WolfPlanner: Personalized Task Scheduling Bot

Neel Kapadia
North Carolina State University
Raleigh, NC, US
ntkapadi@ncsu.edu

Rohan Chandavarkar
North Carolina State University
Raleigh, NC, US
rgchanda@ncsu.edu

Sainag Shetty
North Carolina State University
Raleigh, NC, US
sgshetty@ncsu.edu

Rohit Naik
North Carolina State University
Raleigh, NC, US
rtnaik@ncsu.edu

ABSTRACT

With the number of tasks increasing day-by-day, it is becoming difficult for students to manage so many tasks considering the varying amount of time available and required to complete them. Students have to manage many tasks such as attend classes, complete assignments before deadlines, work on projects and attend meetings, prepare for interviews. With such a large number of tasks lined up for the student to do in a fixed time frame, it becomes very hard for the student to come up with an ideal time table which would help the student complete all the activities and also maintain its quality to an optimum level. It requires continuous and active effort by the student to create weekly schedules leading to wastage of valuable time. Manual effort in scheduling might lead to poor planning which in turn will result in excessive workloads.

In today's age where time is a critical asset for a student's success, we have come with an idea for replacing and reducing human effort by a bot which generates a personalized schedule for the student. The bot makes a schedule for the students taking into account their deadlines and dependencies between tasks. WolfPlanner, an interactive chatbot specifically made for NC State students generates a schedule for the student based on upcoming assignments, projects, interviews, events, etc (information about which will be given to the system by the student). Furthermore, to keep up with the current technology trend of interactive assistants, we have integrated this service with Slack thereby avoiding downloading of stand-alone application. For now, we restrict our domain of users to students for ease of data collection and implementation.

Keywords

Scheduler, Chatbot, Calendar, Time-table, Planner, Slack

1. INTRODUCTION

In this paper, we discuss WolfPlanner - a personalized task scheduling bot for students, which generates a weekly schedule of tasks and guarantees completion of tasks within the respective deadlines without compromising on the quality of work.

In order to get the best out of their education, it is important that students focus on important tasks such as their coursework and studies and not spend time on lesser menial tasks like creating a time-table. The best way forward is to automate such tasks and WolfPlanner does exactly the same. Today, it is a world of personal interactive assistants and according to Gartnerś report[1], 85% of customer interactions will be managed without a human by 2020. So we have developed a chatbot that is able to assist the user in scheduling a wide of range of day-to-day tasks for a student which includes include classes, assignments, project meetings, discussions, readings to name a few. The application is a chatbot interface which will take in all of these conditions as input from students to help them manage their time efficiently. While generating the task schedule, we have also taken care of time spent in activities like sleep time, eating time, buffer time between two tasks such as commuting time from classroom to the destination where a project meet is scheduled. The bot will also give the student a free time slot in which the student can do any other activity which he wishes to do.

In universities, students are involved in many group activities, and scheduling these meetups are often challenging due to conflicting schedules. They take up a lot of time and lead to hassles. So, having a collaborative approach is needed. The bot is of a great help as it has the data of the weekly schedule of all its users. The schedule generated will help the students come up with a suitable time for arranging meetings. All user will know their free slots and can agree on a common time accordingly.

The motive of our work was to develop a chatbot with which a user can communicate, provide data, ask for change in schedule, etc. thus saving time for schedule generation. A chatbot feature also guarantees ease of use of application.The bot generates schedules on a weekly or biweekly basis as per user preference. The user requests a set of tasks every week and the priority is set according to deadline. As per the user preference, the bot will generate an appropriate schedule taking into account all of the details. Further, this generated schedule may be exported to Google Calendar for easy access across all the devices.

We have hosted our database on Mlab.com where the user data is stored on MongoDB. Mlab provides scalability and reliability to our system, and security to users' data. The user details are stored as objects on the NoSQL MongoDB environment making it convenient to access the details of

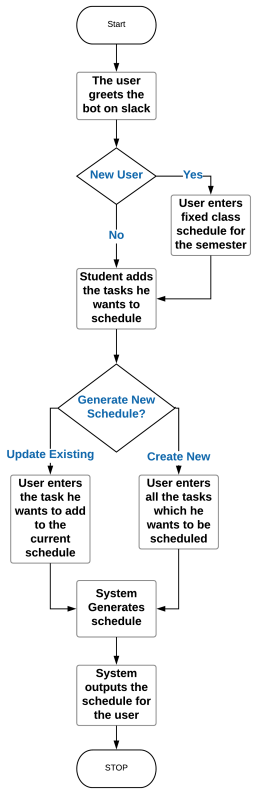

Figure 1: Activity Diagram

the user who is logged in without any requirement of costly join operations which would have been needed in case of relational databases. Also storing user data in tables might lead to redundancy and lot of NULL data. Hence, we decided to use MongoDB as our database system.

To provide a gist of what the bot does:

- Confirms unity id of a new user to check whether the user is a NCSU student.
- Gives options of commands to the user such as add courses, add tasks, view courses, view tasks, fetch schedule.
- Provides a dialog to the user to enter details while adding a course/task.
- Displays added courses and tasks when the user enters the appropriate commands.
- Generates schedule using the scheduling algorithm and display it to the user.
- Add the schedule to google calendar.

