IŞIK UNIVERSITY
COMPUTER
SCIENCE AND
ENGINEERING

INSTITUTE AUTOMATION SYSTEM

Bachelor's Thesis

Oğuzhan Ulusoy
214CS2015

Supervised by
Taner Eskil

June 2018

INSTITUTE AUTOMATION SYSTEM

Abstract

Institute Automation System is a web-based application that provides online campus interface to academic staffs, institute staffs and grad students. The main goal of system is to satisfy easier and faster operations. Institute Automation System is basically online student information and registration system. Institute Automation System consists of three different modules. We have developed institute staff module.

Acknowledgements

To my lovely family,

They have always supported to me during my university education. They have got encouraged and motivated me for I will finish the university in three years. And today, I already finished the study my bachelor's thesis. I appreciate to you. Thank you!

To my supervisor and my close friends,

I am happy and grateful for your support. Thanks very much for all of you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

DEFINITIONS, ACRONYMS AND ABBREVIATIONS

- IAS: Institute Automation System that is going to be developed by us.
- Academic personal: People on system that have lectures.
- Grad students: People who are master and doctorate students.
- Institute staff: People who are secretes, manager and other.
- Existing system: The system that has University of Işık.

CHAPTER ONE
INTRODUCTION

1.1 Purpose of the System

Institute Automation System is a web-based application that provides online campus interface to grad students, academic instructors and institute staffs. The main goal of the mentioned interface is to satisfy to make easier and faster their activities.

1.2 Scope of the System

As mentioned at Purpose of the System section, Institute Automation System is a web- based application that satisfies a school interface to actors in system. The main target of these interfaces is to provide to make easier and faster their activities.

Institute Automation System has four major actors which are grad student, visitor, staff and system administrator fundamentally. Staff is divided into academic staff and institute staff. Academics staff might be an instructor or advisor. System administrator has full authority to manage the whole system. There are three different interfaces for grad students, academic staffs and institute staffs.

Each interface is going to be had many useful functions according to actor's role in institute. We are going to be develop a system that contains main purposes of existing system and also useful different functions.

Grad student is able to register and pre-register on school. Grad student is able to access own CCR, transcript, graduation time, curriculum, weekly schedule, lecture lists, personal information, notifications. In addition to these, they can create weekly schedule as live preview.

Academic staff is able to display own courses, students in her courses. Academic staff is able to give letter grade for students. If she is also advisor, she can accept or reject the weekly schedule of the student and can access to transcripts, weekly schedules, ccrs and personal information of own students and can open special quota.

Institute staff has full authority to manage the all secretarial and instituted tasks. Institute staff is able to display institutes, departments, programs, curriculums, courses, course types and sections with own details individually. In addition to this, institute staff is able to add new institute, department, program, curriculum and course type with own details. Moreover, institute staff is able to define a new course and section for a specific course. Existing courses might be opened, closed or removed by institute staff. Existing sections are removed by institute staff individually or completely. Institute staff is able to display all academic staffs with details, institute heads, program heads, department heads and quota managers. Also, new quota manager is defined by institute staff. Institute staff is able to display all grad students with details including personal information, transcripts, programs, ccrs individually and applications from visitors. These applications might be accepted or rejected by institute staff. Institute staff is able to reach to all completed courses as filtering. Institute staff is able to send mail for everyone in system or an actor group. Institute staff is able to define exam dates and places. Institute staff has a query screen to find information fast.

1.3 Objectives and Success Criteria of the Project

We are going to improve a system that is more exclusive and capable than existing system. Some functions are mentioned at Scope of the System section are included on existing system. Whereas, some functions are mentioned at Scope of the System section are completely new and useful function. It should be more reliable, efficient and lossless data.

The implementation of Institute Automation System should be understandable, clear end efficient. The all needs should be planned to relevant domain. Security is another objective when we develop the system. The all information is kept protected in secure and the system should be guaranteed this.

Success criteria can be listed as:

- Develop the functions of existing system
- Develop more secure, reliable and faster system
- Develop completely new functions to provide automation
- Decrease waiting time in queue and prevent deadlock, or errors

CHAPTER TWO
LITERATURE REVIEW

There is an existing system belonging to University of Işık that is called Campus Online. It is developed by S.Kuru, M.Yildiz and G.Erdogan in 2001. It is developed with ASP.Net. Campus Online provides to registration for courses, display own transcript, CCR, weekly schedule and curriculum in terms of students. Instructor is able to give letter grade for students. Also, instructor is able to view more information about own students and accept or reject weekly schedule of each student. If an instructor is also quota manager, she can open a special quota for specific student and increase or decrease quota quantity. Staff is able to make query to search anything on system. This is summary of Campus Online. There are no big changes in Campus Online from first live to today. Therefore, it is not more interactive and capable system. [1]

Istanbul Aydin University has own online campus interface. We have observed and analyzed it as from student side. It has more interaction among between student and university. Online student system of Istanbul Aydin University is named UBIS. UBIS provides more interactive and functional menu. In terms of student side; registered courses, online courses, weekly schedule, attendance lists, grades, home works and projects, departmental courses, course criteria, transcript, all exam schedule, exam results, notifications about school – department - program and courses, online CV maker, finance information and so on are displayed. We have to opportunity to ask for institute staff side, and we learnt that there is a big query screen and file management system in terms of staff side. [2]

Galatasaray University has own a system for master science applications. We have observed and analyzed it from visitor side. We analyzed how the application page has designed and what information is required. [3]

CHAPTER THREE

PROPOSED SYSTEM [4, 5]

3.1 Functional Requirements

Functional requirements for institute staff are given in below:

1. Institute staff shall be able to login and logout.
2. Institute staff shall be able to display institutes, departments, programs, curriculums, courses, available courses, course types and sections with own details on Institute section. (Essential)
3. Institute staff shall be able to define a new course. (Essential)
4. Institute staff shall be able to open or close an existing source. (Essential)
5. Institute staff shall be able to define a section for a course. (Optional)
6. Institute staff shall be able to display all academic staff, institute heads, department heads, program heads and quota managers with own details on Academic Staff section. (Optional)
7. Institute staff shall be able to define a new academic staff. (Optional)
8. Institute staff shall be able to define a new quota manager for a department. (Optional)
9. Institute staff shall be able to display all institute staffs with own details. (Optional)
10. Institute staff shall be able to define an institute staff. (Optional)
11. Institute staff shall be able to display all grad students, applications with own details on Grad Student section. (Optional)
12. Institute staff shall be able to accept or reject an application. (Essential)
13. Institute staff shall be able to display all completed courses, all curriculums, all CCRs, all transcripts with own details. (Optional)
14. Institute staff shall be able to withdraw for a student. (Essential)
15. Institute staff shall be able to reach to statistics. (Optional)
16. Institute staff shall be able to define exam dates. (Optional)
17. Institute staff shall be able to make notification. (Optional)

Table 3.1 – Functional Requirements

3.2 Nonfunctional Requirements

Usability

Institute Automation System should have three different interfaces for all sub-actors. All functions are divided in to the capabilities of actors. Each division should represent an interface. By the side of Institute Staff, she is able to do in activities in mentioned at Functional Requirement part. By the side of Institute Staff, she has a menu navigator in top of the page. This menu contains sub-actor names. Each menu has own sub menu in body part of the page. The sub menu is rendered by request of menu. Interfaces should be having similar blocks and design. That is important to usability. There should no guide for all actors, because Institute Automation System should be created easier.

Reliability

The Institute Automation System should be provided to access for all actors by secure. By the side of management, in other words automation side, should be lossless data. We are going to develop the system according to Service Oriented Architecture and Object Relation Model. We have no query-based a database. We designed a relational database model, therefore we defined objects with own attributes. We are not going to work class-based programming. Unlike, we are going to work with service-based programming. We are going to use Proxy design pattern to prevent to reach directly to objects.

Performance

Model-view-template design pattern is going to be implemented in this project. This design pattern satisfies relation among these entities bidirectionally. There are five associations among model-view-template entities. The connection -in other words association- should be faster. Moreover, observer and listener design pattern structures are going to be used to satisfy real-time synchronization and orchestration. The system is powerful with Python. Thanks to Django, the system uses less hardware of the server. Therefore, the system will be more performance.

Supportability

The system should be reachable over browser in standard computer and mobile devices. All management of Institute Automation System belongs to the system administrator. All support operations can be handled by system administrator, if it is necessary. Because of the

implementation of system is developed with strategy design pattern, the implementation folder has a layout structure. This makes to easier supportability. Thanks to Django Template Language, update operations is done easily and rapidly. Furthermore, default Django Administration Panel has a basic management structure.

Implementation

Institute Automation System is going to be implemented by using Python language.

All back-end side of project runs with virtual environment (virtualenv). Back-end side of the project is going to be developed with Python. Django framework is going to be used. Django has own model-view-controller structure. When Django framework sets up, it comes with a default project folder that includes settings, urls and init files. When Django app is created, it comes with model, view, admin, tests, apps and init files. All of these files are Python files. Because of there is no complex back-end structure, we are going to work with single app that is called Institute. By the back-end side, Regular Expressions (RegEx) is going to be used for model structure. Also, back-end needs Pillow library of Python.

