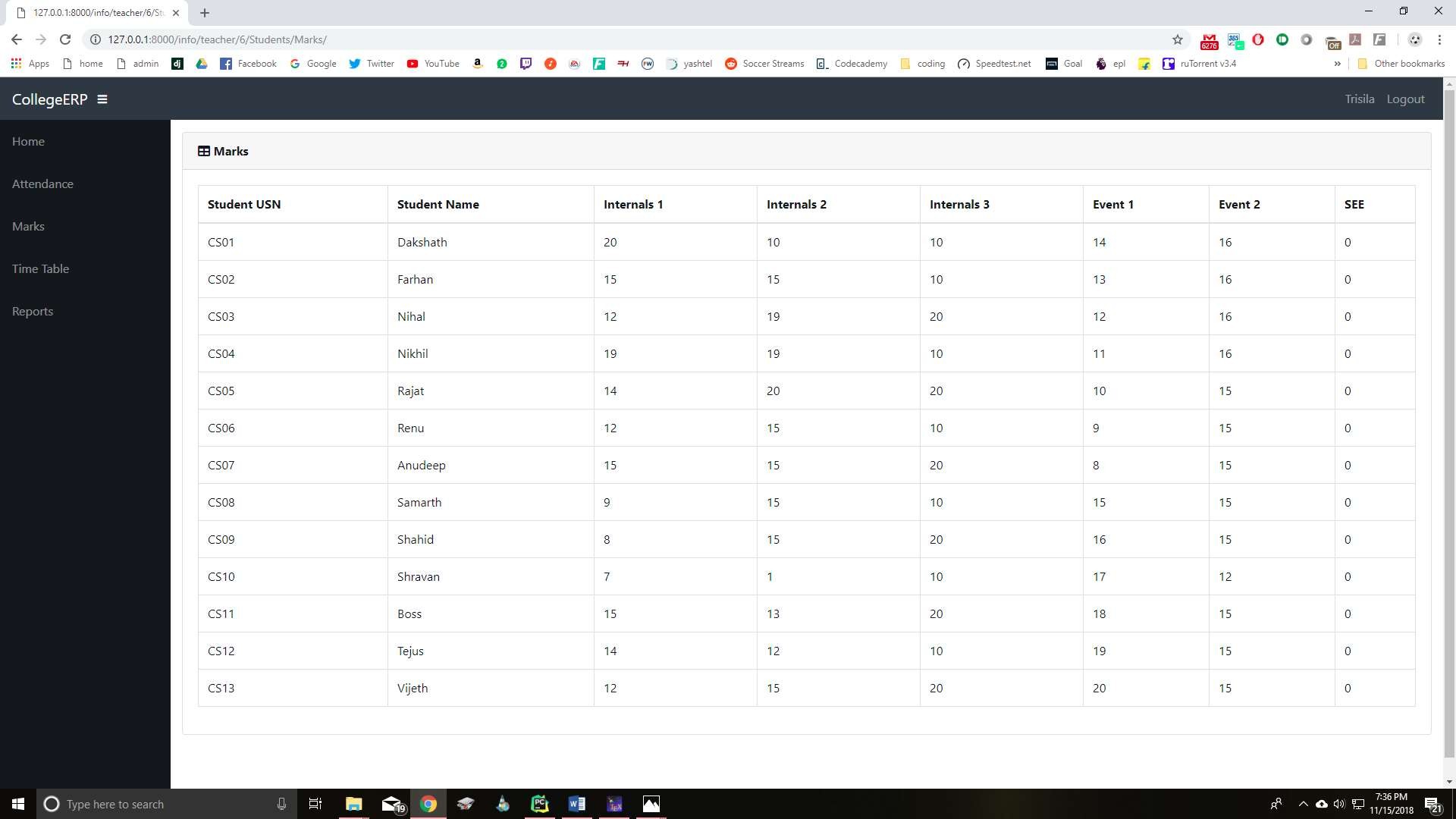


Figure 4.15: Marks of all the students in a class

**Free teachers**

For each entry in the table, the list of free teachers can be generated. Free teachers are the teachers who assigned to the class and are free for that time slot on that day. This is very useful for the teachers particularly when they are on leave as it helps them find a suitable replacement are that class.

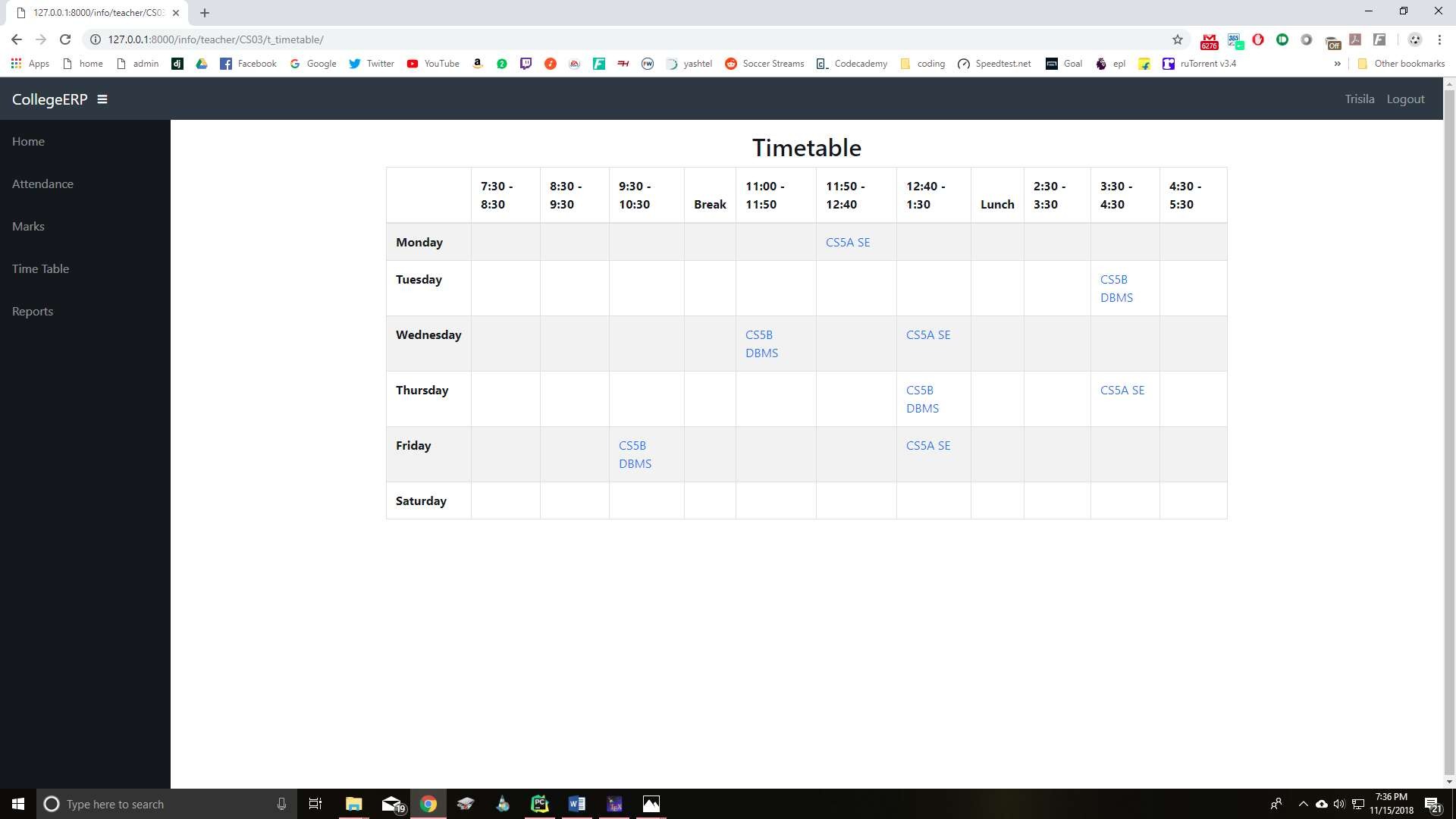


Figure 4.16: Teacher Timetable

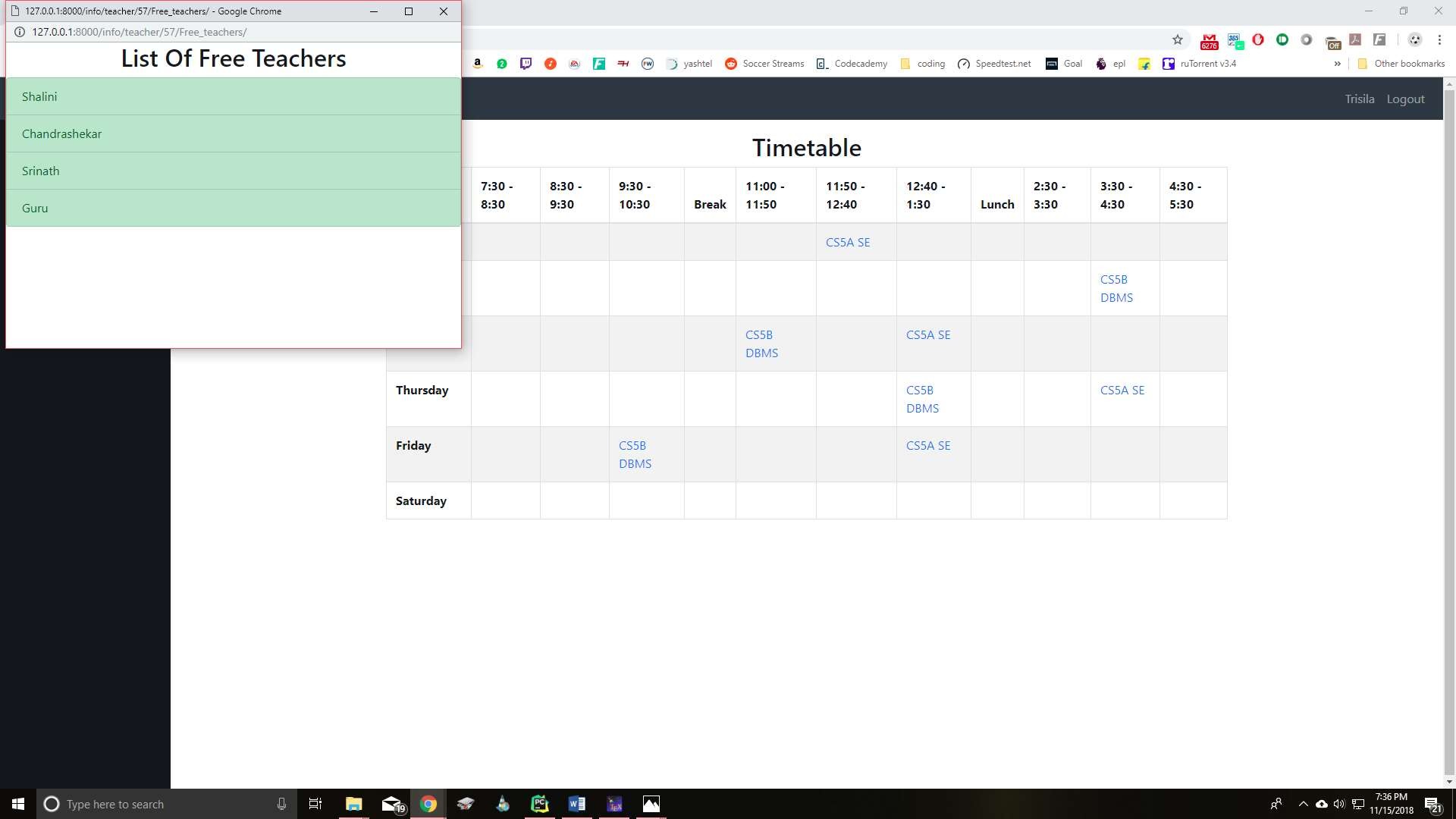


Figure 4.17: List of free teachers for a time slot

##### Reports

The last page for the teachers is used to generate reports for each class. The report specifies the list of students in that class and their respective CIE and attendance percentage. CIE is the average of the marks obtained from the tests, 3 internals and 2 events. The CIE is out of 50 and the students with CIE below 25 are marked in red and are not eligible to write the semester end exam. Also, the attendance

percentage is displayed with students below 75% marked in red.

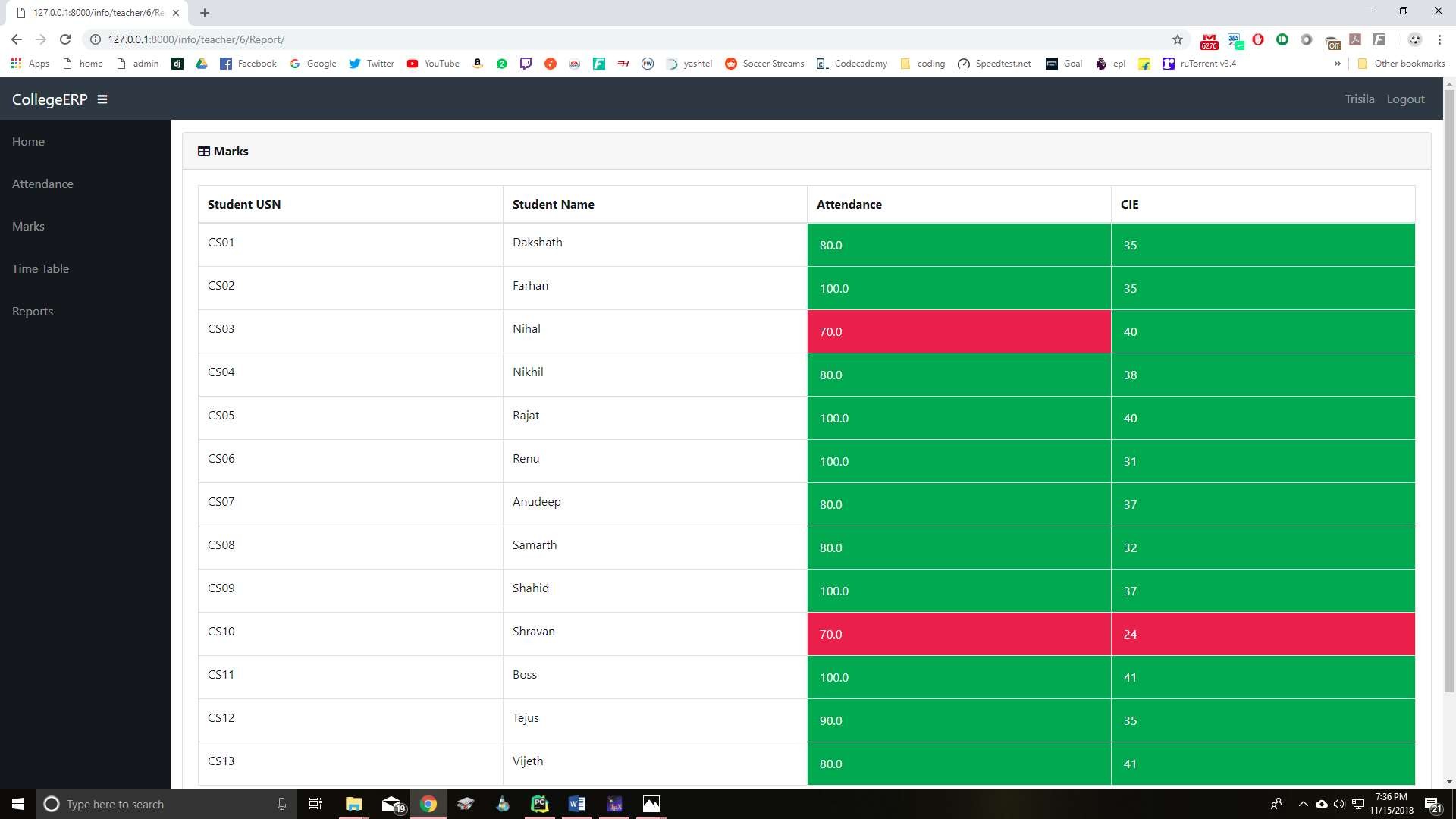


Figure 4.18: CIE and attendance for a class of students

#### Administrator

The administrator is responsible for adding and maintaining all the departments, students, teachers, classes and courses. All this data is stored in the database in their respective tables. The admin is also responsible for adding and maintaining the list of teachers assigned to class with a course and the timings. This information is stored in the Assign table. The admin also has access to the marks and attendance of each student and can modify them.

There are several features in place to ensure that querying the database is quick and efficient for the administrator. As the database has the potential to become huge, there is a search feature for every table including student, teacher etc. The search has get a specific record based on name or id. Also, it can filter the record based on department, class etc.

Figure 3.19 shows the homepage for the admin, it lists all the different tables in the database. Figure 3.20 shows the details of the class table. Each class consists of a list of students as shown.

