BoilerBot

Team 1: Aditya Dhingra, Rahul Pai, Eehita Parameswaran,
Kush Rustagi, Devaunsh Sambhav, Yash Shiroya

February 6th, 2017

Design Document

Contents

1 Purpose **3**

1.1 Description . . . 3

1.2 Functional Requirements . . . 3

1.3 Non-Functional Requirements . . . 5

2 Design Outline **5**

2.1 Database . . . 6

2.2 Conversational UX . . . 6

2.3 Natural Processing Model . . . 7

2.4 Context Management . . . 7

3 Design Issues **8**

3.1 Functional Issues . . . 8

3.2 Non-Functional Issues . . . 8

4 Design Details **10**

4.1 Database Layer . . . 10

4.2 Classes and their Relations . . . 11

4.3 Sequence Diagrams . . . 13

4.4 UI Mockups . . . 15

1 Purpose

1.1 Description

Balancing multiple applications for different functionality on a daily basis is a tedious task for Purdue students. Accessing information about courses, dining court menus, crime statistics on different portals increases memory usage on the individual's device and incurs a toll on battery life. With that said, there are several service aggregator chat bots that help people on a daily basis. For instance, Google Home has doubled up to be a personal assistant keeping the user informed about news, weather and other Google related features. Even Alexa has similar functionality with Amazon. More specifically, these bots integrate only features available through a single platform. So, we believe there is a demand for a chat bot that can integrate various features for Purdue students.

Planning ahead for a semester and staying ahead of the game is extremely important for Purdue students. We intend to build **BoilerBot**, a Facebook Messenger chat bot that answers course related questions and efficiently provides students various Purdue resources at their fingertips. BoilerBot uniquely provides an aggregation of Purdue services by giving users quick access to useful information in one window. Thus, the bot will serve as an everyday companion to enhance the student experience at Purdue.

1.2 Functional Requirements

1. The user interaction capabilities of the chatbot:
   As a user,

   - I would like the bot to introduce itself by stating its role.
   - The first time I use the bot, I would like it to provide me with instructions or a start-up guide on how to use the application.
   - I would like the bot to repeat help instructions when asked.
   - I expect the bot to handle the conversation properly even when I send an irrelevant message.

   As a developer,

   - I would like to be able to handle the User Interface (UI) for mobile and web appropriately.

2. The device permissions that the chatbot will have:
   As a user,

   - I would like to have access to the internet.
   - I would like to be able to access the bot through my Facebook account.
   - I would like to be able to access the bot's Facebook FAQ page through my account.

3. The prominent attributes of the chatbot:
   As a user,

   - I want the chatbot to be platform-independent so that I can use it on any device.

- I would also like the bot to keep a log of all messages and settings across all conversations that I have.
- I would like to be able to view the menu for each individual dining court.
- I would like to have access to the timings of the individual dining courts.
- I would like to have access to the available seating in the dining courts.
- I would like to be able to see the current courses being offered for a given semester.
- As a student at Purdue, I would like the bot to provide me with the seats available in a specific course.
- As a student at Purdue, I would like the bot to provide me with information about the different professors teaching a course.
- As a student at Purdue, I would like the bot to provide me with information about the credit hours offered for taking a course.
- As a student at Purdue, I would like the bot to provide me with information about the different times at which a specific course is being offered.
- As a student at Purdue, I would like the bot to provide me with information about the different rooms in which a specific course is being taught.
- I would like be able to view the last crime committed at Purdue.
- I would like to have access to monthly crime statistics in a current year at Purdue.
- I would like to have access to the active warrants at Purdue.
- I would like to have access to the yearly crime statistics in a current year at Purdue.

4. Assistance providing capabilities of the bot:
   As a user,

   - I would like to have a facebook page for FAQs.
   - I would like to participate in a forum-like environment facilitated by the developers for any discussions.
   - I would like to have a tutorial on how the bot works. (if time permits)

5. Feedback capabilities of the chatbot:
   As a user,

   - I would like to be able to suggest some new features that the bot can provide. I expect the BoilerBot team to consider the request and then introduce them in future updates.

   As a developer,

   - I would like to receive feedback on popular features users may request.
   - I would like to receive user feedback through the Facebook page.
   - I would like to use the feedback and store it for further fixes and enhancements.
   - I would like to come up with regular updates to the chatbot that include more features based on user recommendations.

6. Security measures adopted by the chatbot:
   As a user,

   - I would like to use my Facebook credentials i.e. I want to login to my Facebook account to use the chatbot.

1.3 Non-Functional Requirements

1. Performance:
   As a user,
   - I would like the bot to respond quickly to complex queries too.
   - I would like the bot to inform me of any downtime due to maintenance.
   - I would like the bot to respond with descriptive error/instructions in case of any undefined requests.

   As a developer,
   - I would like the bot to allow multiple connections/users to run without delay between responses.
   - In case of any planned maintenance, our team intends the downtime to be minimized for 3-5 hours/month.
2. Design:
   As a developer,
   - I would like the client to experience the least computational load by only posting processed and finalized data from APIs (on the server side).
   - I would like the FAQ pages, instructions, any external redirects to also follow the design principles/choices used in the bot.
3. Security:
   As a developer,
   - I would like the user details and other personal information to be stored securely and only be accessible by the server when required for authentication.
   - Any storage of user metadata or user statistics(if recorded) should be stored privately in the database.

2 Design Outline

The BoilerBot team is implementing a Facebook Messenger chat bot that answers course related questions and efficiently provides students various Purdue resources at their fingertips. Our project follows the Multi Layer Pattern. When a user first interacts with the chatbot, a webhook on our backend receives the message and begins to parse it. We refer to the parser as our 'Conversational UX', which is an umbrella term for our Natural Language Processing and 'Context Management' modules. Our backend also heavily relies on a fully functional database layer to be used in conjunction with our 'Conversational UX', as well as a Natural Language Processing Module designed to exclusively handle all the message processing components of the system.

