International Institute of Information Technology, Bangalore

Software Production Engineering Mini Project

IIIT-B Timetable Manager

Mentor	**Teaching Assistant**
Dr. B Thangaraju	Ansh Goyal

Mudita Baid	Shreya Singh	Atibhi Agrawal
IMT2016038	IMT2016045	IMT2016096

Table of Contents

1. Abstract

This report is a brief summary of our Software Production Engineering mini-project. This talks about the application we developed, the IIIT-B time table manager, which is a platform where students and faculties can visualize and manage their semester's time table. We have explained the features of our application and as well as explained important parts of the code. Moreover, we have explained our SDLC in detail. We have also gone through the tools that we used for setting up our pipeline. Given below are some important links relevant to our project:

1. https://bit.ly/300RRCb - Our Application is deployed here.
2. https://bit.ly/2ABdXRa - GitHub
3. https://dockr.ly/2XRbJow - DockerHub
4. https://bit.ly/3coLcnL - Travis

2. Introduction

As the name suggests, this application intends to provide a time table management system for IIITB students. Presently, we don't have any such tool where we can organise our timetable. For 1st and 2nd year students, it is easy for them to track their timetable as they have fixed timetables and subjects. From 3rd year onwards, we have many electives to choose from, where we have to select courses whose timings are not clashing with any other course. Presently, we first write the subjects manually with their respective timings and then check which subjects are not clashing with each other and then choose whichever elective we want to take. This procedure is quite tedious and time consuming. IIIT-B Timetable Manager makes it easy for students and faculty to check and manage their timetable accordingly.
Also, sometimes, when we want to meet any professor but we don't have any idea where the professor is, we just go and wait for them outside their cabin without knowing when they will be coming back. This takes a lot of time, sometimes without even meeting them. IIIT-B Timetable Manager solves this problem by publicly providing every professor's timetable and venue.

2.1 Plan to add value to the existing solution/your plan to work on this problem.
We got the inspiration from an existing application https://whatslot.metakgp.org/ developed by IIT Kharagpur, which visualizes the timetable of every course at IIT Kharagpur.
IIIT-B Timetable manager is an extension of the Whatslot, where we added the following new features:

1. Hassle-free way of choosing electives: It provides students with the facility of choosing electives by selecting the subjects from the drop-down list. If the student selects any subject whose timing clashes with any of the other selected electives, it gives a warning and displays that particular slot with red color. It then provides an option to deselect any of the selected subjects.
2. Timetable for professors: It provides faculty members and students to search for the timetable of any professor.
3. Downloading an ics file: It provides faculty and students to download their timetable in .ics format and then adding it to the Google calendar or their respective calendars.
4. Exporting timetable directly to the calendars.

3. System Configuration

```
Operating System : Ubuntu 18.04
Kernel:  5.3.0-53-generic
RAM: 4GB RAM
```

4. Continuous Integration and Deployment (CI/CD)

Continuous Integration or CI is when a developer's code is merged to the main branch multiple times a day. A build is created for each commit, tests are run to validate each commit before merging. This allows the developers to build products faster and more efficiently. Continuous Delivery makes sure that the company can release software easily whenever they want.

Here is a brief summary of our CI/CD process :

1. Created GitHub Repository and used Git for SCM.
2. Used Pytest to write tests for our flask application,
3. Configures **Travis CI** to build and test our commits.
4. Created a **docker image** and pushed docker image to **DockerHub**.
5. Finally, deploy docker image to **Heroku**.
6. Our Travis CI builds the flask application, runs automated tests in pytest, creates a Docker image using the DockerFile, pushes the image to DockerHub, and deploys docker on Heroku. This process is done for every commit that is merged to master branch as well as for every Pull Request.
7. Monitoring and Logging with **ELK stack**.