Django framework uses SQLite3 relational database as default, it can be changed later. We store all data as encrypted for security of the system. Moreover, to prevent to reach to directly to objects in the database, proxy design-pattern is going to be used. It is satisfied by Django.

We are going to use HTML5 and Bootstrap (*to satisfy responsiveness for computers and mobile devices*) to implement front-end side of the project. Settling of files should be set up according to strategy design-pattern. Django Template Language (DTL) is also going to be implemented. It is called jinja2. It has own tags like HTML, those tags are placed in to relevant HTML files to satisfy relation among back-end and front-end. Therefore, observer and listener design-patterns have been satisfied.

We are going to work Object Relational Model (ORM) because of Django is object-oriented framework. We are going to work service-based programming, unlike class-based programming. This is called Service Oriented Architecture (SOA).

We are going to follow Iterative Development Process/Approach throughout whole improvement period. All major activities should be divided into smaller activities, then they

should be divided into tasks. This is called Work Breakdown Structure (WBS). This process makes easier to implementation.

Interface

The interface should be quite simple for all actors, because usability has to be easier. There should no be guide. The interface should be less colorful. The colors of school should be used. Menu navigation should change according to actor. Similar tasks, functions should be put together.

Packaging

When the project finishes, the requirements should be created. To do that, "freeze" command runs and creates the requirements.txt file. When we start to project to implement, we create a virtual environment (*using virtualenv library*). We can make container as the project finishes as by using Docker.

3.3 System Models

System models section has scenarios and use-case scenarios, use case model, object model and dynamic model.

3.3.1 Scenarios and Use-Case Scenarios

This section has both scenario and use case. Use-case is representation association among user and system. A scenario is an instance of a use case.

Scenario Name: Login and Logout
Participating Actor Instance(s): Oğuzhan: Visitor
Flow of Events:

1. Oğuzhan is a visitor who is not defined as an actor in the system. By the side of the project, he is defined as institute staff. He should be logged in to the system to remember as an actor. Therefore, he wants to log in to the system.
2. It is assumed that he has entered to web-address of the project over browser in his computer.
3. He clicks to "Login" linked text label.
4. "Authentication" page is loaded. When it comes, there is a form with own attributes that are e-mail and password fields.

5. Oğuzhan types own e-mail address and password. Then, he clicks to “Submit” button.
6. If there is no error about authentication, he is redirected to inside of the system. Otherwise, he should be typed e-mail and password again.
7. When Oğuzhan desires to log out from the system, he clicks to “Logout” linked text label on menu navigator. Therefore, he could have logged out of the system.

Use-case Name: Login and Logout

Participating Actor(s): Visitor

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier.

Flow of Events:

1. Visitor is a person who is not defined as an actor in the system. By the side of the project, he is defined as institute staff. He should be logged in to the system to remember as an actor. Therefore, he wants to log in to the system. He clicks to “Login” linked text label.
2. The relevant method in back-end receives the request, and prepares a form including get-message with this request. Then, it returns rendering the content that includes request, link and created form. The link is a new link that it is incoming page after clicking “Login” linked text label.
3. When incoming page is loaded, visitor types his own authentication information to the form, then clicks to “Submit” button.
4. Still, the relevant method is running in back-end. It receives the typed fields and cleans data, then checks the records on database. If there is matching typed information with records on database, visitor is redirected to home page as logged successfully. If there is a problem about authentication, he is redirected to again same page with error tag.
5. If institute staff desires to log out of the system, clicks to “Logout” linked text label in menu navigator.
6. Then, system receives the request and transmits it to logout method in Django. Therefore, the person could have logged out of the system.

Entry Condition(s):

- Institute Automation System must be opened over browser.

Exit Condition(s):

- Institute staff can log in to the system.

- ❖ Institute staff can log out of the system.

Exceptional Case(s):

- ❖ The form is sensitive for invalid input.
- ❖ Server might be down.

Scenario Name: Display to Institutes/Departments/Programs/Curriculums/Courses/ Available Courses/Course Types/Sections

Participating Actor Instance(s): Oğuzhan: Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Displaying to institutes, departments, programs, curriculums, courses, available courses, course types and sections have same scenario. So, only "Displaying Courses" is going to introduce.
2. Oğuzhan clicks to "Institute" linked text label on menu navigation. Then, incoming page has own sub-menu. That includes Institutes, Departments, Programs, Curriculums, Courses (*also, it has own sub-categories*), Available Courses, Course Types and Sections respectively. Oğuzhan clicks to "Courses" linked text label.
3. Incoming page has own table. The table contains code, title, program and view as columns. For instance, "CSE501" for code, "Advanced Software Engineering" for title, "Computer Engineering" for program. Oğuzhan clicks to view button of this row.
4. Incoming page has own attributes, these are course number, name, code, description, credit, program, university, is valid, is deleted, created date. Course number is identification number of this data on system. Name is title of course, that is "Advanced Software Engineering". Code is "CSE501". Description includes briefly information about the lecture. Program is "Computer Engineering", it is not department. There is chance to define a course from another university, so university should be defined. "University of Isik" is for university. Is valid is to show its statue (open or close), is deleted is to show its statue (removed or not removed). Created date is default established date.
5. We have mentioned at first item, displaying to institutes, departments, programs, curriculums, courses, available courses, course types and sections have same scenario. They are similar to each other about to show data and view details of them. There is just a difference, that is to add new item. Institutes, departments, programs, curriculums, course types and sections have same scenario about adding new item. Adding course scenario is

going to introduced later in another case. We are going to introduce adding new item in terms of departments.

6. Oğuzhan clicks to “Departments” linked text label from sub-menu of Institute section. Incoming page has own table and there is a button that is named “Add New Department”, below the table. Oğuzhan clicks to “Add New Department” button. Then, a modal is opened.
7. The opened modal has own attributes. In terms of Department, it has name, institute and head. Oğuzhan types “Computer Science” for name, “Natural Science” for institute and “Olcay Taner Yildiz” for head. Then, clicks to “Submit” button. If there is no error about adding new item, it is added to relevant model in database successfully.

Use-case Name: Display to Institutes/Departments/Programs/Curriculums/Courses/ Available Courses/Course Types/Sections

Participating Actor(s): Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Institute Staff is a person who is defined as an actor in the system. By the side of the project, he must be logged in to the system to make process. It is assumed that he has entered to web-address of the project {Scenario and Use-case: Login and Logout}. Institute staff clicks to “Institute” linked text label from menu navigation bar.
2. The request is received, and Institute section is opened after it is rendering. It has own sub-menu and details.
3. Institute staff reaches to sub-menu including institutes, departments, programs, courses, course types and sections. Moreover, courses item has own sub-menu. All of these items have same structure. They have a table and modal. Therefore, dictionary data structure and Django form structure are used in these cases. Institute staff clicks to “Courses” item.
4. The request is received and relevant method in back-end is called, then this request is transmitted to this method. This method gets all data from relevant model, in this case it is Course model. Order changes according to item, such as Courses case gets data according to created date, Departments cases gets data according to alphabetical. Furthermore, these methods create a Django form with request. It is going to be explained with details later. Finally, the method returns rendering content that includes request, url, and all courses dictionary. Therefore, the content can be handled in front-end side successfully.

5. Then, Institute Staff should add to new department item. He clicks to “Department” item from sub-menu of Institute section.
6. The request is received and relevant method in back-end is called, then this request is transmitted to this method. As we have mentioned earlier, this method has two code blocks. First is to get all data from relevant model, second is to create a form to add new item.
7. Incoming page has a table and button to add new item. Institute staff clicks to “Add New Department” button.
8. Thus, a modal is opened by front-end side. The form is created by Django form and it is showed with own attributes on modal. In this process, institute staff types information to relevant fields. If the form is valid, relevant fields are cleaned and saved. If there is an invalid input, institute staff is redirected to invalid page. Otherwise, error is represented on modal.

Entry Condition(s):

- ❖ Institute Automation System must be opened over browser.

Exit Condition(s):

- ❖ Institute staff can display any item on Institute section.
- ❖ Institute staff can add new item for any item on Institute section.

Exceptional Case(s):

- ❖ The form is sensitive for invalid input.
- ❖ Server might be down.

Scenario Name: Define A New Course

Participating Actor Instance(s): Oğuzhan: Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Oğuzhan desires to add new course item. So, he goes to “Define Course” item under Course item under Institute section.
2. Incoming page own table and button. The table is related to courses. It has code, title, program and view as columns. For instance: “CSE501” is for code, “Advanced Software Engineering” is for title, “Computer Engineering” is for program.

3. Button is related to add new item. When it is clicked, modal is opened. Oğuzhan clicks to "Define Course" button.
4. Modal is opened, and it has own attributes. These are course code, course name, course description, course credit, course ETCS credit, program, university and submit button. Oğuzhan types "CSE520" for course code, "Advanced Database Systems" for course name, "This course contains advanced database systems" for course description, "3" for course credit, "6" for course ETCS credit, "Computer Engineering" for program, "Isik University" for university. Then, he clicks to submit button to finish the operation.