The 'Conversation UX' and 'Database' Layers together form the Model-View-Controller (MVC) framework that is the main architectural component of our system. Together all this information can be summarized by the document (UML diagram) below:

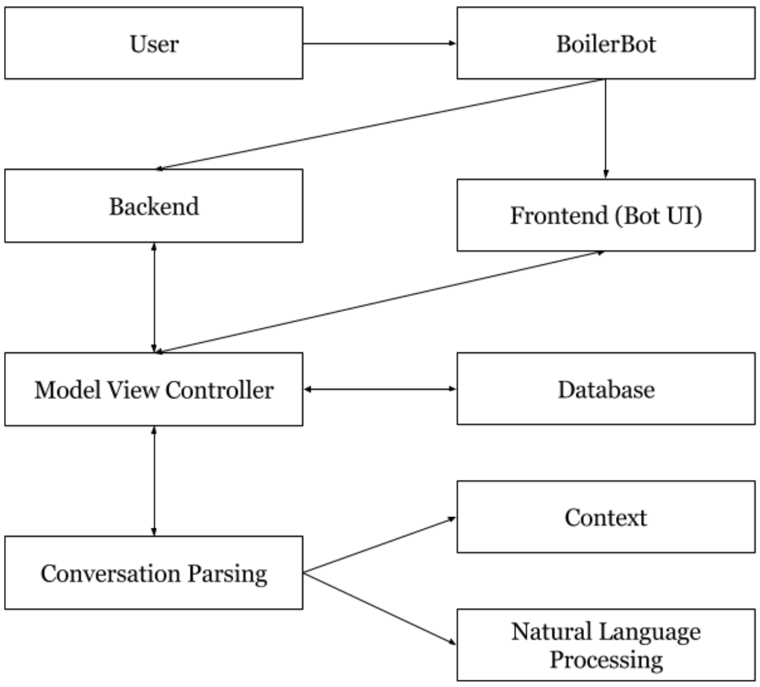


Figure 1: UML Diagram

2.1 Database

We will be using a SQL database. We will be storing all of our tables in a MySQL database. We believe that MySQL offers the most flexibility and is the most feature rich database in regards to our requirements. The database helps us store the feature requests sent by the user. The bot and Natural Language Processing model can query relevant information based on the user's request. While some of the information will be returned by scraping pages/making multiple API calls, other time-consuming and data-intensive requests would need to be queried from our database of pre-stored information.

Our database will typically store the user details, feedback, etc. In addition, it will store the user's usage of our chatbot, so that we can run analytics and refine their experience with the data from our database.

2.2 Conversational UX

The conversational UX is built around the Messenger platform. In essence, the goal is to make it feel like the user is chatting with just another friend. To achieve this and have it work effectively, we are using a Natural Language Processing API in the backend that parses the user's messages and understands its intent. The parsed intent(what we believe the user is trying to achieve) is then used to do one of the following things:

- Ask the user for which dining court they want details about.
- Ask the user for which course(s) they want details about.
- Ask the user for the crime statistics(monthly or yearly) they want details about.

- Ask the user to provide feedback on their chat experience with the bot.
- Respond with standardized error message for irrelevant messages.
- Exchange greetings and answer questions regarding chatbot.

Apart from the different types of responses that are required for the main interaction (regarding message processing), the chatbot would also be able to handle small talk to some extent. This means that the bot would be able to react to certain 'curveball' messages effectively and elegantly, thus maintaining a rich user experience.

2.3 Natural Processing Model

The Natural Language Processing for our chatbot is an amalgamation of Python libraries like NLTK and freely available APIs like Recast.AI. Combining the two would help us better understand the user's intent since this would allow us to personalize the bot to help Purdue students specifically. The API would make it easier to train BoilerBot as well as make it understand different types of sentences the user could use. When in use, the API parses the sentence and gives us an intent and a list of entities, if any. Intent is what the API determines the user's intention to be and entities are a list of things that the user wants to perform actions on.

For instance, User: Give me the courses available next semester.

This message would be parsed for intent and entities. The intent here would be 'courses' and the entity would be 'next semester'. This would tell our bot's backend that the user wants the list of courses available for the next semester. This results in the following pattern being followed for every single message:

- User sends a message. The message is sent to the backend.
- The message is first broken down, tokenized and formatted using NLTK, and relevant information is then sent to Recast for analysis.
- NLTK would give us insight into the message and help construct a query best fit for the request.
- Recast replies with an intent and a list of entities, if any.
- We use that information on the backend to take the appropriate actions and provide a response.

2.4 Context Management

Context Management is basically remembering the flow of the conversation. The bot would remember what the user had asked for in the previous conversation. An example for this is as follows:

User: Give me the courses available next semester.

Bot returns a couple of courses and provides the option to choose a specific subject for the courses.

User: Okay, give me the Computer Science courses available.

This is where the context management kicks in. The bot would essentially remember the intent of the previous conversation. It would know that the user wanted specifically Computer Science courses and BoilerBot would provide that list.

3 Design Issues

3.1 Functional Issues

1. How are we going to handle messages from the user that are not relevant to Purdue and not related to the functionality of the bot?

   Option 1: A standardized reply, for any message that the bot cannot understand as determined by a low probability score by our Natural Language Processing API.
   Option 2: NLP API to understand the user's message and also have a column in our database to store frequently sent messages apart from the one related to the functionality so that we update our chatbot to handle such messages from the user.

   Decision: We decided to respond with a standardized reply for any message that the bot can't understand. There are way too many options to consider if we try to understand what a user is saying. Also, there is a high chance of getting spam messages. So, we decided to keep it simple and reply with a default message such as "Sorry, I am not able to understand you. Do you want help?".

2. How should users access extra features like help and feedback?
   Option 1: Have a dedicated menu associated with all external functionalities.
   Option 2: Have the user send their commands as just another message.
   Decision: Having the user send their commands(help/feedback) as another message complements our bots functionality as being the user's companion. 'help' would redirect the users to the FAQ page which provides a comprehensive list of everything the bot can do. Entering 'feedback' would instruct the users to be able to submit constructive feedback in a comprehensible format.

3. How to display dining courts to the user?

   Option 1: Display no image and just provide a text list with number options.
   Option 2: Display images of dining courts with their locations and the option to view their menus.

   Decision: The best choice would be the option 2, as it would make the user familiar with the dining courts appearance and the food available there to make an informed decision.

3.2 Non-Functional Issues

1. What type of database would be best for us?

   Option 1: SQL database type: MySQL
   Option 2: NoSQL database type: Firebase

Decision: We will use a MySQL database because the database allows safe storage of user details. Also, a MySQL database can be easily implemented with Flask. Flask has great object relational mapping(ORM) which will help ease our query.

2. Which framework is the best for our chatbot backend?

   Option 1: Flask - Python Web framework.
   Option 2: Express - Node.js Web framework.

   Decision: Although we have previous experience with Node.js and are more comfortable with Express framework, we decided to use Flask as our framework. As our Natural Language Processing model code base is being implemented in Python, we chose to use Flask as our framework because it allows us to keep our entire backend in Python instead of switching between two different languages.

