



College of Engineering
COMP 491 – Computer Engineering Design Project
Final Report

KUSISTANT

Participant information:

Alkan AKISU
Burak YILDIRIM
İsmail Hakkı YEŞİL
Mete UZ

Project Advisor(s):

Gözde Gül Şahin

Spring 2022

Table of contents	Page
I. Abstract	3
II. Introduction	4
III. System Design	5
1- Application Login	5
2- Bot Interaction and NLP	6
3- Bridge API and Web-Scraping API	8
4- Flutter and User Interface	9
5- MongoDB	10
IV. Analysis and Results	10
V. Conclusion	12
VI. References	13

I. Abstract

A variety of chatbots and dialog agents are currently being used as automation tools. In an online-based education, students must visit many websites for various academic assignments. The main goal of KUsistant is to help Koç University students with their academic challenges. It serves as a link between many academic websites. Similar chatbots are used to aid teaching assistants or students with course-related issues. Our focus is on distributed and computerized academic systems, not instructional chatbots.

The system design of KUsistant combines NLU, backend, and frontend. Each component serves a purpose in the system. NLU for intent recognition. APIs and databases are for data retrieval and storage. The front-end is for user interaction with the login and chat screen. In general, each component has a link with one or more others and KUsistant's system has a linear structure and modular structure

Analysis and results of KUsistant were mostly positive. At first, KUsistant's linear design works correctly and data transfer between each of the components is successful. KUsistant delivers 2 running servers 1 database for user id and cookie storage, and deliverables: KUsistant mobile app, running servers for cookies. Tests are performed on different cases and results from them were used to improve our application.

Simulation is performed at the poster event. The people who joined the simulation are Koç University students, teaching assistants, and professors. They were very positive about the KUsistant. Positive feedback and improvements are noted and will be used in future research.

Our user research revealed that KUsistant made a positive effect on students since they all had the same problem. For future work, better APIs, and voice and multi-intent recognition can be added. More works and examples of personal student assistants are needed for a better analysis of KUsistant's functionality.

II. Introduction

With the rapid growth of the internet and its reach to everyone, companies are trying to automate not only the production phase but also the communication with the user. With this automation process, we are currently faced with different types of chatbots and dialog agents. We mainly researched the usage of chatbots and dialog agents in education. Since education is becoming more and more digital, students need to follow different websites for different works. Distributed academic structure imposes another burden on students. Students need to check one website for their exam schedule, another one for grades, and another for different academic works.

Although the analysis study by Winkler and M. Söllner [1] shows that chatbots play an important role in education and their role will be increased in the future, it is important to note that the analysis of chatbots in this study is about learning assistants. We believe it is important to describe clearly KUsistant's position as an important matter in our report so that we can compare what KUsistant can do and talk about KUsistant. In our project, our scope is not educational chatbots but distributed and digitized academic systems. KUsistant is a solution to the problems created by distributed academic systems of Koç University, not to educational problems created by digitized education.

KUsistant's main aim is to solve the distributed academic problems of Koç University students. Similar student assistants are for course-related help to teaching assistants or the students. Their main goal is to help students with the course or assignment materials. KUsistant chooses another way to help students. Acting as a bridge between different academic websites. Koç University students have 2 main academic websites to check for academic works: KUSIS [2] and Blackboard [3]. Taking 4 or 5 courses, and for even more extreme cases this division creates problems. KUsistant acts as a bridge between these two websites, a connection that gets the necessary information for students' needs. Taking letter grades of a course from KUSIS API and course deadlines from BlackBoard API. Combining KUSIS and BlackBoard may be another solution to this problem and it needs further studies.

III. System Design

KUistant's system design is the combination of NLU, backend, and frontend. Each part has its role in the system. If we need to examine the system in more detail, we would see 5 components: Front-end (app-side), Google Dialogflow, Web-Scraping API, Bridge API, and MongoDB database. Generally speaking, each 5 of the components has a relationship with one or more components. Moreover, it is important to note that KUistant's system has a linear structure as illustrated in [Fig. 1]: Answer/s from the requested component need to be valid for the system's integrity.