Use-case Name: Define A New Course

Participating Actor(s): Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Institute Staff is a person who is defined as an actor in the system. By the side of the project, he must be logged in to the system to make process. It is assumed that he has entered to web-address of the project {Scenario and Use-case: Login and Logout}. Institute staff clicks to "Institute" linked text label from menu navigation bar.
2. The request is received, and Institute section is opened after it is rendering. It has own sub-menu and details.
3. Institute staff reaches to sub-menu including institutes, departments, programs, courses, course types and sections. Moreover, courses item has own sub-menu. Courses item have "define course", "open/close course", "available courses", "remove course" items. All of these items have same structure. Institute staff should define a new course, so clicks to "Define Course" item under Course item.
4. The request is received and relevant method in back-end is called, then this request is transmitted to this method. This method creates a Django from that runs with post method. It returns rendering content that includes request, url, dictionary including form.
5. Institute staff types information to relevant fields on modal screen, then clicks to "Submit" button to finish the operation.
6. Still, the relevant method is running in back-end side of the project. If the form is valid, it receives typed information, cleans them and saves to database. If there is an invalid input, it is redirected invalid page. Otherwise, error tags are displayed on modal screen.

Entry Condition(s):

- ❖ Institute Automation System must be opened over browser.

Exit Condition(s):

- ❖ Institute staff can define new course.

Exceptional Case(s):

- ❖ The form is sensitive for invalid input.
- ❖ Server might be down.

Scenario Name: Open or Close Course

Participating Actor Instance(s): Oğuzhan: Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Oğuzhan desires to open or close an existing course. So, he goes to "Open/Close Course" item under Course item under Institute section.
2. There is a table including code, title, program and view columns. "CSE501" for code, "Advanced Software Engineering" for title, "Computer Engineering" for program. Oğuzhan should open this course. If there is no data, it does not have to define again. Therefore, he clicks to view linked text label.
3. Incoming page has details of selected course. These are course number, name, code, description, credit, program, university, is valid, is deleted, created date. Course number is identification number of this data on system. Name is title of course, that is "Advanced Software Engineering". Code is "CSE501". Description includes briefly information about the lecture. Program is "Computer Engineering", it is not department. There is chance to define a course from another university, so university should be defined. "University of Isik" is for university. Is valid is to show its statue (open or close), is deleted is to show its statue (removed or not removed). Created date is default established date.
4. Above the details content, there is a button is named "Open/Close Course". Oğuzhan clicks to button. Then, a modal is opened. There are three components which are is valid, edited date and submit button. True value is for to open the course, false value is for to close the course. Edited date is necessary. Then, Oğuzhan puts tick to is valid field, that means true and types a date YYYY-MM-DD format. Finally, clicks to submit button to finish the operation.

Use-case Name: Open or Close Course

Participating Actor(s): Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Institute Staff is a person who is defined as an actor in the system. By the side of the project, he must be logged in to the system to make process. It is assumed that he has entered to web-address of the project {Scenario and Use-case: Login and Logout}. Institute staff clicks to "Institute" linked text label from menu navigation bar.
2. The request is received, and Institute section is opened after it is rendering. It has own sub-menu and details.
3. Institute staff should open, or close, an existing course. Therefore, he goes to Open/Close Course item under Courses item Institute section.
4. The request is received, and the relevant method in back-end is called, then this request is transmitted to the method. All data in Courses model is showed in table in here.
5. Institute staff clicks to any item on table.
6. The request and id are received, and the relevant method is called, and these are transmitted to this method. Course details are got according to identification number of selected course. They are displayed on content block.
7. Institute staff clicks to "Open/Close Course" button. A modal is opened.
8. Modal is handled in front-end side. There is a similar form. It is handled mentioned method. When the institute staff does something and clicks to submit button, if the form is valid, data cleans and saves to database. If there is an invalid input, he is redirected to invalid page. Otherwise, if there is an error, it is showed with error tag.

Entry Condition(s):

- ❖ Institute Automation System must be opened over browser.

Exit Condition(s):

- ❖ Institute staff can open or close an existing course.

Exceptional Case(s):

- ❖ The form is sensitive for invalid input.
- ❖ Server might be down.

- ❖ Edited date must be YYYY-MM-DD format.

Scenario Name: Define Section

Participating Actor Instance(s): Oğuzhan: Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Oğuzhan desires to define a section for specific course. So, he goes to "Add Section" item on Section item under Institute section.
2. Incoming page has a table and a button. The table contains own attributes which are course, number, instructor, year/semester and view columns. {Sections part has been introduced in scenario and use case: display to institutes - departments - programs curriculums - courses - available courses - course types and sections.
3. Oğuzhan should define a section for a specific course. Therefore, he clicks to add section button to adding new item.
4. A modal is opened. Incoming page has own components which are course, section number, quota quantity, instructor, semester and submit button.
5. Oğuzhan selects a course from dropdown menu, "Artificial Intelligence" for course Then, he types "1" for section number, "20" for quota quantity. He selects an instructor from dropdown menu, "Taner Eskil" for instructor. He selects a semester from dropdown menu, "2017/Winter" for semester. Then, he clicks to submit button to finish the operation, finally.

Use-case Name: Define Section

Participating Actor(s): Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Institute Staff is a person who is defined as an actor in the system. By the side of the project, he must be logged in to the system to make process. It is assumed that he has entered to web-address of the project {Scenario and Use-case: Login and Logout}. Institute staff clicks to "Institute" linked text label from menu navigation bar.
2. The request is received, and Institute section is opened after it is rendering. It has own sub-menu and details.

3. Institute staff should define a section for a specific course. Therefore, he goes to Section item under Institute section. There is a structure that has been explained in earlier. Institute staff clicks to "Add Section" button.
4. A modal is opened. It is handled by front-end. There is a form which is created by Django. Components in here are course, section number, quota quantity, instructor, semester and submit button.
5. Institute staff types, selects and clicks to submit button to finish the operation.
6. The request is received, and the relevant method in back-end is called, and the request is transmitted to this method. If there is no problem such as invalid input, data is cleaned and saved in to database, if the form is valid.

Entry Condition(s):

- ❖ Institute Automation System must be opened over browser.

Exit Condition(s):

- ❖ Institute staff can define a section for a specific course.

Exceptional Case(s):

- ❖ The form is sensitive for invalid input.
- ❖ Server might be down.

Scenario Name: Display to All Academic Staffs, Institute Heads, Department Heads, Program Heads, Quota Managers

Participating Actor Instance(s): Oğuzhan: Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Oğuzhan desires to display all academic staffs, institute heads, department heads, program heads and quota managers. So, he goes to "Academic Staffs" on menu navigator.
2. Incoming page has own sub-menu under Academic Staffs section. The sub-menu contains all academic staffs, institute heads, department heads, program heads, and quota managers. All of these items have same structure, the structure has a table. Academic staff item has also view column in table to display details of every row.
3. Oğuzhan desires to display department heads, so clicks to "Department Heads" linked text label. Incoming page has department name, head and institute as table. "Computer

Science" for department name, "Olcay Taner Yildiz" for head and "Natural Science" for institute.

Use-case Name: Display to All Academic Staffs, Institute Heads, Department Heads, Program Heads, Quota Managers

Participating Actor(s): Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Institute Staff is a person who is defined as an actor in the system. By the side of the project, he must be logged in to the system to make process. It is assumed that he has entered to web-address of the project {Scenario and Use-case: Login and Logout}. Institute staff clicks to "Academic Staffs" linked text label from menu navigation bar.
2. The request is received, and Academic Staffs section is opened after it is rendering. It has own sub-menu and details. Sub-menu contains all academic staffs, institute heads, department heads, program heads and quota managers items.
3. Institute staff should display all academic staffs or institute heads, department heads, program heads, and quota managers. He clicks to "Department Heads" linked text label.
4. The request is received, the relevant method in back-end is called and the request is transmitted to that method. The method gets all data of Department model, selects some of them. It prepares a dictionary and return rendering content that is including request, url, and the dictionary. Each item on sub-menu under Academic Staff section has own method to handle the self-task.

Entry Condition(s):

- Institute Automation System must be opened over browser.

Exit Condition(s):

- Institute staff can display any item on sub-menu.

Exceptional Case(s):

- Server might be down.

Scenario Name: Add Quota Manager

Participating Actor Instance(s): Oğuzhan: Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Oğuzhan desires to add new quota manager, so clicks to "Quota Managers" linked text label.
2. Incoming page has a table including quota manager and program, and a button.
3. Oğuzhan clicks to "Add New Quota Manager" button.
4. A modal opens and has two components which are dropdown menu from academic staff list and submit button. Oğuzhan selects "Fahrettin Ay" for quota manager, then clicks to submit button to finish the operation.