3. How are we going to host our backend services?

   Option 1: Amazon Web Services.
   Option 2: Heroku

   Decision: Heroku is accessible by any language and Python is one of the many frameworks that Heroku supports. Heroku is a hosting service built for students and learners with free models available to be able to use most features without restrictions. It is easier and cleaner to set up and is built for smaller and scalable applications. AWS on the other hand is a resource intensive hosting platform that provides much more than we need at this point. Although AWS is better prepared for handling resource intensive tasks, Heroku is a better fit for the services we are going to provide through our bot.

4. Which NLP API would be the best to have a conversational chat-bot?

   Option 1: Recast.ai
   Option 2: Wit.ai
   Option 3: Api.ai

   Decision: Recast.ai is an extremely powerful API that makes it easier for developers to train their bots on different sentences and possible interactions. It's overall design makes it faster to set up different intents and work on their analysis. The API is also free to use, while being extremely robust and fast, thus making it the obvious choice.

5. How will we be collecting user feedback from our chat bot?

   Option 1: Ask user after every conversation snippet to rate their messages.
   Option 2: Check their usage statistics and see what a user prefers.

   Decision: Rather than prompting the user to give feedback every single time, we believe that making assumptions on their texts based on their usage would give us a better idea of what features are preferred by the users.

4 Design Details

4.1 Database Layer

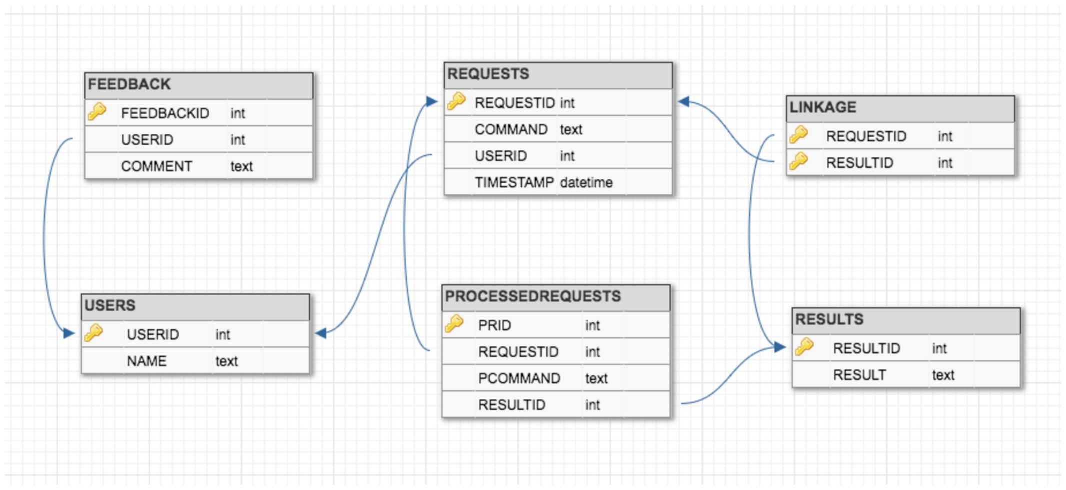


Figure 2: Database Schema

The above diagram illustrates our database mockup. Things may change over time, but we believe that our current schema is a best fit for our requirements.

Users:

- Every user entry corresponds to a Facebook messenger user interacting with BoilerBot.
- USERID is a unique integer created for each user that works as the Primary Key.
- The NAME field stores each user's name to create the conversational aspect.
- User table will be linked with the Feedback table and the Requests table
  - If the User gives an input that is an information request, then correspondence will occur with the Requests table.
  - If the User prompts the feedback command and gives their command in the feeback format, then correspondence with the Feedback table will commence.

Feedback:

- Represents the feedback for the features provided by BoilerBot.
- Contains the feedback given by the user that is sent as a text message.
- FEEDBACKID works as the Primary Key and USERID is the Foreign Key.

Requests:

- Represents the message sent by the user.
- REQUESTID is a unique integer generated for each message that works as the Primary Key.
- USERID acts as a Foreign Key for the Request table.
- Timestamp field contains the received date-time of the message.

Processed Requests:

- Represents the intent of the messages sent by the user.
- Saves intent of requests parsed by Recast.ai.
- PRID is the unique Process Request ID that works as the Primary Key.
- REQUESTID is the foreign key for the Processed Requests table.

Results:

- Represents the result from processed requests.
- RESULTID is a unique integer generated for each result.

Linkage:

- REQUESTID and RESULTID together make the Primary Key.
- Represents the table that links request and results to their respective queries.

4.2 Classes and their Relations

The main reason we chose Flask is because of its elegant Object Relational Mapping (ORM). This ORM means that the class structure is very similar to the way the database handles our information. So the various one-to-many and many-to-one relations have already been described in the database details. However, there are some additional classes we have added to help our product be more robust and thus, improve codebase quality. The objects are described below:

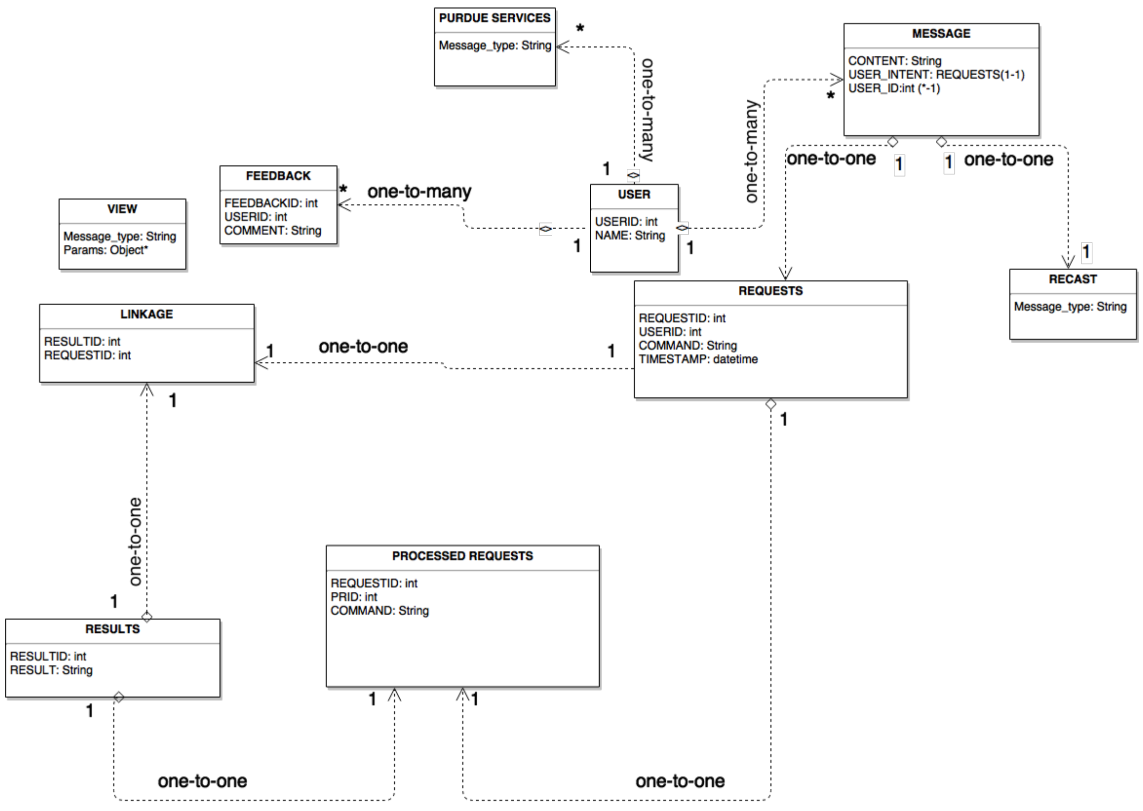


Figure 3: Detailed Schema Relations

User
The User table contains the information of the user like user name and an auto-generated integer user ID. From this table, we have three one-to-many relations. The Message, Purdue Services and Feedback table originate from the User table and the User table has the following fields:

- USERID (one-to-many)
- Name i.e. String

Message
It will store the probability that the user is trying to perform a query. This is important due to the way Natural Language has multiple interpretations. We can have a simple check in the future to only store relevant messages that our NLP API is confident about. It will have the following fields:

- Content String
- User_intent Requests (one-to-one)
- User_id id (many-to-one)

Purdue Services
This table contains the information that we collect by scraping the Crime Statistics webpage, as well as details of Purdue Services by using API calls. It has the following fields:

- Returns the Message_type String (many-to-one)

Feedback
The feedback table contains the texts that the user sends BoilerBot in a particularly formatted way. This information is then accessed by the developer team for future releases. It has the following fields:

- FEEDBACKID (many-to-one)
- USERID which is an integer
- COMMENT i.e. String

Requests
The request object is designed to abstract the details as well as provide some basic helper functions. It is designed to encapsulate a single interaction of text that a user sends to our chatbot. It will have the following fields:

- REQUESTID (one-to-one)
- COMMAND i.e. String
- USERID (one-to-one)
- Timestamp

Processed Requests
The processed request object contains the parsed information from the Natural Language Processing API. We parse the message into multiple entities by using Recast.ai and then save the intent of the message in this table. It will have the following fields:

- REQUESTID (one-to-one)
- PRID i.e. Processed Request ID (one-to-one)

- COMMAND i.e. String

Results
The processed request contains the intent of the message. Then, this message is parsed by algorithms on our side and our team saves the results in this table. It will have the following fields:

- RESULTID (one-to-one)
- RESULT i.e. String

Recast
A Recast object is the object that connects to our selected NLP API (Recast.ai). It's main function is to take in input from the message object, connect to the API to process the message text and return intent and entity attributes to the message object. Our Recast object would have the following fields:

- Message_type String (one-to-one)
- Returns String (intent) and Array (entities)

Linkage
This table updates the REQUESTID and RESULTID and this change shows up in the Results table and Requests table. Linkage table has the following fields:

- REQUESTID (one-to-one)
- RESULTID (one-to-one)

View
Our chatbot views are designed to be very modular. We can send various types of views ranging from simple text messages to multi-card galleries for the dining courts. In order to create a general view class that can have subclasses for various purposes, we have written a view class with parameters for an array of objects, and a message type. It's various field and types are listed below:

- Message_type String
- Params Object*

4.3 Sequence Diagrams

The following sequence diagrams detail out the steps of the bot-user interaction.