2. MOTIVATION

A major problem in student's life is managing homeworks, projects, interviews, meetings etc. and still get some free time. Our study revealed that majority of survey respondents (87%) want an application/bot which generates a schedule for them. However, a lot of people (67%) don't use any calendar for organizing their day. This shows that people wanted an application for a functionality which other calendars do not provide. Hence we came up with the idea to fulfil the user's requirement by creating an interactive bot which generates schedules for the user and also organizes meeting times for them. This will help the users manage their time efficiently and work productively.

3. LITERATURE REVIEW

Despite the success of calendar tools in organizing and tracking activities, a lot of time is spent by people on managing their time properly. Berry et al.[2] came up with PTime, a personalized time management agent which reduces this problem of time management considerably, when it comes to arranging meetings. PTime schedules meetings based on factors like participants, time and locations. Our bot will go a step further and automatically schedule each and every task of a user along with the meetings.

The biggest challenges in any task planner are the constraints in schedules and which scheduling method is used.

The performance of a scheduling algorithm depends on the following:

- Constraints in scheduling
- Preemptive tasks
- Penalty for errors in scheduling

Traditional algorithms face a lot of problems when it comes to preemptive scheduling. For example, schedule quality of FCFS scheduling algorithm is highly sensitive to the preemption strategy used. Schwiegelshohn et al.[3] demonstrated that by doing certain improvements in FCFS it can be better than any of the other preemptive algorithms but the parameter settings and efficiency depends heavily on the workload. As our task scheduler deals with multiple tasks which can arrive at any time, using FCFS might lead to compromising efficient task scheduling as our workload itself is dynamic in nature.

After reviewing some papers we found that the algorithm implemented by L. J. Wilkerson et al.[4] will be an effective algorithm for our task scheduler. We then compared this algorithm with First Fit algorithm[5] and found that First Fit Algorithm will be the most suitable algorithm for our application.

4. IMPLEMENTATION

4.1 Software Engineering Methodologies

4.1.1 Agile Methodology

We used Agile methodology for development of our application.So the features were built initially as small incremental features, and then the final one was built on top of those. We made use of the storyboard feature provided by github to form issues and then resolve them.

4.1.2 Code Review

Code Review practices were followed by creating different branches in github other than master and working on those before pushing on master.In this way, we managed to reduce bugs and error in our code and code was reviewed by the members before commiting to the master.

4.2 Working of Bot

Steps:

1. Authenticate the slack user using the OAuth handler of Botkit and Slack

   1.1 When a user starts communication with the bot, the user is first authenticated. Since, we have used the BotKit toolkit, it provides a handler which helps us authenticate the user.
   1.2 Slack sends a OAuth request '/oauth' which is caught at our server and the user is verified.

2. For every new user create a new document in the MongoDB MLab collection

   2.1 When the user enters on the Slack interface, Slack sends POST to the server at '/slack/receive'. These are events(message.im and message.channel) which the bot has subscribed to. So whenever there is a message posted our server is notified.
   2.2 The user is checked if an entry is already present in the MLab 'student' collection. If the user if present then the user details for the further steps.
   2.3 If the user document is not present, then create one. Initial entry into the user document is the user's slackID and the user's unityID.

3. Get the user input(either Tasks or Courses) and store them into the user's Mlab entry.

   3.1 When the user requests to add tasks/courses, Slack sends a HTTP POST to our server at '/message'. These are classified as Interactive Messages.
   3.2 The bot sends a request to open up a dialog using the dialog.open Slack API.
   3.3 The user fills up the necessary details and this is received by the bot controller. The bot connects to the MLab using mongoose, and the document is updated.

4. When 'fetch schedule' command is invoked, the bot sends the user identification details to the scheduling algorithm.

   4.1 When the user requests for fetch schedule, the bot fetches the user's unityID and generates a day-date dictionary which is the current week's dates(according to which the schedule is to be generated)
   4.2 Now, the bot uses the python-shell package to call the core-scheduling algorithm, passing the above variables as arguments to it.
   4.3 The algorithm working is described below. The output of the algorithm is a generated schedule which is stored in the respective user document.

5. The bot displays the generated schedule on the Slack Interface.

   5.1 Once the schedule is generated, the bot fetches the schedule from the MLab user document.
   5.2 This fetched scheduled is formatted in a way which is easily understandable to the user and the bot displays this formatted schedule.

4.2.1 Scheduling Algorithm

The algorithm is as follows:

1. Get student data from MLab mongoDB database

   1.1 This step uses NoSQL find query to retrieve the data object for the student. The unique identifier student id is received as a command line argument to the function.
   1.2 The data is returned as a JSON object. We use the python json library to parse this object and make it compatible for use in python code.

2. Check if the data object contains a 'freeTime' attribute

   2.1 This step uses NoSQL find query to retrieve the data object for the student. The unique identifier student id is received as a command line argument to the function.