Use-case Name: Add Quota Manager

Participating Actor(s): Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Institute Staff clicks to "Quota Manager" under Academic Staff section.
2. The request is received and the relevant method in back-end is called and the request is transmitted to the method. The content is displayed on table.
3. Institute Staff clicks to "Add New Quota Manager" button.
4. A modal is opened, still the relevant method is running on back-end side. A form is created by the method.
5. Institute Staff does something and clicks to submit button.
6. The relevant method controls the form, if it is valid, received data is saved to database.

Entry Condition(s):

- Institute Automation System must be opened over browser.

Exit Condition(s):

- Institute staff can add new quota manager.

Exceptional Case(s):

- Server might be down.
- Dropdown menu might be empty.

Scenario Name: Add New Academic Staff

Participating Actor Instance(s): Oğuzhan: Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Oğuzhan desires to add new academic staff, so clicks to "All Academic Staffs" linked text label.
2. Incoming page has two components which are table and button.
3. Oğuzhan clicks to "Add New Academic Staff" button.
4. A modal is opened. The modal has four components which are staff, university, institute and submit button.
5. Oğuzhan selects "Ahmet Feyzi Ateş" is for staff, "MEF University" is for university, "Natural Science" is for institute. Then, Oğuzhan clicks to submit button.

Use-case Name: Add New Academic Staff

Participating Actor(s): Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Institute Staff clicks to "All Academic Staff" item under Academic Staff section.
2. The request is received and the relevant method in back-end is called and the request is transmitted to the method. The content is displayed on table.
3. Institute Staff clicks to "Add New Quota Manager" button.
4. A modal is opened, still the relevant method is running on back-end side. A form is created by the method.
5. Institute Staff does something and clicks to submit button.
6. The relevant method controls the form, if it is valid, received data is saved to database.

Entry Condition(s):

- ❖ Institute Automation System must be opened over browser.

Exit Condition(s):

- ❖ Institute staff can add new quota manager.

Exceptional Case(s):

- ❖ Server might be down.

- ❖ Dropdown menu might be empty.

Scenario Name: Accept or Reject an Application

Participating Actor Instance(s): Oğuzhan: Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Oğuzhan should accept or reject an application, so clicks to "Grad Students" linked text label on menu navigation.
2. Incoming page has own sub-menu that includes all grad students, applications, and so on. Then, Oğuzhan clicks to Applications item.
3. Incoming page has own table with self-attributes which are full name, application date, ALES, YDS, GPA, degree and view columns. Oğuzhan sees some data "Gözde Ninda Şakir" for full name, "15 May 2018" for application date, "80" for ALES, "70" for YDS, "3" for GPA, "University Degree – Bachelor Science" for degree.
4. Oğuzhan goes inside of selected data over view linked text label. Incoming page has content block that includes many information about person who did application.
5. Moreover, there are two buttons to accept or reject. When the accept button is clicked, a modal is opened. It has own components which are student ID, school e-mail address, curriculum, program, advisor, hold state, registration open statue, approval statue and submit button. Oğuzhan enters information about new student, Gözde Ninda Şakir. Oğuzhan types "214CS2015" for student ID, gozde.sakir@isik.edu.tr for school e-mail address, "CSE2016" for curriculum, "Computer Engineering" for program, "Emine Ekin" for advisor, "true" for hold state, "false" for registration open statue and "true" for approval statue. Then, Oğuzhan clicks to submit button to finish the operation.
6. Oğuzhan sees some data "Çağrı Kandaş" for full name, "15 May 2018" for application date, "56" for ALES, "60" for YDS, "2.8" for GPA, "University Degree – Bachelor Science" for degree. Oğuzhan clicks to view button for this data, then goes to inside of selected data.
7. He clicks to "Reject" button. When a modal is loaded, there is a variable is "called is removed". Oğuzhan selects it as "Yes", then clicks to submit button.

Use-case Name: Add New Academic Staff

Participating Actor(s): Institute Staff

Pre-condition: It is assumed that he has entered to web-address of the project over browser in his computer earlier, then he has logged in to the system.

Flow of Events:

1. Institute Staff clicks to "Applications" item under Academic Staff section.
2. The request is received and returns Application page with own components.
3. Institute Staff selects a data to go inside.
4. The request is received, and the relevant method in back-end is called and the request is transmitted to this method. It gets all data about selected item by using primary key, identification number.
5. Institute Staff clicks to "Accept" button. When he clicked to the button, Django form is created by relevant handling method. Thus, "student ID, school e-mail address, curriculum, program, advisor, hold state, registration open statue, approval statue" are displayed. They are checked on relevant method in back-end. If there is no invalid input or null point, that means that if the form is valid, the all data is received, cleaned and saved into database. Otherwise, when he clicks to "Reject" button, again Django form is created, only "is removed" variable is displayed.

Entry Condition(s):

- Institute Automation System must be opened over browser.

Exit Condition(s):

- Institute staff can accept or reject an application.

Exceptional Case(s):

- Server might be down.
- Dropdown menu might be empty.
- Conflict may be occurring.

3.3.2 Use Case Model

Use case model is a model that includes all use cases individually. Use case model represents the association among actors and use cases. It is reachable in attachments file of the project.

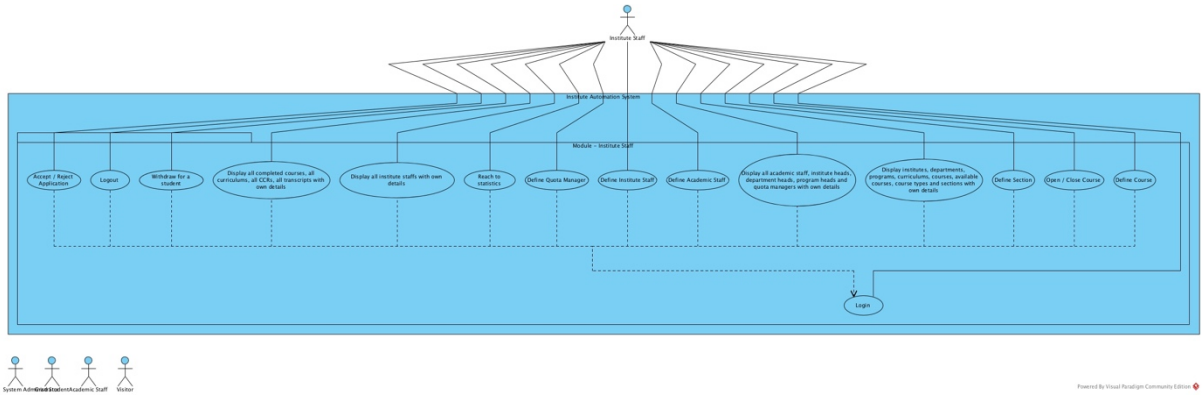

Figure 3.3.2 - Use-Case Model

3.3.3 Object Model

The analysis object model, depicted with UML class diagrams, includes classes, attributes, and operations. The analysis object model is a visual dictionary of the main concepts visible to the user. It is reachable in attachments file of the project.

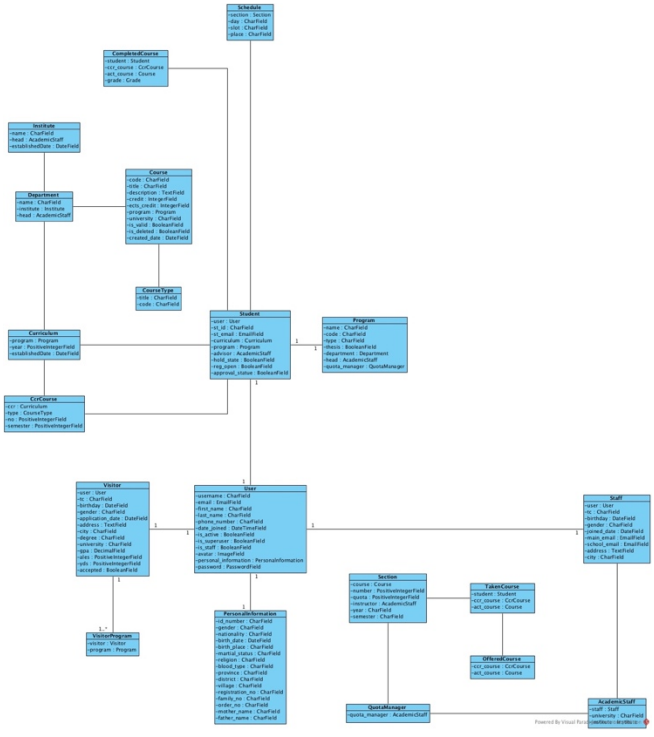

Figure 3.3.3 - Object Model

3.3.4 Dynamic Model

Sequence diagrams represent the interactions among a set of objects during a single use case. The dynamic model serves to assign responsibilities to individual classes and, in the process, to identify new classes, associations, and attributes to be added to the analysis object model. All sequence diagrams are given in attachments folder.

This sequence diagram represents the relationship among institute staff and system about accepting or rejecting an application comes from visitor.