5. Continuous Integration and Deployment Tool - Travis

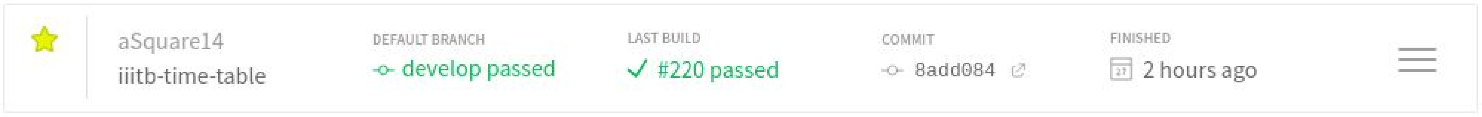
We used Travis for our CI/CD pipeline. Travis CI, Circle CI and Jenkins are some of the most popular CI tools. We decided to choose Travis CI for our application because of the following reasons :

- Jenkins makes sense when any project is hosted in house and it requires a lot of configuration, preparation of jobs etc.
- Our project was hosted on GitHub and Travis is very simple to set up as it requires just a travis.yml file. After creating travis.yml, we need to log in to travis-ci.com and activate our repository. Travis CI is free for open source.

- Moreover, we had used Jenkins and Rundeck in our past assignment and we wanted to explore something new.

5.1 Setting up Travis

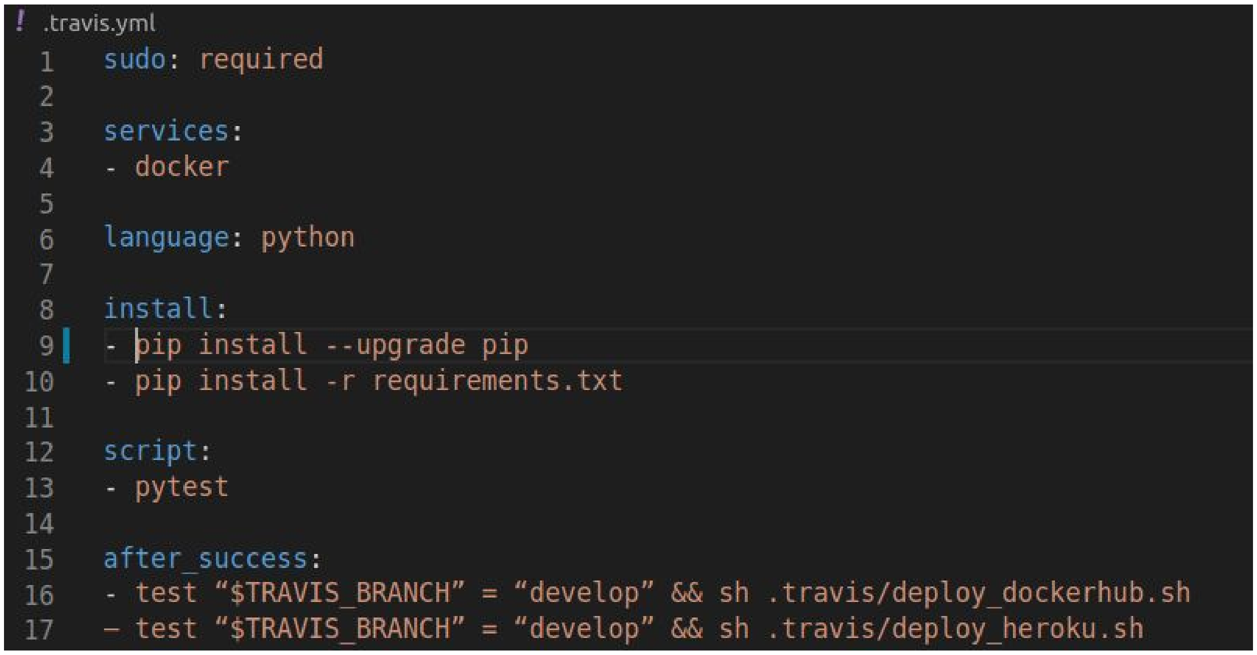
1. We go to travis-ci.com and then log in using our GitHub account. There we can see a list of all our public GitHub repositories. We turn on the switch next to our repository to enable Travis for that repository. Figure 1 below shows us how it is visible.