3. If no, then generate 'freeTime' and update the corresponding student in database

   3.1 The freeTime array is generated once per semester per user. Basically, it contains the time(s) of the day when the student is 'free' i.e. has no lectures, no predefined fix task. For example if the student has lectures on monday from 8am-10am and 12pm-2pm, then the freeTime array contains 10am-12pm and 2pm-10pm.
   3.2 We also introduce a user-input for buffer i.e. if he wants to keep some buffer time before/after lectures for travelling, relaxation etc.

4. Generate the schedule using the information in the data object.

   4.1 This step generates the schedule for the student. Basically it checks the freeTime array and the set of tasks from the data object and assigns tasks to freeTime slots linearly. The tasks are sorted based on deadline and duration so that the algorithm schedules in such a way that the chance of missing the deadline is minimized.
   4.2 After sorting, we use the first-fit algorithm to generate schedule. We fit the tasks in the free slots preemptively until it is completed and then proceed to next task and so on. For example consider freeTime for monday to be 9am-11am, 2pm-5pm and 7pm-10pm
   4.3 The reason for the shift in the original plan of using the algorithm in the paper [4] to using first-fit algorithm [5]:
   1) we didn't have the kind of input data required
   2) due to time constraints we decided to opt out the complex algorithm
   3) we focused on deploying a basic but complete functionality with all the planned features for the 1st phase.

5. Add the generated schedule to the data object and update it in database.

   5.1 In this step we then add the generated schedule to the data object and updates the database.

5. INITIAL SURVEY

We took a survey to find out how people are tackling issues of time management and whether they need such a product or not. The questions which we asked are as follows:

1. **How often do you face difficulties in completing tasks because of not being able to manage time properly?**

The purpose of this question was to understand if the people face difficulties in time management.

From the survey, the results tells us that there are not many people who do not experience the problem of task completion due to ineffective time management, so our bot might help to solve this problem.

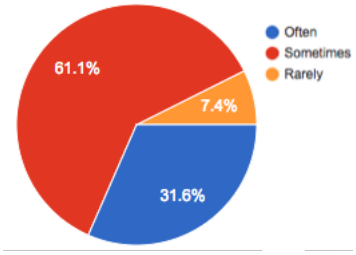

Figure 2: Response for Question 1

2. **Do you face an issue to set up project (or other) meetings due to conflicting schedules?**

   The purpose of this question so that we could evaluate the need of collaborative scheduling methods.

   The results suggest that people do face the issue of setting up meetings as their schedules conflict. Hence, indicating a need of a collaborative personal scheduler.

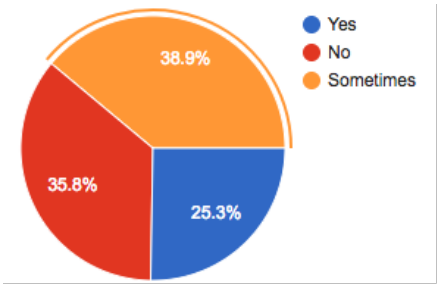

Figure 3: Response for Question 2

3. **How much do you rely on Google Calendar (or similar) for planning your day-to-day tasks?**

   This question was to understand the percentage of people depending on Calendar applications for daily use.

   This result tell us that people do not rely on calendar applications for planning their activities.But the following question (Q4) tells us that they do want a software which makes schedule for them.

4. **Do you feel the need for a software which makes a weekly schedule for you taking into account your pending tasks and preferences?**

   The results of this question showed us that more than three-fourth of the population need an application that helps the user.

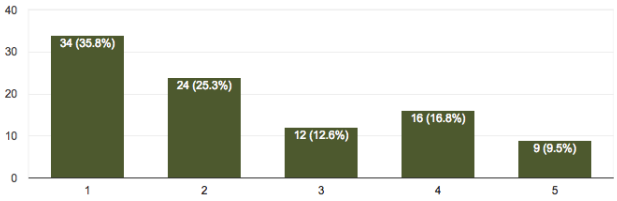

Figure 4: Response for Question 3

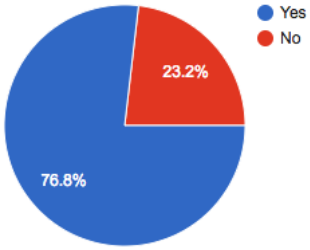

Figure 5: Response for Question 4

Survey Conclusion

Based on the 92 responses we got from our survey we can conclude that a personalized time management bot will be very helpful for the people. Most of the respondents do face a problem of completing tasks efficiently due to lack of proper time management. Using a bot which will schedule tasks for the individual will help with proper management of tasks and also save the user's time as there is no need to schedule tasks manually. The bot will also take care of meeting scheduling which is hassle for some of the respondents. The survey tells us that not many people rely on calendar applications like Google Calendar for task scheduling but would like to use a software like our bot. From this result we can conclude that people will start using softwares similar to calendar applications for task scheduling if there are more important features like dynamic schedule generation .

6. TECHNOLOGIES USED

1. **MongoDB**

   - MongoDB[6] is a NoSQL database platform. It uses JSON-like documents with schemas.
   - We used MongoDB for storing user details and their tasks. We used MongoDB so that we get the advantages of document oriented storage and indexing on any attribute.
   - One document for each user help us cleanly maintain all the information about the users in one place without the need of joins to get the data.
   - Using such a database gave us the advantage of easily adding entries from NodeJS and accessing that entry using a Python script. Schema-less JSON-like documents made it easy for us define relationships between users, courses and tasks.