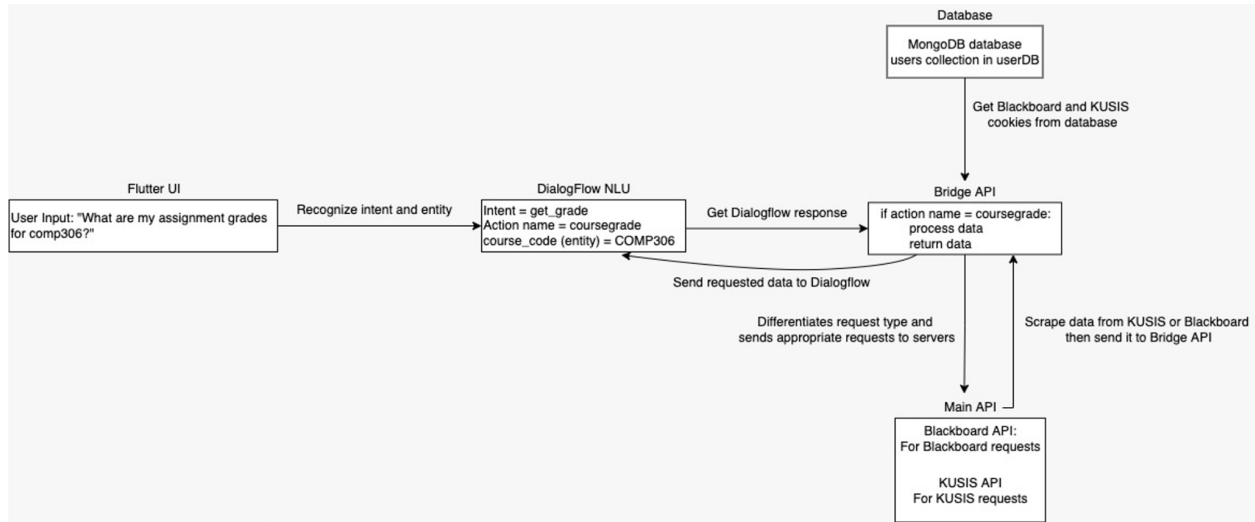


Figure 1: System Design of KUistant.

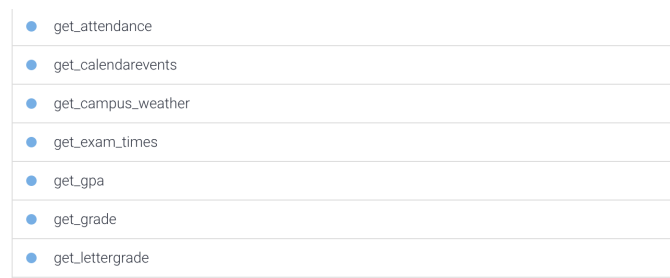
1- Application Login

Before any of the system components interact with one another, users need to log in to Blackboard and KUSIS with their credentials so the app can get their cookies and store them in a database with a unique user id. For each of the users using KUistant, there is a unique randomized id stored in local. We only delete the user's id from our database when the user deletes the application. This way we guarantee that every user will have a unique id so that cookies will be mapped to one and only one user.

2- Bot Interaction and NLP

After logging in, the user can ask a question to the chatbot. Multiple questions can be asked to the bot such as Weather conditions at Koç University, GPA, Letter grade of a specific course, and so on. Afterward, the application will send this text to Dialogflow to extract intent and necessary data. For instance, if a user asks “What is my COMP491 grade?” to the bot, ask_grade and necessary data which is the course code of COMP49 will be the extracted intent and data by the Dialogflow respectively.