aSquare14 iiitb-time-table	DEFAULT BRANCH	LAST BUILD	COMMIT	FINISHED
	develop passed	✓ #220 passed	8add084	2 hours ago

Figure1. Travis-CI Repository

2. After activating our repository, we add a .travis.yml file to our code. We will be explaining each section of the .travis.yml in detail later on. In the figure 2 below you can see the travis.yml



```
.travis.yml
sudo: required

services:
- docker

language: python

install:
- pip install --upgrade pip
- pip install -r requirements.txt

script:
- pytest

after_success:
- test “$TRAVIS_BRANCH” = “develop” && sh .travis/deploy_dockerhub.sh
– test “$TRAVIS_BRANCH” = “develop” && sh .travis/deploy_heroku.sh
```

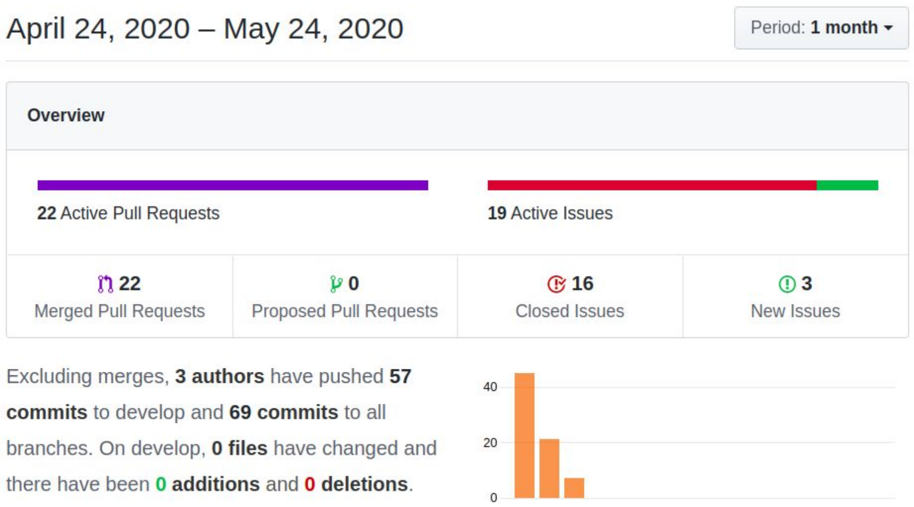
Figure 2. .travis.yml

3. We use pip for dependency management and for that we need a requirements.txt file. This file lists all packages needed by our application. Since we are using pip for

dependency management, the requirements.txt can be generated as follows : pip3 freeze > requirements.txt

6. Source Code Management

Source Code Management is a component of software configuration management. It is also known as version control(VCS) and helps in the management of changes to documents, computer programs and large websites. Git is the most commonly used VCS. We used git extensively in the development of our software. We have also followed the good practices of SCM by tracking our issues, testing before merging any Pull Request, using different branches etc. In figure 3 below you can see the insights from our github repository.