2. **Mlab.com**

- mLab is a fully connected, managed cloud database that hosts MongoDB databases.
- It has partnered with platform-as-a-service providers and uses AWS, Google and Azure. As we have stored all users' data on MongoDB, we wanted a platform which is reliable and secure. mLab.com took care of the security of the details of the users. Using a platform like mLab.com ensures BASE properties of a NoSQL database, secures data on MongoDB and makes the database available online.

3. **Slack**
   - We used the Slack platform for deploying the Chatbot[7].
   - The main reason was its increasing popularity[8], ease of use for students and unique interface. Students, who are the main users of the application widely use Slack for their classes and project discussions and hence it will be convenient for them to just add this bot to their workspace to generate schedules for them while they manage their own work.
   - We defined a basic interaction interface using the dialogs feature provided by the slack API[9]. Apart from that we used various other features such as -
     - Interactive Component: Used this feature to handle the message buttons, menus, or dialogs
     - Events: Used for handling events such as when the bot is part of the conversations, whenever the bot is part of the event Slack sends a HTTP POST to the url specified.
4. **Node.js**
   - Node.js is an event driven asynchronous Javascript runtime[10]. Node allows for concurrently handling many connections.
   - We have used Node.js to run the entire SlackBot mainly because of the availability of BotKit - an open-source SDK for developing bots[11] available in Node.js.
   - We have used the tools provided in BotKit to handle the interactions with the bot. We created a BotKit controller which connects to Slack and provides core features and functionality.
   - Initially, we had decided to use DialogFlow to handle conversations with the bot. However, we did not find it absolutely necessary to add another dependency to the project, to handle tasks which could be done with BotKit.
5. **Python**
   - Python is powerful and fast, plays well with others, runs everywhere, is friendly & easy to learn, is open source[12]. Python provide plethora of libraries which can be used to solve almost every problem. Also, it is an easy to use and easy to learn language.
   - The primary reasons for selecting python to develop the core algorithm were: 1) suitable libraries available 2) team comfort with python 3) easy compatibility with Node.js.
   - We used this language for implementing our core scheduling algorithm, for the API calls (Google Calendar and Node.js), for establishing connection with mLab MongoDB Database and to make any updates if required to the database.
6. **Google Calender API**
   - The Google Calendar API [13] lets developers add the functionality of linking their applications, with Google Calendars of the users of their application. This gives the developers a great tool to make their application much more productive by letting user add reminders and events to their calendars which the user can view any time, even if he/she is not using the developer's application.
   - Google Calendar was the obvious choice because of its immense popularity and the number of users that increasingly use this platform. Users will not have to register for any other calendar application but just link their Google account with the bot.
   - We used the API to provide an option to the users for adding their scheduled tasks as events [14] to a widely used calendar service for easy integration between users.

How the technologies interact with each other -

- The Slack bot has been developed mainly in Node.js.
- Whenever a new user enters the unity ID, a document for that user is created.
- The input which the user adds via the bot is stored in MongoDB documents.
- When the user calls the fetch schedule functionality, the python code for scheduling is called and a field for schedule is added in the document of that user. The bot then retrieves it from the document and displays the schedule on the slack bot.

7. FEATURES

1. **User Details Retrieval**

   We fetch details based on the Unity ID mentioned by the user when he/she first uses the chatbot, so that we do not need to ask him/her to enter the details already available on the university portal. It is easy to retrieve these details by making a request call to the University Campus Directory. If the fetch is not successful then an error is returned indicating that the unityID is wrong. If the fetch is successful, then the student details are parsed from the HTML Page.

2. **New User Entry**

   When a new user interacts with our bot for the first time, the bot asks for the unity id. The bot then confirms unity id and goes ahead if the unity id is correct.

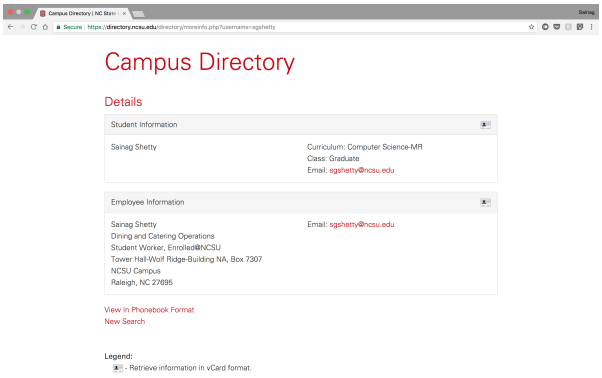

Figure 6: User Details Retrieval

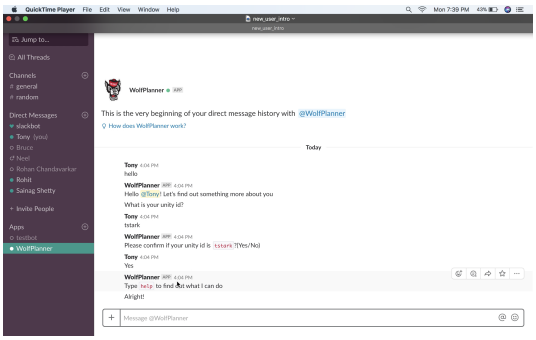

Figure 7: New User Entry

3. **Add Courses**

When the user types 'add courses' a new dialog will open up with a form. The student can put the details of the courses from that form and submit it. Details such as course number, course name, days of week and timings can be entered. The added courses are added in the MongoDB database.