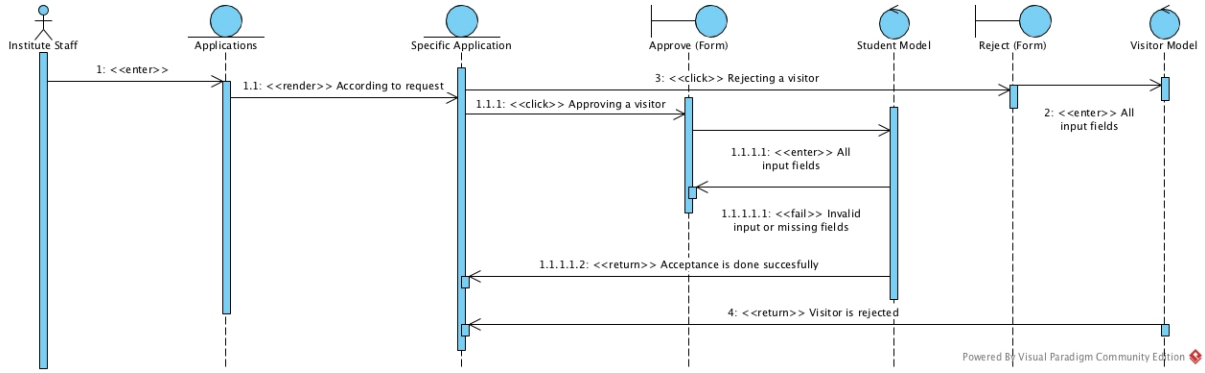

Figure 3.3.4.1 – Sequence diagram: Accept or reject an application of visitor

This sequence diagram represents the relationship among institute staff and system about displaying all entities which are institutes, departments, program, curriculums, courses, course types and sections, and also this diagram includes adding new item scenario.

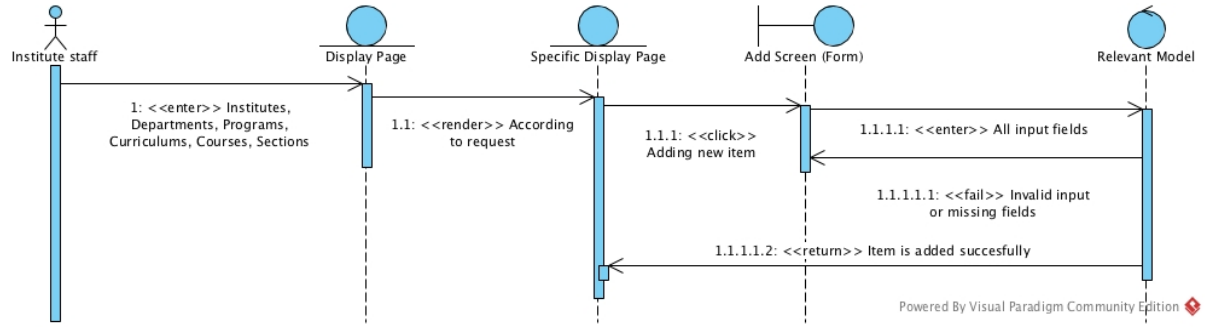

Figure 3.3.4.2 – Sequence diagram: Display all entities and add new item

This sequence diagram represents the relationship among institute staff and system about defining a new course.

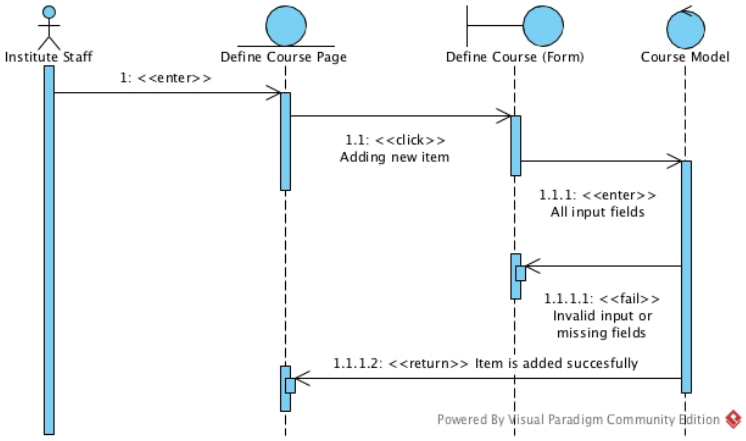

Figure 3.3.4.3 – Sequence diagram: Define a new course

This sequence diagram represents the relationship among institute staff and system about opening or closing an existing course in system.

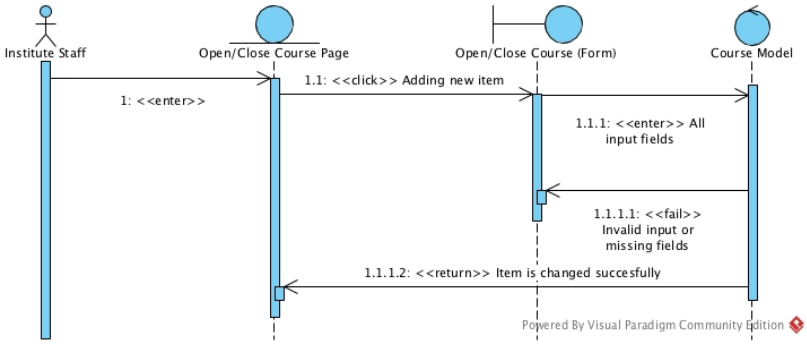

Figure 3.3.4.4 – Sequence diagram: Open or close an existing course

3.4 System Decomposition

Institute Automation System is made of four major actors that are named grad students, staffs, system administrators and visitor. Staffs are divided into academic staffs and institute staffs. Each actor in the system has different authorities. Operations are divided into actor authorities. We have four actors to implement, so we have four different sub-systems. We are going to develop institute staff side.

Institute staff is able to manage institute, academic staffs, institute staffs, grad students and applications. Therefore, institute staff module, in other words sub-system, is divided into five sub-system, another level.

- Institute sub-system includes management of institutes, departments, programs, curriculums, courses, course types and sections.
    - Institutes tab under institute sub-system contains to display all data and add a new institute with own attributes.
    - Departments tab under institute sub-system contains to display all data and add a new department with own attributes.
    - Programs tab under institute sub-system contains to display all data and add a new program with own attributes.
    - Curriculums tab under institute sub-system contains to display all data and add a new curriculum with own attributes.
    - Courses tab under institute sub-system contains to display all data, display available courses, define a new course with own attributes, open or close an existing course and remove an existing course.
    - Course types tab under institute sub-system contains to display all data and add a new course type with own attributes.
    - Sections tab under institute sub-system contains to display all data, add a new section with own attributes and remove all data.
- Academic staff sub-system includes management of all academic staffs, institute heads, department heads, program heads and quota managers.
    - All academic staffs tab under academic staff sub-system contains to display all data and add new academic staff with own attributes.
    - Institute heads tab under academic staff sub-system contains to display all data.
    - Department heads tab under academic staff sub-system contains to display all data.
    - Program heads tab under academic staff sub-system contains to display all data.
    - Quota managers tab under academic staff sub-system contains to display all data and add new quota manager with own attributes.
- Institute staff sub-system includes management of all institute staffs.
    - All institute staffs tab under institute staff sub-system contains to display all data and provide acceptance operations.
- Grad student sub-system include management of all grad students.

    - All grad students tab under grad student sub-system contains to display all data.
    - All completed courses tab under grad student sub-system contains to display all data about completed courses by department, program, courses, students.
- Application sub-system includes management of all applications from visitors.
    - Applications tab under application sub-system contains to display all data and accept, reject or remove operations.
    - Remove all applications tab under application sub-system contains to remove all applications from system forever.

3.5 Hardware – Software Mapping

We are able to examine hardware-software mapping by two sides that are actors and system.

- In terms of actors, there is no any constraint on hardware-software mapping. Institute Automation System is a web-based application that developed by Python infrastructure. All different type actor is able to reach to the system with ordinary computer, or mobile devices with internet-connection.
- In terms of system, we are going to demonstrate the system with Heroku Cloud technology. Full version of Institute Automation System might able to demonstrate with Digital Ocean, Google or Amazon Cloud services, and so on.

The important point in live version is to run the system with virtual environment and Python infrastructure.

3.6 Persistent Data Management

We are going to work with SQLite3 database. It comes as default database system in Python. SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects. [3]

We are going to work with object relational model, in shortly ORM. Object relational model is to more power, greater flexibility, better performance and greater data integrity then those

that came before it. Some of the benefits that are offered by the Object-Relational Model include:

- Extensibility – We are able to extend the capability of the database server; this can be done by defining new data types, as well as user-defined patterns. This allows the us to store and manage data.
- Complex types – It allows us to define new data types that combine one or more of the currently existing data types. Complex types aid in better flexibility in organizing the data on a structure made up of columns and tables.
- Inheritance – We are able to define objects or types and tables that procure the properties of other objects, as well as add new properties that specific to the object that been defined.