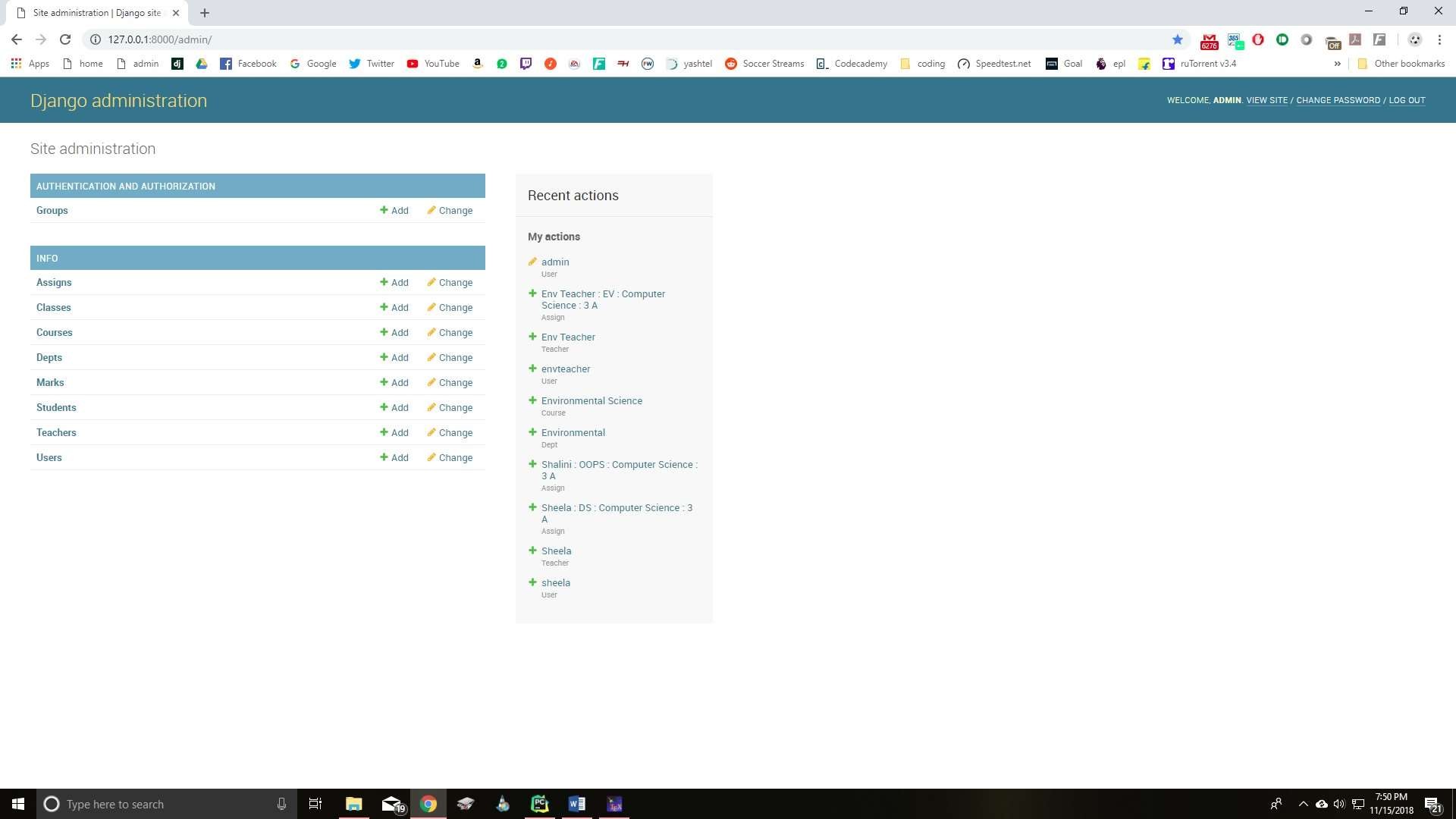


Figure 4.19: Admin homepage

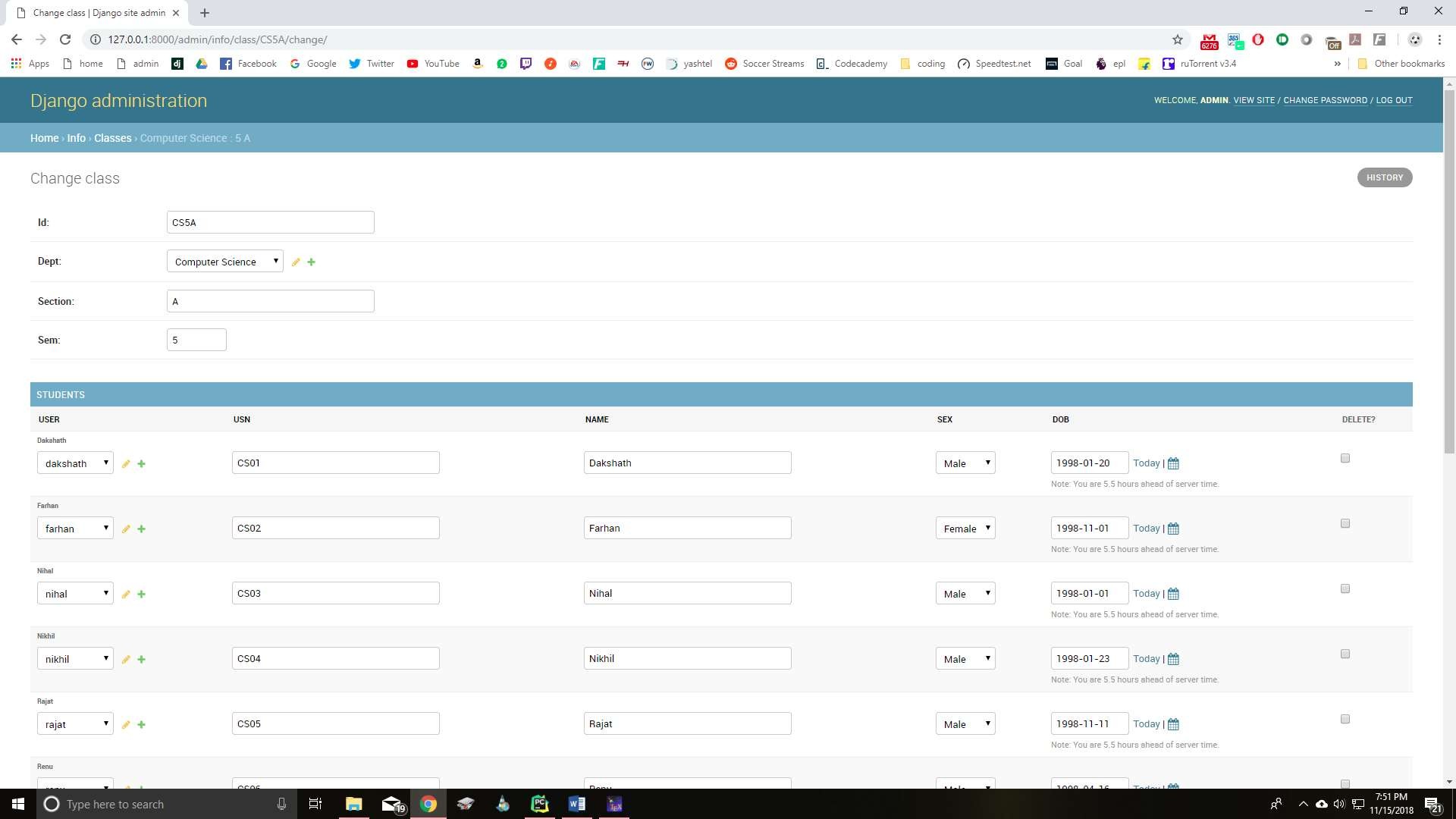


Figure 4.20: Admin students table page

**Chapter 4 Layout**

**Screenshots of the implemented system**

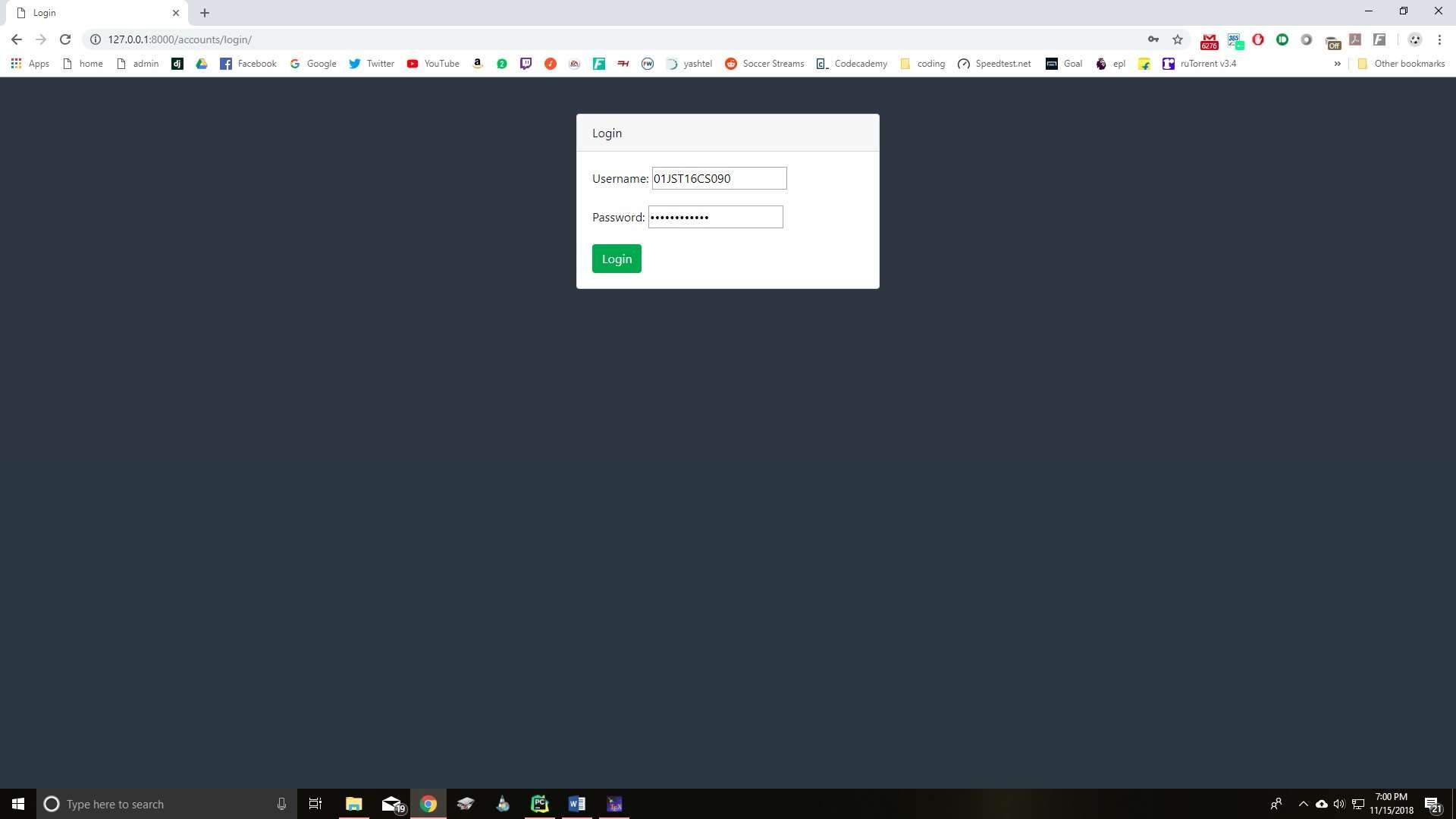


Figure 8: Student Login Page

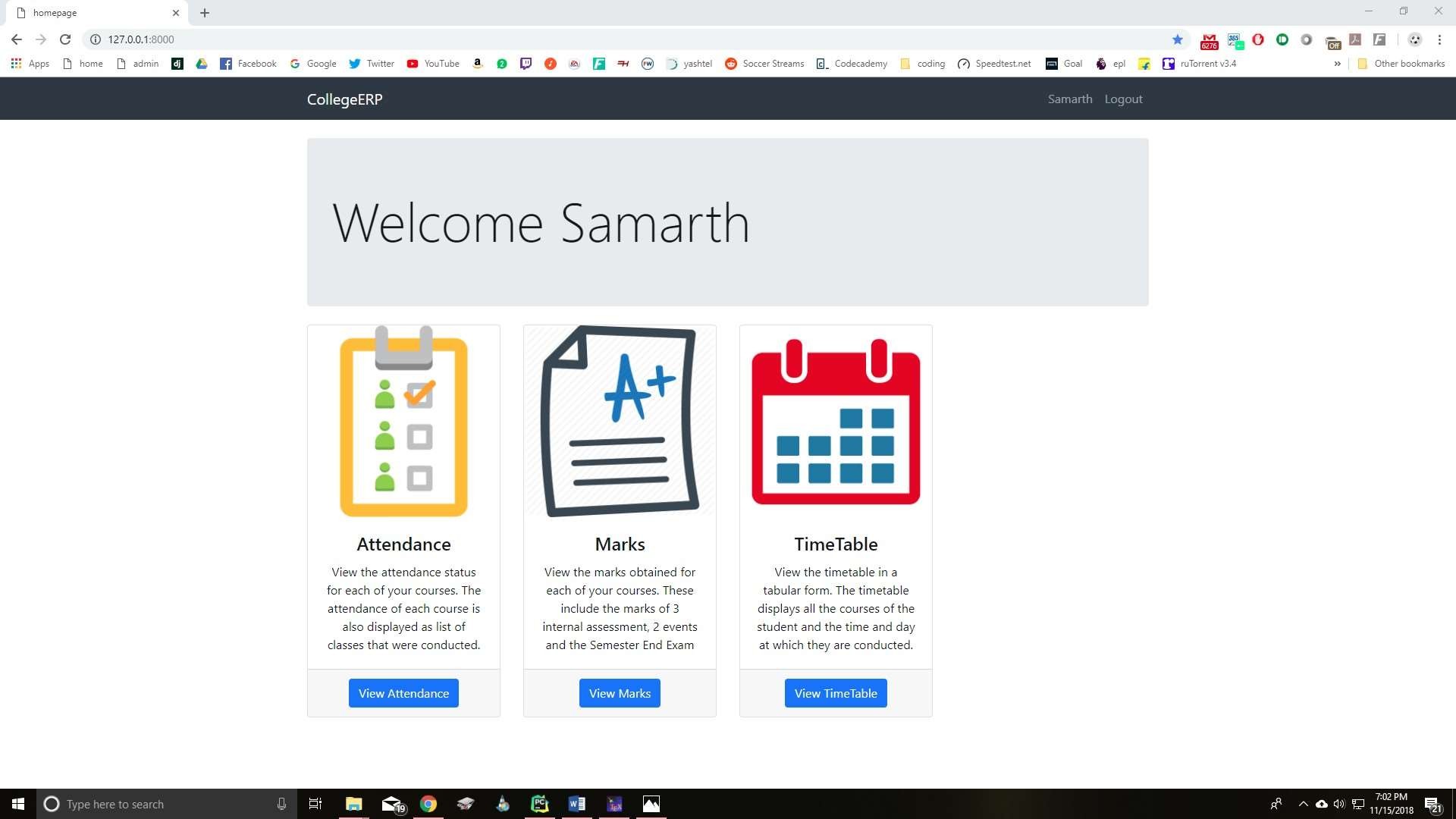


Figure 9: Student Home Page

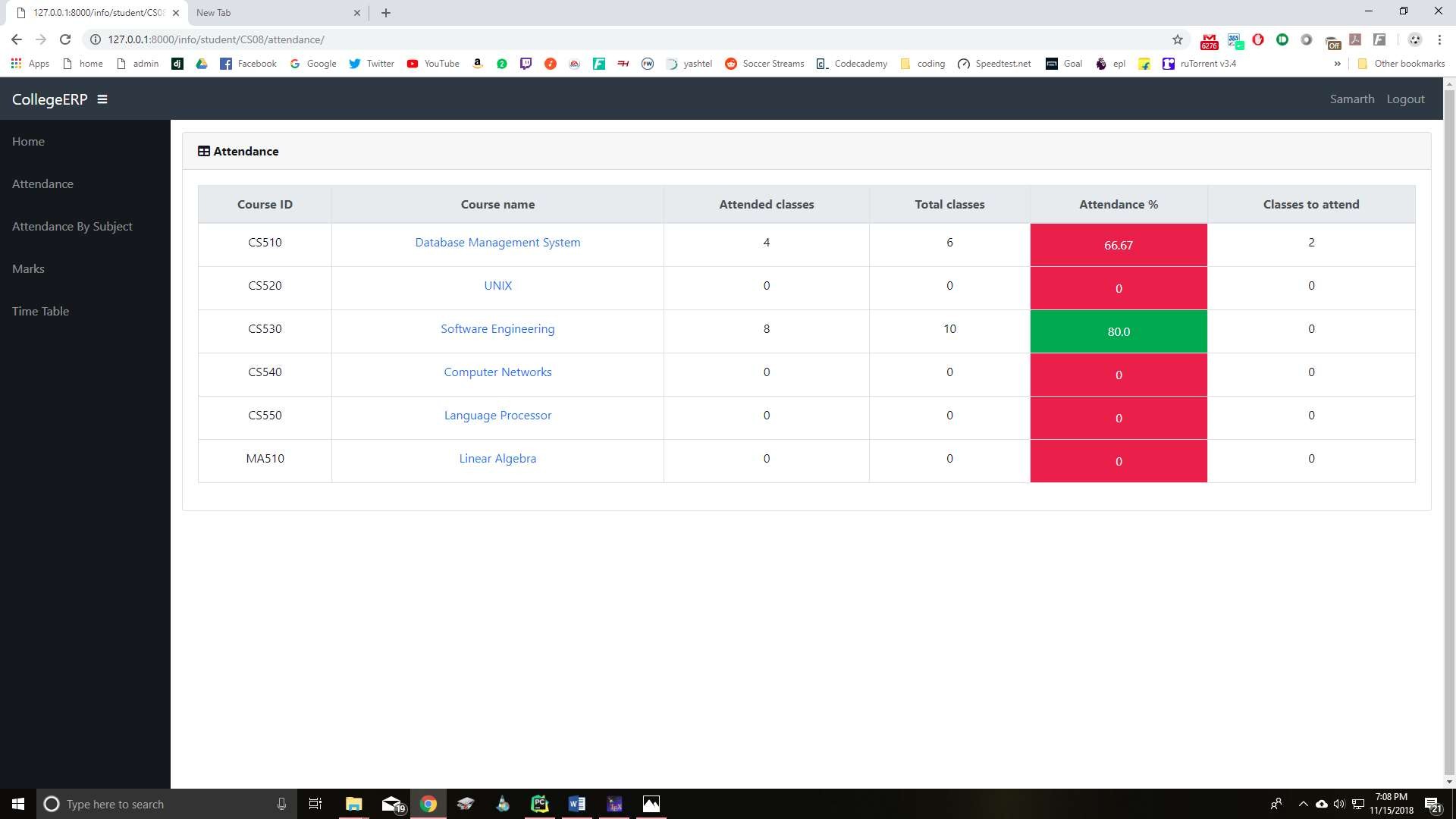


Figure 10: Student Attendance Page

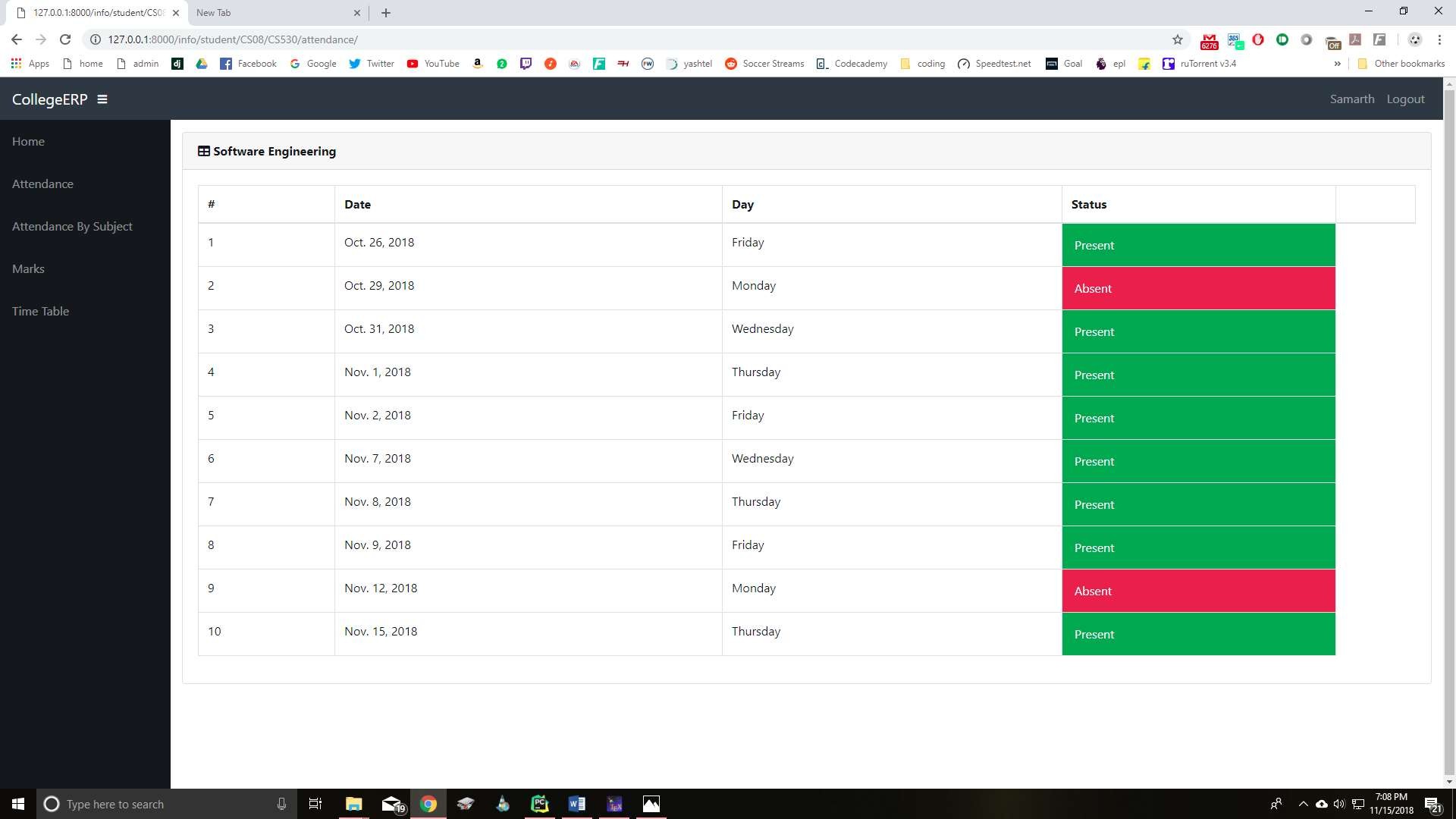


Figure 11: Student Attendance Detail Page

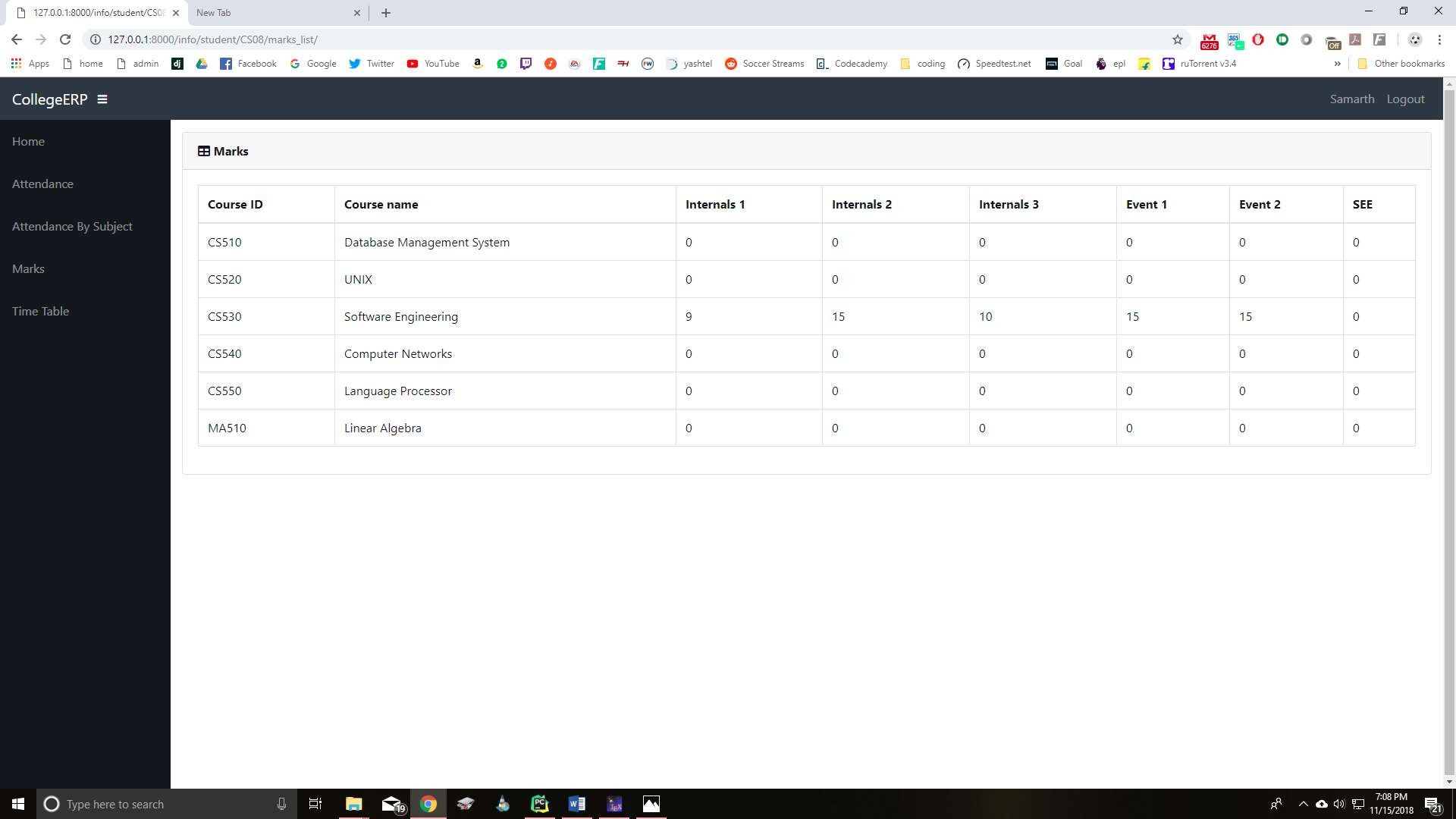


Figure 12: Student Marks Page

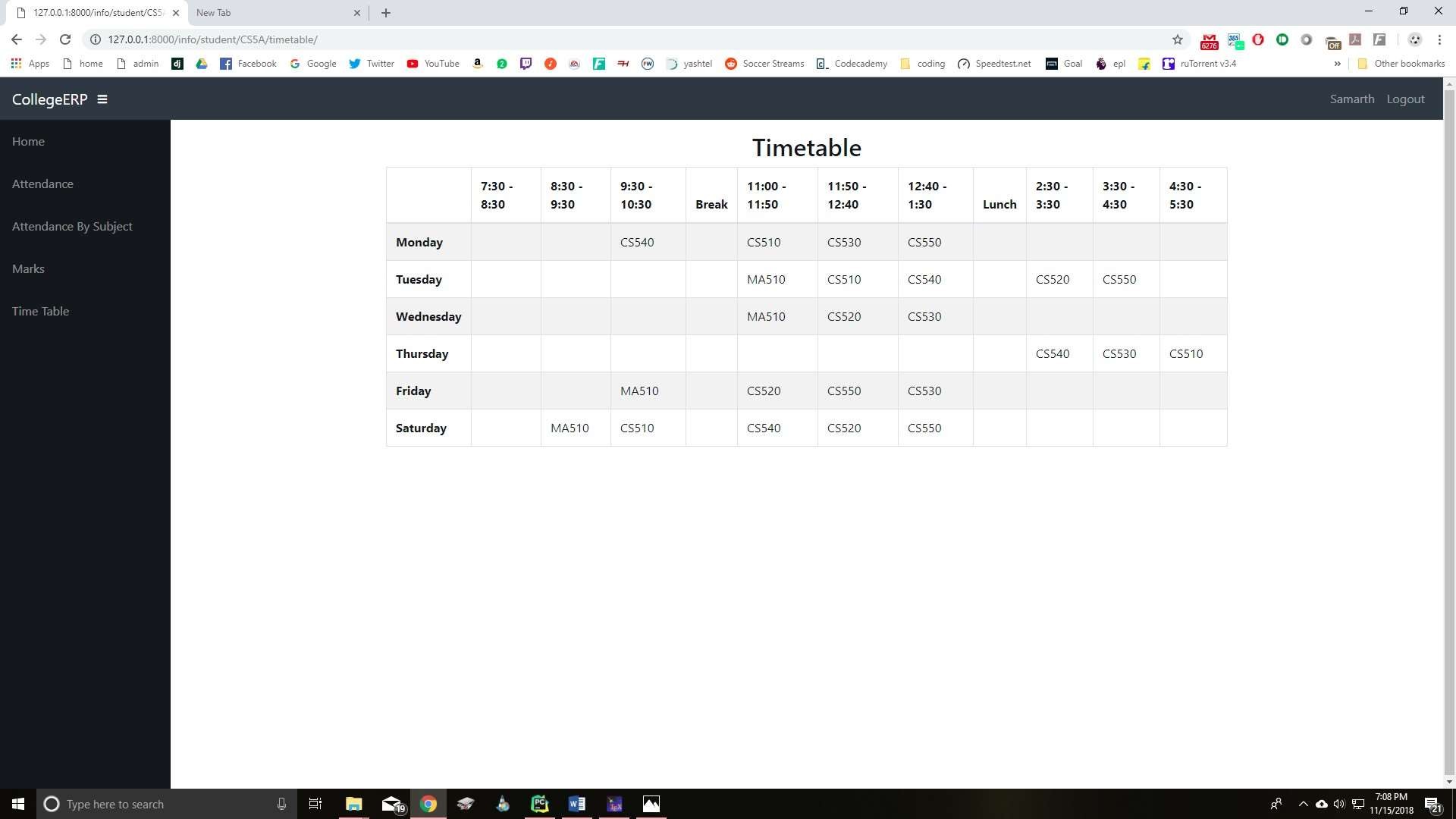


Figure 13: Student Timetable

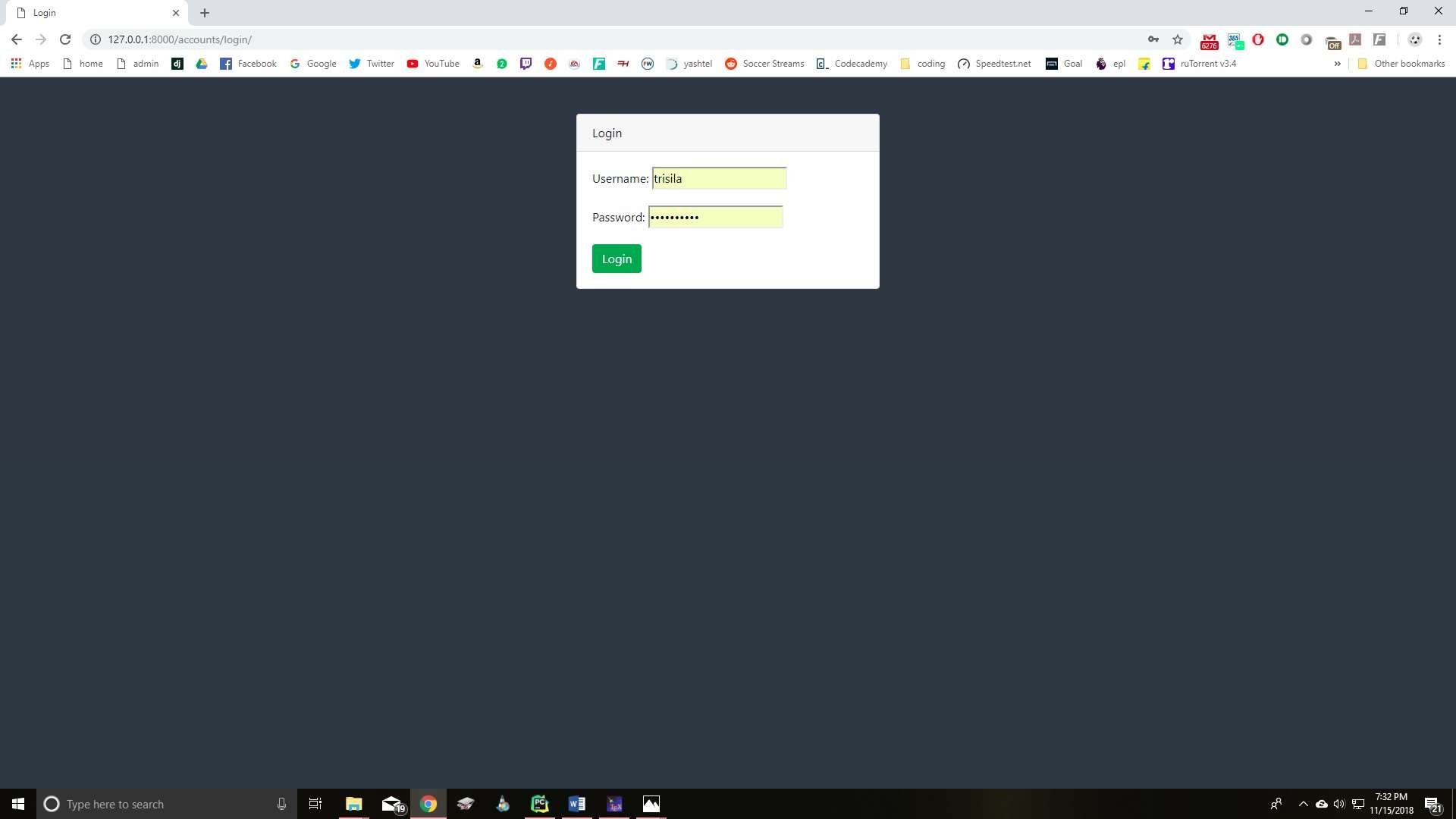


Figure 14: Teacher Login

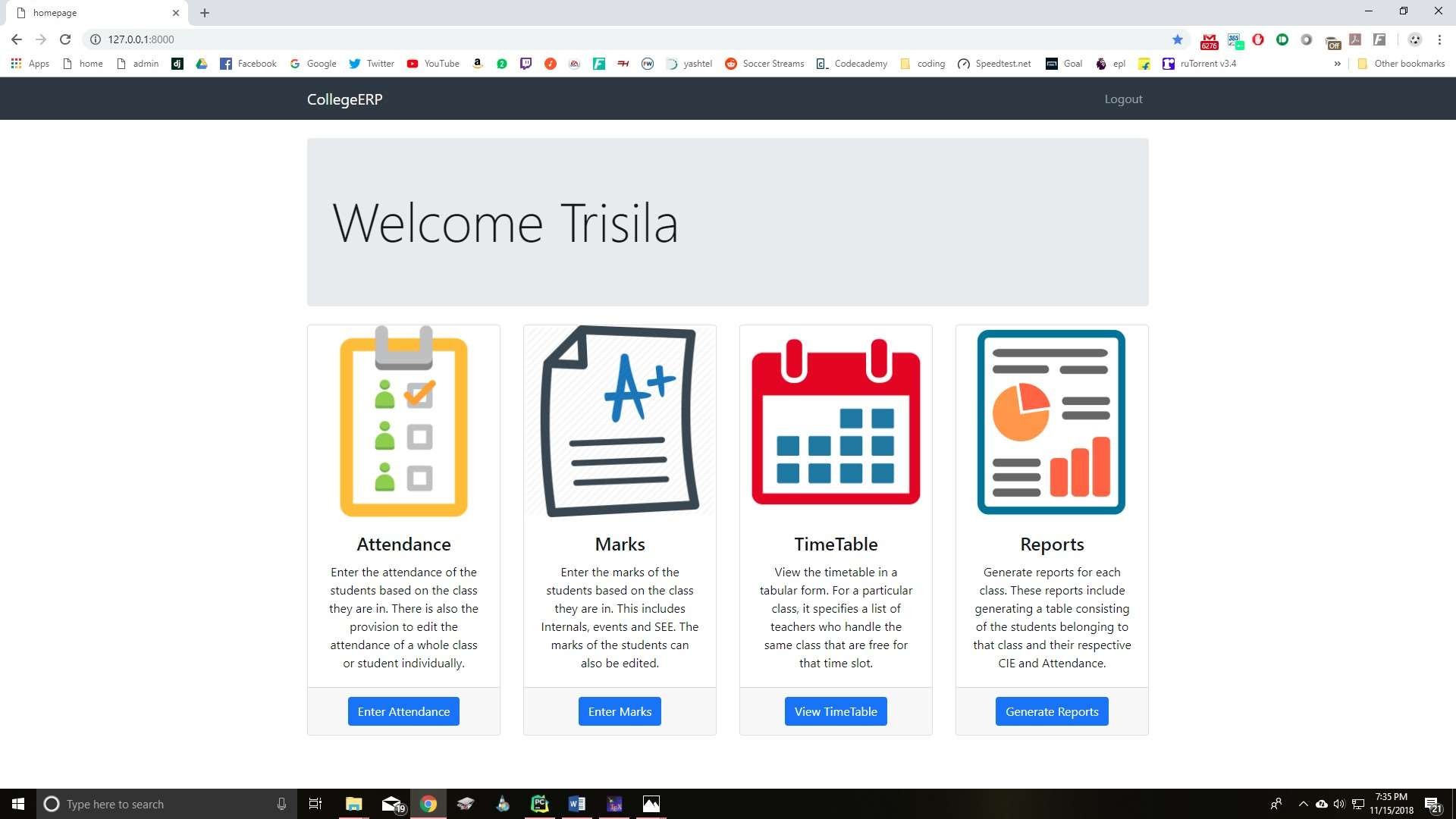


Figure 15: Teacher homepage

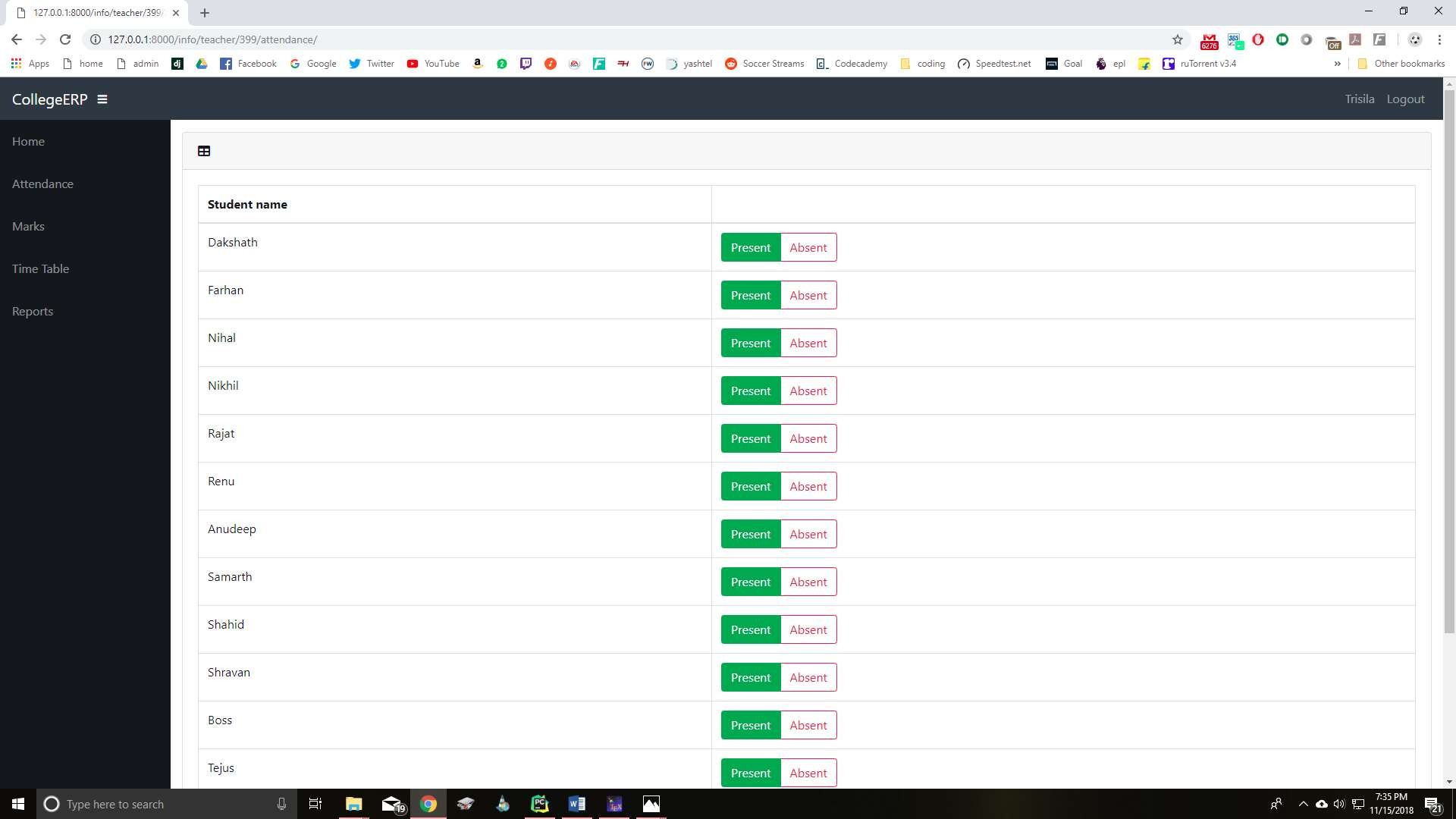


Figure 16: Entering attendance

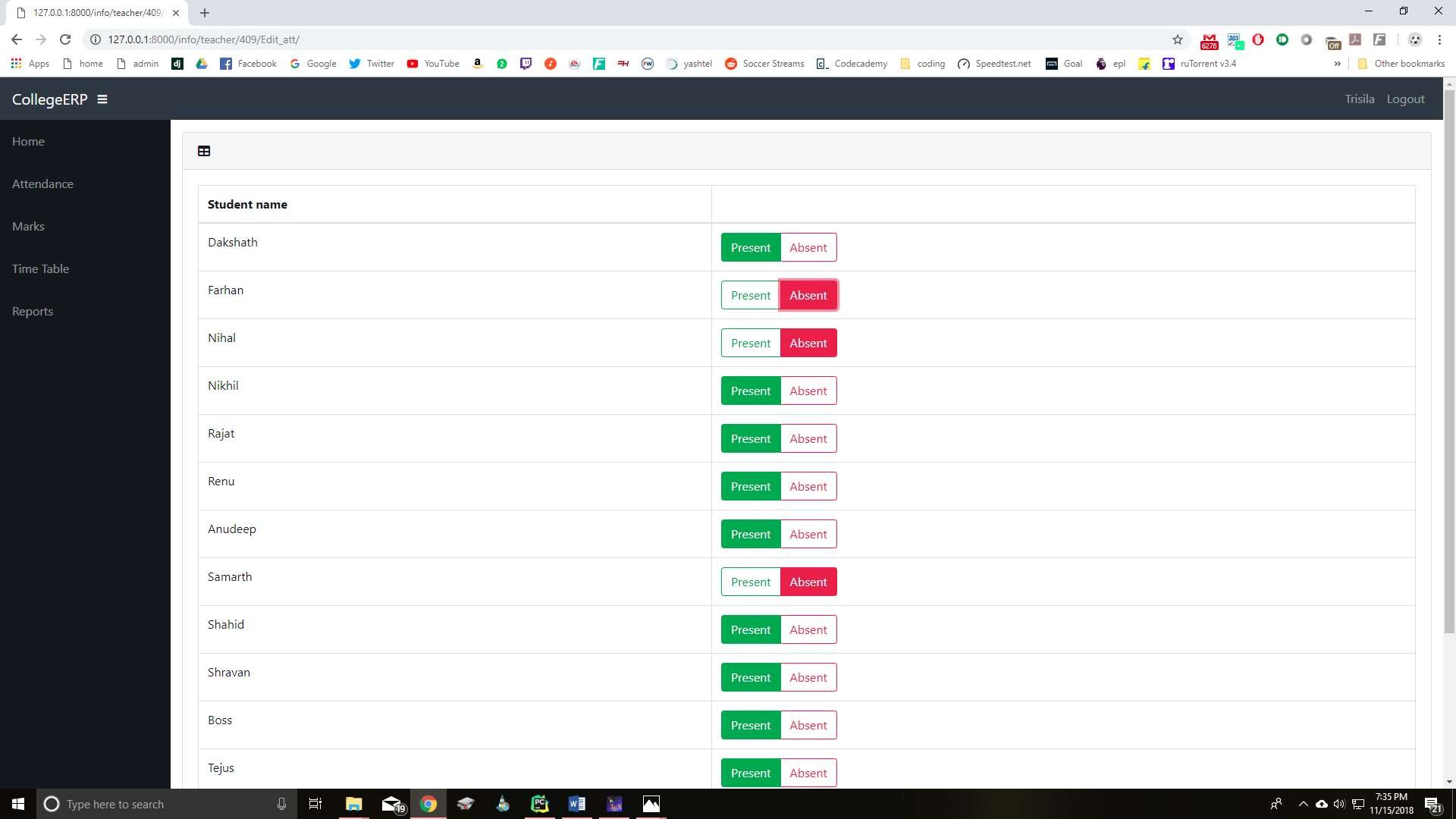


Figure 17: Editing attendance

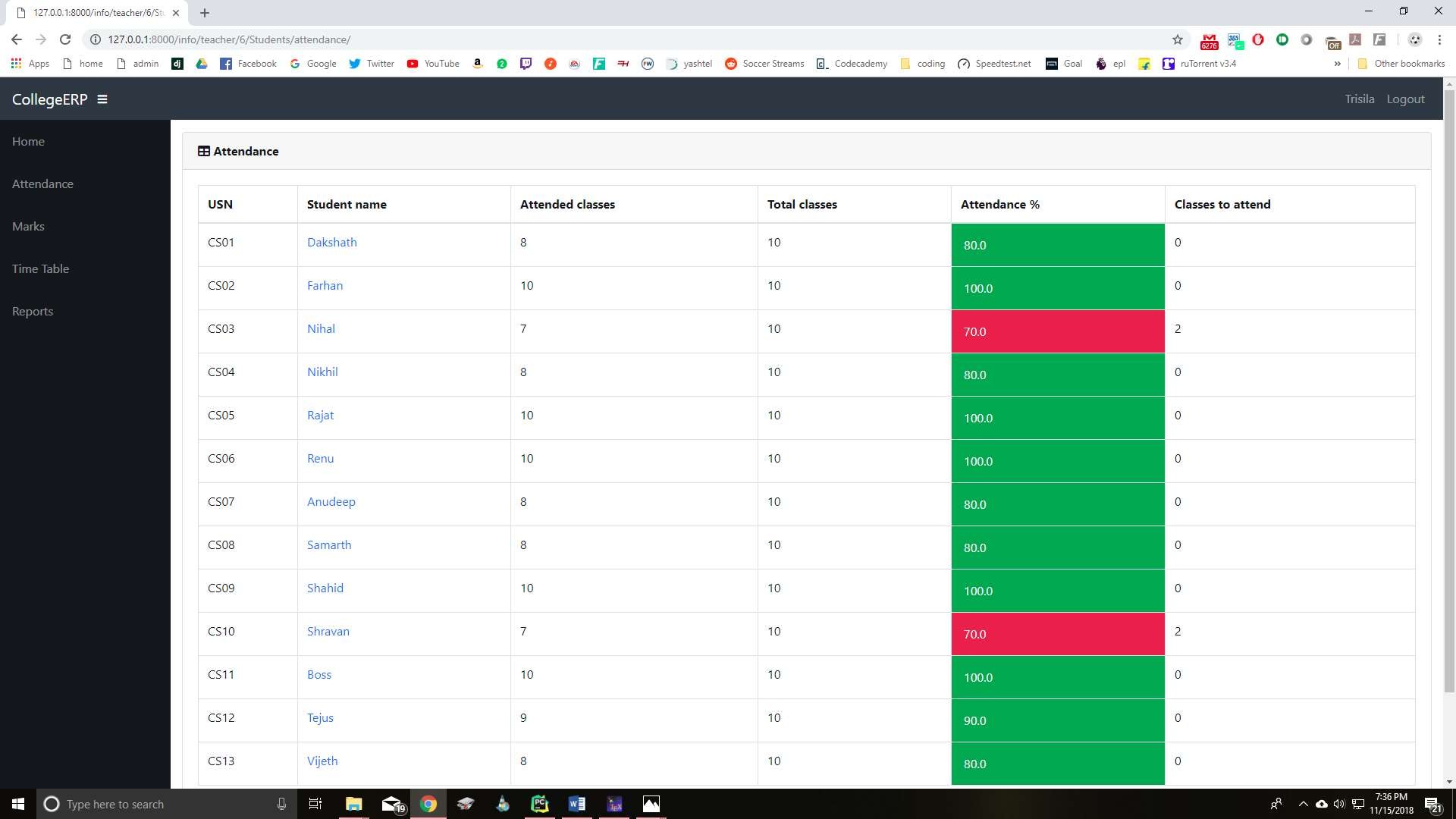


Figure 18: Attendance of students in a class

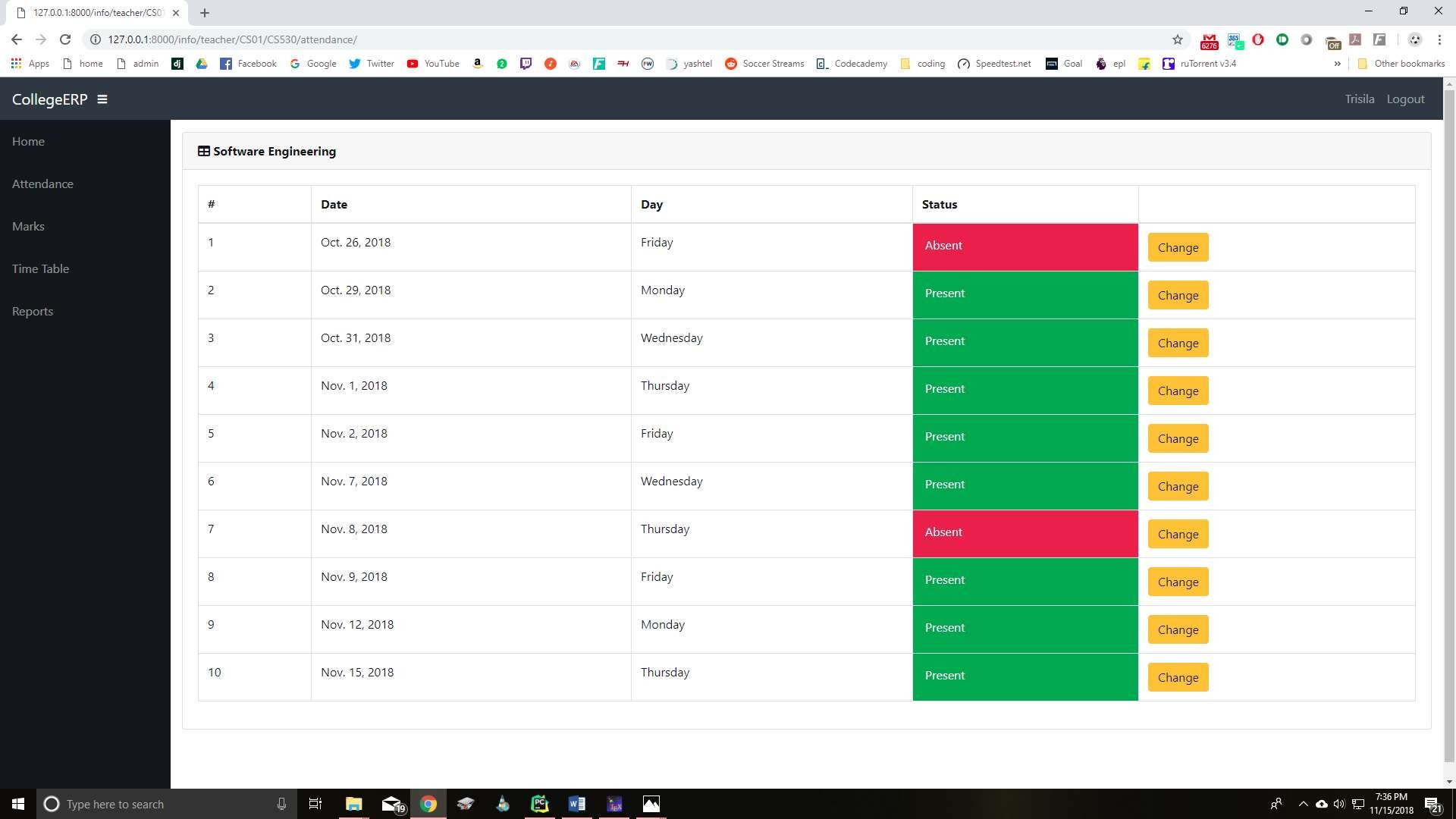


Figure 19: Attendance details of an individual student

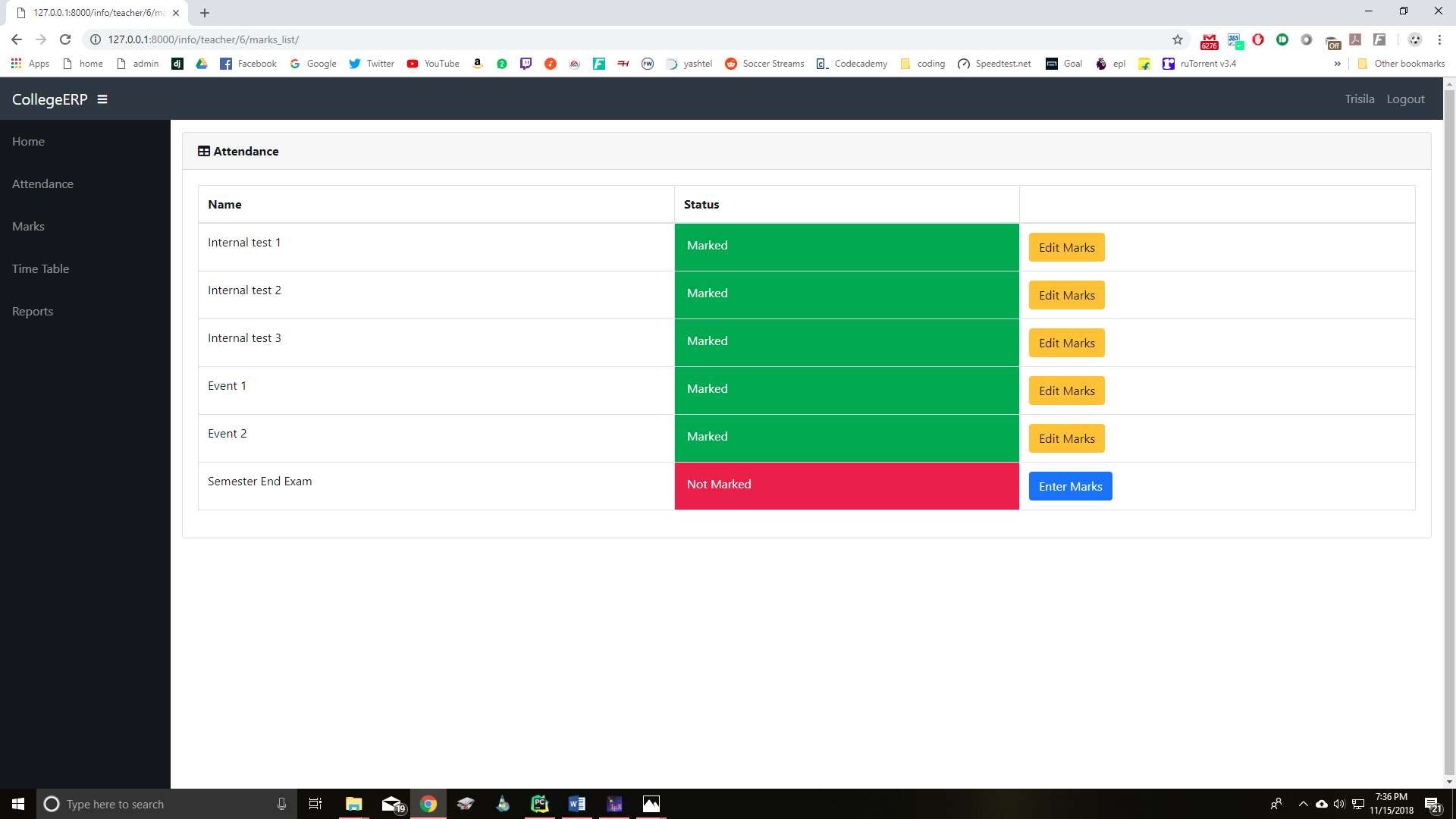


Figure 20: Entering marks

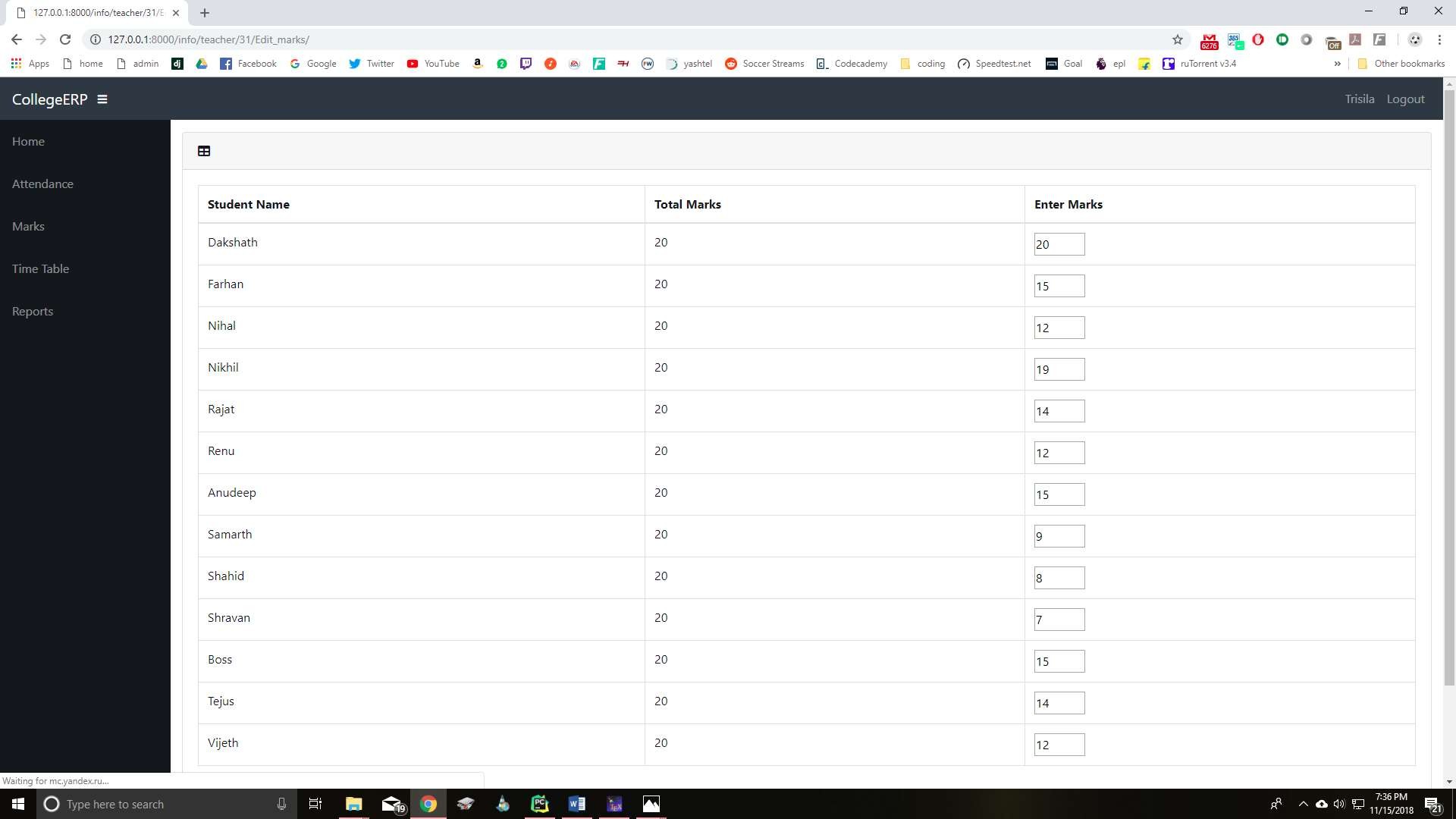


Figure 21: Editing marks

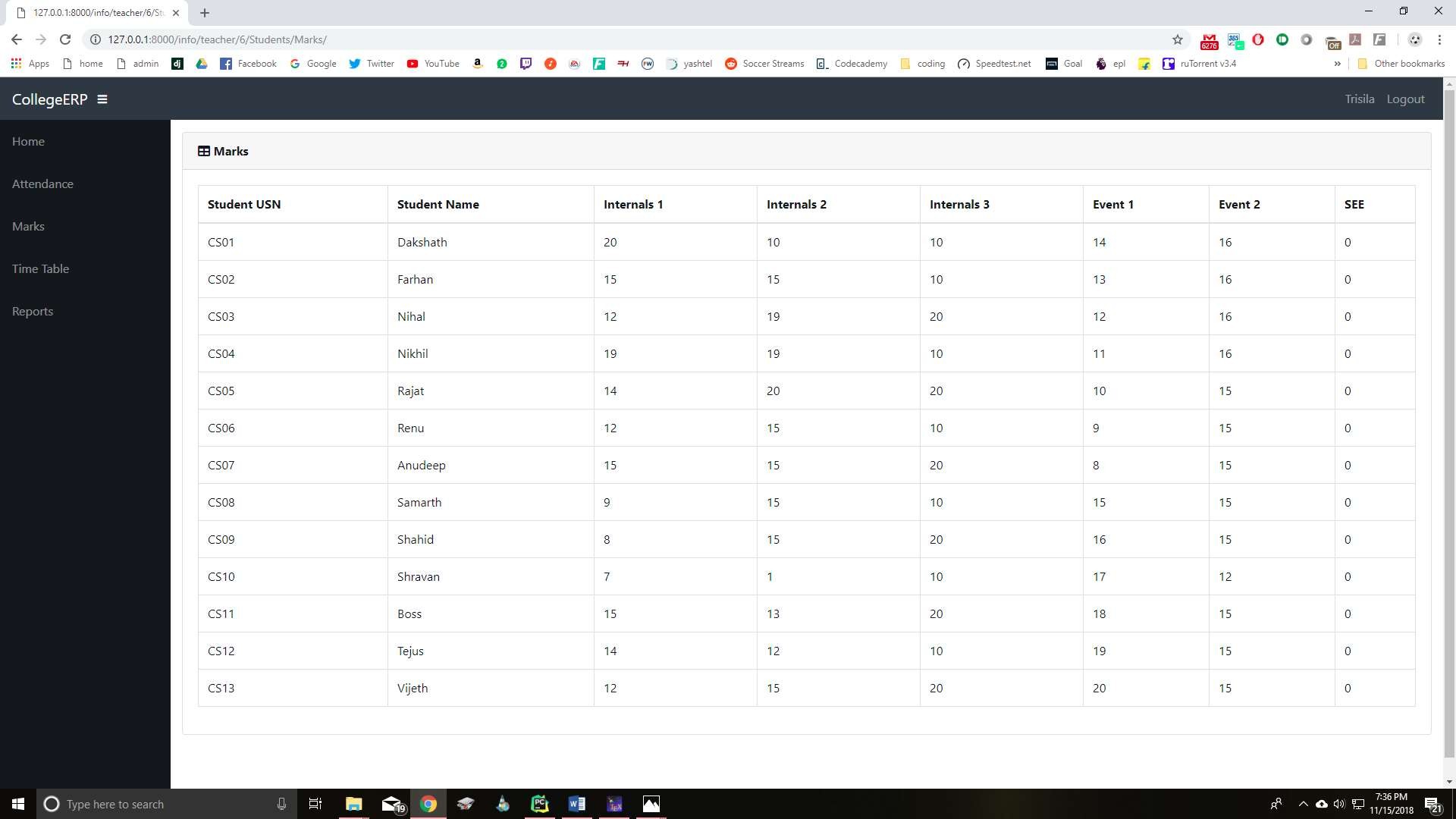


Figure 22: Marks of all the students in a class

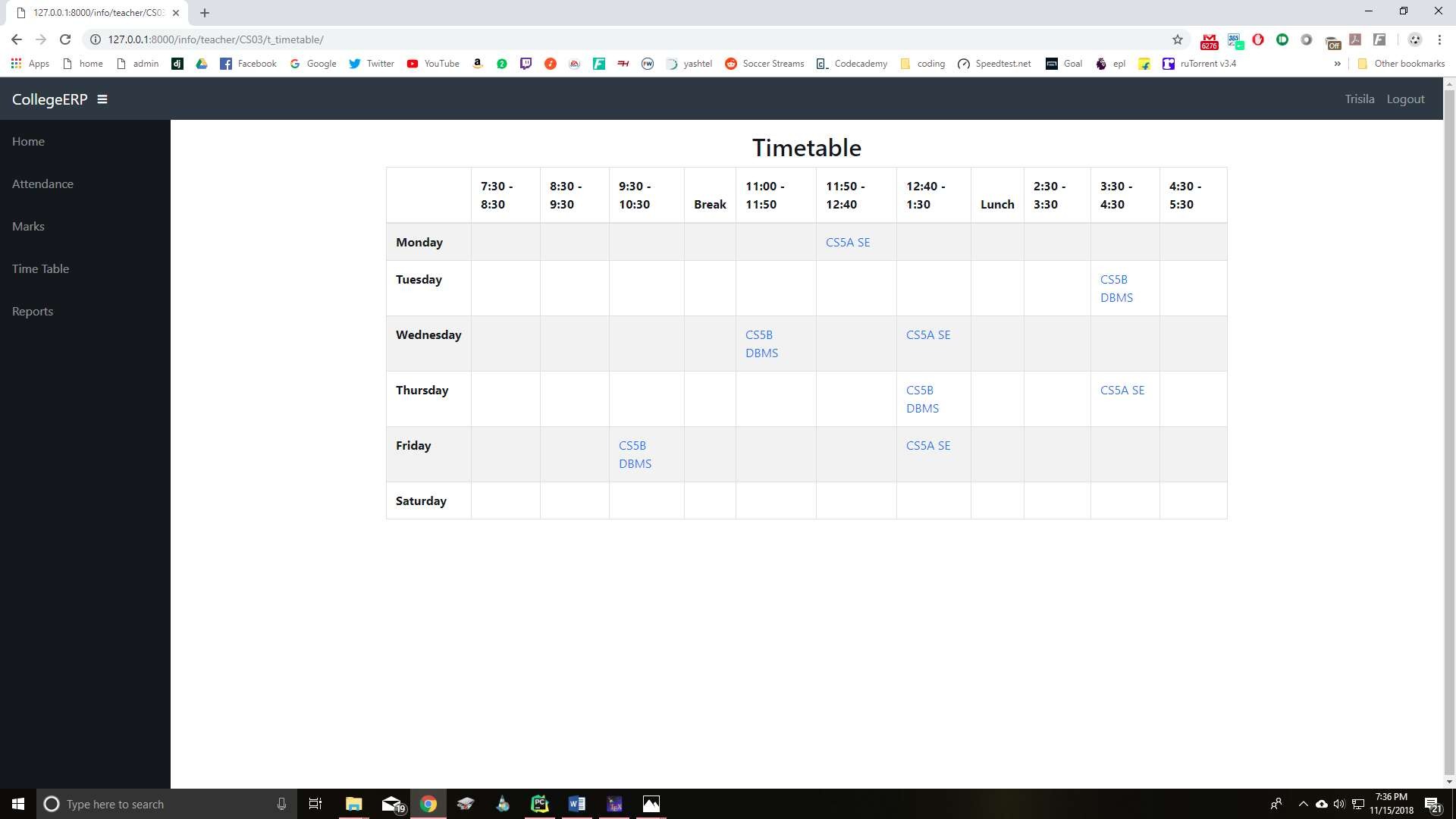


Figure 23: Teacher Timetable

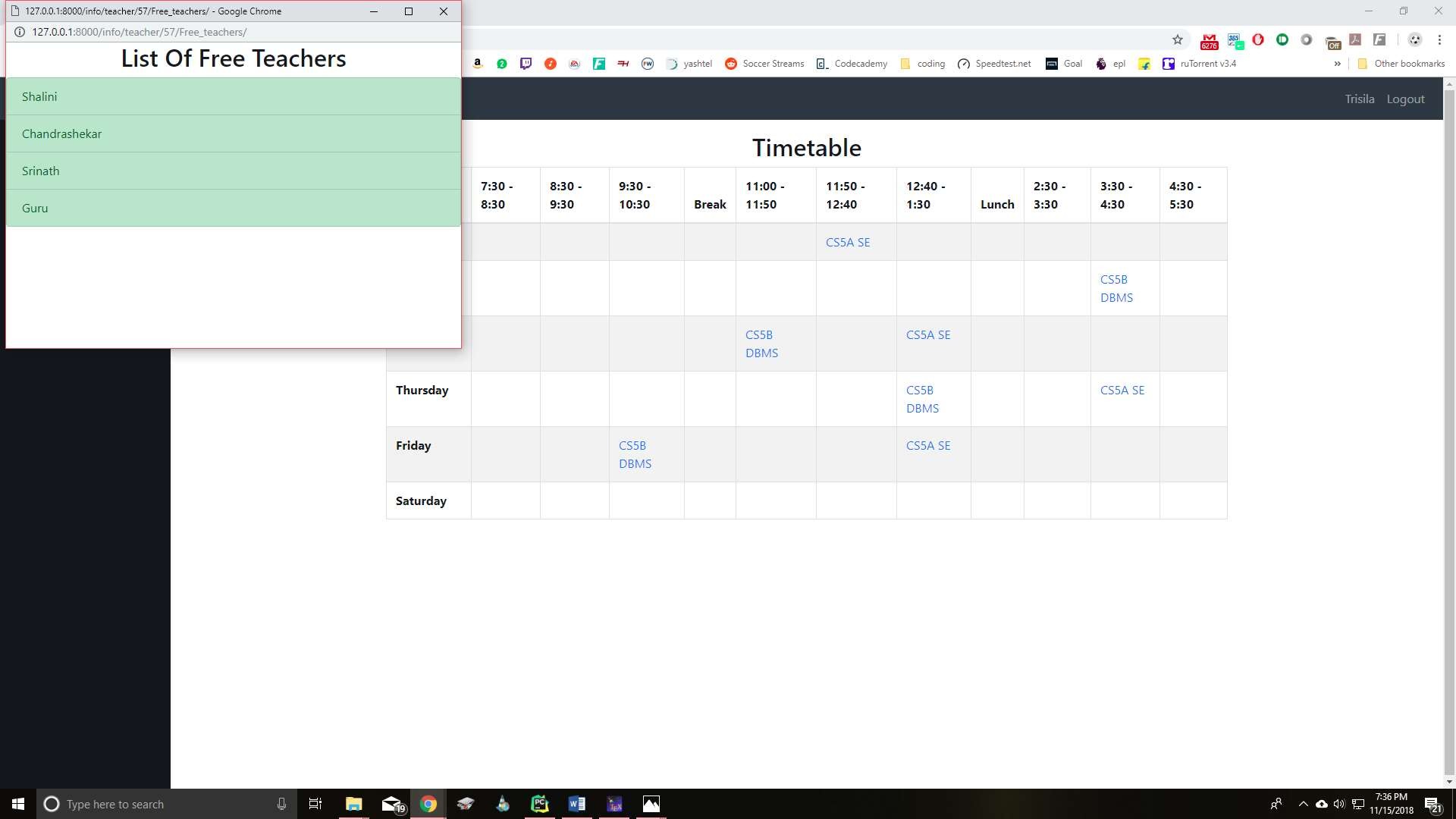


Figure 24: List of free teachers for a time slot

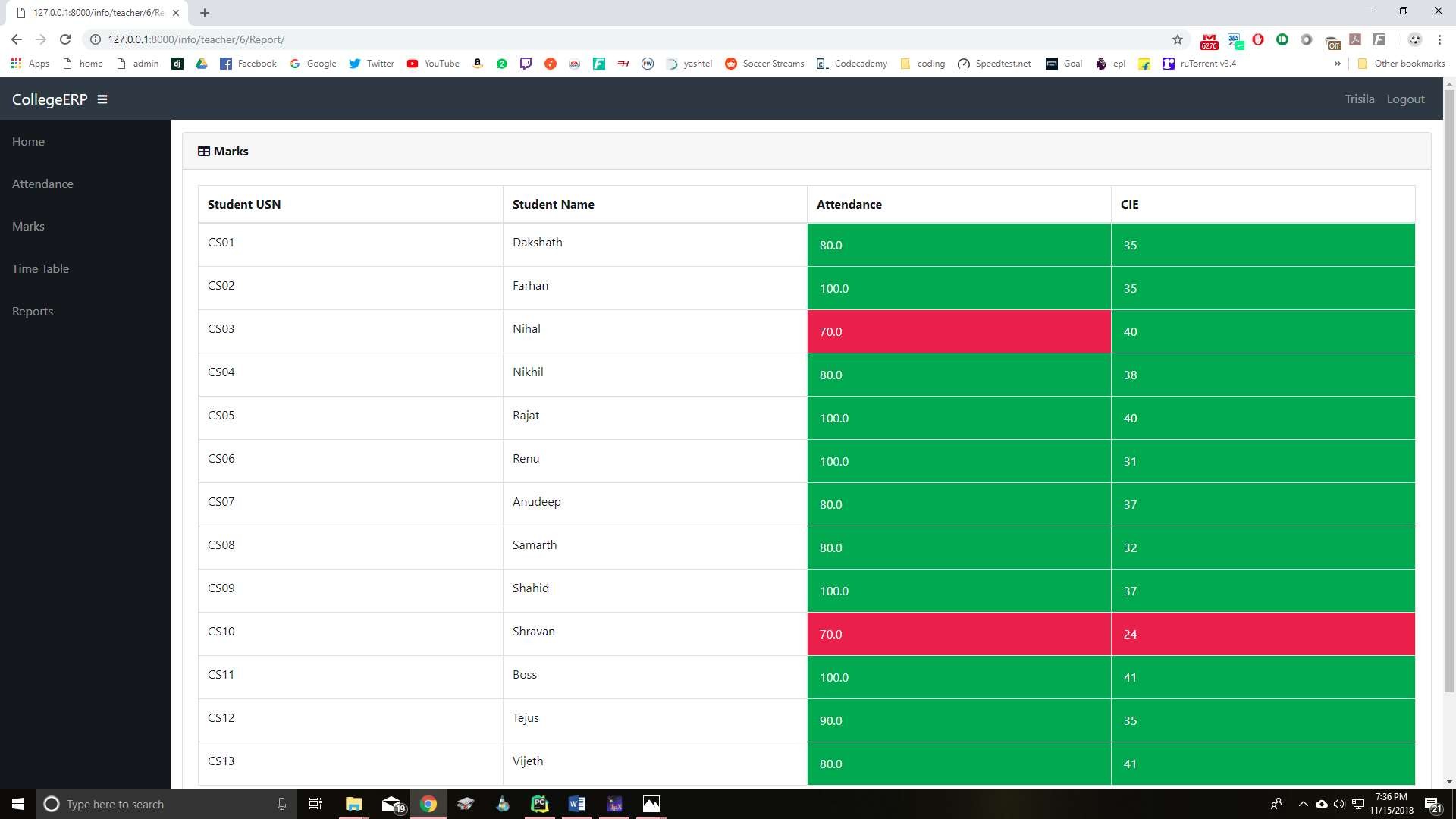


Figure 25: CIE and attendance for a class of students

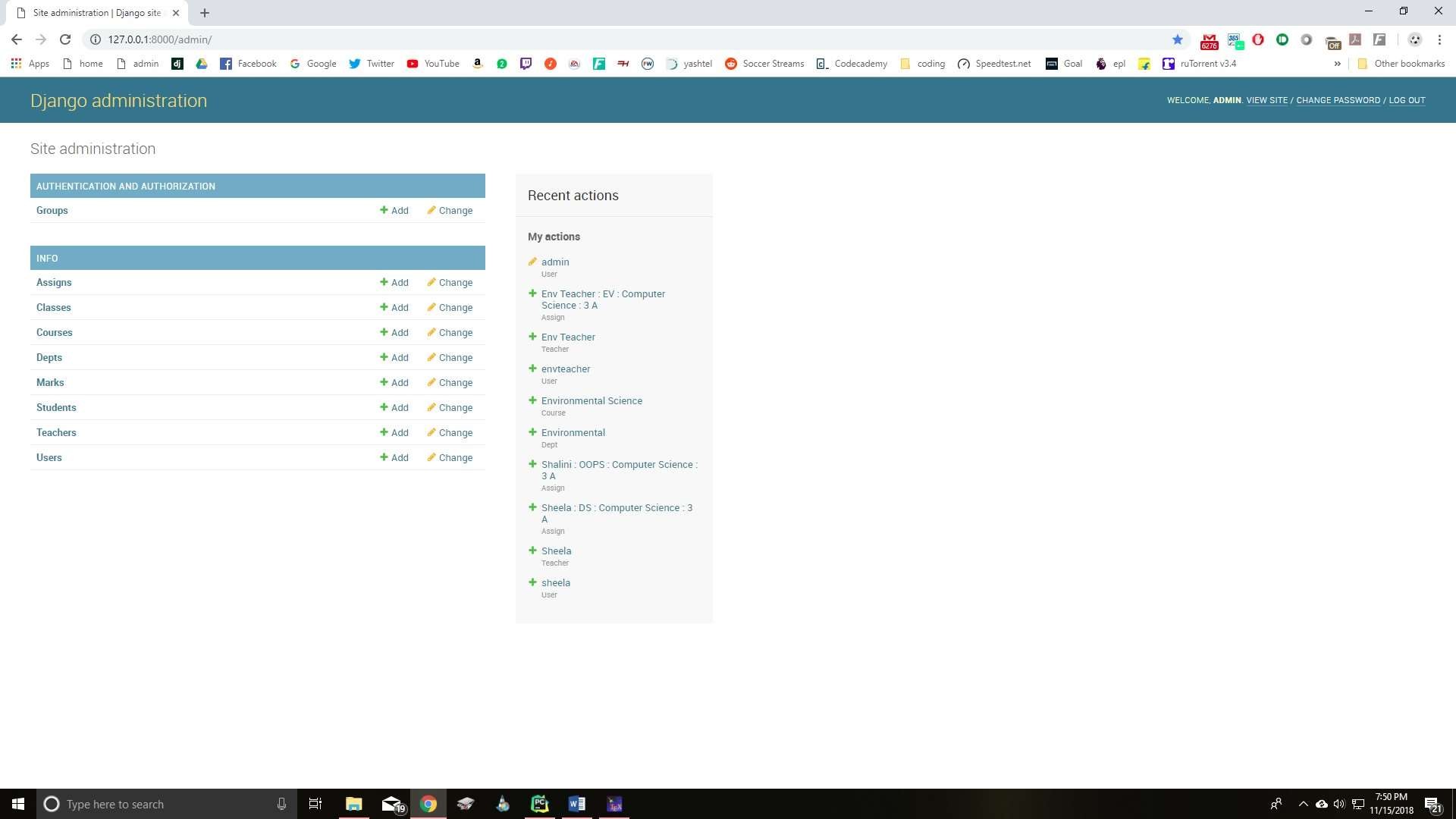


Figure 26: Admin homepage

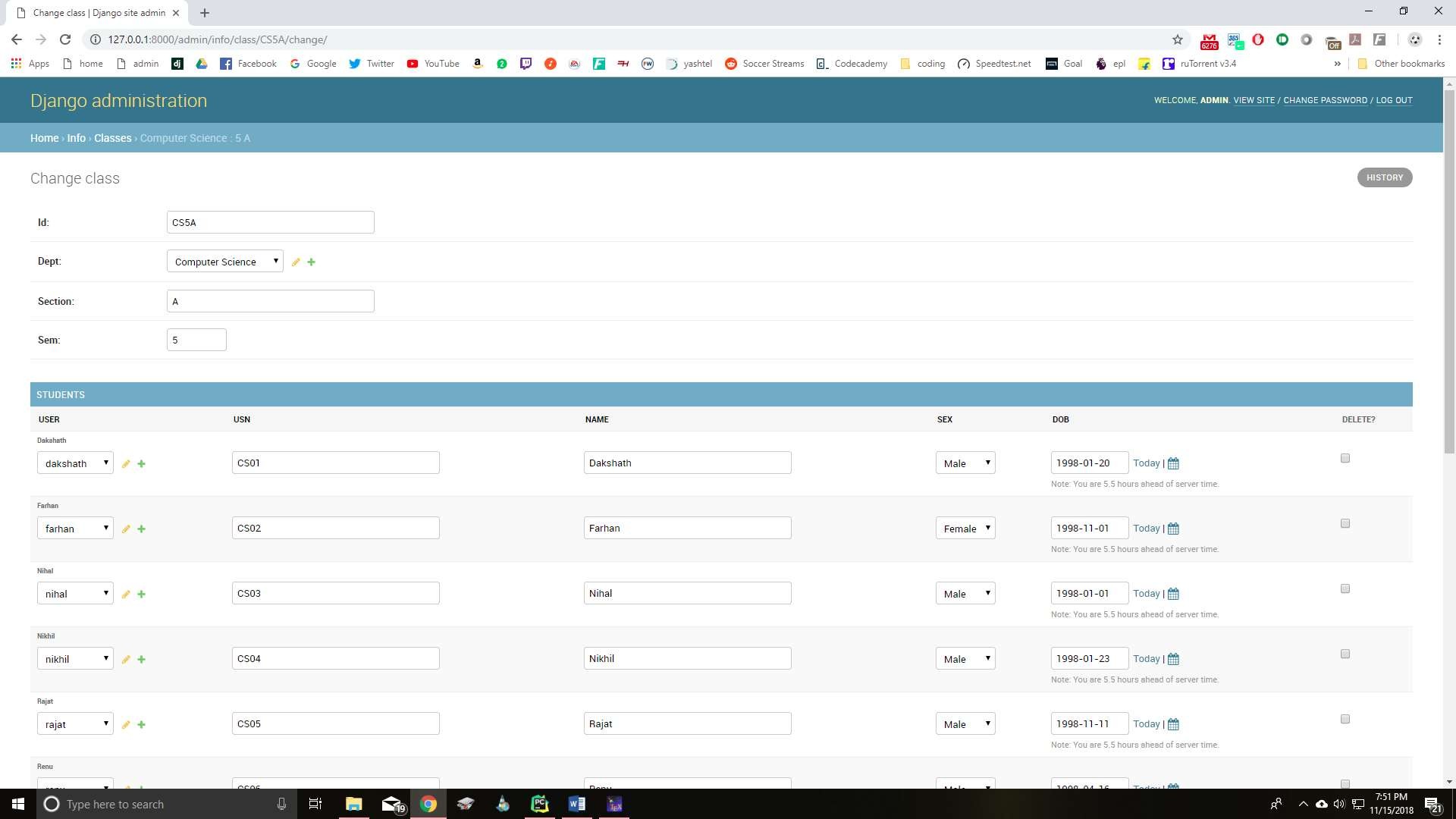


Figure 27: Admin students table page

### Coding

Step 1:

* Install Django.

Step 2:

* Create a folder with the name College\_Management\_System and open it with VS Code.

Step 3:

* Open the terminal and create a new project “student\_management\_project” using the

below command.

django-admin startproject student\_management\_project

Step 4:

* Enter inside the folder “student\_management\_project” and create the app

“student\_management\_app”.

python manage.py startapp student\_management\_app

Step 5:

* Go to student\_management\_project -> settings.py -> INSTALLED\_APPS and add our app

‘student\_management\_app’.

•

Step 6:

* Go to urls.py of student\_management\_project and add the below path in urlpatterns.

(Note – Import include as from django.urls import path, include)

* path('', include('student\_management\_app.urls'))

Step 7:

* Now enter the views that are going to use in views.py of student\_management\_app.

from django.shortcuts import render,HttpResponse, redirect,HttpResponseRedirect from django.contrib.auth import logout, authenticate, login

from .models import CustomUser, Staffs, Students, AdminHOD from django.contrib import messages

def home(request):

return render(request, 'home.html')

def contact(request):

return render(request, 'contact.html')

def loginUser(request):

return render(request, 'login\_page.html') def doLogin(request):

print("here")

email\_id = request.GET.get('email') password = request.GET.get('password')

# user\_type = request.GET.get('user\_type')

print(email\_id) print(password) print(request.user)

if not (email\_id and password):

messages.error(request, "Please provide all the details!!") return render(request, 'login\_page.html')

user = CustomUser.objects.filter(email=email\_id, password=password).last() if not user:

messages.error(request, 'Invalid Login Credentials!!') return render(request, 'login\_page.html') login(request, user)

print(request.user)

if user.user\_type == CustomUser.STUDENT:

return redirect('student\_home/')

elif user.user\_type == CustomUser.STAFF:

return redirect('staff\_home/')

elif user.user\_type == CustomUser.HOD:

return redirect('admin\_home/') return render(request, 'home.html')

def registration(request):

return render(request, 'registration.html')

def doRegistration(request):

first\_name = request.GET.get('first\_name') last\_name = request.GET.get('last\_name') email\_id = request.GET.get('email') password = request.GET.get('password')

confirm\_password = request.GET.get('confirmPassword') print(email\_id)

print(password) print(confirm\_password) print(first\_name) print(last\_name)

if not (email\_id and password and confirm\_password): messages.error(request, 'Please provide all the details!!') return render(request, 'registration.html')

if password != confirm\_password:

messages.error(request, 'Both passwords should match!!') return render(request, 'registration.html')