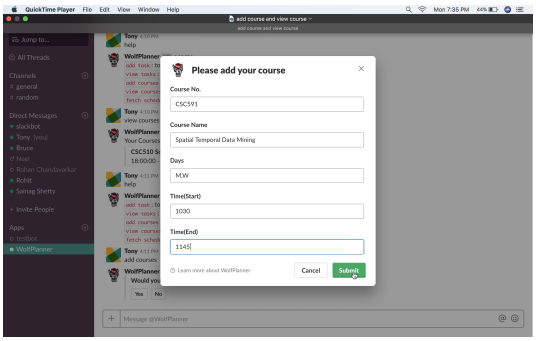

Figure 8: Add courses

4. **View Courses**

When the user types 'view courses', all the courses of the user are retrieved from the database. A list of all courses added is displayed along with the days and timings.

5. **Add Tasks**

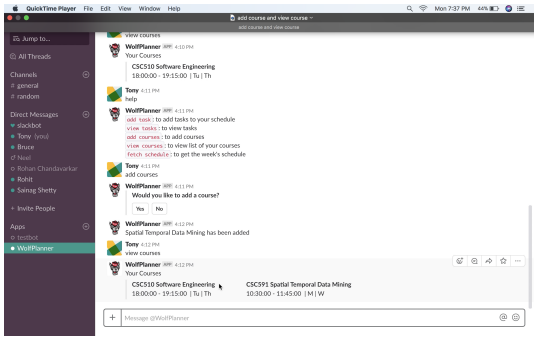

Figure 9: View courses

When the user types 'add task' a new dialog will open up with a form. The student can put the details of the task from that form and submit it. Details such as task name, task type, task duration and task deadline can be entered from that from. . The added tasks are added in the MongoDB database.

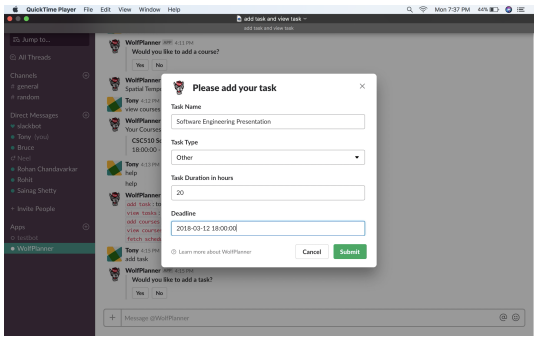

Figure 10: Add Tasks

6. **View Tasks**

When the user types 'view tasks', all the tasks of the user are retrieved from the database. A list of all tasks added is displayed along with the hours scheduled for the task.

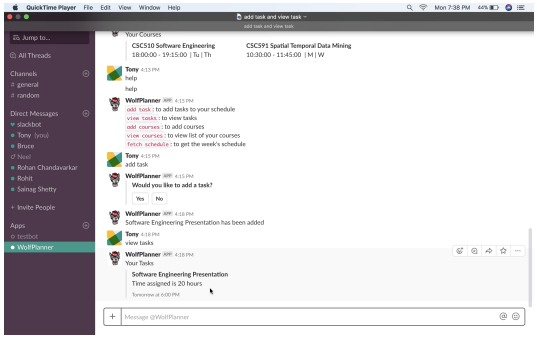

Figure 11: View Tasks

7. **Fetch Schedule**

When the user types 'view tasks', all the tasks of the user are retrieved from the database. A list of all tasks

added is displayed along with the hours scheduled for the task.

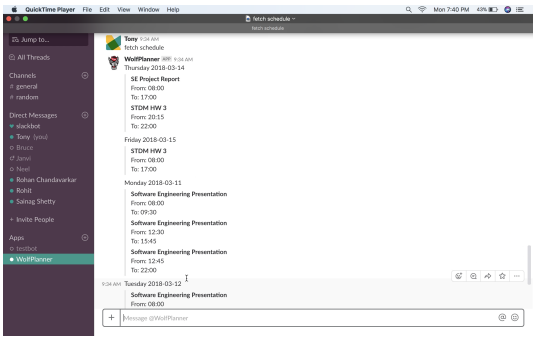

Figure 12: Fetch Schedule

8. EVALUATION

8.1 Evaluation Method

We used Beta Testing[15] as a evaluation measure for our system. Beta testing is also sometimes referred to as user acceptance testing (UAT) or end user testing. In this phase of software development, applications are subjected to real world testing by the intended audience for the software. The experiences of the early users are forwarded back to the developers who make final changes before releasing the software commercially.