April 24, 2020 – May 24, 2020

Period: **1 month**

Overview

22 Active Pull Requests

19 Active Issues

22 Merged Pull Requests

0 Proposed Pull Requests

16 Closed Issues

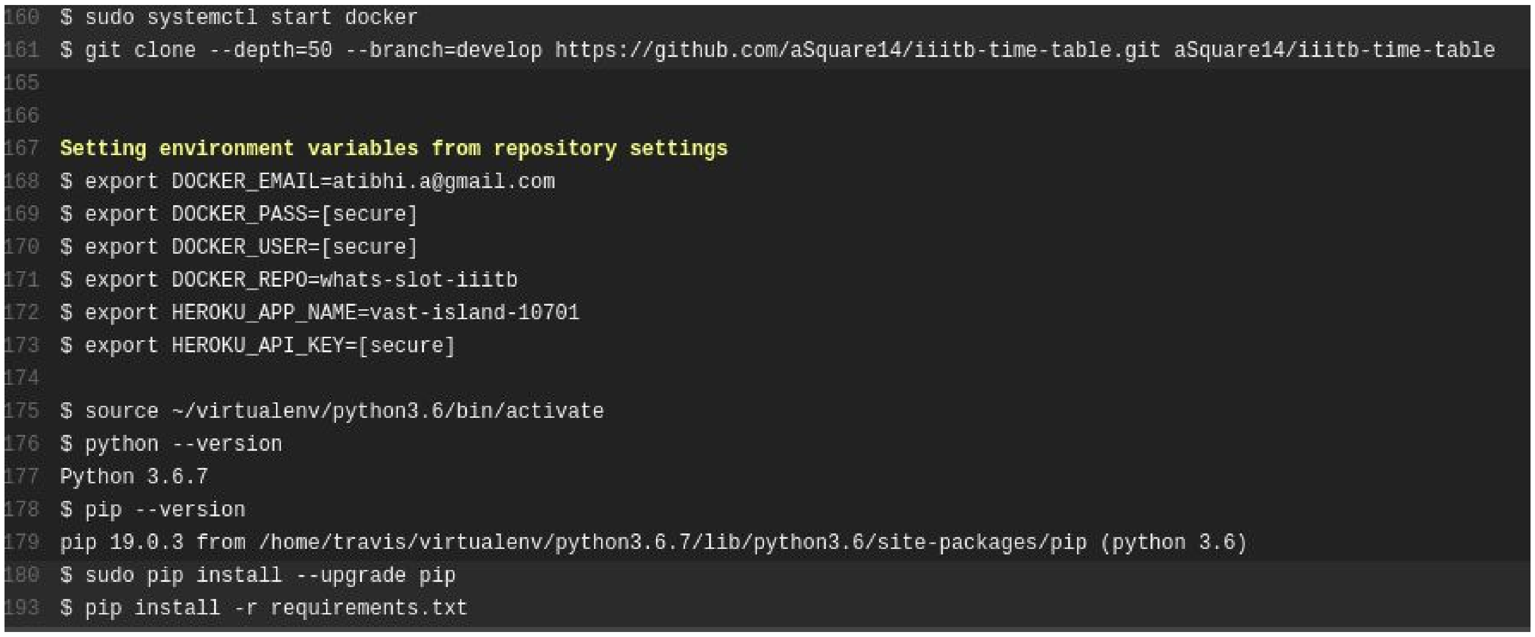
3 New Issues

Excluding merges, **3 authors** have pushed **57 commits** to develop and **69 commits** to all branches. On develop, **0 files** have changed and there have been **0 additions** and **0 deletions**.

Figure 3. Insights from our github repository

7. Build

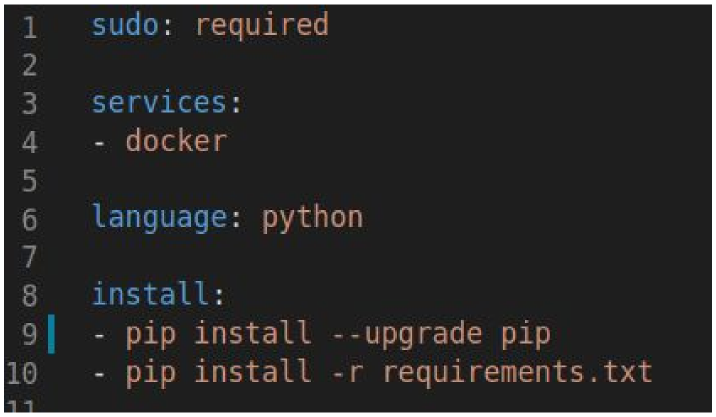
The install section in the .travis.yml takes care of building the project. It installs the dependencies from requirements.txt and uses virtualenv for managing the python environment. As you can see, Travis clones the GitHub repository (line 161), exports the ENVIRONMENT variable (lines 167-173), creates virtualenv (line 175), installs dependencies (line 179). This is how Travis builds the project.



```
$ sudo systemctl start docker
$ git clone --depth=50 --branch=develop https://github.com/aSquare14/iiitb-time-table.git aSquare14/iiitb-time-table


Setting environment variables from repository settings
$ export DOCKER_EMAIL=atibhi.a@gmail.com
$ export DOCKER_PASS=[secure]
$ export DOCKER_USER=[secure]
$ export DOCKER_REPO=whats-slot-iiitb
$ export HEROKU_APP_NAME=vast-island-10701
$ export HEROKU_API_KEY=[secure]

$ source ~/virtualenv/python3.6/bin/activate
$ python --version
Python 3.6.7
$ pip --version
pip 19.0.3 from /home/travis/virtualenv/python3.6.7/lib/python3.6/site-packages/pip (python 3.6)
$ sudo pip install --upgrade pip
$ pip install -r requirements.txt
```