is\_user\_exists = CustomUser.objects.filter(email=email\_id).exists() if is\_user\_exists:

messages.error(request, 'User with this email id already exists. Please proceed to login!!') return render(request, 'registration.html')

user\_type = get\_user\_type\_from\_email(email\_id) if user\_type is None:

messages.error(request, "Please use valid format for the email id: '<username>.<staff|student|hod>@<college\_domain>'")

return render(request, 'registration.html') username = email\_id.split('@')[0].split('.')[0]

if CustomUser.objects.filter(username=username).exists():

messages.error(request, 'User with this username already exists. Please use different username')

return render(request, 'registration.html') user = CustomUser()

user.username = username user.email = email\_id user.password = password user.user\_type = user\_type user.first\_name = first\_name user.last\_name = last\_name user.save()

if user\_type == CustomUser.STAFF: Staffs.objects.create(admin=user)

elif user\_type == CustomUser.STUDENT: Students.objects.create(admin=user) elif user\_type == CustomUser.HOD: AdminHOD.objects.create(admin=user)

return render(request, 'login\_page.html')

def logout\_user(request):

logout(request)

return HttpResponseRedirect('/')

def get\_user\_type\_from\_email(email\_id):

"""

Returns CustomUser.user\_type corresponding to the given email address email\_id should be in following format: '<username>.<staff|student|hod>@<college\_domain>'

eg.: 'abhishek.staff@jecrc.com' """

try:

email\_id = email\_id.split('@')[0] email\_user\_type = email\_id.split('.')[1]

return CustomUser.EMAIL\_TO\_USER\_TYPE\_MAP[email\_user\_type] except:

return None

Step 8:

* Create or Go to urls.py of student\_management\_app and add the following URLs. from django.contrib import admin

from django.urls import path, include from . import views

from .import HodViews, StaffViews, StudentViews urlpatterns = [

path('admin/', admin.site.urls), path('', views.home, name="home"),

path('contact', views.contact, name="contact"), path('login', views.loginUser, name="login"),

path('logout\_user', views.logout\_user, name="logout\_user"), path('registration', views.registration, name="registration"), path('doLogin', views.doLogin, name="doLogin"), path('doRegistration', views.doRegistration, name="doRegistration"),

# URLS for Student

path('student\_home/', StudentViews.student\_home, name="student\_home"), path('student\_view\_attendance/', StudentViews.student\_view\_attendance,

name="student\_view\_attendance"),

path('student\_view\_attendance\_post/', StudentViews.student\_view\_attendance\_post, name="student\_view\_attendance\_post"),

path('student\_apply\_leave/', StudentViews.student\_apply\_leave, name="student\_apply\_leave"),

path('student\_apply\_leave\_save/', StudentViews.student\_apply\_leave\_save, name="student\_apply\_leave\_save"),

path('student\_feedback/', StudentViews.student\_feedback, name="student\_feedback"), path('student\_feedback\_save/', StudentViews.student\_feedback\_save,

name="student\_feedback\_save"),

path('student\_profile/', StudentViews.student\_profile, name="student\_profile"), path('student\_profile\_update/', StudentViews.student\_profile\_update,

name="student\_profile\_update"),

path('student\_view\_result/', StudentViews.student\_view\_result, name="student\_view\_result"),

# URLS for Staff

path('staff\_home/', StaffViews.staff\_home, name="staff\_home"), path('staff\_take\_attendance/', StaffViews.staff\_take\_attendance,

name="staff\_take\_attendance"),

path('get\_students/', StaffViews.get\_students, name="get\_students"), path('save\_attendance\_data/', StaffViews.save\_attendance\_data,

name="save\_attendance\_data"),

path('staff\_update\_attendance/', StaffViews.staff\_update\_attendance, name="staff\_update\_attendance"),

path('get\_attendance\_dates/', StaffViews.get\_attendance\_dates, name="get\_attendance\_dates"),

path('get\_attendance\_student/', StaffViews.get\_attendance\_student, name="get\_attendance\_student"),

path('update\_attendance\_data/', StaffViews.update\_attendance\_data, name="update\_attendance\_data"),

path('staff\_apply\_leave/', StaffViews.staff\_apply\_leave, name="staff\_apply\_leave"), path('staff\_apply\_leave\_save/', StaffViews.staff\_apply\_leave\_save,

name="staff\_apply\_leave\_save"),

path('staff\_feedback/', StaffViews.staff\_feedback, name="staff\_feedback"), path('staff\_feedback\_save/', StaffViews.staff\_feedback\_save,

name="staff\_feedback\_save"),

path('staff\_profile/', StaffViews.staff\_profile, name="staff\_profile"), path('staff\_profile\_update/', StaffViews.staff\_profile\_update,

name="staff\_profile\_update"),