We invited users to test our application. All the testers, received an invite to the Slack Channel which hosts our WolfPlanner Bot, along with a readme containing instructions and a Google evaluation form. They were asked to follow the steps described in the readme on the WolfPlanner bot. Once they completed testing, testers filled the evaluation form which asked some basic questions about the functioning of the application. Questions were based on ease of use, correctness of results, understandability of the README etc. The evaluation form and responses are covered in detail in the next section.

8.2 Evaluation Questions and Results

The questions which we asked for our evaluation are as follows:

1. **Rate the ease of use of our application.**

   We had asked the respondents to rate the ease of use. The ease of interaction and accessibility of the system. The results showed that most of the testers found it easy to use and interact.

2. **The content of the README was enough for me to understand the system.**

   We had provided the testers with a file containing instructions to follow for effective evaluation of the project. The main purpose of this question was to understand if those instructions were good enough for the testers to understand the system.

   From the results we obtained, it was clear that the README was well described so that any user can understand the system.

Test name	Test description
Add User	Is profile created for a new user
Data Persistence	Once user inputs data, it stays persistent
Add courses	Is user able to add courses
Add tasks	Is user able to add tasks
View Courses	Is user able to view list of courses he has entered
View Tasks	Is user able to view list of tasks he has entered
Fetch Schedule	Is user able to request a schedule
Correctness of schedule	Is user satisfied with the schedule generated

All the above test cases passed and ran successfully.

Figure 13: Summary of Tests

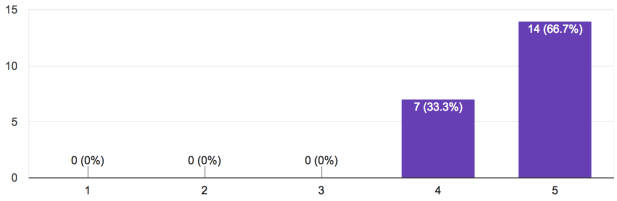

Figure 14: Response for Question 1

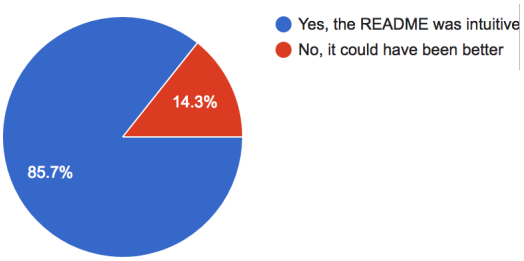

Figure 15: Response for Question 2

3. **Were you able to understand the general flow of the system?**

   We wanted to make sure that the testers could understand the general flow of the system i.e start with adding their unityID, followed by the tasks and courses, and finally fetching the schedule. The question was aimed at understanding if this requirement was satisfied.

   So, after the evaluation it could be seen that most of the testers had a grasp of the flow of the system.

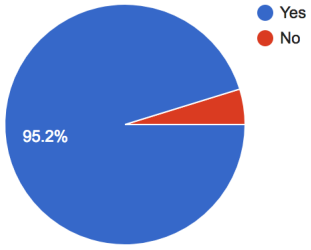

Figure 16: Response for Question 3

4. **Do you think the schedule generated by the bot is feasible and can be followed?**

   The results of this question answered a very important requirement of the system. The requirement being that whether the final schedule generated was feasible or not.

   Most of the testers felt that the schedule which was generated was feasible.

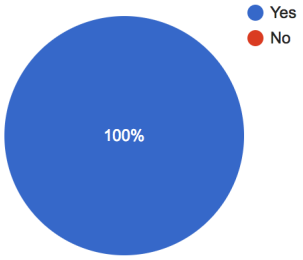

Figure 17: Response for Question 4

5. **Do you think you will use this bot for making schedules for you?**

   This question was to know if people would use the system in the future probably as a weekly scheduler. The respondents have replied on a positive note, showing that many of them will use the WolfPlanner bot for their schedule generation.

   Most of the testers felt that the schedule which was generated was feasible.

We also recieved feedback regarding the issues faced and improvements suggested as discussed in the following subsections.

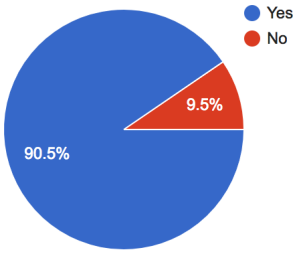

Figure 18: Response for Question 5

8.3 Issues Faced and Solutions

1. **The application allowed the users to add any dates in the field for deadline (for user tasks), without any control over the dates they are entering.**

   Solution: The field for entering dates was a text field and the placeholder had the date format in which we wanted the user to enter the date. The dialogs in Slack API did not allow for calendar input and hence we had the option of either text or dropdown dates only (years, months, days) [9]. We initially made a text field and were aware that this situation might arise. So, we had made a dropdown option (a better version) of the deadline field which we have released together along with few other fixes after the evaluations.

2. **The application allows the user to use 'fetch schedule' feature without adding tasks or courses.**

   Solution: The commands for performing tasks have not been linked with each other. Hence, the order of the functionalities has not been defined in the flow of the application, but has been mentioned in the documentation of the application (README file). If someone misses this detail in the documentation, there is a chance of confusion and possible error in output. We have made the fix that does not allow the user to fetch schedule in a similar way as we have implemented for view tasks vs add tasks (If you try to view tasks without adding, the bot will tell you that give you an option to add tasks and will not give an error/no output) and view courses vs add courses, and have released it along with few other fixes.