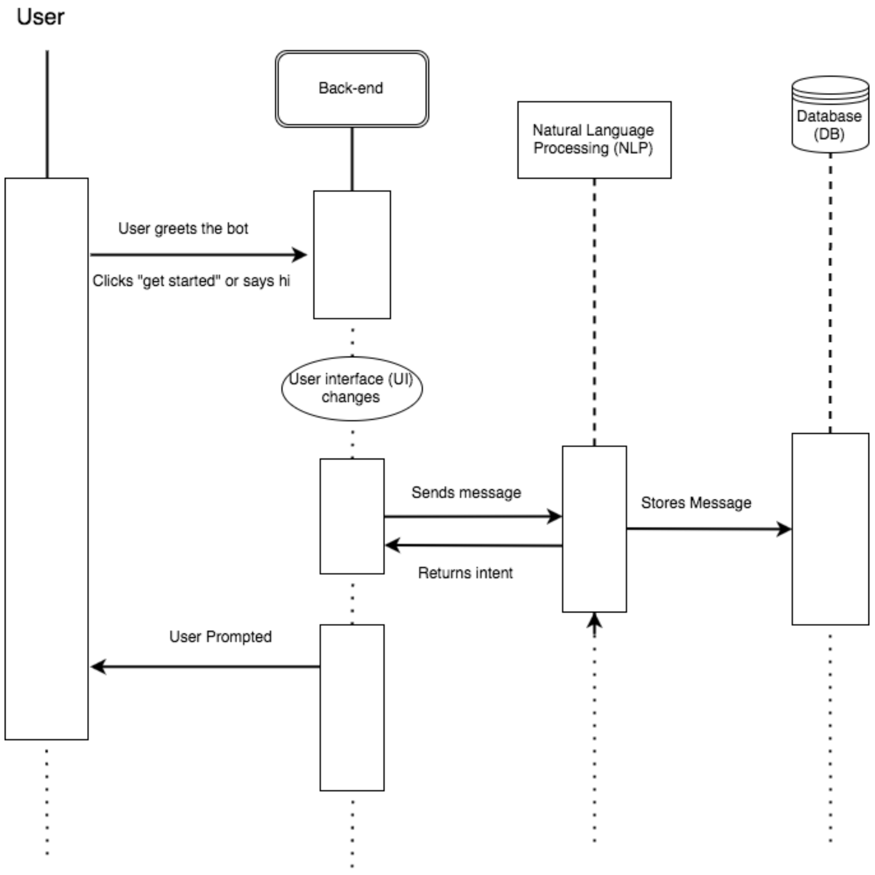


Figure 4: User sends message to BoilerBot

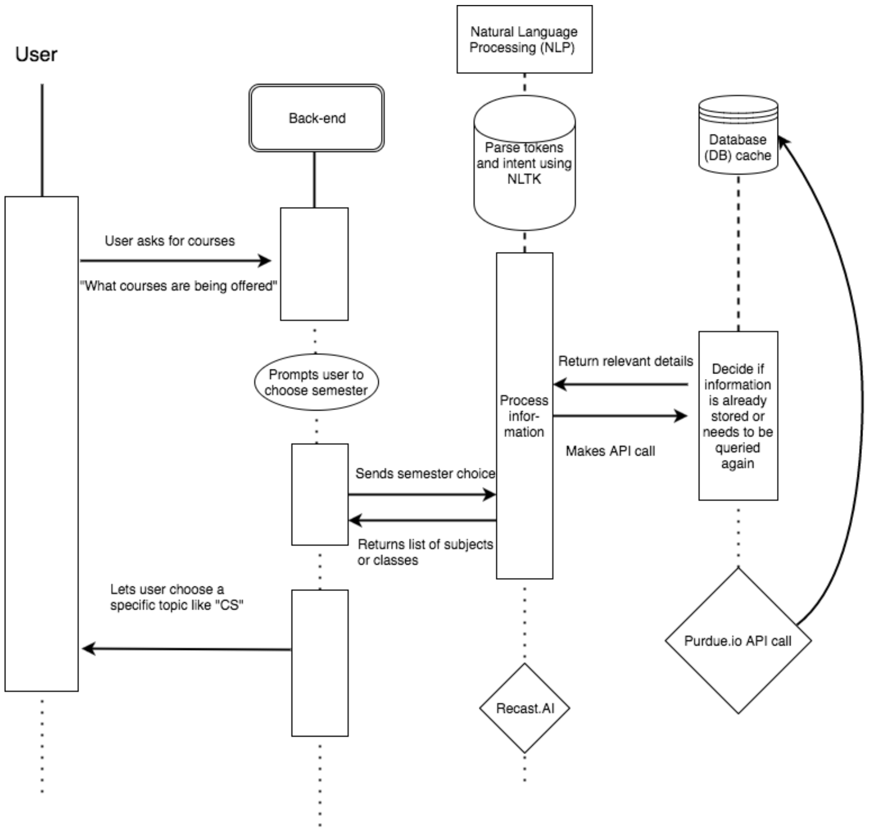

User
Natural Language Processing (NLP)
Back-end
Parse tokens and intent using NLTK
Database (DB) cache
User asks for courses
"What courses are being offered"
Return relevant details
Prompts user to choose semester
Process infor- mation
Decide if information is already stored or needs to be queried again
Makes API call
Sends semester choice
Returns list of subjects or classes
Lets user choose a specific topic like "CS"
Purdue.io API call
Recast.AI

Figure 5: User requests for courses from BoilerBot

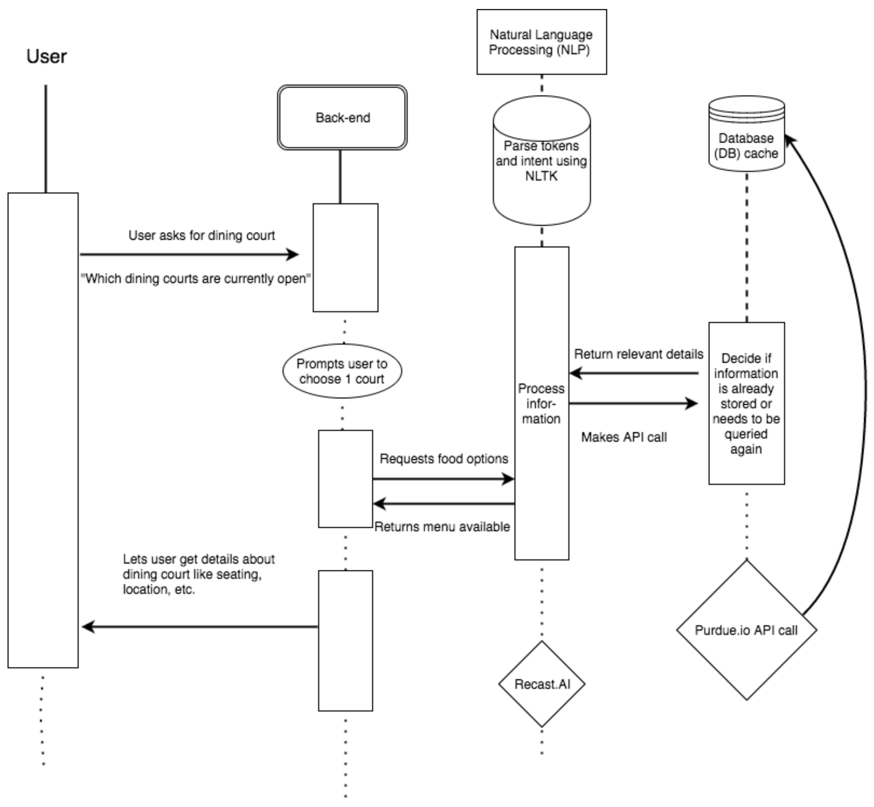
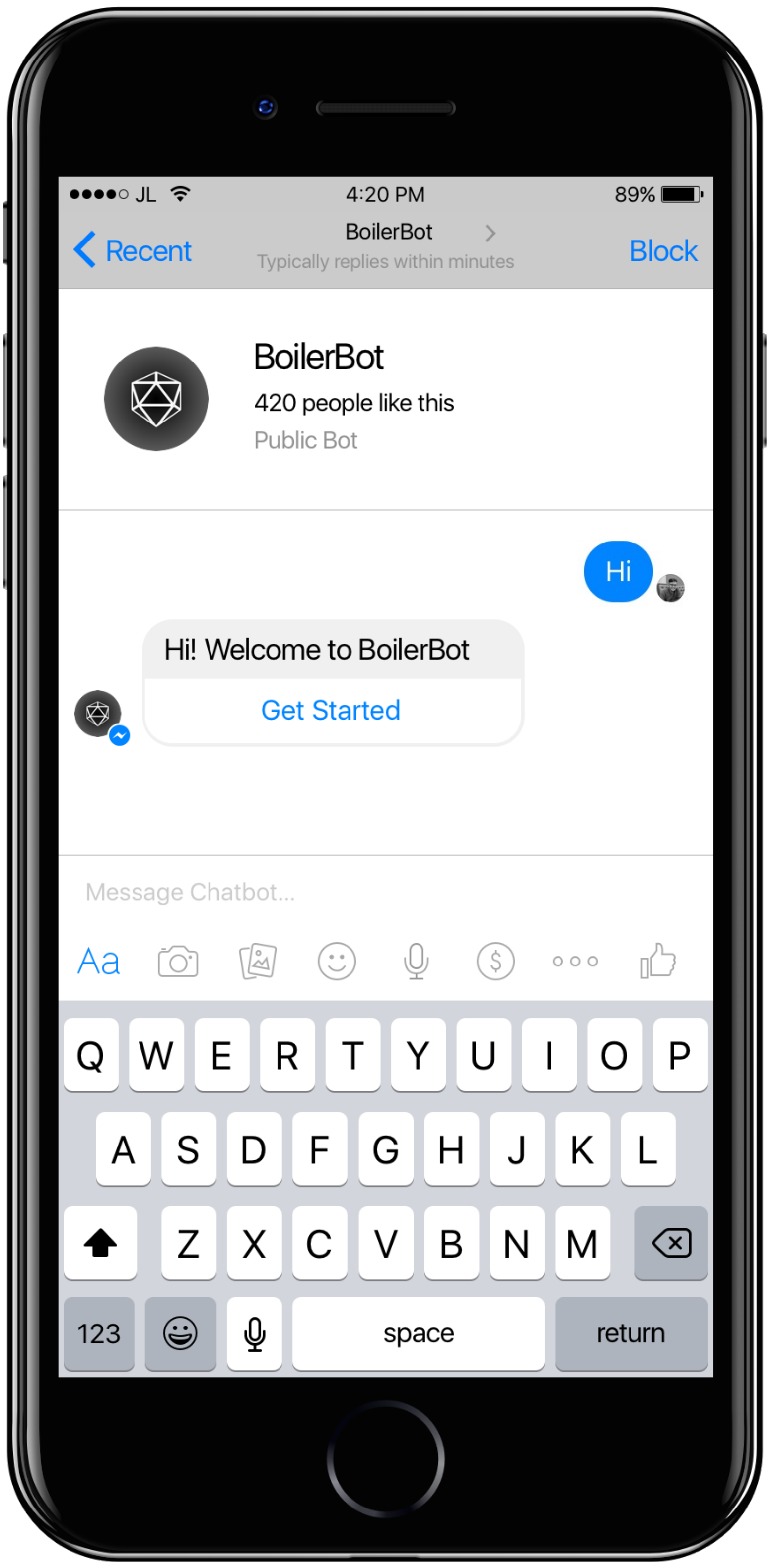