In the Dialogflow, multiple intents are created such as get_gpa, get_weather, get_grade, etc. For each use case, necessary intents are created as seen in [4, Fig 2]. For intent matching Dialogflow uses rule-based grammar matching and ML matching [4]. This ensures that grammar mistakes or the position of the words will not be a problem for the NLP model. We preferred Dialogflow as the NLU framework because it is more convenient for our application. Other reasons behind our selection are accurate entity and intent recognition, fuzzy matching for more accurate matching since fuzzy matching detects similar words or phrases corresponding to the input of the user and webhook support. Webhook support is important in our case since KUsistant has a lot of different APIs for different data-gathering operations.



• get_attendance
• get_calendarevents
• get_campus_weather
• get_exam_times
• get_gpa
• get_grade
• get_lettergrade

Figure 2: Sample Section of Dialogflow Intent Dashboard [4].

Another important aspect of our Dialogflow model is entity extraction with regular expressions. Koç University course codes have a specific format: the First 4 characters are letters and the last 3 characters are digits. To extract this format, regular expressions are used. This formatting can be seen in [4, Fig. 3]. For some other entities, the fuzzy matching feature of Dialogflow is used to achieve better results.

course_code

SAVE

☒ Define synonyms ?
☒ Regexp entity ?
☐ Allow automated expansion

☐ Fuzzy matching ?

\S{4}\s*\d{3}

Enter value

Figure 3: Dialogflow's Entity Matching [4].

Dialogflow intents need to be trained with a dataset of training phrases. These training phrases are acquired by finding our potential sentences and conducting a user study. In this training operation, parameters are introduced and the action name is set to a constant value to match in the API later on. In addition, the fulfillment option is turned on to make the model learn custom responses. Moreover, the follow-up intent is added so that if Dialogflow doesn't detect any parameters although it requires one, it prompts to ask the parameter separately.

get_grade

SAVE

hey what is my MGMT 304 grade?

what scores for mava303

1 OF 2

Action and parameters

coursegrade

REQUIRED ?	PARAMETER NAME ?	ENTITY ?	VALUE	IS LIST ?	PROMPTS ?
<input checked="" type="checkbox"/>	course_cr	@course_code	\$course_code	<input type="checkbox"/>	Which course wo...
<input type="checkbox"/>	Enter nan	Enter enti	Enter value	<input type="checkbox"/>	—

+ New parameter

Responses ?

Fulfillment ?

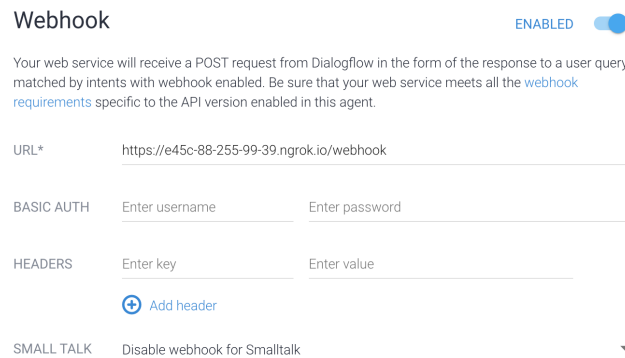
Enable webhook call for this intent

Enable webhook call for slot filling

Figure 4: Dialogflow's Intent Settings [4].

3- Bridge API and Web-Scraping API

Thereafter intent recognition, our APIs written in python [5] start functioning. First, Dialogflow will request a webhook to our Bridge API [4, Fig. 5]. Bridge API is our bridge between KUSIS and Blackboard API, and Dialogflow API. When it gets a request, the intent and user's cookies are sent to Web-Scraping API so it can get the data. Then Web-Scraping API sends a request to Blackboard or KUSIS. This request makes them send the HTML of the page. This HTML page contains the data the user needs so the Web-Scraping API uses the BeautifulSoup library[6] to scrape the HTML to get the information we need. Data is acquired at this point so Web-Scraping API will send back the data to Bridge API. Moreover, OpenWeather API [7] is used to get weather information on campus. Finally, Bridge API sends the data to Dialogflow as responses which will eventually be written to the chat screen.