3. **The format in which deadlines are to be added is difficult to remember.**

   Solution: The format for adding deadlines is a little complicated (YYYY-MM-DD HH:MM:SS) which becomes difficult for a new user to remember if it is not in front of his/her eyes while entering it. We have added the example format along with the 'Deadline' title in addition to the one in the placeholder of the field to make it easier for the user.

4. **I added M,w instead of M,W and it did not give W in the resulting course. But it works if I put small and capital letters for other days.**

   Solution: When the user entered M, w while defining tasks, only M got stored and w was not recognized

by the bot. On the other hand, small letters worked for all other days and all combinations were tested and were working fine. This was because of a small error in which we had missed adding a condition for detecting 'w' in the regular expressions for the input from this field. This was something we missed accidentally but then immediately fixed it.

5. **The users had to use the ĥelp ćommand continuously to know the exact keywords to enter.**

   Solution: This issue arose because of slight inconsistency in the names of the keywords to be added for functionality. Some keywords had a singular noun such as ädd taskbut similar keywords had a plural noun such as ádd courses. This confused the user slightly and hence had to refer to the help command continuously. This was a mistake on our part and we have rectified it and made the commands consistent to avoid such confusion.

6. **Duration of tasks might not be known always.**

   Solution: The add task module requires the user to input the number of hours expected to be needed to complete that particular task. This is necessary because the number of hours of each task are a deciding factor for generating a schedule for that user. But, it is difficult for the user to know this before hand and it may change eventually. To solve this problem, we plan to add a modify task module which will allow the user to change the duration of the task according to new knowledge so that he/she can make a better estimate. We can also implement some techniques from the future scope to solve this problem.

7. **Some users could not understand which task to perform at first and the flow of tasks.**

   Solution: The README provides the steps to be followed to use the bot. But, if someone does not go through it, then it might be difficult for them to know what needs to be done to start conversing with the bot. We have added a 'Hello!' command from the bot whenever a user starts the application, so that there is a starting point for the conversation, which can then continue easily.

8.4 Improvements suggested and comments

1. **Add a way to make collaborative meetings by asking the application to take at other students schedules.**

   a. The users wanted a feature which let them schedule team meetings.
   b. Our initial idea was implementing this feature. But we had to prioritize our tasks i.e. have a bot which first made schedules for an individual and fix any issues that may arise at this stage. However, it could be added as a future improvement.
   c. This feature can be added by asking the user for a list of people who will be part of the meeting. By having this input, the scheduling algorithm can access all of their schedules and find free times where the said meeting could occur.

2. **Improve on the interface making it more intuitive and have a different approach to asking the user to enter the tasks/courses.**

   a. A user can interact with an application in different ways, such as a Web Page, Form-entry, etc. Of the different methods that we could think of, interaction with the bot was found to be most intuitive.
   b.Although, taking in specific inputs such as task or course input, could be improved by linking the student to the university database which enables the bot to fetch the course and assignment details directly.

3. **Displaying the schedule in a better manner rather than a text-table form. Also, allowing to add the schedule to my calendar.**

   a. Now the application uses a text-table based form to print the student's schedule with every day as a separate output along with the day's tasks.
   b. We were considering an approach which integrates with the Google Calendar [13] to display the schedule. Using this approach will allow the user to link his/her Google Calendar with the bot. By doing this, it will make it easier to export the generated schedule.

4. **Could have added a sample example besides deadlines instead of just as a placeholder to make it easier for the user.**

   a. When the user fills in the data, the examples are put in as placeholders. For ex, Deadline field has a placeholder: 2018-03-12 09:00:00. However, the placeholder disappears when the user starts entering in the field. Hence, making it difficult for the user.
   b. A possible solution to this is to add the example along with the header rather than just as a placeholder. So, the example will always be visible for the user to refer to.

5. **Some form of control which will make sure I have everything entered correctly before I try to fetch schedule.**

   a. There is requirement to validate the contents of the input before passing the data to the database. Such as the deadline may be entered in a different manner rather than the required one and thus causing error in generating the schedule.
   b. This validation could be done by restricting the user to a choice from which the option can be chosen. It could also be done by enforcing a method which checks if the input is done correctly and only then allowing the user to submit the data entered.

9. STEPS FOR FUTURE CONTRIBUTION

To continue development of WolfPlanner, there are a few things that need to be done:

1. Install **Python 3.6** - `https://www.python.org/downloads/`
2. Install **Node.js** - `https://nodejs.org/en/download/package-manager/`
3. Clone the WolfPlanner repository - **git clone** `https://github.com/neelkapadia/WolfPlanner.git`

4. Make sure you have the correct version of pip for Python 3.6. (Ideally pip is bundled with Python 3.4+ so it should be available directly).

5. Install the required packages using **pip install <package-name>**.

6. Create a **.env** file (if not present) which store all the access codes and environment details.

7. Install **Ngrok** - `https://ngrok.com/download`

8. Run a local server for development by running the command - **nrgok http 3000**. Specify the url obtained, in the slack app api page(Event Subscription, Interactive Components, and OAuth & Permissions) for receiving HTTP POST requests.