Description of the encapsulation of the database:

We are going to work with Django user. We will use it as abstract user, then other actors are inherited by this abstract user. Grad student, staff and visitor are inherited by abstract user. Academic staff is inherited by staff. Quota manager is inherited by academic staff. Inheritance is provided with foreign key feature of Django model.

- User model - User model needs e-mail, first name, last name, password, phone number.
- Visitor model – Visitor model needs user as foreign key, citizenship number as char field, birthday as date field, gender as char field, application date as date field, address as text field, city as char field, degree as char field, university as char field, general point average as decimal field, ales as positive integer field, yds as positive integer field, acceptance as boolean field, program as foreign key.
- Student model: Student needs user as foreign key, student identification number as char field, student e-mail as e-mail field, curriculum as foreign key, program as foreign key, advisor as foreign key and check variables as boolean fields.
- Staff model: Staff model needs user as foreign key, citizenship number as char field, birthday as date field, gender as char field, main e-mail as e-mail field, school mail as e-mail field, address as text field and city as char field.
- Academic Staff model: Academic Staff model needs staff as one-to-one field from Staff model, university as car field and institute as foreign key.
- Institute model: Institute model needs name as char field, head as foreign key and established date as date field.

- Department model: Department model needs a name as char field, institute as foreign key and head as foreign key.
- Program model: Program model needs name as char field, code as char field, type as char field, thesis as boolean field, department as foreign key, head as foreign key, quota manager as foreign key.
- Curriculum model: Curriculum model needs program as foreign key, year as integer field.
- Course model: Course model needs code as char field, title as char field, description as text field, credit as integer field, ects credit as integer field, program as foreign key, university as char field and check variables as boolean fields. Course model acts a archive for all courses from beginning to now.
- Quota Manager model: Quota Manager model needs quota manager as foreign key. It is inherited from academic staff model.
- Section model: Section model needs course as foreign key, number as positive integer field, quota as positive integer field, year as char field, instructor as foreign key, semester as char field, special quota as many-to-many field, students as many-to-many field. Section is open course.
- Schedule model: Schedule model needs section as foreign key, day as char field, slot as char field and place as char field. Instances of Schedule model are to show where and when a section is.
- Course Type model: Course Type model needs title as char field and code as char field.
- CCR Course model: CCR Course model needs curriculum as foreign key, type as foreign key, no as positive integer field and semester as positive integer field. This model is to complete all slots with courses in curriculum.
- Offered Course model: Offered Course model needs ccr course as foreign key, active course as a foreign key. Ccr course is inherited from ccr course model and active course is inherited by course model.
- Taken Course model: Taken Course model needs student as foreign key, ccr course as foreign key, active course as foreign key and acceptance as null boolean field. Ccr course is a course that has defined according to curriculum. Active course is a course that is inherited from Section, student desire to take. Acceptance is statue of the data that is required why advisor permission. Uniqueness is checked.

- Completed Course model – Completed Course model needs student as foreign key, ccr course as foreign key, active course as foreign key and grade as char field. Ccr course is a course that has defined according to curriculum. Active course is a course that student has passed. Grade is a semester point of the active course. Uniqueness is checked.

All uniqueness check is controlled in models. All lengths are set up in models. Proxy design-pattern is used to prevent to models directly.

3.7 Access Control and Security

All defined users on Institute Automation System, those are academic staffs, institute staffs and grad students, are able to login in to the system with school e-mail that is defined in registration process and own password in secure.

System administrator that is a user has all management authentication, can reach in to the control management mechanism of Institute Automation System over Django administrator panel with own username and password.

Visitors are able to reach to application forms to become grad student directly without any authentication mechanism.

All permissions are defined as groups on Django administrator panel by system administrator. These groups are set up through actors.

Actor name	**Group name**
Academic Staff	Academic Staffs
Institute Staff	Institute Staffs
Grad Student	Grad Students

Table 3.7 – Actors and groups

Staffs can be divided into another level in later with same logic. After division of groups, operations have been defined. Accordingly, institute staff can make adding, displaying or deleting operations anymore. Thus, because of front-end structure, anybody cannot reach another actor's permissions.

3.8 Global Software Control

There is required internet-connection that was established earlier, to reach to Institute Automation System. Because of HTTP messages, people can access to Institute Automation System. Session is detected on authentication. After authentication, person is redirected in to another page. Her group is detected, so that she cannot access to page that she does not have permission to display.

Accessible pages are synchronized over internet-connection. To succeed this, database connection is also required. After database connection has been done, synchronization operations are satisfied.

3.9 Boundary Conditions

- Start-up:
  - There is required a virtual private server, in shortly VPS, that has Python 3 version. Then, Django 2 version must be set up.
  - Project's requirements must be documented with freeze command. It creates a text file that is named requirements.
  - Debug feature in project settings must be converted from true to false.
  - Project must be transferred from local repository in to this virtual private server completely.
  - Created requirements text file must be called with command line. Thus, other requirements are downloaded from relevant libraries.
  - The server must be run finally.

- Shutdown:
  - Shutdown might occur with two-ways.
    I. Physical hardware might be interrupted.
    II. Virtual private server might be stopped.

- Error behavior:
  - Internet-connection might be unconnected.

- Physical hardware might be interrupted.
- Virtual private serve might be stopped.
- Database might be lost.
- Database connection might be unconnected.

3.10 Subsystem Services

- Institute sub-system includes management of institutes, departments, programs, curriculums, courses, course types and sections.
    - Institutes tab under institute sub-system contains to display all data and add a new institute with own attributes. Adding operation creates institute object, then saves into Institute model.
    - Departments tab under institute sub-system contains to display all data and add a new department with own attributes. Adding operation creates department object, then saves into Department model.
    - Programs tab under institute sub-system contains to display all data and add a new program with own attributes. Adding operation creates program object, then saves into Program model.
    - Curriculums tab under institute sub-system contains to display all data and add a new curriculum with own attributes. Adding operation creates curriculum object, then saves into Curriculum model.
    - Courses tab under institute sub-system contains to display all data, display available courses, define a new course with own attributes, open or close an existing course and remove an existing course. Adding operation creates course object, then saves into Course model. Removing operation changes a check variable from false to true. Open/close operation changes a check variable from false to true bidirectionally.
    - Course types tab under institute sub-system contains to display all data and add a new course type with own attributes. Adding operation creates course type object, then saves into Course Type model.
    - Sections tab under institute sub-system contains to display all data, add a new section with own attributes and remove all data. Adding operation

creates section object, then saves into Section model. Removing operation removes from database forever.

- Academic staff sub-system includes management of all academic staffs, institute heads, department heads, program heads and quota managers.
    - All academic staffs tab under academic staff sub-system contains to display all data and add new academic staff with own attributes. Adding operation creates academic staff object, then saves into Academic Staff model.
    - Institute heads tab under academic staff sub-system contains to display all data.
    - Department heads tab under academic staff sub-system contains to display all data.
    - Program heads tab under academic staff sub-system contains to display all data.
    - Quota managers tab under academic staff sub-system contains to display all data and add new quota manager with own attributes. Adding operation creates quota manager object, then saves into Quota Manager model.
- Institute staff sub-system includes management of all institute staffs.
    - All institute staffs tab under institute staff sub-system contains to display all data and provide acceptance operations.
- Grad student sub-system include management of all grad students.
    - All grad students tab under grad student sub-system contains to display all data.
    - All completed courses tab under grad student sub-system contains to display all data about completed courses by department, program, courses, students.
- Application sub-system includes management of all applications from visitors.
    - Applications tab under application sub-system contains to display all data and accept, reject or remove operations. Accepting operation creates student object, then saves into Student model. Removing operation removes the selected instance from database forever.
    - Remove all applications tab under application sub-system contains to remove all applications from system forever.

CHAPTER FOUR
IMPLEMENTATION DETAILS
AND TESTS [6]

This chapter contains highly detailed implementation of Institute Automation System. UML diagrams, pseudocodes, and syntaxes used to represent subsystem and algorithms to visualize.

This chapter contains:

- Object design trade-offs
- Interface documentation guideline
- Model class interfaces
- Implementation planning and view class interfaces
- Package schema
- System test results

4.1 Object Design Trade-offs

- We implemented the “active course” attribute in Taken Course model with Course foreign key. We have changed it as Section foreign key, instead Course foreign key. Semantically, Taken Course represents students have registered courses currently.
- We have worked with Django User model. We used it as abstract user model. Phone number attribute was char field. We thought people might type invalid input for phone number. Therefore, we have evaluated by regular expression.
- We have added some extra attributes in to Student model. These are orderly “hold state”, “reg open statue” and “approval statue”. Without these, the controlling mechanism is weak.
- Instead of defining academic staff completely, we have satisfied to inherit from Staff model. So that, complexity decreased.
- Instead of removing an existing course, we have defined an attribute that is called “is deleted”. Thanks to this variable, we will not remove a course from the database. It is stored in database similar to archive.