The screenshot shows the 'Webhook' configuration panel in Dialogflow. At the top, there is a toggle switch labeled 'ENABLED' which is turned on. Below this, a text box explains: 'Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.' The main configuration area includes a 'URL*' field with the value 'https://e45c-88-255-99-39.ngrok.io/webhook'. Below the URL field are two input fields for 'BASIC AUTH': 'Enter username' and 'Enter password'. There is also a 'HEADERS' section with 'Enter key' and 'Enter value' fields, and a blue '+ Add header' button. At the bottom, there is a 'SMALL TALK' dropdown menu currently set to 'Disable webhook for Smalltalk'.

Figure 5: Dialogflow's Fulfillment Panel. [4]

In the Bridge API, the Flask library [8] is used to run it as a backend server. It checks the action name to execute the corresponding code fragment in [Source code 1]. Then, it sends requests to the Web-scraping API. Web-Scraping API returns the result. Finally, the JSON response format is prepared and sent to the Dialogflow.

```
if req.get("queryResult").get("action") == "coursegrade":
    result = req.get("queryResult")
    parameters = result.get("parameters")
    courseCode = parameters['course_code'].upper()
    URL = 'https://comp491.alkanakis.repl.co/getCourseID'
    PARAMS = {'courseName':courseCode}
    HEADERS = {'Cookie':bbCookie}
    r = requests.get(url = URL, params = PARAMS, headers=HEADERS)
    data = r.json()
    res = getCourseGradeResult(data['courseId'], bbCookie)
    return res
```

Source Code 1: Bridge API Handling Intent.

Web-scraping API scrapes the KUSIS and Blackboard websites to get the requested data by sending GET requests to them and parsing data from HTML elements. The Flask library is used to run it as a backend server. Methods in BlackboardAPI and KusiAPI are handled by a route [Source Code 2]. Each API has different functions such as KUSIS has GPA function and Blackboard has grade function. Using those functions, responses are returned to the Bridge API.

```
@app.route('/mygrades/<course_id>')
def getGrade(course_id):
    cookie = request.headers.get('Cookie')
    bb_API = BlackBoardAPI(cookie)
    grades = bb_API.getGrades(course_id)
    json_string = json.dumps([ob.__dict__ for ob in grades])
    r = make_response( json_string )
    r.mimetype = 'application/json'
    return r
```

Source Code 2: Web Scraping API Handling a Request.

4- Flutter and User Interface

In the Flutter application [9], a login screen welcomes our users. In this screen, users are asked to log in to their Blackboard and KUSIS accounts as seen in [Fig. 6]. Once they logged in, their unique id along with their Blackboard and KUSIS cookies are inserted into the MongoDB database [10]. After logging in, the user can start chatting with the bot and it will retrieve the data from either KUSIS or Blackboard. Sample chat in [Fig. 6]. Furthermore, since Flutter supports both IOS and Android, it would spread the usability and accessibility of KUsistant.

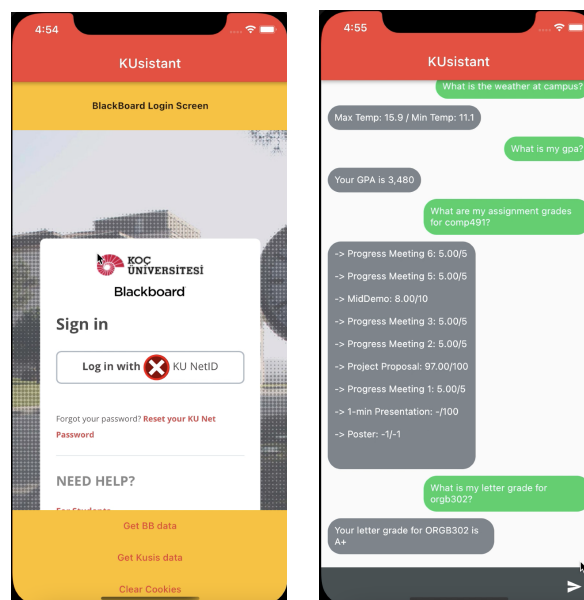


Figure 6: KUsistant Login and Chat Screen.