Figure 4. Image from Travis CI pipeline - Building the project

For the build part our .travis.yml has the following section (as shown in figure 5 below).



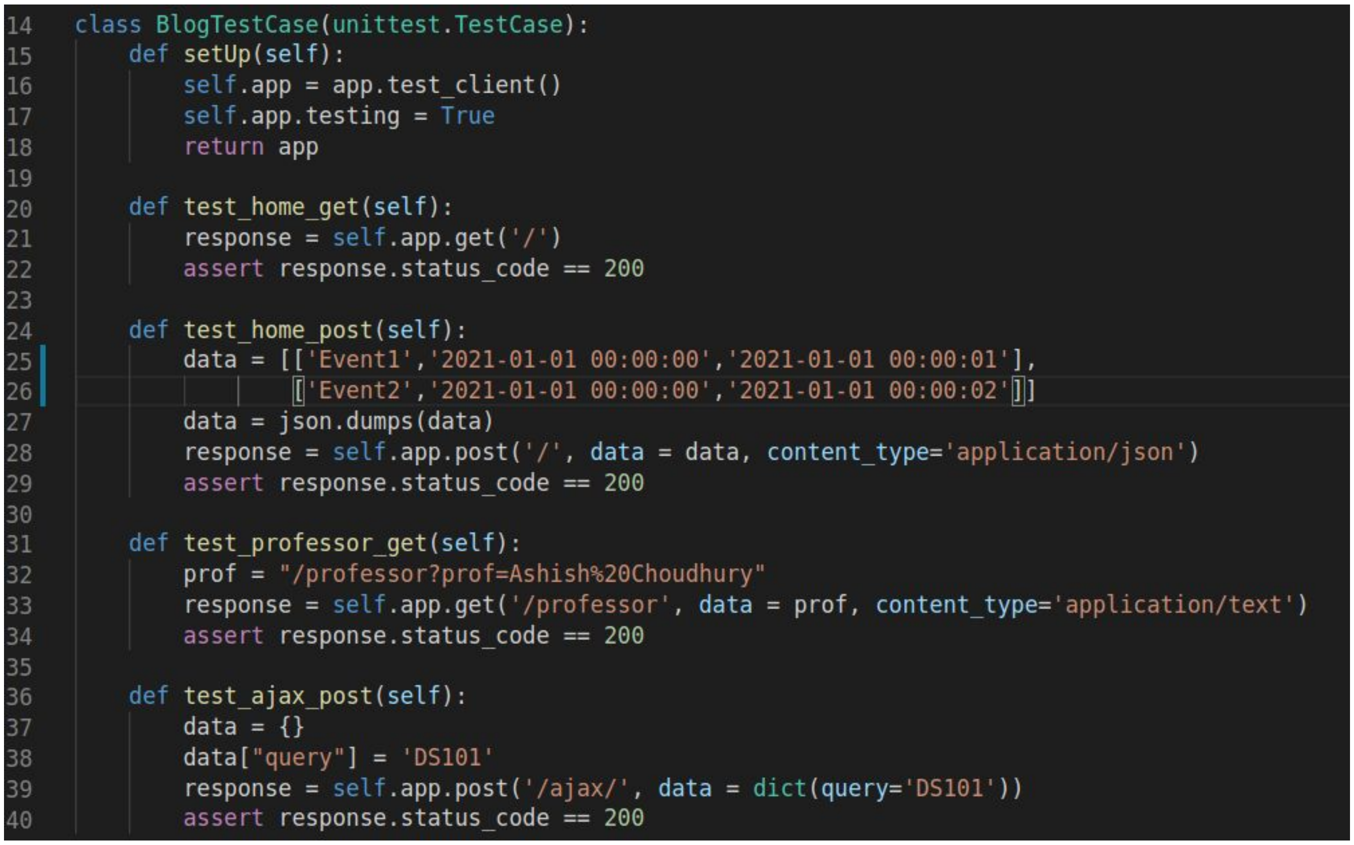
```yaml
sudo: required

services:
- docker

language: python

install:
- pip install --upgrade pip
- pip install -r requirements.txt
```