path('staff\_add\_result/', StaffViews.staff\_add\_result, name="staff\_add\_result"), path('staff\_add\_result\_save/', StaffViews.staff\_add\_result\_save,

name="staff\_add\_result\_save"),

# URL for Admin

path('admin\_home/', HodViews.admin\_home, name="admin\_home"), path('add\_staff/', HodViews.add\_staff, name="add\_staff"), path('add\_staff\_save/', HodViews.add\_staff\_save, name="add\_staff\_save"), path('manage\_staff/', HodViews.manage\_staff, name="manage\_staff"), path('edit\_staff/<staff\_id>/', HodViews.edit\_staff, name="edit\_staff"), path('edit\_staff\_save/', HodViews.edit\_staff\_save, name="edit\_staff\_save"), path('delete\_staff/<staff\_id>/', HodViews.delete\_staff, name="delete\_staff"), path('add\_course/', HodViews.add\_course, name="add\_course"), path('add\_course\_save/', HodViews.add\_course\_save, name="add\_course\_save"), path('manage\_course/', HodViews.manage\_course, name="manage\_course"), path('edit\_course/<course\_id>/', HodViews.edit\_course, name="edit\_course"),

path('edit\_course\_save/', HodViews.edit\_course\_save, name="edit\_course\_save"), path('delete\_course/<course\_id>/', HodViews.delete\_course, name="delete\_course"), path('manage\_session/', HodViews.manage\_session, name="manage\_session"), path('add\_session/', HodViews.add\_session, name="add\_session"), path('add\_session\_save/', HodViews.add\_session\_save, name="add\_session\_save"), path('edit\_session/<session\_id>', HodViews.edit\_session, name="edit\_session"), path('edit\_session\_save/', HodViews.edit\_session\_save, name="edit\_session\_save"), path('delete\_session/<session\_id>/', HodViews.delete\_session, name="delete\_session"), path('add\_student/', HodViews.add\_student, name="add\_student"), path('add\_student\_save/', HodViews.add\_student\_save, name="add\_student\_save"), path('edit\_student/<student\_id>', HodViews.edit\_student, name="edit\_student"), path('edit\_student\_save/', HodViews.edit\_student\_save, name="edit\_student\_save"), path('manage\_student/', HodViews.manage\_student, name="manage\_student"), path('delete\_student/<student\_id>/', HodViews.delete\_student, name="delete\_student"), path('add\_subject/', HodViews.add\_subject, name="add\_subject"), path('add\_subject\_save/', HodViews.add\_subject\_save, name="add\_subject\_save"), path('manage\_subject/', HodViews.manage\_subject, name="manage\_subject"), path('edit\_subject/<subject\_id>/', HodViews.edit\_subject, name="edit\_subject"), path('edit\_subject\_save/', HodViews.edit\_subject\_save, name="edit\_subject\_save"), path('delete\_subject/<subject\_id>/', HodViews.delete\_subject, name="delete\_subject"), path('check\_email\_exist/', HodViews.check\_email\_exist, name="check\_email\_exist"), path('check\_username\_exist/', HodViews.check\_username\_exist,

name="check\_username\_exist"),

path('student\_feedback\_message/', HodViews.student\_feedback\_message, name="student\_feedback\_message"),

path('student\_feedback\_message\_reply/', HodViews.student\_feedback\_message\_reply, name="student\_feedback\_message\_reply"),

path('staff\_feedback\_message/', HodViews.staff\_feedback\_message, name="staff\_feedback\_message"),

path('staff\_feedback\_message\_reply/', HodViews.staff\_feedback\_message\_reply, name="staff\_feedback\_message\_reply"),

path('student\_leave\_view/', HodViews.student\_leave\_view, name="student\_leave\_view"), path('student\_leave\_approve/<leave\_id>/', HodViews.student\_leave\_approve,

name="student\_leave\_approve"),

path('student\_leave\_reject/<leave\_id>/', HodViews.student\_leave\_reject, name="student\_leave\_reject"),

path('staff\_leave\_view/', HodViews.staff\_leave\_view, name="staff\_leave\_view"), path('staff\_leave\_approve/<leave\_id>/', HodViews.staff\_leave\_approve,

name="staff\_leave\_approve"),

path('staff\_leave\_reject/<leave\_id>/', HodViews.staff\_leave\_reject, name="staff\_leave\_reject"),

path('admin\_view\_attendance/', HodViews.admin\_view\_attendance, name="admin\_view\_attendance"),

path('admin\_get\_attendance\_dates/', HodViews.admin\_get\_attendance\_dates, name="admin\_get\_attendance\_dates"),

path('admin\_get\_attendance\_student/', HodViews.admin\_get\_attendance\_student, name="admin\_get\_attendance\_student"),

path('admin\_profile/', HodViews.admin\_profile, name="admin\_profile"), path('admin\_profile\_update/', HodViews.admin\_profile\_update,

name="admin\_profile\_update"),

]

Step 9:

* + Now create a file StudentViews.py. It contains the views that are used on the student

Interface.

from django.shortcuts import render, redirect

from django.http import HttpResponse, HttpResponseRedirect from django.contrib import messages