5- MongoDB

In MongoDB [10], there is a user database that stores users' cookies. In the user database, we mapped each user's cookie to a unique id for Blackboard and KUSIS cookies [10, Fig. 7] to make the retrieval process easier and to stop confusion. MongoDB is selected due to the advantages of a No-SQL like having the flexibility to store as JSON. It makes it easier to insert and receive data from Flask-based APIs and Flutter.

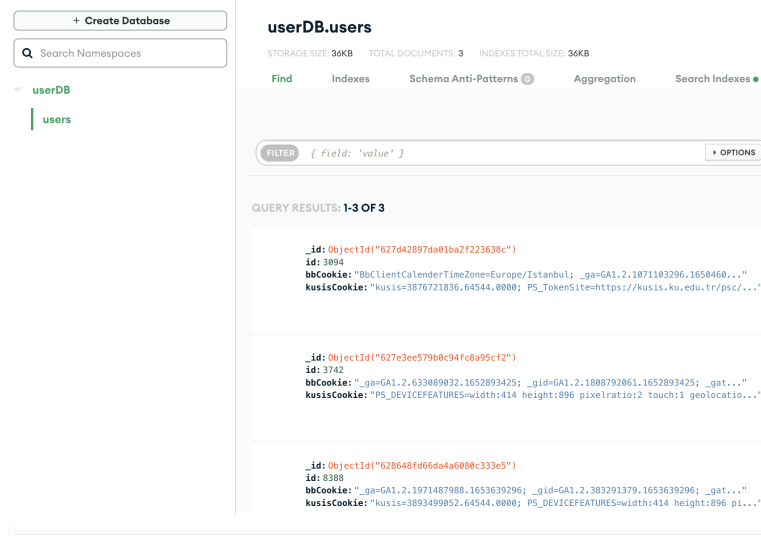


Figure 7: MongoDB Dashboard [10].

IV. Analysis and Results

At the end of our project, we ended up with different implementations. KUsistant now has 2 running servers for backend-related operations on KUSIS and Blackboard 1 MongoDB database for user id and cookie storage, and deliverables: KUsistant mobile app, running servers for cookies, MongoDB database and running Flutter-based mobile application. Every component is running correctly and transferring data between each other for users' requests. We saw that the design of KUsistant is very successful. Linear transferring and request operations make the correct gathering of user data from KUSIS and Blackboard.

Since our users are students of Koç University students, we conducted a user study among 12 Koç University students. Different tasks are given to the students to evaluate our application. These tasks consist of questions such as retrieval of a letter grade for a course, retrieval of

cumulative GPA, asking for assignment deadlines in 2 weeks, and finding the location of the Pandora store. Results revealed that all participants performed the first task on their first try, 7 participants performed the second task on their first try, 10 participants accomplished the third task on their first, and lastly, all participants completed the tasks on their first try. Our training phrases are expanded with the inputs of our test users.

Furthermore, the score for the easiness of our application is 4, performance satisfaction is 3.75, usability in Koç life is 4, and visually appealing is 4.33 averagely regarding the user study. For further improvements, users suggested adding voice recognition, and notifications, and improving the current app in terms of performance, usability, and interface. Generally, users gave positive feedback and said that it can be really helpful for any university student as it provides many features in just a single mobile application.

Analysis and simulation of KUsistant have been performed on both user study and “Final Project Poster Event of Koç University Engineering Faculty”. But before that, we have done extensive tests for a correct working application. Some of these tasks are as follows:

- Grammar mistakes check for use cases. This one is for making our model correctly evaluate the intent from even the grammatical mistakes of users’ text.
- Unique user id checks: This one is for making sure of every user will have a unique user id on their first login and this doesn’t check until the user deletes the KUsistant from their phone.
- Bridge and Web-scraping API checks: Since our application revolves around getting the correct information from KUSIS and Blackboard, we checked whether all of our use case intents are correct or not. Moreover, we also test the Bridge API on our servers and experiment on which information is coming to the Web-scraping API and Dialogflow. This way we understand clearly what is going on with our requests.