Figure 5. Build Section from travis.yml

8. Test

We have written tests using pytest for this project. The code snippet can be found in figure 6 below. Pytest helps us write small tests, yet scales to support complex functional testing for applications.
The setup() function (line 15) helps us define our test client.

Then we have defined a function for each route and its respective methods, are sending dummy data when required as an argument, and testing if the status code returning is 200 or not, as a HTTP status code of 200 means that the request has succeeded.

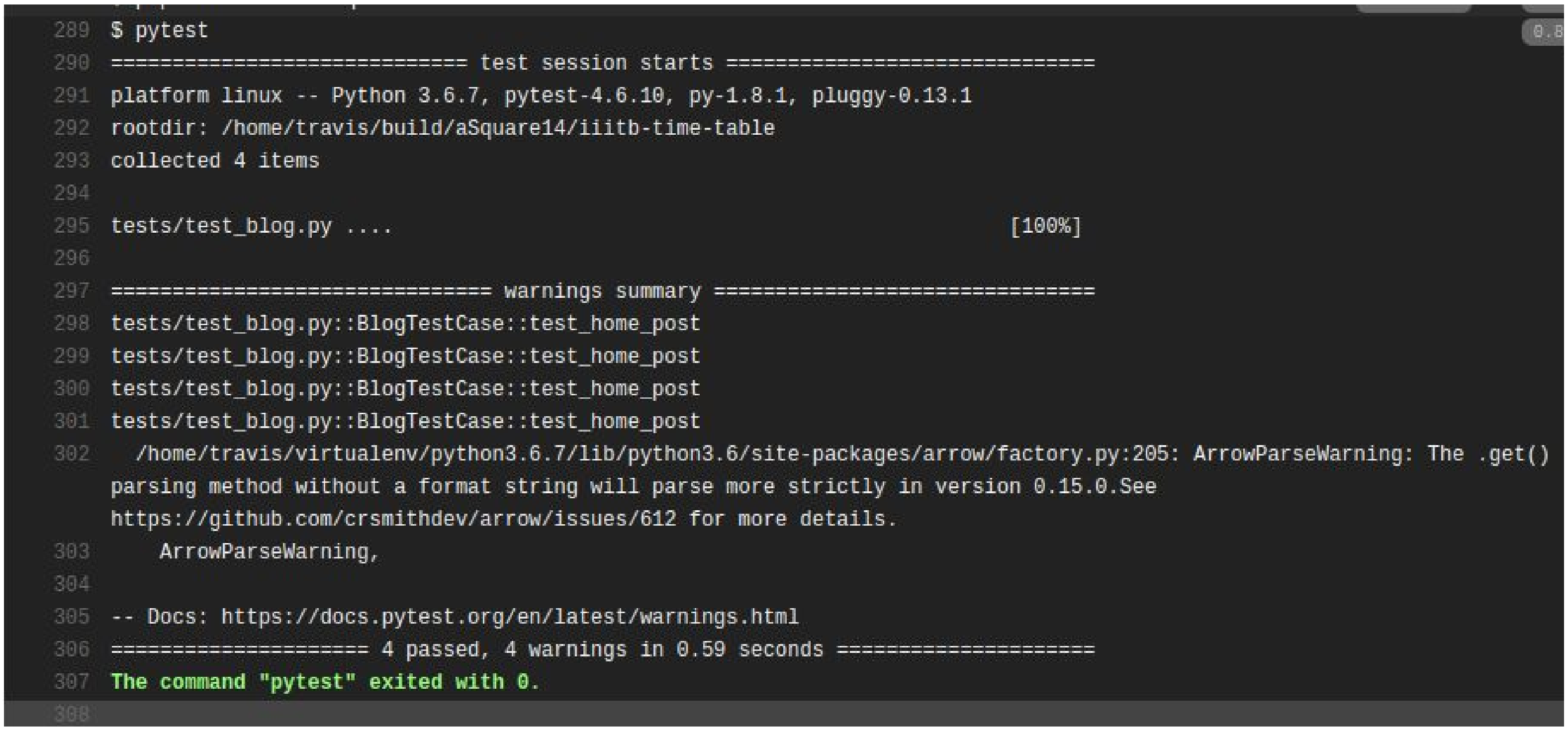


```
14  class BlogTestCase(unittest.TestCase):
15      def setUp(self):
16          self.app = app.test_client()
17          self.app.testing = True
18          return app
19
20      def test_home_get(self):
21          response = self.app.get('/')
22          assert response.status_code == 200
23
24      def test_home_post(self):
25          data = [['Event1','2021-01-01 00:00:00','2021-01-01 00:00:01'],
26                  ['Event2','2021-01-01 00:00:00','2021-01-01 00:00:02']]
27          data = json.dumps(data)
28          response = self.app.post('/', data = data, content_type='application/json')
29          assert response.status_code == 200
30
31      def test_professor_get(self):
32          prof = "/professor?prof=Ashish%20Choudhury"
33          response = self.app.get('/professor', data = prof, content_type='application/text')
34          assert response.status_code == 200
35
36      def test_ajax_post(self):
37          data = {}
38          data["query"] = 'DS101'
39          response = self.app.post('/ajax/', data = dict(query='DS101'))
40          assert response.status_code == 200
```