from django.core.files.storage import FileSystemStorage from django.urls import reverse

import datetime

from .models import CustomUser, Staffs, Courses, Subjects, Students, Attendance,

AttendanceReport, LeaveReportStudent, FeedBackStudent, StudentResult def student\_home(request):

student\_obj = Students.objects.get(admin=request.user.id)

total\_attendance = AttendanceReport.objects.filter(student\_id=student\_obj).count() attendance\_present = AttendanceReport.objects.filter(student\_id=student\_obj,

status=True).count()

attendance\_absent = AttendanceReport.objects.filter(student\_id=student\_obj, status=False).count()

course\_obj = Courses.objects.get(id=student\_obj.course\_id.id) total\_subjects = Subjects.objects.filter(course\_id=course\_obj).count() subject\_name = []

data\_present = [] data\_absent = []

subject\_data = Subjects.objects.filter(course\_id=student\_obj.course\_id) for subject in subject\_data:

attendance = Attendance.objects.filter(subject\_id=subject.id) attendance\_present\_count =

AttendanceReport.objects.filter(attendance\_id in=attendance, status=True, student\_id=student\_obj.id).count()

attendance\_absent\_count = AttendanceReport.objects.filter(attendance\_id in=attendance,

status=False, student\_id=student\_obj.id).count()

subject\_name.append(subject.subject\_name) data\_present.append(attendance\_present\_count) data\_absent.append(attendance\_absent\_count)

context={

"total\_attendance": total\_attendance, "attendance\_present": attendance\_present, "attendance\_absent": attendance\_absent, "total\_subjects": total\_subjects, "subject\_name": subject\_name, "data\_present": data\_present, "data\_absent": data\_absent

}

return render(request, "student\_template/student\_home\_template.html")

def student\_view\_attendance(request):

# Getting Logged in Student Data

student = Students.objects.get(admin=request.user.id)

# Getting Course Enrolled of LoggedIn Student course = student.course\_id

# Getting the Subjects of Course Enrolled

subjects = Subjects.objects.filter(course\_id=course) context = {

"subjects": subjects

}

return render(request, "student\_template/student\_view\_attendance.html", context)

def student\_view\_attendance\_post(request):

if request.method != "POST":

messages.error(request, "Invalid Method") return redirect('student\_view\_attendance') else:

# Getting all the Input Data

subject\_id = request.POST.get('subject') start\_date = request.POST.get('start\_date') end\_date = request.POST.get('end\_date') # Parsing the date data into Python object

start\_date\_parse = datetime.datetime.strptime(start\_date, '%Y-%m-%d').date() end\_date\_parse = datetime.datetime.strptime(end\_date, '%Y-%m-%d').date() # Getting all the Subject Data based on Selected Subject

subject\_obj = Subjects.objects.get(id=subject\_id)

# Getting Logged In User Data

user\_obj = CustomUser.objects.get(id=request.user.id)

# Getting Student Data Based on Logged in Data stud\_obj = Students.objects.get(admin=user\_obj)

# Now Accessing Attendance Data based on the Range of Date # Selected and Subject Selected

attendance = Attendance.objects.filter(attendance\_date range=(start\_date\_parse, end\_date\_parse),

subject\_id=subject\_obj)

# Getting Attendance Report based on the attendance # details obtained above

attendance\_reports = AttendanceReport.objects.filter(attendance\_id in=attendance, student\_id=stud\_obj)

context = {

"subject\_obj": subject\_obj, "attendance\_reports": attendance\_reports

}

return render(request, 'student\_template/student\_attendance\_data.html', context)

def student\_apply\_leave(request):

student\_obj = Students.objects.get(admin=request.user.id)

leave\_data = LeaveReportStudent.objects.filter(student\_id=student\_obj) context = {

"leave\_data": leave\_data

}

return render(request, 'student\_template/student\_apply\_leave.html', context)

def student\_apply\_leave\_save(request):

if request.method != "POST": messages.error(request, "Invalid Method") return redirect('student\_apply\_leave') else:

leave\_date = request.POST.get('leave\_date') leave\_message = request.POST.get('leave\_message') student\_obj = Students.objects.get(admin=request.user.id) try:

leave\_report = LeaveReportStudent(student\_id=student\_obj, leave\_date=leave\_date,

leave\_message=leave\_message, leave\_status=0)

leave\_report.save()

messages.success(request, "Applied for Leave.") return redirect('student\_apply\_leave')

except:

messages.error(request, "Failed to Apply Leave") return redirect('student\_apply\_leave')

def student\_feedback(request):

student\_obj = Students.objects.get(admin=request.user.id) feedback\_data = FeedBackStudent.objects.filter(student\_id=student\_obj) context = {