Finally, simulation is performed on the poster event. We tested our simulation with people passing by the poster event, teaching assistants, and professors. Every one of them was positive about our project and we only get positive feedback on what we can do to improve our project that is already good. Some even stated that we can create a subscription-based application in the future.

V. Conclusion

In conclusion, KUsistant creates a solution to the distributed academic structure of Koç University. KUsistant design is flexible and modular. It uses NLP for intent recognition, backend for data retrieval, and frontend for user interface, chat, and login screen. Our user study showed us that KUsistant created a very good impression among students since each has the same common problem: Clicking a lot of links consecutively for even a simple operation. KUsistant wants to make GPA, deadline, or letter grade retrieval processes much more efficient than its current format and it achieves it to a certain degree.

The main weakness of KUsistant is its modular and linear structure. This makes KUsistant work as intended. However, it also creates a very big problem: the Data retrieval process can be more than the Dialogflow's time-out time. If this happens, blank space will be printed on the chat screen and it will be a bad experience for our users. To solve this problem, a more sophisticated API can be designed and implemented. This way the same API can get the intent and data from Dialogflow, retrieve and parse the requested data and get it back to Dialogflow. With this implementation, KUsistant will be much more efficient in both times and use cases.

For future works, voice recognition, more use cases, and multi-intent recognition can be added. Voice recognition is a very useful tool for chatbots. More use cases can be implemented with a more strong API. Finally, multi-intent recognition is the missing link in KUsistant's academic function. A student might say get my grades and deadlines for the same or different classes. The current chat structure of KUsistant is back and forth communication of the user with the bot. This is a very useful way to get the requested information from the servers. However, it can be more effective if it has multi-intent recognition. Additionally, more follow-up intents can be implemented.

In the end, KUsistant is a good working mobile application with different APIs. It is a simple but combined and efficient solution to the distributed academic structure of Koç University. With more improvements to its design, it can be the ultimate personal student assistant for not only Koç University but also other universities inside and outside of Turkey.

VI. References

- [1] R. Winkler and M. Söllner, Unleashing the Potential of Chatbots in Education: A State-Of-The-Art Analysis, *Academy of Management Proceedings*. 2018. 15903. Apr-2018. Accessed: 05-Jun-2022. DOI:[10.5465/AMBPP.2018.15903abstract](https://doi.org/10.5465/AMBPP.2018.15903abstract) [Online]. Available: https://www.researchgate.net/publication/326281264_Unleashing_the_Potential_of_Chatbots_in_Education_A_State-Of-The-Art_Analysis.
- [2] *KUSIS*. Accessed: 05-Jun-2022 [Online]. Available: <https://kuis.ku.edu.tr/>.
- [3] Blackboard, *Blackboard learn*. Accessed: 05-Jun-2022. [Online]. Available: <https://ku.blackboard.com/>.
- [4] “Dialogflow ES documentation” *Google*. [Accessed: 08-Jun-2022]. [Online]. Available: <https://cloud.google.com/dialogflow/es/docs/>.
- [5] “Welcome to Python.org,” *Python.org*. Accessed: 08-Jun-2022 [Online]. Available: <https://www.python.org/>.
- [6] “Beautiful Soup documentation,” *Beautiful Soup 4.9.0 documentation*. Accessed: 08-Jun-2022 [Online]. Available: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>.
- [7] OpenWeatherMap.org, “Weather API,” *OpenWeatherMap*. Accessed: 08-Jun-2022 [Online]. Available: <https://openweathermap.org/api>.
- [8] “Flask,” *Welcome to Flask - Flask Documentation (2.1.x)*. Accessed: 08-Jun-2022 [Online]. Available: <https://flask.palletsprojects.com/>.
- [9] *Flutter*. Accessed: 08-Jun-2022 [Online]. Available: <https://flutter.dev/>.
- [10] *MongoDB*. Accessed: 08-Jun-2022 [Online]. Available: <https://mongodb.com/>.