User
Back-end
Natural Language Processing (NLP)
Parse tokens and intent using NLTK
Database (DB) cache
User asks for dining court
"Which dining courts are currently open"
Prompts user to choose 1 court
Return relevant details
Process information
Makes API call
Decide if information is already stored or needs to be queried again
Requests food options
Returns menu available
Lets user get details about dining court like seating, location, etc.
Purdue.io API call
Recast.AI

Figure 6: User requests for dining options from BoilerBot

4.4 UI Mockups

The following mockups detail out the various functionalities of BoilerBot.



BoilerBot

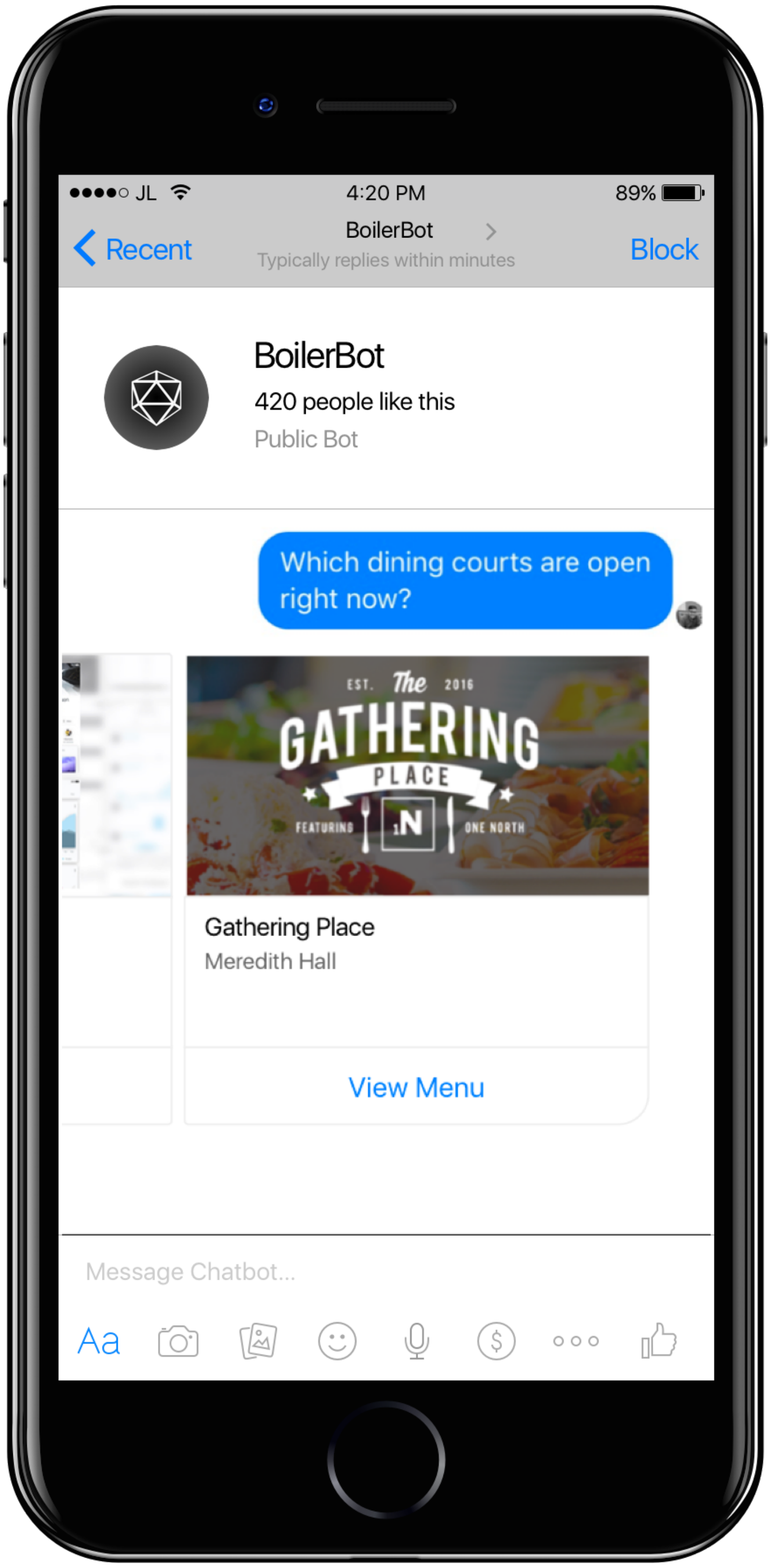
420 people like this

Public Bot

Hi

Hi! Welcome to BoilerBot

Figure 7: User and Bot first-time interaction


●●●●○ JL 4:20 PM 89%

Recent | BoilerBot | Block

Typically replies within minutes

BoilerBot

420 people like this

Public Bot

Which dining courts are open right now?