"feedback\_data": feedback\_data

}

return render(request, 'student\_template/student\_feedback.html', context) def student\_feedback\_save(request):

if request.method != "POST": messages.error(request, "Invalid Method.") return redirect('student\_feedback')

else:

feedback = request.POST.get('feedback\_message') student\_obj = Students.objects.get(admin=request.user.id) try:

add\_feedback = FeedBackStudent(student\_id=student\_obj, feedback=feedback,

feedback\_reply="") add\_feedback.save()

messages.success(request, "Feedback Sent.") return redirect('student\_feedback')

except:

messages.error(request, "Failed to Send Feedback.") return redirect('student\_feedback')

def student\_profile(request):

user = CustomUser.objects.get(id=request.user.id) student = Students.objects.get(admin=user) context={

"user": user, "student": student

}

return render(request, 'student\_template/student\_profile.html', context)

def student\_profile\_update(request):

if request.method != "POST": messages.error(request, "Invalid Method!") return redirect('student\_profile')

else:

first\_name = request.POST.get('first\_name') last\_name = request.POST.get('last\_name') password = request.POST.get('password') address = request.POST.get('address')

try:

customuser = CustomUser.objects.get(id=request.user.id) customuser.first\_name = first\_name customuser.last\_name = last\_name

if password != None and password != "": customuser.set\_password(password) customuser.save()

student = Students.objects.get(admin=customuser.id) student.address = address

student.save()

messages.success(request, "Profile Updated Successfully") return redirect('student\_profile')

except:

messages.error(request, "Failed to Update Profile") return redirect('student\_profile')

def student\_view\_result(request):

student = Students.objects.get(admin=request.user.id) student\_result = StudentResult.objects.filter(student\_id=student.id) context = {

"student\_result": student\_result,

}

return render(request, "student\_template/student\_view\_result.html", context)

Step 10:

* + Now add the StaffViews.py. It contains the views of the staff interface.

from django.shortcuts import render, redirect

from django.http import HttpResponse, HttpResponseRedirect, JsonResponse from django.contrib import messages

from django.core.files.storage import FileSystemStorage from django.urls import reverse

from django.views.decorators.csrf import csrf\_exempt from django.core import serializers

import json

from .models import CustomUser, Staffs, Courses, Subjects, Students, SessionYearModel,

Attendance, AttendanceReport, LeaveReportStaff, FeedBackStaffs, StudentResult

def staff\_home(request):

# Fetching All Students under Staff print(request.user.id)

subjects = Subjects.objects.filter(staff\_id=request.user.id) print(subjects)

course\_id\_list = []

for subject in subjects:

course = Courses.objects.get(id=subject.course\_id.id) course\_id\_list.append(course.id)

final\_course = []

# Removing Duplicate Course Id for course\_id in course\_id\_list:

if course\_id not in final\_course: final\_course.append(course\_id)

print(final\_course)

students\_count = Students.objects.filter(course\_id in=final\_course).count() subject\_count = subjects.count()

print(subject\_count) print(students\_count)

# Fetch All Attendance Count

attendance\_count = Attendance.objects.filter(subject\_id in=subjects).count()

# Fetch All Approve Leave # print(request.user)

print(request.user.user\_type)

staff = Staffs.objects.get(admin=request.user.id)

leave\_count = LeaveReportStaff.objects.filter(staff\_id=staff.id, leave\_status=1).count()

# Fetch Attendance Data by Subjects subject\_list = []

attendance\_list = [] for subject in subjects:

attendance\_count1 = Attendance.objects.filter(subject\_id=subject.id).count() subject\_list.append(subject.subject\_name) attendance\_list.append(attendance\_count1)

students\_attendance = Students.objects.filter(course\_id in=final\_course) student\_list = []

student\_list\_attendance\_present = [] student\_list\_attendance\_absent = [] for student in students\_attendance:

attendance\_present\_count = AttendanceReport.objects.filter(status=True, student\_id=student.id).count()

attendance\_absent\_count = AttendanceReport.objects.filter(status=False, student\_id=student.id).count()

student\_list.append(student.admin.first\_name+" "+ student.admin.last\_name) student\_list\_attendance\_present.append(attendance\_present\_count) student\_list\_attendance\_absent.append(attendance\_absent\_count)

context={

"students\_count": students\_count, "attendance\_count": attendance\_count, "leave\_count": leave\_count, "subject\_count": subject\_count, "subject\_list": subject\_list, "attendance\_list": attendance\_list, "student\_list": student\_list,

"attendance\_present\_list": student\_list\_attendance\_present, "attendance\_absent\_list": student\_list\_attendance\_absent

}

return render(request, "staff\_template/staff\_home\_template.html", context)

def staff\_take\_attendance(request):

subjects = Subjects.objects.filter(staff\_id=request.user.id) session\_years = SessionYearModel.objects.all()

context = { "subjects": subjects,

"session\_years": session\_years

}

return render(request, "staff\_template/take\_attendance\_template.html", context)

def staff\_apply\_leave(request): print(request.user.id)

staff\_obj = Staffs.objects.get(admin=request.user.id) leave\_data = LeaveReportStaff.objects.filter(staff\_id=staff\_obj) context = {

"leave\_data": leave\_data

}

return render(request, "staff\_template/staff\_apply\_leave\_template.html", context)

def staff\_apply\_leave\_save(request):

if request.method != "POST": messages.error(request, "Invalid Method") return redirect('staff\_apply\_leave')

else:

leave\_date = request.POST.get('leave\_date') leave\_message = request.POST.get('leave\_message') staff\_obj = Staffs.objects.get(admin=request.user.id) try:

leave\_report = LeaveReportStaff(staff\_id=staff\_obj, leave\_date=leave\_date, leave\_message=leave\_message, leave\_status=0)

leave\_report.save()

messages.success(request, "Applied for Leave.") return redirect('staff\_apply\_leave')

except:

messages.error(request, "Failed to Apply Leave") return redirect('staff\_apply\_leave')

def staff\_feedback(request):

return render(request, "staff\_template/staff\_feedback\_template.html")

def staff\_feedback\_save(request):

if request.method != "POST": messages.error(request, "Invalid Method.") return redirect('staff\_feedback')

else:

feedback = request.POST.get('feedback\_message') staff\_obj = Staffs.objects.get(admin=request.user.id) try:

add\_feedback = FeedBackStaffs(staff\_id=staff\_obj, feedback=feedback,

feedback\_reply="") add\_feedback.save()

messages.success(request, "Feedback Sent.") return redirect('staff\_feedback')

except:

messages.error(request, "Failed to Send Feedback.") return redirect('staff\_feedback')

@csrf\_exempt

def get\_students(request):

subject\_id = request.POST.get("subject") session\_year = request.POST.get("session\_year") # Students enroll to Course, Course has Subjects

# Getting all data from subject model based on subject\_id

subject\_model = Subjects.objects.get(id=subject\_id) session\_model = SessionYearModel.objects.get(id=session\_year)

students = Students.objects.filter(course\_id=subject\_model.course\_id, session\_year\_id=session\_model)

# Only Passing Student Id and Student Name Only list\_data = []

for student in students: data\_small={"id":student.admin.id,

"name":student.admin.first\_name+" "+student.admin.last\_name} list\_data.append(data\_small)

return JsonResponse(json.dumps(list\_data), content\_type="application/json", safe=False)

@csrf\_exempt

def save\_attendance\_data(request):

# Get Values from Staf Take Attendance form via AJAX (JavaScript) # Use getlist to access HTML Array/List Input Data

student\_ids = request.POST.get("student\_ids") subject\_id = request.POST.get("subject\_id") attendance\_date = request.POST.get("attendance\_date") session\_year\_id = request.POST.get("session\_year\_id") subject\_model = Subjects.objects.get(id=subject\_id)

session\_year\_model = SessionYearModel.objects.get(id=session\_year\_id) json\_student = json.loads(student\_ids)

try:

# First Attendance Data is Saved on Attendance Model attendance = Attendance(subject\_id=subject\_model,

attendance\_date=attendance\_date, session\_year\_id=session\_year\_model)

attendance.save()

for stud in json\_student:

# Attendance of Individual Student saved on AttendanceReport Model student = Students.objects.get(admin=stud['id'])

attendance\_report = AttendanceReport(student\_id=student, attendance\_id=attendance,

status=stud['status']) attendance\_report.save() return HttpResponse("OK") except:

return HttpResponse("Error")

def staff\_update\_attendance(request):

subjects = Subjects.objects.filter(staff\_id=request.user.id) session\_years = SessionYearModel.objects.all()

context = { "subjects": subjects,

"session\_years": session\_years

}

return render(request, "staff\_template/update\_attendance\_template.html", context) @csrf\_exempt

def get\_attendance\_dates(request):

# Getting Values from Ajax POST 'Fetch Student' subject\_id = request.POST.get("subject") session\_year = request.POST.get("session\_year\_id") # Students enroll to Course, Course has Subjects

# Getting all data from subject model based on subject\_id subject\_model = Subjects.objects.get(id=subject\_id) session\_model = SessionYearModel.objects.get(id=session\_year) attendance = Attendance.objects.filter(subject\_id=subject\_model,

session\_year\_id=session\_model)

# Only Passing Student Id and Student Name Only list\_data = []

for attendance\_single in attendance: data\_small={"id":attendance\_single.id,

"attendance\_date":str(attendance\_single.attendance\_date), "session\_year\_id":attendance\_single.session\_year\_id.id}

list\_data.append(data\_small)

return JsonResponse(json.dumps(list\_data), content\_type="application/json", safe=False)

@csrf\_exempt

def get\_attendance\_student(request):

# Getting Values from Ajax POST 'Fetch Student' attendance\_date = request.POST.get('attendance\_date') attendance = Attendance.objects.get(id=attendance\_date)

attendance\_data = AttendanceReport.objects.filter(attendance\_id=attendance) # Only Passing Student Id and Student Name Only

list\_data = []

for student in attendance\_data: data\_small={"id":student.student\_id.admin.id,

"name":student.student\_id.admin.first\_name+" "+student.student\_id.admin.last\_name, "status":student.status}

list\_data.append(data\_small)

return JsonResponse(json.dumps(list\_data), content\_type="application/json", safe=False)

@csrf\_exempt

def update\_attendance\_data(request): student\_ids = request.POST.get("student\_ids")

attendance\_date = request.POST.get("attendance\_date") attendance = Attendance.objects.get(id=attendance\_date) json\_student = json.loads(student\_ids)

try:

for stud in json\_student:

# Attendance of Individual Student saved on AttendanceReport Model student = Students.objects.get(admin=stud['id'])

attendance\_report = AttendanceReport.objects.get(student\_id=student, attendance\_id=attendance)

attendance\_report.status=stud['status'] attendance\_report.save()

return HttpResponse("OK") except:

return HttpResponse("Error")

def staff\_profile(request):

user = CustomUser.objects.get(id=request.user.id) staff = Staffs.objects.get(admin=user)

context={ "user": user, "staff": staff

}

return render(request, 'staff\_template/staff\_profile.html', context)

def staff\_profile\_update(request):

if request.method != "POST": messages.error(request, "Invalid Method!") return redirect('staff\_profile')

else:

first\_name = request.POST.get('first\_name') last\_name = request.POST.get('last\_name') password = request.POST.get('password') address = request.POST.get('address')

try:

customuser = CustomUser.objects.get(id=request.user.id) customuser.first\_name = first\_name customuser.last\_name = last\_name

if password != None and password != "": customuser.set\_password(password) customuser.save()

staff = Staffs.objects.get(admin=customuser.id) staff.address = address

staff.save()

messages.success(request, "Profile Updated Successfully") return redirect('staff\_profile')

except:

messages.error(request, "Failed to Update Profile") return redirect('staff\_profile')

def staff\_add\_result(request):

subjects = Subjects.objects.filter(staff\_id=request.user.id) session\_years = SessionYearModel.objects.all()

context = { "subjects": subjects,

"session\_years": session\_years,

}

return render(request, "staff\_template/add\_result\_template.html", context)

def staff\_add\_result\_save(request):

if request.method != "POST":

messages.error(request, "Invalid Method") return redirect('staff\_add\_result')

else:

student\_admin\_id = request.POST.get('student\_list') assignment\_marks = request.POST.get('assignment\_marks') exam\_marks = request.POST.get('exam\_marks')

subject\_id = request.POST.get('subject')

student\_obj = Students.objects.get(admin=student\_admin\_id) subject\_obj = Subjects.objects.get(id=subject\_id)

try:

# Check if Students Result Already Exists or not

check\_exist = StudentResult.objects.filter(subject\_id=subject\_obj, student\_id=student\_obj).exists()

if check\_exist:

result = StudentResult.objects.get(subject\_id=subject\_obj, student\_id=student\_obj)

result.subject\_assignment\_marks = assignment\_marks result.subject\_exam\_marks = exam\_marks result.save()

messages.success(request, "Result Updated Successfully!") return redirect('staff\_add\_result')

else:

result = StudentResult(student\_id=student\_obj, subject\_id=subject\_obj, subject\_exam\_marks=exam\_marks, subject\_assignment\_marks=assignment\_marks)

result.save()

messages.success(request, "Result Added Successfully!") return redirect('staff\_add\_result')

except:

messages.error(request, "Failed to Add Result!") return redirect('staff\_add\_result')

Step 11:

* + Now add the HodViews.py. It contains the views of the HOD interface.

from django.shortcuts import render, redirect

from django.http import HttpResponse, HttpResponseRedirect, JsonResponse from django.contrib import messages

from django.core.files.storage import FileSystemStorage from django.urls import reverse

from django.views.decorators.csrf import csrf\_exempt import json

from .forms import AddStudentForm, EditStudentForm

from .models import CustomUser, Staffs, Courses, Subjects, Students, SessionYearModel, FeedBackStudent, FeedBackStaffs, LeaveReportStudent, LeaveReportStaff, Attendance,

AttendanceReport

def admin\_home(request):

all\_student\_count = Students.objects.all().count() subject\_count = Subjects.objects.all().count() course\_count = Courses.objects.all().count() staff\_count = Staffs.objects.all().count() course\_all = Courses.objects.all() course\_name\_list = []

subject\_count\_list = [] student\_count\_list\_in\_course = [] for course in course\_all:

subjects = Subjects.objects.filter(course\_id=course.id).count() students = Students.objects.filter(course\_id=course.id).count() course\_name\_list.append(course.course\_name) subject\_count\_list.append(subjects) student\_count\_list\_in\_course.append(students)

subject\_all = Subjects.objects.all() subject\_list = [] student\_count\_list\_in\_subject = [] for subject in subject\_all:

course = Courses.objects.get(id=subject.course\_id.id)

student\_count = Students.objects.filter(course\_id=course.id).count() subject\_list.append(subject.subject\_name) student\_count\_list\_in\_subject.append(student\_count)

# For Saffs staff\_attendance\_present\_list=[] staff\_attendance\_leave\_list=[] staff\_name\_list=[]

staffs = Staffs.objects.all() for staff in staffs:

subject\_ids = Subjects.objects.filter(staff\_id=staff.admin.id)

attendance = Attendance.objects.filter(subject\_id in=subject\_ids).count() leaves = LeaveReportStaff.objects.filter(staff\_id=staff.id,

leave\_status=1).count() staff\_attendance\_present\_list.append(attendance) staff\_attendance\_leave\_list.append(leaves) staff\_name\_list.append(staff.admin.first\_name)

# For Students student\_attendance\_present\_list=[] student\_attendance\_leave\_list=[] student\_name\_list=[]

students = Students.objects.all() for student in students:

attendance = AttendanceReport.objects.filter(student\_id=student.id, status=True).count()

absent = AttendanceReport.objects.filter(student\_id=student.id, status=False).count()

leaves = LeaveReportStudent.objects.filter(student\_id=student.id, leave\_status=1).count()

student\_attendance\_present\_list.append(attendance) student\_attendance\_leave\_list.append(leaves+absent) student\_name\_list.append(student.admin.first\_name)

context={

"all\_student\_count": all\_student\_count, "subject\_count": subject\_count, "course\_count": course\_count, "staff\_count": staff\_count, "course\_name\_list": course\_name\_list, "subject\_count\_list": subject\_count\_list,

"student\_count\_list\_in\_course": student\_count\_list\_in\_course, "subject\_list": subject\_list,

"student\_count\_list\_in\_subject": student\_count\_list\_in\_subject, "staff\_attendance\_present\_list": staff\_attendance\_present\_list, "staff\_attendance\_leave\_list": staff\_attendance\_leave\_list, "staff\_name\_list": staff\_name\_list, "student\_attendance\_present\_list": student\_attendance\_present\_list, "student\_attendance\_leave\_list": student\_attendance\_leave\_list, "student\_name\_list": student\_name\_list,

}

return render(request, "hod\_template/home\_content.html", context)

def add\_staff(request):

return render(request, "hod\_template/add\_staff\_template.html")

def add\_staff\_save(request):

if request.method != "POST": messages.error(request, "Invalid Method ") return redirect('add\_staff')

else:

first\_name = request.POST.get('first\_name') last\_name = request.POST.get('last\_name') username = request.POST.get('username') email = request.POST.get('email')

password = request.POST.get('password') address = request.POST.get('address') try:

user = CustomUser.objects.create\_user(username=username, password=password,

email=email, first\_name=first\_name, last\_name=last\_name, user\_type=2)

user.staffs.address = address user.save()

messages.success(request, "Staff Added Successfully!") return redirect('add\_staff')

except:

messages.error(request, "Failed to Add Staff!") return redirect('add\_staff')

def manage\_staff(request):

staffs = Staffs.objects.all() context = {

"staffs": staffs

}

return render(request, "hod\_template/manage\_staff\_template.html", context)

def edit\_staff(request, staff\_id):

staff = Staffs.objects.get(admin=staff\_id) context = {

"staff": staff, "id": staff\_id

}

return render(request, "hod\_template/edit\_staff\_template.html", context)

def edit\_staff\_save(request):

if request.method != "POST":

return HttpResponse("<h2>Method Not Allowed</h2>") else:

staff\_id = request.POST.get('staff\_id') username = request.POST.get('username') email = request.POST.get('email') first\_name = request.POST.get('first\_name') last\_name = request.POST.get('last\_name') address = request.POST.get('address')

try:

# INSERTING into Customuser Model

user = CustomUser.objects.get(id=staff\_id) user.first\_name = first\_name user.last\_name = last\_name

user.email = email user.username = username user.save()

# INSERTING into Staff Model

staff\_model = Staffs.objects.get(admin=staff\_id) staff\_model.address = address staff\_model.save()

messages.success(request, "Staff Updated Successfully.") return redirect('/edit\_staff/'+staff\_id)

except:

messages.error(request, "Failed to Update Staff.") return redirect('/edit\_staff/'+staff\_id)

def delete\_staff(request, staff\_id):

staff = Staffs.objects.get(admin=staff\_id)

try:

staff.delete()

messages.success(request, "Staff Deleted Successfully.") return redirect('manage\_staff')

except:

messages.error(request, "Failed to Delete Staff.") return redirect('manage\_staff')

def add\_course(request):

return render(request, "hod\_template/add\_course\_template.html")

def add\_course\_save(request):

if request.method != "POST":

messages.error(request, "Invalid Method!") return redirect('add\_course')

else:

course = request.POST.get('course') try:

course\_model = Courses(course\_name=course) course\_model.save()

messages.success(request, "Course Added Successfully!") return redirect('add\_course')

except:

messages.error(request, "Failed to Add Course!") return redirect('add\_course')

def manage\_course(request):

courses = Courses.objects.all() context = {

"courses": courses

}

return render(request, 'hod\_template/manage\_course\_template.html', context)

def edit\_course(request, course\_id):

course = Courses.objects.get(id=course\_id) context = {

"course": course, "id": course\_id

}

return render(request, 'hod\_template/edit\_course\_template.html', context)

def edit\_course\_save(request):

if request.method != "POST": HttpResponse("Invalid Method") else:

course\_id = request.POST.get('course\_id') course\_name = request.POST.get('course') try:

course = Courses.objects.get(id=course\_id)

course.course\_name = course\_name course.save()

messages.success(request, "Course Updated Successfully.") return redirect('/edit\_course/'+course\_id)

except:

messages.error(request, "Failed to Update Course.") return redirect('/edit\_course/'+course\_id)

def delete\_course(request, course\_id):

course = Courses.objects.get(id=course\_id) try:

course.delete()

messages.success(request, "Course Deleted Successfully.") return redirect('manage\_course')

except:

messages.error(request, "Failed to Delete Course.") return redirect('manage\_course')

def manage\_session(request):

session\_years = SessionYearModel.objects.all() context = {

"session\_years": session\_years

}

return render(request, "hod\_template/manage\_session\_template.html", context)

def add\_session(request):

return render(request, "hod\_template/add\_session\_template.html")

def add\_session\_save(request):

if request.method != "POST": messages.error(request, "Invalid Method") return redirect('add\_course')

else:

session\_start\_year = request.POST.get('session\_start\_year') session\_end\_year = request.POST.get('session\_end\_year') try:

sessionyear = SessionYearModel(session\_start\_year=session\_start\_year, session\_end\_year=session\_end\_year)

sessionyear.save()

messages.success(request, "Session Year added Successfully!") return redirect("add\_session")

except:

messages.error(request, "Failed to Add Session Year") return redirect("add\_session")

def edit\_session(request, session\_id):

session\_year = SessionYearModel.objects.get(id=session\_id) context = {

"session\_year": session\_year

}

return render(request, "hod\_template/edit\_session\_template.html", context)

def edit\_session\_save(request):

if request.method != "POST": messages.error(request, "Invalid Method!") return redirect('manage\_session')

else:

session\_id = request.POST.get('session\_id') session\_start\_year = request.POST.get('session\_start\_year') session\_end\_year = request.POST.get('session\_end\_year') try:

session\_year = SessionYearModel.objects.get(id=session\_id) session\_year.session\_start\_year = session\_start\_year session\_year.session\_end\_year = session\_end\_year session\_year.save()

messages.success(request, "Session Year Updated Successfully.") return redirect('/edit\_session/'+session\_id)

except:

messages.error(request, "Failed to Update Session Year.") return redirect('/edit\_session/'+session\_id)

def delete\_session(request, session\_id):

session = SessionYearModel.objects.get(id=session\_id) try:

session.delete()

messages.success(request, "Session Deleted Successfully.") return redirect('manage\_session')

except:

messages.error(request, "Failed to Delete Session.") return redirect('manage\_session')

def add\_student(request):

form = AddStudentForm() context = {

"form": form

}

return render(request, 'hod\_template/add\_student\_template.html', context)

def add\_student\_save(request):

if request.method != "POST": messages.error(request, "Invalid Method") return redirect('add\_student')

else:

form = AddStudentForm(request.POST, request.FILES) if form.is\_valid():

first\_name = form.cleaned\_data['first\_name'] last\_name = form.cleaned\_data['last\_name'] username = form.cleaned\_data['username'] email = form.cleaned\_data['email']

password = form.cleaned\_data['password']

address = form.cleaned\_data['address'] session\_year\_id = form.cleaned\_data['session\_year\_id'] course\_id = form.cleaned\_data['course\_id']

gender = form.cleaned\_data['gender']

if len(request.FILES) != 0:

profile\_pic = request.FILES['profile\_pic'] fs = FileSystemStorage()

filename = fs.save(profile\_pic.name, profile\_pic) profile\_pic\_url = fs.url(filename)

else:

profile\_pic\_url = None

try:

user = CustomUser.objects.create\_user(username=username, password=password,

email=email, first\_name=first\_name, last\_name=last\_name, user\_type=3)

user.students.address = address

course\_obj = Courses.objects.get(id=course\_id) user.students.course\_id = course\_obj

session\_year\_obj = SessionYearModel.objects.get(id=session\_year\_id) user.students.session\_year\_id = session\_year\_obj user.students.gender = gender

user.students.profile\_pic = profile\_pic\_url user.save()

messages.success(request, "Student Added Successfully!") return redirect('add\_student')

except:

messages.error(request, "Failed to Add Student!") return redirect('add\_student')

else:

return redirect('add\_student')

def manage\_student(request):

students = Students.objects.all() context = {

"students": students

}

return render(request, 'hod\_template/manage\_student\_template.html', context)

def edit\_student(request, student\_id):