9. Do a **npm install & npm start** from the WolfPlanner directory to install all the packages required by Node.js.

After this, the bot is ready to use.

For testing individual files locally, steps 1-5 are sufficient. If you want to deploy the application, and test the entire bot flow, you can run steps 6-8 as well.

10. FUTURE SCOPE

1. **Integrating with the University Portal**

   Using the University portal, will save the students' time and effort. As the user will not have to enter details such as Courses, Exams and Assignment schedules. By integrating the application with the Portal will enable the Bot to fetch all of these details automatically.

2. **Using Machine Learning Approaches**

   The bot can predict time required to complete a certain task by a specific user using Machine Learning approaches and set task duration accordingly. The bot will first learn the behavior of a certain user using initial tasks and then give personalized schedule for subsequent tasks. For example, if a user X takes more than the average hours to complete AI Homework 1, then the bot will assign more time than normal for user X in case of AI Homework 2.

3. **Planning a group meeting**

   The bot has the data of the weekly schedule of all its users. So it can suggest a common free slot so that a meeting cian be organized. It can also include the meeting venue and time needed to reach that venue. If a group has to meet to work on a particular project, this feature will give the group members an idea to plan the stages of the project as they know beforehand when the meetings can be scheduled.

4. **Integrating the bot with Google maps to find time required for commute**

   We can find time required from position A to position B using Google Maps API and the bot can take that time into consideration and schedule tasks if they are at different geographical locations. So if a student has a lecture at location A and then has to attend a project meeting at location B, the task scheduler will consider time to go from A to B and reserve time for it in the schedule.

11. CONCLUSIONS

We have successfully implemented a bot which makes a time-table for the student. The bot takes as input the set of tasks to be scheduled and the set of courses the student has taken. The bot also communicates with the user regarding the action to be performed. User can perform various tasks like add course, add task, view courses, view tasks and fetch schedule. Before fetching schedule, the user has to provide courses and tasks. The user describes what he wants to do, the bot extracts the intent from the user input and takes necessary actions. If required, it makes changes to database and/or calls core modules. If the user asks for a schedule, it calls the scheduling module and returns its output in an user-friendly display.

Finally, the bot also adds the schedule to the student's google calendar (linked with the university email). This helps the user in keeping a track of his tasks efficiently.

12. REFERENCES

[1] Gartner.com, *Gartner Customer 360 Summit 2011.* [Online]. Retrieved from: `http://www.gartner.com/imagesrv/summits/docs/na/customer-360/C360_2011_brochure_FINAL.pdf` [Accessed: 23-Jan-2018].

[2] Berry, Pauline, et al. *Deploying a personalized time management agent.* Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. ACM, 2006.

[3] Schwiegelshohn, Uwe, and Ramin Yahyapour. *Analysis of first-come-first-serve parallel job scheduling.* SODA. Vol. 98. 1998

[4] Wilkerson, L. J., and J. D. Irwin. *An improved method for scheduling independent tasks.* AIIE transactions 3.3 (1971): 239-245.

[5] William Stallings *Operating Systems: Internals and Design Principles 6th* Prentice Hall Press Upper Saddle River, NJ, USA Â©2008 ISBN:0136006329 9780136006329

[6] MongoDB Documentation, Retrieved from: `https://docs.mongodb.com/` [Accessed:26-Jan-2018]

[7] Slack Documentation *Slack Reference Documentation* Retrieved from: `https://api.slack.com/getting-started` [Accessed: 26-Jan-2018]

[8] Department of Product *Why Slack is popular?* Retrieved from: `https://api.slack.com/getting-started`[Accessed:Mar-10-2018]

[9] Slack Documentation *Interacting with users through dialogs.* Retrieved from: `https://api.slack.com/dialogs`[Accessed:Feb-20-2018]

[10] Node.js Documentation *About Node.js* Retrieved from:`https://nodejs.org/en/about/` [Accessed: 5-Mar-2018]

[11] Botkit Documentation *About Botkit* Retrieved

from:https://botkit.ai/docs/ [Accessed: 5-Mar-2018]
[12] Python Documentation *Python* Retrieved from: https://www.python.org/about/ [Accessed: 6-Mar-2018]
[13] Google Calender Documentation *Google Calendar API Quickstart for Node.js* Retrieved from: https://developers.google.com/calendar/quickstart/nodejs [Accessed: 1-Mar-2018]
[14] Google Calender Documentation *Creating Events using Google Calendar API* Retrieved from: https://developers.google.com/calendar/create-events [Accessed: 1-Mar-2018]
[15] TechTarget *What is Beta Testing?* Retrieved from: http://whatis.techtarget.com/definition/beta-test [Accessed: 1-Mar-2018]

13. CHITS

1. MYP
2. KKB
3. AZM
4. QTZ
5. MVR
6. GRC
7. KCA
8. MCI
9. WSJ
10. MRS
11. PEG
12. WUG
13. ZMT
14. XYU
15. BRM
16. NLK
17. PAR
18. ZAR
19. OXH
20. LMX
21. DZJ