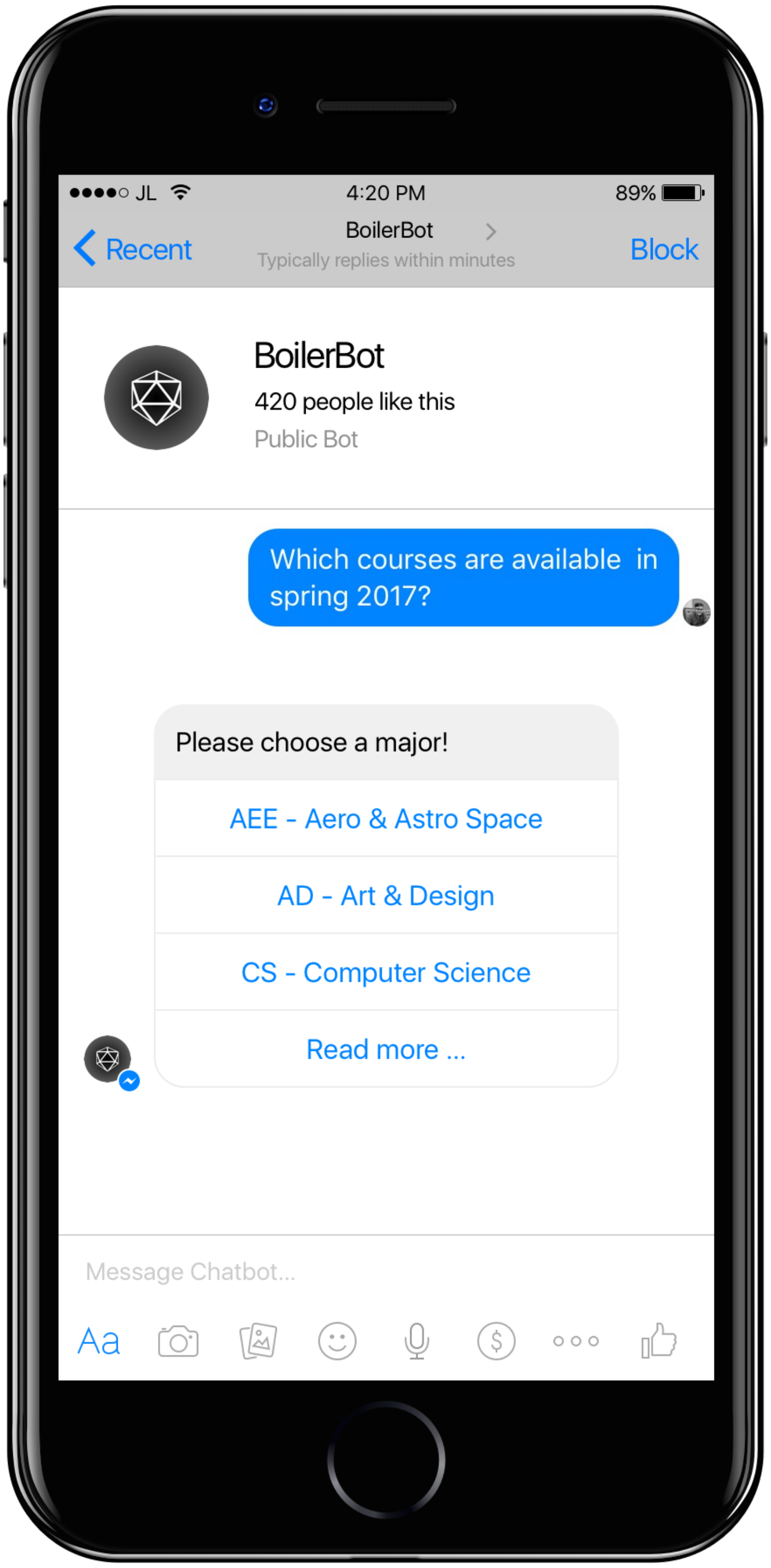
Gathering Place

Meredith Hall

View Menu

Message Chatbot...

Figure 8: User requests for dining courts from BoilerBot



●●●●○ JL 4:20 PM 89%

Recent BoilerBot Typically replies within minutes Block

BoilerBot

420 people like this

Public Bot

Which courses are available in spring 2017?

Please choose a major!