# Adding Student ID into Session Variable request.session['student\_id'] = student\_id

student = Students.objects.get(admin=student\_id) form = EditStudentForm()

# Filling the form with Data from Database form.fields['email'].initial = student.admin.email

form.fields['username'].initial = student.admin.username form.fields['first\_name'].initial = student.admin.first\_name form.fields['last\_name'].initial = student.admin.last\_name form.fields['address'].initial = student.address form.fields['course\_id'].initial = student.course\_id.id form.fields['gender'].initial = student.gender form.fields['session\_year\_id'].initial = student.session\_year\_id.id context = {

"id": student\_id,

"username": student.admin.username, "form": form

}

return render(request, "hod\_template/edit\_student\_template.html", context)

def edit\_student\_save(request):

if request.method != "POST":

return HttpResponse("Invalid Method!") else:

student\_id = request.session.get('student\_id') if student\_id == None:

return redirect('/manage\_student')

form = EditStudentForm(request.POST, request.FILES) if form.is\_valid():

email = form.cleaned\_data['email'] username = form.cleaned\_data['username']

first\_name = form.cleaned\_data['first\_name'] last\_name = form.cleaned\_data['last\_name'] address = form.cleaned\_data['address'] course\_id = form.cleaned\_data['course\_id'] gender = form.cleaned\_data['gender']

session\_year\_id = form.cleaned\_data['session\_year\_id'] # Getting Profile Pic first

# First Check whether the file is selected or not # Upload only if file is selected

if len(request.FILES) != 0:

profile\_pic = request.FILES['profile\_pic'] fs = FileSystemStorage()

filename = fs.save(profile\_pic.name, profile\_pic) profile\_pic\_url = fs.url(filename)

else:

profile\_pic\_url = None try:

# First Update into Custom User Model

user = CustomUser.objects.get(id=student\_id) user.first\_name = first\_name

user.last\_name = last\_name user.email = email user.username = username user.save()

# Then Update Students Table

student\_model = Students.objects.get(admin=student\_id)

student\_model.address = address

course = Courses.objects.get(id=course\_id) student\_model.course\_id = course

session\_year\_obj = SessionYearModel.objects.get(id=session\_year\_id) student\_model.session\_year\_id = session\_year\_obj student\_model.gender = gender

if profile\_pic\_url != None: student\_model.profile\_pic = profile\_pic\_url student\_model.save()

# Delete student\_id SESSION after the data is updated del request.session['student\_id']

messages.success(request, "Student Updated Successfully!") return redirect('/edit\_student/'+student\_id)

except:

messages.success(request, "Failed to Uupdate Student.") return redirect('/edit\_student/'+student\_id)

else:

return redirect('/edit\_student/'+student\_id)

def delete\_student(request, student\_id):

student = Students.objects.get(admin=student\_id) try:

student.delete()

messages.success(request, "Student Deleted Successfully.") return redirect('manage\_student')

except:

messages.error(request, "Failed to Delete Student.") return redirect('manage\_student')

def add\_subject(request):

courses = Courses.objects.all()

staffs = CustomUser.objects.filter(user\_type='2') context = {

"courses": courses, "staffs": staffs

}

return render(request, 'hod\_template/add\_subject\_template.html', context)

def add\_subject\_save(request):

if request.method != "POST": messages.error(request, "Method Not Allowed!") return redirect('add\_subject')

else:

subject\_name = request.POST.get('subject') course\_id = request.POST.get('course') course = Courses.objects.get(id=course\_id)

staff\_id = request.POST.get('staff')

staff = CustomUser.objects.get(id=staff\_id) try:

subject = Subjects(subject\_name=subject\_name, course\_id=course,

staff\_id=staff) subject.save()

messages.success(request, "Subject Added Successfully!") return redirect('add\_subject')

except:

messages.error(request, "Failed to Add Subject!") return redirect('add\_subject')

def manage\_subject(request):

subjects = Subjects.objects.all() context = {

"subjects": subjects

}

return render(request, 'hod\_template/manage\_subject\_template.html', context)

def edit\_subject(request, subject\_id):

subject = Subjects.objects.get(id=subject\_id) courses = Courses.objects.all()

staffs = CustomUser.objects.filter(user\_type='2') context = {

"subject": subject, "courses": courses, "staffs": staffs, "id": subject\_id

}

return render(request, 'hod\_template/edit\_subject\_template.html', context)

def edit\_subject\_save(request):

if request.method != "POST": HttpResponse("Invalid Method.") else:

subject\_id = request.POST.get('subject\_id') subject\_name = request.POST.get('subject') course\_id = request.POST.get('course') staff\_id = request.POST.get('staff')

try:

subject = Subjects.objects.get(id=subject\_id) subject.subject\_name = subject\_name course = Courses.objects.get(id=course\_id) subject.course\_id = course

staff = CustomUser.objects.get(id=staff\_id) subject.staff\_id = staff

subject.save()

messages.success(request, "Subject Updated Successfully.")

return HttpResponseRedirect(reverse("edit\_subject", kwargs={"subject\_id":subject\_id}))

except:

messages.error(request, "Failed to Update Subject.") return HttpResponseRedirect(reverse("edit\_subject",

kwargs={"subject\_id":subject\_id}))

def delete\_subject(request, subject\_id):

subject = Subjects.objects.get(id=subject\_id) try:

subject.delete()

messages.success(request, "Subject Deleted Successfully.") return redirect('manage\_subject')

except:

messages.error(request, "Failed to Delete Subject.") return redirect('manage\_subject')

@csrf\_exempt

def check\_email\_exist(request):

email = request.POST.get("email")

user\_obj = CustomUser.objects.filter(email=email).exists() if user\_obj:

return HttpResponse(True) else:

return HttpResponse(False)

@csrf\_exempt

def check\_username\_exist(request):

username = request.POST.get("username")

user\_obj = CustomUser.objects.filter(username=username).exists() if user\_obj:

return HttpResponse(True) else:

return HttpResponse(False)

def student\_feedback\_message(request):

feedbacks = FeedBackStudent.objects.all() context = {

"feedbacks": feedbacks

}

return render(request, 'hod\_template/student\_feedback\_template.html', context)

@csrf\_exempt

def student\_feedback\_message\_reply(request): feedback\_id = request.POST.get('id') feedback\_reply = request.POST.get('reply')

try:

feedback = FeedBackStudent.objects.get(id=feedback\_id) feedback.feedback\_reply = feedback\_reply feedback.save()

return HttpResponse("True")

except:

return HttpResponse("False")

def staff\_feedback\_message(request):

feedbacks = FeedBackStaffs.objects.all() context = {

"feedbacks": feedbacks

}

return render(request, 'hod\_template/staff\_feedback\_template.html', context)

@csrf\_exempt

def staff\_feedback\_message\_reply(request):

feedback\_id = request.POST.get('id') feedback\_reply = request.POST.get('reply') try:

feedback = FeedBackStaffs.objects.get(id=feedback\_id) feedback.feedback\_reply = feedback\_reply feedback.save()

return HttpResponse("True") except:

return HttpResponse("False")

def student\_leave\_view(request):

leaves = LeaveReportStudent.objects.all() context = {

"leaves": leaves

}

return render(request, 'hod\_template/student\_leave\_view.html', context) def student\_leave\_approve(request, leave\_id):

leave = LeaveReportStudent.objects.get(id=leave\_id) leave.leave\_status = 1

leave.save()

return redirect('student\_leave\_view')

def student\_leave\_reject(request, leave\_id):

leave = LeaveReportStudent.objects.get(id=leave\_id) leave.leave\_status = 2

leave.save()

return redirect('student\_leave\_view')

def staff\_leave\_view(request):

leaves = LeaveReportStaff.objects.all() context = {

"leaves": leaves

}

return render(request, 'hod\_template/staff\_leave\_view.html', context)

def staff\_leave\_approve(request, leave\_id):

leave = LeaveReportStaff.objects.get(id=leave\_id) leave.leave\_status = 1

leave.save()

return redirect('staff\_leave\_view')

def staff\_leave\_reject(request, leave\_id):

leave = LeaveReportStaff.objects.get(id=leave\_id) leave.leave\_status = 2

leave.save()

return redirect('staff\_leave\_view')

def admin\_view\_attendance(request):

subjects = Subjects.objects.all()

session\_years = SessionYearModel.objects.all() context = {

"subjects": subjects, "session\_years": session\_years

}

return render(request, "hod\_template/admin\_view\_attendance.html", context)

@csrf\_exempt

def admin\_get\_attendance\_dates(request):

subject\_id = request.POST.get("subject") session\_year = request.POST.get("session\_year\_id") # Students enroll to Course, Course has Subjects

# Getting all data from subject model based on subject\_id subject\_model = Subjects.objects.get(id=subject\_id) session\_model = SessionYearModel.objects.get(id=session\_year) attendance = Attendance.objects.filter(subject\_id=subject\_model,

session\_year\_id=session\_model)

# Only Passing Student Id and Student Name Only list\_data = []

for attendance\_single in attendance: data\_small={"id":attendance\_single.id,

"attendance\_date":str(attendance\_single.attendance\_date), "session\_year\_id":attendance\_single.session\_year\_id.id}

list\_data.append(data\_small)

return JsonResponse(json.dumps(list\_data), content\_type="application/json", safe=False)

@csrf\_exempt

def admin\_get\_attendance\_student(request):

# Getting Values from Ajax POST 'Fetch Student' attendance\_date = request.POST.get('attendance\_date') attendance = Attendance.objects.get(id=attendance\_date)

attendance\_data = AttendanceReport.objects.filter(attendance\_id=attendance) # Only Passing Student Id and Student Name Only

list\_data = []

for student in attendance\_data: data\_small={"id":student.student\_id.admin.id,

"name":student.student\_id.admin.first\_name+" "+student.student\_id.admin.last\_name, "status":student.status}

list\_data.append(data\_small)

return JsonResponse(json.dumps(list\_data), content\_type="application/json", safe=False)

def admin\_profile(request):

user = CustomUser.objects.get(id=request.user.id) context={

"user": user

}

return render(request, 'hod\_template/admin\_profile.html', context)

def admin\_profile\_update(request):

if request.method != "POST":

messages.error(request, "Invalid Method!") return redirect('admin\_profile')

else:

first\_name = request.POST.get('first\_name') last\_name = request.POST.get('last\_name') password = request.POST.get('password') try:

customuser = CustomUser.objects.get(id=request.user.id) customuser.first\_name = first\_name customuser.last\_name = last\_name

if password != None and password != "": customuser.set\_password(password) customuser.save()

messages.success(request, "Profile Updated Successfully") return redirect('admin\_profile')

except:

messages.error(request, "Failed to Update Profile") return redirect('admin\_profile')

def staff\_profile(request):

pass

def student\_profile(requtest):

pass

Step 12:

* + Now add models.py to our project. It stores all the models that will be used in our project.

from django.contrib.auth.models import AbstractUser from django.db import models

from django.db.models.signals import post\_save from django.dispatch import receiver

class SessionYearModel(models.Model): id = models.AutoField(primary\_key=True) session\_start\_year = models.DateField() session\_end\_year = models.DateField()

objects = models.Manager()

# Overriding the Default Django Auth

# User and adding One More Field (user\_type) class CustomUser(AbstractUser):

HOD = '1'

STAFF = '2'

STUDENT = '3'

EMAIL\_TO\_USER\_TYPE\_MAP = { 'hod': HOD,

'staff': STAFF, 'student': STUDENT

}

user\_type\_data = ((HOD, "HOD"), (STAFF, "Staff"), (STUDENT, "Student")) user\_type = models.CharField(default=1, choices=user\_type\_data, max\_length=10)

class AdminHOD(models.Model):

id = models.AutoField(primary\_key=True)

admin = models.OneToOneField(CustomUser, on\_delete = models.CASCADE) created\_at = models.DateTimeField(auto\_now\_add=True)

updated\_at = models.DateTimeField(auto\_now=True) objects = models.Manager()

class Staffs(models.Model):

id = models.AutoField(primary\_key=True)

admin = models.OneToOneField(CustomUser, on\_delete = models.CASCADE) address = models.TextField()

created\_at = models.DateTimeField(auto\_now\_add=True) updated\_at = models.DateTimeField(auto\_now=True) objects = models.Manager()

class Courses(models.Model):

id = models.AutoField(primary\_key=True) course\_name = models.CharField(max\_length=255)

created\_at = models.DateTimeField(auto\_now\_add=True) updated\_at = models.DateTimeField(auto\_now=True) objects = models.Manager()

class Subjects(models.Model):

id =models.AutoField(primary\_key=True) subject\_name = models.CharField(max\_length=255)

# need to give default course

course\_id = models.ForeignKey(Courses, on\_delete=models.CASCADE, default=1) staff\_id = models.ForeignKey(CustomUser, on\_delete=models.CASCADE) created\_at = models.DateTimeField(auto\_now\_add=True)

updated\_at = models.DateTimeField(auto\_now=True) objects = models.Manager()

class Students(models.Model):

id = models.AutoField(primary\_key=True)

admin = models.OneToOneField(CustomUser, on\_delete = models.CASCADE) gender = models.CharField(max\_length=50)

profile\_pic = models.FileField() address = models.TextField()

course\_id = models.ForeignKey(Courses, on\_delete=models.DO\_NOTHING, default=1) session\_year\_id = models.ForeignKey(SessionYearModel, null=True,

on\_delete=models.CASCADE)

created\_at = models.DateTimeField(auto\_now\_add=True) updated\_at = models.DateTimeField(auto\_now=True) objects = models.Manager()

class Attendance(models.Model):

# Subject Attendance

id = models.AutoField(primary\_key=True)

subject\_id = models.ForeignKey(Subjects, on\_delete=models.DO\_NOTHING) attendance\_date = models.DateField()

session\_year\_id = models.ForeignKey(SessionYearModel, on\_delete=models.CASCADE) created\_at = models.DateTimeField(auto\_now\_add=True)

updated\_at = models.DateTimeField(auto\_now=True) objects = models.Manager()

class AttendanceReport(models.Model):

# Individual Student Attendance

id = models.AutoField(primary\_key=True)

student\_id = models.ForeignKey(Students, on\_delete=models.DO\_NOTHING) attendance\_id = models.ForeignKey(Attendance, on\_delete=models.CASCADE) status = models.BooleanField(default=False)

created\_at = models.DateTimeField(auto\_now\_add=True) updated\_at = models.DateTimeField(auto\_now=True) objects = models.Manager()

class LeaveReportStudent(models.Model):

id = models.AutoField(primary\_key=True)

student\_id = models.ForeignKey(Students, on\_delete=models.CASCADE) leave\_date = models.CharField(max\_length=255)

leave\_message = models.TextField() leave\_status = models.IntegerField(default=0)

created\_at = models.DateTimeField(auto\_now\_add=True) updated\_at = models.DateTimeField(auto\_now=True) objects = models.Manager()

class LeaveReportStaff(models.Model):

id = models.AutoField(primary\_key=True)

staff\_id = models.ForeignKey(Staffs, on\_delete=models.CASCADE) leave\_date = models.CharField(max\_length=255)

leave\_message = models.TextField() leave\_status = models.IntegerField(default=0)

created\_at = models.DateTimeField(auto\_now\_add=True) updated\_at = models.DateTimeField(auto\_now=True) objects = models.Manager()

class FeedBackStudent(models.Model):

id = models.AutoField(primary\_key=True)

student\_id = models.ForeignKey(Students, on\_delete=models.CASCADE) feedback = models.TextField()

feedback\_reply = models.TextField()

created\_at = models.DateTimeField(auto\_now\_add=True) updated\_at = models.DateTimeField(auto\_now=True) objects = models.Manager()

class FeedBackStaffs(models.Model):

id = models.AutoField(primary\_key=True)

staff\_id = models.ForeignKey(Staffs, on\_delete=models.CASCADE) feedback = models.TextField()

feedback\_reply = models.TextField()

created\_at = models.DateTimeField(auto\_now\_add=True) updated\_at = models.DateTimeField(auto\_now=True) objects = models.Manager()

class NotificationStudent(models.Model):

id = models.AutoField(primary\_key=True)

student\_id = models.ForeignKey(Students, on\_delete=models.CASCADE) message = models.TextField()

created\_at = models.DateTimeField(auto\_now\_add=True) updated\_at = models.DateTimeField(auto\_now=True) objects = models.Manager()

class NotificationStaffs(models.Model):

id = models.AutoField(primary\_key=True)

stafff\_id = models.ForeignKey(Staffs, on\_delete=models.CASCADE) message = models.TextField()

created\_at = models.DateTimeField(auto\_now\_add=True) updated\_at = models.DateTimeField(auto\_now=True) objects = models.Manager()

class StudentResult(models.Model):

id = models.AutoField(primary\_key=True)

student\_id = models.ForeignKey(Students, on\_delete=models.CASCADE) subject\_id = models.ForeignKey(Subjects, on\_delete=models.CASCADE, default=1) subject\_exam\_marks = models.FloatField(default=0)

subject\_assignment\_marks = models.FloatField(default=0) created\_at = models.DateTimeField(auto\_now\_add=True) updated\_at = models.DateTimeField(auto\_now=True) objects = models.Manager()

#Creating Django Signals @receiver(post\_save, sender=CustomUser) # Now Creating a Function which will

# automatically insert data in HOD, Staff or Student

def create\_user\_profile(sender, instance, created, \*\*kwargs): # if Created is true (Means Data Inserted)

if created:

# Check the user\_type and insert the data in respective tables if instance.user\_type == 1: AdminHOD.objects.create(admin=instance)

if instance.user\_type == 2: Staffs.objects.create(admin=instance) if instance.user\_type == 3:

Students.objects.create(admin=instance, course\_id=Courses.objects.get(id=1), session\_year\_id=SessionYearModel.objects.get(id=1), address="",

profile\_pic="", gender="")

@receiver(post\_save, sender=CustomUser)

def save\_user\_profile(sender, instance, \*\*kwargs):

if instance.user\_type == 1:

instance.adminhod.save() if instance.user\_type == 2:

instance.staffs.save()

if instance.user\_type == 3:

instance.students.save()

Step 13:

* + Go to student\_management\_project -> setting and add AUTH\_USER\_MODEL = ‘student\_management\_app.CustomUser’.

Step 14:

* + Now create forms.py

from django import forms

from .models import Courses, SessionYearModel

class DateInput(forms.DateInput):

input\_type = "date"

class AddStudentForm(forms.Form):

email = forms.EmailField(label="Email", max\_length=50,

widget=forms.EmailInput(attrs={"class":"form-control"})) password = forms.CharField(label="Password",

max\_length=50, widget=forms.PasswordInput(attrs={"class":"form-control"}))

first\_name = forms.CharField(label="First Name",

max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

last\_name = forms.CharField(label="Last Name", max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

username = forms.CharField(label="Username", max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

address = forms.CharField(label="Address", max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

#For Displaying Courses try:

courses = Courses.objects.all() course\_list = []

for course in courses:

single\_course = (course.id, course.course\_name) course\_list.append(single\_course)

except:

print("here") course\_list = []

#For Displaying Session Years try:

session\_years = SessionYearModel.objects.all() session\_year\_list = []

for session\_year in session\_years:

single\_session\_year = (session\_year.id, str(session\_year.session\_start\_year)+" to "+str(session\_year.session\_end\_year))

session\_year\_list.append(single\_session\_year)

except:

session\_year\_list = []

gender\_list = ( ('Male','Male'),

('Female','Female')

)

course\_id = forms.ChoiceField(label="Course", choices=course\_list, widget=forms.Select(attrs={"class":"form-control"}))

gender = forms.ChoiceField(label="Gender", choices=gender\_list, widget=forms.Select(attrs={"class":"form-control"}))

session\_year\_id = forms.ChoiceField(label="Session Year", choices=session\_year\_list, widget=forms.Select(attrs={"class":"form-control"}))

profile\_pic = forms.FileField(label="Profile Pic", required=False, widget=forms.FileInput(attrs={"class":"form-control"}))

class EditStudentForm(forms.Form):

email = forms.EmailField(label="Email", max\_length=50,

widget=forms.EmailInput(attrs={"class":"form-control"})) first\_name = forms.CharField(label="First Name",

max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

last\_name = forms.CharField(label="Last Name", max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

username = forms.CharField(label="Username", max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

address = forms.CharField(label="Address", max\_length=50, widget=forms.TextInput(attrs={"class":"form-control"}))

# For Displaying Courses try:

courses = Courses.objects.all() course\_list = []

for course in courses:

single\_course = (course.id, course.course\_name) course\_list.append(single\_course)

except:

course\_list = []

# For Displaying Session Years try:

session\_years = SessionYearModel.objects.all() session\_year\_list = []

for session\_year in session\_years:

single\_session\_year = (session\_year.id, str(session\_year.session\_start\_year)+" to "+str(session\_year.session\_end\_year))

session\_year\_list.append(single\_session\_year)

except:

session\_year\_list = []

gender\_list = ( ('Male','Male'),

('Female','Female')

)

course\_id = forms.ChoiceField(label="Course", choices=course\_list, widget=forms.Select(attrs={"class":"form-control"}))

gender = forms.ChoiceField(label="Gender", choices=gender\_list, widget=forms.Select(attrs={"class":"form-control"}))

session\_year\_id = forms.ChoiceField(label="Session Year",

choices=session\_year\_list, widget=forms.Select(attrs={"class":"form-control"}))

profile\_pic = forms.FileField(label="Profile Pic", required=False, widget=forms.FileInput(attrs={"class":"form-control"}))

Step 15:

* + Now register the models in admin.py

from django.contrib import admin

from django.contrib.auth.admin import UserAdmin

from .models import CustomUser, AdminHOD, Staffs, Courses, Subjects, Students, Attendance, AttendanceReport, LeaveReportStudent, LeaveReportStaff, FeedBackStudent,

FeedBackStaffs, NotificationStudent, NotificationStaffs # Register your models here.

class UserModel(UserAdmin):

pass

admin.site.register(CustomUser, UserModel) admin.site.register(AdminHOD) admin.site.register(Staffs) admin.site.register(Courses) admin.site.register(Subjects) admin.site.register(Students) admin.site.register(Attendance) admin.site.register(AttendanceReport) admin.site.register(LeaveReportStudent) admin.site.register(LeaveReportStaff) admin.site.register(FeedBackStudent) admin.site.register(FeedBackStaffs) admin.site.register(NotificationStudent) admin.site.register(NotificationStaffs)

Step 16:

* + Now Create a new folder as templates which includes Student\_template, Hod\_template, Staff\_template folders. It contains the different templates used in each interface.
  + Create another folder as static which also includes some files.

Step 17:

* + Add media, static URLs, and root path. import os

MEDIA\_URL="/media/" MEDIA\_ROOT=os.path.join(BASE\_DIR,"media")

STATIC\_URL="/static/" STATIC\_ROOT=os.path.join(BASE\_DIR,"static")

Step 18:

* + Now create a base.html page.