- Instead of defining another model for available courses, we have defined an attribute that is called “is valid” in Course model.
- Instead of defining all courses in Curriculum model, we have followed a strategy to solve this. We implemented CCR Course model and Curriculum model. We are able to add a course in CCR Course for a selected curriculum.
- Instead of defining a role model, we used Django permissions.

4.2 Interface Documentation Guideline

Front-end design:

- Static content, such as CSS and Bootstrap files, is made a folder that is called static, in main.
- Django Template Language (DTL), in other words jinga2, is used for public side of project.
- There are five bases. Child bases are inherited from parent base.
- Common areas where assets – footer, header, head, navigation - and body, in other say content area, are tagged with jinga2.
- All pages extend a base through the activity.
- Django forms are used for input operations. They are implemented in pages as classical.
- Django Group Permissions are implemented in pages for security.

4.3 Model Class Interfaces

Model name: User

Model attributes – explanation – dependencies:

Model attributes – explanation:

- Username – Unique username, it should be defined as university identification number.
- E-mail – E-mail address of the user.
- First name – First name of the user.
- Last name – Last name of the user.
- Phone regex – It is regular expression to get valid phone number.
- Phone number – Phone number of the user.
- Date joined – Date that user has been joined into the system.

- Is active – It is for ban.
- Is super user – It is for defining system administrator.
- Is staff – It is for defining staff.
- Avatar – Picture of the user
- Personal information – All personal information of the user, it comes from another model.

Dependencies:

- It is dependent with Personal information model.

Model operations:

- Generate unique username – It is for generating unique username.
- Full name – It is toString method.
- E-mail user – It is for sending an e-mail to user to give as password.
- Save – Saves the all data.

Model name: Visitor

Model attributes – explanation – dependencies:

Model attributes – explanation:

- User – All user information of visitor, it comes from User model.
- TC – Turkey Citizenship number of visitor.
- Birthday – Birthday of visitor.
- Gender – Gender of visitor.
- Application Date – Auto generated date for application.
- Address – Home address of visitor.
- City – City where visitor has been lived.
- Degree – Last education degree of visitor.
- University – University that visitor has been graduated.
- GPA – General point average of visitor that is belonging to university.
- ALES – ALES exam point of visitor.
- YDS – YDS exam point of visitor.
- Acceptance – It is statue for defining the application.
- Program – University program that visitor has been applied.

Dependencies:

- It is dependent with User, Personal Information and Program models.

Model operations:

- There is no model operation.

Model name: Student

Model attributes – explanation – dependencies:

Model attributes – explanation:

- User – All user information of student, it comes from User model.
- Student ID – Unique student identification number.
- Student E-mail – An e-mail address is defined by school for student.
- Curriculum – Curriculum that is actual for a program. It is automatically defined according to the last curriculum in a program.
- Program – Program that student has been applied to.
- Advisor – It is defined by school, in other words academic advisor of a student.
- Hold State – Boolean field to check financial statue of student.
- Reg Open Statue – Boolean field to check registration time of student.
- Approval Statue – Boolean field to check approval statue of student.

Dependencies:

- It is dependent with User, Curriculum, Program, Academic Staff models.

Model operations:

- There is no model operation.

Model name: Staff

Model attributes – explanation – dependencies:

Model attributes – explanation:

- User – All user information of staff, it comes from User model.
- TC – Turkey citizenship number of staff.
- Birthday – Birthday of staff.
- Gender – Gender of staff.

- Joined date – It is obtained automatically by system, the date that staff has been joined to system.
- Main E-mail - Personal e-mail address of staff.
- School E-mail – E-mail address of staff, it is defined by school.
- Address – Home address of staff.
- City – City that staff has been lived.

Dependencies:

- It is dependent with User model.

Model operations:

- There is no model operation.

Model name: Academic Staff

Model attributes – explanation – dependencies:

Model attributes – explanation:

- Staff – All academic staff information, it comes from Staff model.
- University – University that academic staff has been worked.
- Institute – Institute that academic staff has been belonged to.

Dependencies:

- It is dependent with Staff (User, Personal Information) and Institute models.

Model operations:

- There is no model operation.

Model name: Institute

Model attributes – explanation – dependencies:

Model attributes – explanation:

- Name – Name of institute
- Head – Head of institute, it comes from Academic Staff model.
- Established Date – Date that it is obtained automatically.

Dependencies:

- It is dependent with Academic Staff model.

Model operations:

- There is no model operation.

Model name: Department

Model attributes – explanation – dependencies:

Model attributes – explanation:

- Name – Name of department.
- Head – Head of department, it comes from Academic Staff model.
- Institute – Institute that department belongs to.

Dependencies:

- It is dependent with Academic Staff and Institute models.

Model operations:

- There is no model operation.

Model name: Quota Manager

Model attributes – explanation – dependencies:

Model attributes – explanation:

- Quota Manager – Name of quota manager.

Dependencies:

- It is dependent with Academic Staff model.

Model operations:

- There is no model operation.

Model name: Program

Model attributes – explanation – dependencies:

Model attributes – explanation:

- Name – Name of program.
- Code – Code of program.
- Type – Type of program.
- Thesis – Boolean field to show program details.

- Department – Department that program belongs to.
- Head – Head of program.
- Quota Manager – Quota Manager of program.

Dependencies:

- It is dependent with Academic Staff, Department, Quota Manager models.

Model operations:

- There is no model operation.

Model name: Curriculum

Model attributes – explanation – dependencies:

Model attributes – explanation:

- Program – Program that curriculum belongs to.
- Year – Time that curriculum has been defined.

Dependencies:

- It is dependent with Program model.

Model operations:

- There is no model operation.

Model name: Course

Model attributes – explanation – dependencies:

Model attributes – explanation:

- Code – Code of course.
- Title – Full title of course.
- Description – Full details, in other say description, of course.
- Credit – Credit of course.
- ECTS Credit – ECTS credit of course.
- Program – Program that course belongs to.
- University – University that course belongs to.
- Is Valid – Boolean field to check course is open.
- Is Deleted – Boolean field to show course has been removed.
- Created Date – Time that course has been created. It is automatically defined by system.

Dependencies:

- It is dependent with Program model.

Model operations:

- There is no model operation.

Model name: Section

Model attributes – explanation – dependencies:

Model attributes – explanation:

- Course – Course is selected to define a section.
- Number – Unique number of section.
- Quota – People amount of section.
- Instructor – Instructor of section.
- Year – Year that section has been defined.
- Semester – Semester that section has been defined.
- Special quota – It is for opening a special quota for a student.
- Students – Full list of students in this section.

Dependencies:

- It is dependent with Course, Academic Staff, Student model.

Model operations:

- There is no model operation.

Model name: Schedule

Model attributes – explanation – dependencies:

Model attributes – explanation:

- Section – Section of a certain schedule.
- Day – Day of section.
- Slot – Slot of section.
- Place – Place of section.

Dependencies:

- It is dependent with Section model.

Model operations:

- Write Roman – It is to write Roman number.

Model name: Course Type

Model attributes – explanation – dependencies:

Model attributes – explanation:

- Title – Full title of course type.
- Code – Code of course type.

Dependencies:

- There is no dependency.

Model operations:

- There is no model operation.

Model name: CCR Course

Model attributes – explanation – dependencies:

Model attributes – explanation:

- Curriculum – To define a certain curriculum.
- Type – To define course type.
- No – Course number.
- Semester – Semester of CCR course.

Dependencies:

- It is dependent with Curriculum and Course Type model.

Model operations:

- There is no model operation.

Model name: Offered Course

Model attributes – explanation – dependencies:

Model attributes – explanation:

- CCR Course – Course that must be taken.
- Active Course – Course that has been taken.

Dependencies:

- It is dependent with CCR Course and Course models.

Model operations:

- There is no model operation.

Model name: Taken Course

Model attributes – explanation – dependencies:

Model attributes – explanation:

- Student – Student that is taking course.
- CCR Course – Course in curriculum of student.
- Active Course – Section of student.
- Accepted – Boolean field to define draft schedule.

Dependencies:

- It is dependent with Student, CCR Course, Section models.

Model operations:

- There is no model operation.

Model name: Completed Course

Model attributes – explanation – dependencies:

Model attributes – explanation:

- Student – Student of completed course.
- CCR Course – Course in curriculum of student.
- Active Course – Course that is selected by student.
- Grade – Grade of student, in that course.

Dependencies:

- It is dependent with Student, CCR Course, Course models.

4.4 Implementation Planning and Backend Details

We have planned service-oriented architecture, not class based. The web-services in project are similar to each other. The system runs with retrieving, adding or deleting and changing data operations.

We already developed a web-service for “Institutes” tab.