AEE - Aero & Astro Space

AD - Art & Design

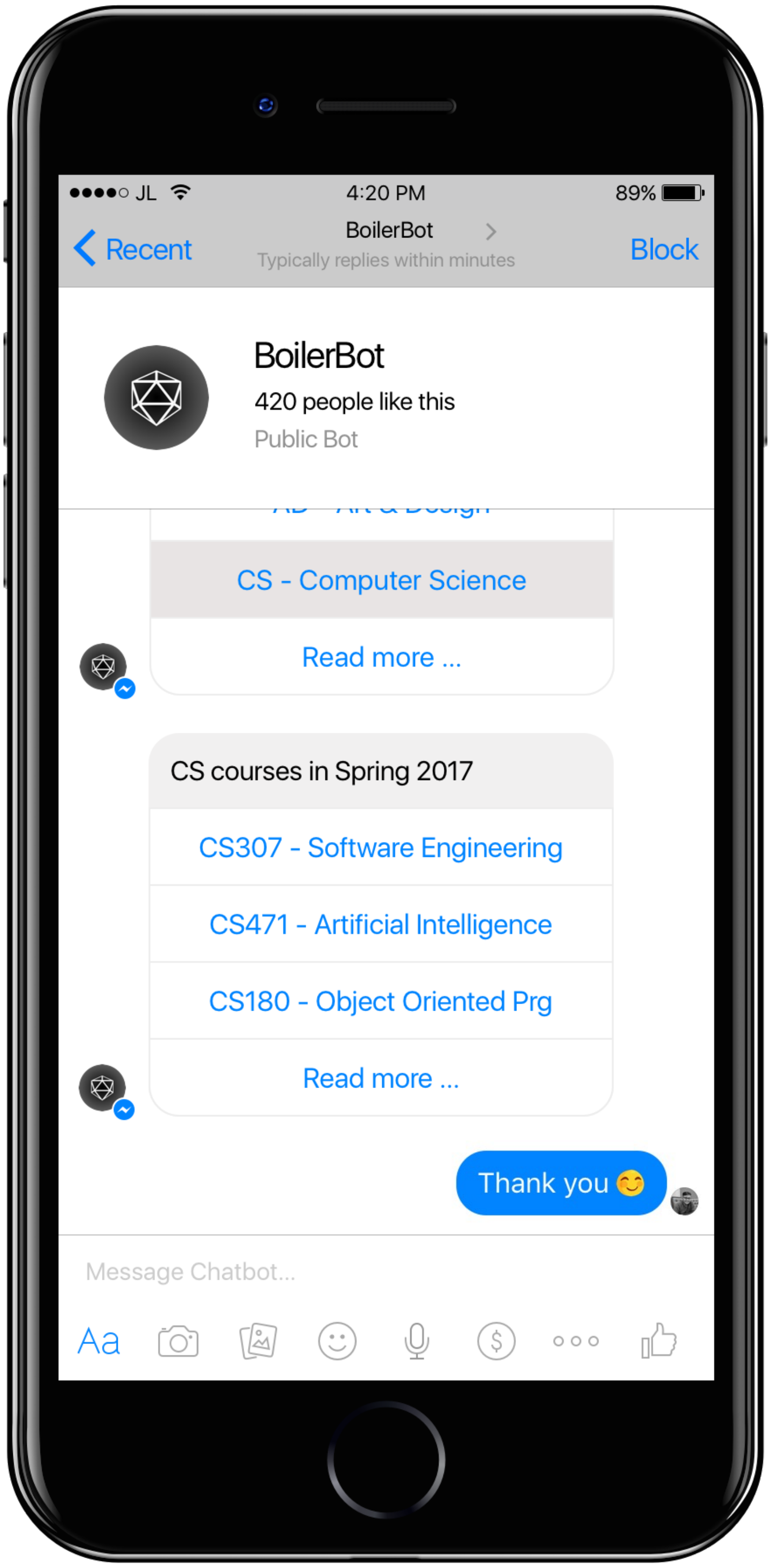
CS - Computer Science

Read more ...

Message Chatbot...

Aa

Figure 9: User requests for courses from BoilerBot


●●●●○ JL

4:20 PM

89%

Recent

BoilerBot

Typically replies within minutes

Block

BoilerBot

420 people like this

Public Bot

CS - Computer Science

Read more ...

CS courses in Spring 2017

CS307 - Software Engineering

CS471 - Artificial Intelligence

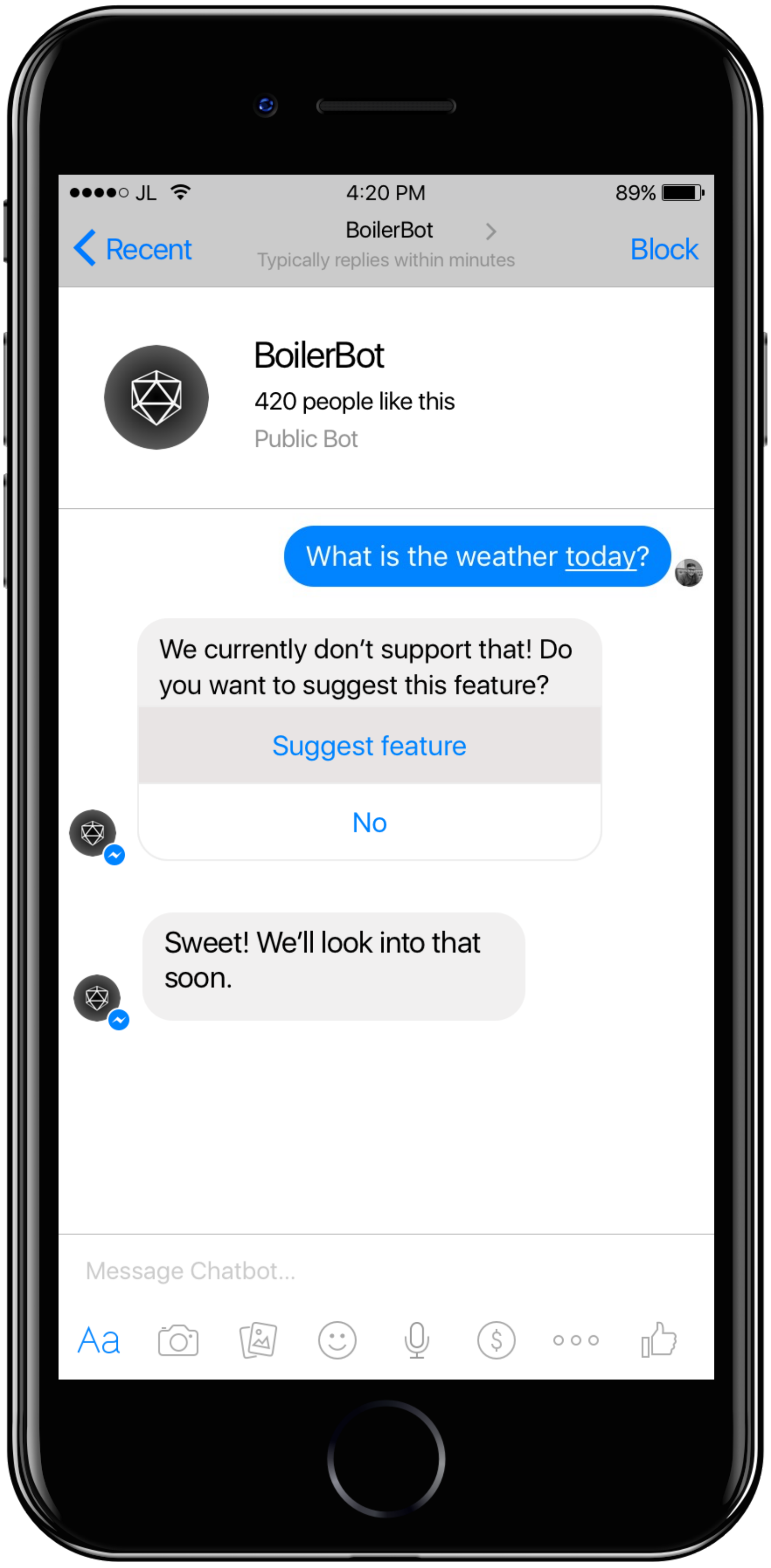
CS180 - Object Oriented Prg

Read more ...

Thank you 😊

Message Chatbot...

Figure 10: Bot provides details for specific courses



●●●●○ JL 4:20 PM 89%

Recent BoilerBot Typically replies within minutes Block

BoilerBot

420 people like this

Public Bot

What is the weather today?

We currently don't support that! Do you want to suggest this feature?

Suggest feature

No

Sweet! We'll look into that soon.

Message Chatbot...

Aa

Figure 11: User submits feature request to bot