{% load static %}

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<title>College Management System | Dashboard</title>

<!-- Tell the browser to be responsive to screen width -->

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta name="viewport" content="width=device-width, initial-scale=1">

<!-- Font Awesome -->

<link rel="stylesheet" href="{% static "fontawesome-free/css/all.min.css" %}">

<!-- Ionicons -->

<!-- Bootstrap CSS -->

<link href="[https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-](https://cdn.jsdelivr.net/npm/bootstrap%405.0.0-) beta3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384- eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6" crossorigin="anonymous">

<link rel="stylesheet" href="https://code.ionicframework.com/ionicons/2.0.1/css/ionicons.min.css">

<!-- Tempusdominus Bbootstrap 4 -->

<link rel="stylesheet" href="{% static 'tempusdominus-bootstrap-4/css/tempusdominus- bootstrap-4.min.css' %}">

<!-- iCheck -->

<link rel="stylesheet" href="{% static "icheck-bootstrap/icheck-bootstrap.min.css" %}">

<!-- JQVMap -->

<link rel="stylesheet" href="{% static "jqvmap/jqvmap.min.css" %}">

<!-- Theme style -->

<link rel="stylesheet" href="{% static 'dist/css/adminlte.min.css' %}">

<!-- overlayScrollbars -->

<link rel="stylesheet" href="{% static "overlayScrollbars/css/OverlayScrollbars.min.css" %}">

<!-- Daterange picker -->

<link rel="stylesheet" href="{% static "daterangepicker/daterangepicker.css" %}">

<!-- summernote -->

<link rel="stylesheet" href="{% static "summernote/summernote-bs4.css" %}">

<!-- Google Font: Source Sans Pro -->

<link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro:300,400,400i,700" rel="stylesheet">

</head>

{% block content %}

{% endblock content %}

<!-- jQuery -->

<!-- Optional JavaScript -->

<!-- jQuery first, then Popper.js, then Bootstrap JS -->

<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384- J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n"

crossorigin="anonymous"></script>

<script src="[https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js](https://cdn.jsdelivr.net/npm/popper.js%401.16.0/dist/umd/popper.min.js)" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>

<script src="{% static "jquery/jquery.min.js" %}"></script>

<!-- jQuery UI 1.11.4 -->

<script src="{% static "jquery-ui/jquery-ui.min.js" %}"></script>

<!-- Resolve conflict in jQuery UI tooltip with Bootstrap tooltip -->

<script>

$.widget.bridge('uibutton', $.ui.button)

</script>

<!-- Bootstrap 4 -->

<script src="{% static "bootstrap/js/bootstrap.bundle.min.js" %}"></script>

<!-- ChartJS -->

<script src="{% static "chart.js/Chart.min.js" %}"></script>

<!-- Sparkline -->

<script src="{% static "sparklines/sparkline.js" %}"></script>

<!-- JQVMap -->

<script src="{% static "jqvmap/jquery.vmap.min.js" %}"></script>

<script src="{% static "jqvmap/maps/jquery.vmap.usa.js" %}"></script>

<!-- jQuery Knob Chart -->

<script src="{% static "jquery-knob/jquery.knob.min.js" %}"></script>

<!-- daterangepicker -->

<script src="{% static "moment/moment.min.js" %}"></script>

<script src="{% static "daterangepicker/daterangepicker.js" %}"></script>

<!-- Tempusdominus Bootstrap 4 -->

<script src="{% static "tempusdominus-bootstrap-4/js/tempusdominus-bootstrap-4.min.js"

%}"></script>

<!-- Summernote -->

<script src="{% static "summernote/summernote-bs4.min.js" %}"></script>

<!-- overlayScrollbars -->

<script src="{% static "overlayScrollbars/js/jquery.overlayScrollbars.min.js" %}"></script>

<!-- AdminLTE App -->

<script src="{% static 'dist/js/adminlte.js' %}"></script>

<!-- AdminLTE dashboard demo (This is only for demo purposes) -->

<script src="{% static 'dist/js/pages/dashboard.js' %}"></script>

<!-- AdminLTE for demo purposes -->

<script src="{% static 'dist/js/demo.js' %}"></script>

</body>

</html>

Step 19:

* + Now create a home.html page of our project in the student\_management\_app\templates

folder.

{% extends 'base.html' %}

{% load static %}

{% block title %}Home{% endblock title %}

{% block content %}

<html>

<head>

<style> img {

background-size: cover;

}

body {background-color: coral;}

</style>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0- alpha.6/css/bootstrap.min.css" integrity="sha384- rwoIResjU2yc3z8GV/NPeZWAv56rSmLldC3R/AZzGRnGxQQKnKkoFVhFQhNUwEyJ" crossorigin="anonymous">

<script src="https://code.jquery.com/jquery-3.1.1.slim.min.js" integrity="sha384- A7FZj7v+d/sdmMqp/nOQwliLvUsJfDHW+k9Omg/a/EheAdgtzNs3hpfag6Ed950n" crossorigin="anonymous"></script>

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.6/js/bootstrap.min.js" integrity="sha384-vBWWzlZJ8ea9aCX4pEW3rVHjgjt7zpkNpZk+02D9phzyeVkE+jo0ieGizqPLForn" crossorigin="anonymous"></script>

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384- Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">

<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script>

<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384- KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>

</head>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href=""><h3>WELCOME TO CMS</h3></a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data- target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav mr-auto">

</ul>

<form class="form-inline my-2 my-lg-0">

<!--<input class="form-control mr-sm-2" type="search" placeholder="Search" aria- label="Search">-->

<a href="/logi" class="btn btn-outline-success my-1 mx-2">Login</a>

<!--<input class="form-control mr-sm-2" type="Register" placeholder="Register" aria- label="Register">-->

<a href="/registration" class="btn btn-outline-success my-1 mx-2">Register</a>

<a href="/contact" class="btn btn-outline-danger my-1 mx-2">Contact Us</a>

</form>

</div>

</nav>

<div id="carouselExampleIndicators" class="carousel slide" data-ride="carousel">

<ol class="carousel-indicators">

</ol>

<div class="carousel-inner">

<div class="carousel-item active">

{% comment %} <img class="d-block w-100" src="https://images.unsplash.com/20/cambridge.JPG?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfH x8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=1030&q=80" alt="First slide"> {% endcomment %}

<img src="{% static 'dist/img/111.png' %}" class="d-block w-100 h-100 size-cover" alt="...">

</div>

<div class="carousel-item">

{% comment %} <img class="d-block w-100" src="https://images.unsplash.com/photo- 1541339907198-

e08756dedf3f?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb- 1.2.1&auto=format&fit=crop&w=1050&q=80" alt="Second slide"> {% endcomment %}

<img src="{% static 'dist/img/re.png' %}" class="d-block w-100 h-100 size-cover " alt="...">

</div>

<div class="carousel-item">

{% comment %} <img class="d-block w-100" src="https://images.unsplash.com/photo- 1541339907198-

e08756dedf3f?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb- 1.2.1&auto=format&fit=crop&w=1050&q=80" alt="Second slide"> {% endcomment %}

<img src="{% static 'dist/img/e.png' %}" class="d-block w-100 h-100 size-cover " alt="...">

</div>

<div class="carousel-item">

{% comment %} <img class="d-block w-100" src="https://images.unsplash.com/photo- 1541339907198-

e08756dedf3f?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb- 1.2.1&auto=format&fit=crop&w=1050&q=80" alt="Second slide"> {% endcomment %}

<img src="{% static 'dist/img/22.png' %}" class="d-block w-100 h-100 size-cover " alt="...">

</div>

<div class="carousel-item">

{% comment %} <img class="d-block w-100" src="https://images.unsplash.com/photo- 1503676260728-

1c00da094a0b?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb- 1.2.1&auto=format&fit=crop&w=1009&q=80" alt="Third slide"> {% endcomment %}

<img src="{% static 'dist/img/33.png' %}" class="d-block w-100 h-100 size-cover" alt="...">

</div>

</div>

<a class="carousel-control-prev" href="#carouselExampleIndicators" role="button" data- slide="prev">

<span class="carousel-control-prev-icon" aria-hidden="true"></span>

<span class="sr-only">Previous</span>

</a>

<a class="carousel-control-next" href="#carouselExampleIndicators" role="button" data- slide="next">

<span class="carousel-control-next-icon" aria-hidden="true"></span>

<span class="sr-only">Next</span>

</a>

</div>

</html>

{% endblock content %} Output :

Step 20:

* + Now create a registration.html page where students, staff, HOD can register themselves.

{% extends 'base.html' %}

{% load static %}

{% block content %}

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Untitled</title>

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter- bootstrap/4.1.3/css/bootstrap.min.css">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/ionicons/2.0.1/css/ionicons.min.css">

<link rel="stylesheet" href="assets/css/style.css">

<style> body{

height:1000px; background:#475d62; background-color: cover; font-family: sans-seriff;

}

.login-dark {

max-width:320px; width:90%;

background-color: #1e2833; padding:40px;

border-radius:4px;

transform: translate(-50%,-50%); position: absolute;

top: 50%;

left: 50%; color:#fff;

box-shadow:3px 3px 4px rgba(0,0,0,0.2);

}

.login-dark form { max-width:320px; width:90%;

background-color:#1e2833; padding:40px;

border-radius:4px; transform:translate(-50%, -50%);

position:absolute; top:50%; left:50%; color:#fff;

box-shadow:3px 3px 4px rgba(0,0,0,0.2);

}

.login-dark .illustration { text-align:center; padding:15px 0 20px; font-size:100px; color:#2980ef;

}

.login-dark form .form-control { background:none; border:none;

border-bottom:1px solid #434a52; border-radius:0;

box-shadow:none; outline:none; color:inherit;

}

.login-dark form .btn-primary { background:#214a80; border:none;

border-radius:4px; padding:11px;

box-shadow:none; margin-top:26px; text-shadow:none; outline:none;

}

.login-dark form .btn-primary:hover, .login-dark form .btn-primary:active { background:#214a80;

outline:none;

}

.login-dark form .forgot { display:block;

text-align:center; font-size:12px; color:#6f7a85; opacity:0.9;

text-decoration:none;

}

.login-dark form .forgot:hover, .login-dark form .forgot:active { opacity:1;

text-decoration:none;

}

.login-dark form .btn-primary:active { transform:translateY(1px);

}

</style>

</head>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="/ ">

<h4>BACK TO HOME</h4>

</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data- target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav mr-auto">

</ul>

<form class="form-inline my-2 my-lg-0">

<!-- <input class="form-control mr-sm-2" type="login" placeholder="login" aria- label="login">-->

<!--<input class="form-control mr-sm-2" type="Register" placeholder="Register" aria- label="Register">-->

<a href="/logi" class="btn btn-outline-success my-1 mx-2">Login Here</a>

<a href="/contact" class="btn btn-outline-danger my-1 mx-2">Contact Us</a>

<!--<input class="form-control mr-sm-2" type="register" placeholder="register" aria- label="register">-->

</form>

</div>

</nav>

<div class="login-dark form-inline py-0 mx-4 my-4 pl-4 pr-4">

<form action="{% url 'doRegistration' %}" method="get">

{% csrf\_token %}

<h1 class="text-center">Signup</h1>

<div class="illustration"><i class="icon ion-ios-locked-outline"></i></div>

<div class="form-group"><input class="form-control mb-2" type="text" name="first\_name" placeholder="First Name"></div>

<div class="form-group"><input class="form-control mb-2" type="text" name="last\_name" placeholder="Last Name"></div>

<div class="form-group"><input class="form-control mb-2" type="email" name="email" placeholder="Email"></div>

<div class="form-group"><input class="form-control mb-2" type="password" name="password" placeholder="Password"></div>

<div class="form-group"><input class="form-control mb-2" type="password" name="confirmPassword" placeholder="Confirm Password"></div>

<div class="form-group"><button class="btn btn-primary btn-block mt-2 ml-2" type="submit">Register</button></div>

{% comment %} Display Messages {% endcomment %}

{% if messages %}

<div class="col-12">

{% for message in messages %}

{% if message.tags == "error" %}

<div class="alert alert-danger alert-dismissible fade show" role="alert" style="margin-top:

10px;">

<b>{{ message }}</b>

<button type="button" class="btn-close" data-bs-dismiss="alert" aria-

label="Close"></button>

</div>

{% endif %}

{% endfor %}

</div>

{% endif %}

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/twitter- bootstrap/4.1.3/js/bootstrap.bundle.min.js"></script>

{% endblock content %}

Output:

Step 21:

* + Now create a login\_page.html where students, staff, HOD can log in themselves.

{% extends 'base.html' %}

{% load static %}

{% block content %}

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1; maximum-scale=1.0; user-scalable=0;">

<title>Untitled</title>

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter- bootstrap/4.1.3/css/bootstrap.min.css">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/ionicons/2.0.1/css/ionicons.min.css">

<link rel="stylesheet" href="assets/css/style.css">

<style> body{

height:1000px; background:#475d62; background-color: cover; font-family: sans-seriff;

}

.login-dark {

max-width:320px; width:90%;

background-color: #1e2833; padding:40px;

border-radius:4px;

transform: translate(-50%,-50%); position: absolute;

top: 50%;

left: 50%; color:#fff;

box-shadow:3px 3px 4px rgba(0,0,0,0.2);

}

.login-dark form {

max-width:320px; width:90%;

background-color:#1e2833; padding:40px;

border-radius:4px; transform:translate(-50%, -50%); position:absolute;

top:50%; left:50%; color:#fff;

box-shadow:3px 3px 4px rgba(0,0,0,0.2);

}

.login-dark .illustration { text-align:center; padding:15px 0 20px; font-size:100px; color:#2980ef;

}

.login-dark form .form-control { background:none; border:none;

border-bottom:1px solid #434a52; border-radius:0;

box-shadow:none; outline:none; color:inherit;

}

.login-dark form .btn-primary { background:#214a80; border:none;

border-radius:4px; padding:11px;

box-shadow:none; margin-top:26px; text-shadow:none; outline:none;

}

.login-dark form .btn-primary:hover, .login-dark form .btn-primary:active { background:#214a80;

outline:none;

}

.login-dark form .forgot { display:block;

text-align:center; font-size:12px; color:#6f7a85; opacity:0.9;

text-decoration:none;

}

.login-dark form .forgot:hover, .login-dark form .forgot:active { opacity:1;

text-decoration:none;

}

.login-dark form .btn-primary:active { transform:translateY(1px);

}

</style>

</head>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a class="navbar-brand" href="/ ">BACK TO HOME</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data- target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav mr-auto">

</ul>

<form class="form-inline my-2 my-lg-0">

<a href="{% url 'contact' %}" class="btn btn-outline-danger my-1 mx-2">Contact Us</a>

</form>

<!-- <form class="form-inline my-2 my-lg-0">-->

</div>

</nav>

<div class="login-dark form-inline py-0 mx-4 my-4 pl-4 pr-4">

<form action="{% url 'doLogin' %}" method="get">

{% csrf\_token %}

<h1 class="text-center">Login</h1>

<div class="illustration"><i class="icon ion-ios-locked-outline"></i></div>

<div class="form-group"><input class="form-control mb-2" type="email" name="email" placeholder="Email"></div>

<div class="form-group"><input class="form-control mb-2" type="password" name="password" placeholder="Password"></div>

<div class="form-group"><button class="btn btn-primary btn-block mb-2 ml-2" type="submit">Log In</button></div>

<a href="/registration" class="forgot">Not Registered Yet? Register Now</a>

</form>

</div>

{% comment %} Display Messages {% endcomment %}

{% if messages %}

<div class="col-12">

{% for message in messages %}

{% if message.tags == "error" %}

{% comment %}

<div class="alert alert-danger alert-dismissible fade show" role="alert" style="margin-top:

10px;">

<b>{{ message }}</b>

<button type="button" class="btn-close" data-bs-dismiss="alert" aria-

label="Close"></button>

</div>

{% endcomment %}

<div class="alert alert-danger alert-dismissible fade show" role="alert">

<strong>Invalid Login Credentials!</strong>

<button type="button" class="close" data-dismiss="alert" aria-label="Close">

<span aria-hidden="true">×</span>

</button>

</div>

{% endif %}

{% endfor %}

</div>

{% endif %}

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/twitter- bootstrap/4.1.3/js/bootstrap.bundle.min.js"></script>

{% endblock content %}

Output:

Step 22:

* + Create contact.html

{% extends 'base.html' %}

{% load static %}

{% block content %}

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<a href="/ " class="btn btn-outline-primary my-1 mx-2">Go Back To Home </a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data- target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav mr-auto">

</ul>

<form class="form-inline my-2 my-lg-0">

<!-- <input class="form-control mr-sm-2" type="login" placeholder="login" aria- label="login">-->

<!--<input class="form-control mr-sm-2" type="Register" placeholder="Register" aria- label="Register">-->

<!--<input class="form-control mr-sm-2" type="register" placeholder="register" aria- label="register">-->

</form>

<form class="form-inline my-2 my-lg-0">

</div>

</nav>

<div class="container-fluid px-0">

<img src="{% static 'dist/img/contact.jpg' %}" class="d-block w-100 mx-0" alt="..." height=450px width=10px>

</div>

<div class="container">

<h1 class="text-center my-3 display-2">

<b>Contact Us</b>

</h1>

<form action="/contact" method="post">

{% csrf\_token %}

<div class="mb-3 py-2">

<label for="exampleFormControlInput1" class="form-label"><b>Name</b></label>

<input type="name" class="form-control" id="exampleFormControlInput1" name = "name" placeholder="Enter your Name">

</div>

<div class="mb-3 py-2">

<label for="exampleFormControlInput1" class="form-label"><b>Email id</b></label>

<input type="email" class="form-control" id="exampleFormControlInpu2"name = "email" placeholder="Enter your Email">

</div>

<div class="mb-3 py-2">

<label for="exampleFormControlInput1" class="form-label"><b>Phone number</b></label>

<input type="number" class="form-control" id="exampleFormControlInput3" name = "phone" placeholder="Enter your Phone number">

</div>

<div class="mb-3 py-2">

<label for="exampleFormControlTextarea1" class="form-label"><b>How can we help you

??</b></label>

<textarea class="form-control" id="exampleFormControlTextarea1" rows="7" name = "desc"></textarea>

</div>

<button type="submit" class="btn btn-primary btn-lg ">Submit</button>

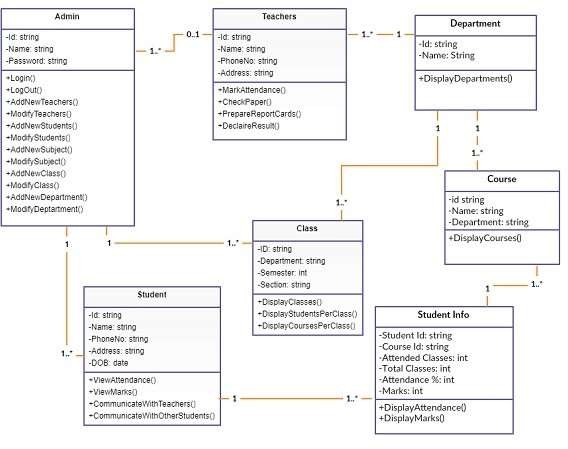
</form>

</div>

{% endblock content%} Step 23:

* + Run these commands to migrate your models into the database. When you successfully do all the steps you will get this type of output in CMD.
  + python manage.py makemigrations python manage.py migrate

### DataBase Table



**Chapter 5 Applications**

Advantages

* College management system completely digitalize the way attendance are managed currently.
* Currently this system is almost managed manually which makes it very tedious to maintain.
* To partially eliminate and minimize the manual work.
* Also to reduce the use of paper.
* To keep track of term details such as faculties, subjects, number of students, etc.
* To keep a record of classroom data.
* Eliminating the inappropriate cases of proxies.
* Since the paper work is very minimal a lot of storage space is saved.
* To make use of latest technologies and updated frameworks and libraries.

Disadvantages

* Will take some time to shift all required informations to this new system.
* Will take some time figure out how to use this new web application.
* Have to spend some time to understand the interface and features of this new system.

Applicability

* Managing college is a very tedious process since it requires lot of manual work and maintenance.
* The main focus of this applciation is to make the whole process of managing college up to 90% digital.
* Also term data will be managed in the system itself.
* Details of faculty teaching various subjects will also be maintained along with the classroom.
* This application is very much important for the colleges and the various other institutes who teaches the self-financed courses such as Bsc IT, BMM etc.
* Many colleges have the rule of copulsory attendance of 75% each month which if failed results in defaulter.
* Every month the defaulter's list is generated which again requires lot of manual work involving mathematical calculations.
* Since there are calcultions involved there is a possibility of calculation error.
* To overcome this hectic process of genereating defaulter's list every month and at the end of term the system will handle the generating of defaulter's list asutomatically.
* This data will be available in excel format.
* At the end of every therm system will automatically generate the term data excel and also the term defaulter's list.
* This system also has a backdoor for the developer to keep maintenance record and quickly fix the system if any bug is found without affecting it.
* Basically the main purpose of this application is to save time, space and to reduce manual work up to 90%.

**Chapter 6**

**Conclusion and future work**

By using Existing System accessing information from files is a difficult task and there is no quick and easy way to keep the records of students and staff. Lack of automation is also there in the Existing System. The aim of Our System is to reduce the workload and to save significant staff time.

Title of the project as College ERP System is the system that deals with the issues related to a particular institution. It is the very useful to the student as well as the faculties to easy access to finding the details. The college ERP provides appropriate in- formation to users based on their profiles and role in the system. This project is designedkeeping in view the day to day problems faced by a college system.

The fundamental problem in maintaining and managing the work by the administra-tor is hence overcome. Prior to this it was a bit difficult for maintaining the time table and also keeping track of the daily schedule. But by developing this web-based applica- tion the administrator can enjoy the task, doing it ease and also by saving the valuable time. The amount of time consumption is reduced and also the manual calculations are omitted, the reports can be obtained regularly and also whenever on demand by the user. The effective utilization of the work, by proper sharing it and by providing the accurate results. The storage facility will ease the job of the operator. Thus the system developedwill be helpful to the administrator by easing his/her task.

This System provide the automate admissions no manual processing is required. Thisis a paperless work. It can be monitored and controlled remotely. It reduces the man power required. It provides accurate information always.. All years together gathered information can be saved and can be accessed at any time. The data which is storedin the repository helps in taking intelligent decisions by the management providing the accurate results. The storage facility will ease the job of the operator. Thus the system developed will be helpful to the administrator by easing his/her task providing the accu-rate results. The storage facility will ease the job of the operator.

This project is successfully implemented with all the features and modules of the college management system as per requirements.

## References

* 1. Elmasri and Navathe: Fundamentals of Database Systems, 7th Edition, PearsonEducation, 2016.
  2. Ian Sommerville: Software Engineering, 10th edition, Person Education Ltd, 2015.
  3. Roger S Pressman: Software Engineering- A Practitioners approach,8th edition,McGraw-Hill Publication, 2015.
  4. https://en.wikipedia.org/wiki/Requirements-engineering
  5. https://web.cs.dal.ca/ hawkey/3130/srs-template-ieee.doc
  6. <http://www.ntu.edu.sg/home/cfcavallaro/Reports/Report%20writing.htmTo> p
  7. https://en.wikipedia.org/wiki/Class diagram
  8. https://[www.djangoproject.com/](http://www.djangoproject.com/)
  9. https://getbootstrap.com/
  10. https://[www.tutorialspoint.com/](http://www.tutorialspoint.com/)
  11. https://creately.com/
  12. https://[www.overleaf.com/project](http://www.overleaf.com/project)