- Pseudocode for “Institutes” tab as follows:

Algorithm: Display institutes and/or add new institute item

---

procedure institutes (*instance name*)
    define a variable for rendering *URL*
    retrieve all objects from *database*
        - **if** there are criteria, it is defined (*i.e. establishing date*)
    the object is added into *dictionary*
    if there is an adding form:
    if request is controlled:
        form is created
        if form is valid:
            form is saved
            data is cleaned
        else:
            return to *invalid* page
    else:
        form is created (*without request type*)
    form is added into *dictionary*
    return with *request, url* and *dictionary*

- Python syntax with explanation for “Institutes” tab as follows:

The web-service is called in a page, receives the incoming HTTP request.
def *institutes(request):*
Rendered URL is defined.
url = 'institutes.html'
The purpose of this page is to display all institutes. Therefore, a typeless variable is defined and all Institute objects are received. It can be received with “.objects.all()” function. We can order them by some feature. We receive Institute objects ordering by “established date” attribute.
institutes = Institute.objects.order_by('-establishedDate')
Normally, we can return “institutes” object, but we want to return a form to add new institute object. We took a request for web-service. If we create a form in secure, we should check it, if it is POST or GET.
if request.method == 'POST':
When request is POST, we define a “form” object. We call relevant form with request, match to this form object.

form = AddInstituteForm(request.POST)
We assume user has filled the form. If the form is valid, we open new indent.
if form.is_valid():
Form is saved.
form.save()
Received data is cleaned. This is required for invalid text.
name = form.cleaned_data['name']
head = form.cleaned_data['head']
else:
Otherwise (form is not valid), it is returned to page that is called invalid.
return redirect('/invalid')
else:
form = AddInstituteForm()
We return everything by rendering. Returning objects are request, url, a dictionary that is including “institutes” and “form” objects.
return render(request, url, {'institutes' : institutes, 'form' : form})

Table 4.4.1 – Python syntax for “Institutes” tab

It might be desired to see details of any item. The web-service is called in “Institutes” page. When any item is clicked, new request goes to relevant method. The web-service receives request and identification number of selected item.
def instituteDetails(request, id=None):
Rendered url is defined.
url = 'institute-details.html'
All item has default unique primary key. All details of selected item is received with “get_object_or_404” function. It can be received with other functions, such “.objects.all() – objects.filter() - .objects.get() – and so on”. “get_object_or_404” needs two fields. These are model name and primary key. Model name is Institute, and pk is identification number of clicked item.
instituteDetails = get_object_or_404(Institute, pk=id)
It is returned by rendering request, url and all content.
return render(request, url, {'instituteDetails' : instituteDetails})

Table 4.4.2 – Python syntax for specific item

4.5 Package Schema

IAS: Institute Automation System

- Media
- IAS
  - init.py
  - settings.py
  - urls.py
  - wsgi.py
- Institute
  - migrations
  - init.py
  - admin.py
  - apps.py
  - forms.py
  - lists.py
  - managers.py
  - models.py
    - user
    - visitor
    - personal information
    - student
    - staff
    - academic staff
    - institute
    - department
    - program
    - quota manager
    - curriculum
    - course
    - section
    - schedule
    - course type
    - ccr course
    - offered course
    - taken course
    - completed course
  - tests.py

- views.py
  - home
  - academic staffs
  - institute staffs
  - grad students
  - institute
  - institutes
  - institute details
  - departments
  - department details
  - programs
  - program details
  - curriculums
  - curriculum details
  - courses
  - course details
  - available courses
  - remove course
  - open/close course
  - course types
  - sections
  - section details
  - academic staff registration
  - all academic staffs
  - academic staff details
  - institute heads
  - department heads
  - program heads
  - quota managers
  - all institute staffs
  - all grad students
  - applications for visitors
  - applications
  - application details
  - remove selected application
  - remove all applications
  - successfully
  - invalid

- all completed courses
- all completed course details
- selected completed course details

- Pages
  - assets
    - contents
      - modal (html)
      - table (html)
    - menu
      - academic staffs (html)
      - grad students (html)
      - institute staffs (html)
      - institute (html)
    - footer (html)
    - head (html)
    - menu (html)
    - navigation (html)
  - registration
    - login (html)
  - academic staff details (html)
  - academic staff registration I (html)
  - academic staff registration II (html)
  - academic staffs (html)
  - all academic staff (html)
  - all completed courses details (html)
  - all grad students (html)
  - all institute staffs (html)
  - all completed courses (html)
  - application details (html)
  - applications (html)
  - apply (html)
  - available courses (html)
  - base (html)
  - curriculum details (html)
  - curriculums (html)
  - close course details (html)
  - close course (html)

- course details (html)
- course types (html)
- courses (html)
- department details (html)
- department heads (html)
- departments (html)
- grad student details (html)
- grad students (html)
- institute details (html)
- institute heads (html)
- institute staffs (html)
- institute (html)
- institutes (html)
- invalid (html)
- program details (html)
- program heads (html)
- programs (html)
- quota managers (html)
- remove course details (html)
- remove course (html)
- section details (html)
- sections (html)
- selected completed course details (html)
- student application I (html)
- student application II (html)
- successfully (html)

- Static
  - css
  - vendor
- Staticdev
  - admin
  - css
  - vendor
- DB.sqlite3
- Manage.py

Table 4.5 - Packaging

4.6 System Test Results

- This screen capture shows that pages change in how much time. This data is retrieved from server side. Each line is a change, actually request. This testing is done over 2 MBPS internet-connection. In less than 1 second time unit, two or three requests are rendered.

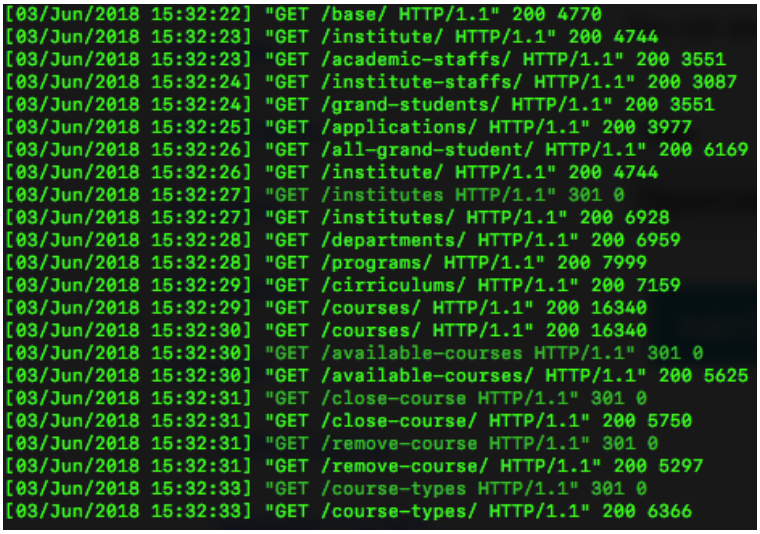

Figure 4.6.1 – Surf on system

- Another test is related adding a new item.
  - Institute staff goes to “Institute” menu, clicks to “Institutes” under Institute tab.
  - Then, “Institutes” page is loaded.
  - Institute staff clicks to “add new institute” button.
  - Modal is loaded.
  - Institute staff types, then clicks to “save” button.

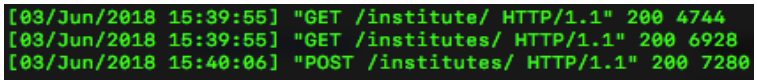

Figure 4.6.2 – Change for adding new item

- Click count for application: 3 clicks, 2 forms
- Click count for accepting or rejecting an application: 5 clicks (*including login*)
- Click count for defining a new course: 5 clicks, 1 form (*including login*)

CHAPTER FIVE
CONCLUSIONS AND FUTURE WORK

Success criteria of my project were:

- Develop the functions of existing system
- Develop more secure, reliable and faster system
- Develop completely new functions to provide automation
- Decrease waiting time in queue and prevent deadlock, or errors

All of these criteria are accomplished:

- Developed the functions of existing system – We just have able to analyze features
- Developed more secure, reliable and faster system – We have implemented proxy design pattern over Django's backend. We have decreased database complexity and database operations; therefore, system could be faster.
- Developed completely new functions to provide automation – We have improved some extra features: defining exam dates, sending notifications and so on.
- Decreased waiting time queue and prevented deadlock, or errors – We have prepared testing environment, then could be compared existing system and proposed system over stopwatch. Decreasing database complexity could prevent deadlock because of database operations.

We have applied and used object-relational model and model-view-template successfully throughout the Project. Service oriented architecture approach is applied partially. Institute Automation System is ready to demonstrate in terms of Institute Staff.

Future work:

- More capable and useful functions are able to be developed.
- Usable and more user-friendly interface is able to be developed.
- Data integration is able to be done.
- The system with all details is able to be deployed.

REFERENCES

1. “Campus Online - University of Isik”, Internet: http://campus.isikun.edu.tr
2. “UBIS - University of Istanbul Aydin”, Internet: https://ubis.aydin.edu.tr
3. “Online Application – University of Galatasaray”, Internet: http://fbe.gsu.edu.tr/tr/basvuru
4. Ulusoy, Oğuzhan - “Requirement Analysis Document of Institute Automation System”, April 20, 2018.
5. Ulusoy, Oğuzhan - “System Design Document of Institute Automation System”, May 2, 2018
6. Ulusoy, Oğuzhan – “Object Design Document of Institute Automation System”, May 4, 2018