Figure 6. (test_blog.py)

For instance, in the function test_home_post(line 24), the app.py function receives a json object of events and time for downloading ics, and returns a response containing a file attachment as header, which users can download. We are sending dummy events and time data, and then checking if the status response code was OK.
Similarly, in the test_professor_get function(line 31), a dummy request argument was sent to app.py, and the response code was checked.

The pytest script in the travis.yml file(line 12, Figure 2) takes care of testing the project. As we can see below in Figure 7, 4 test items are collected which are our functions(line 293), all our 4 tests succeed and pytest exits with success.(line 306-307).

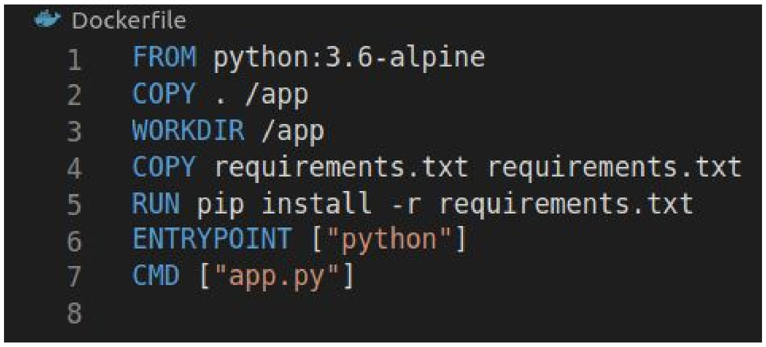


```
$ pytest
============================= test session starts ==============================
platform linux -- Python 3.6.7, pytest-4.6.10, py-1.8.1, pluggy-0.13.1
rootdir: /home/travis/build/aSquare14/iiitb-time-table
collected 4 items

tests/test_blog.py ....                                                  [100%]

=============================== warnings summary ===============================
tests/test_blog.py::BlogTestCase::test_home_post
tests/test_blog.py::BlogTestCase::test_home_post
tests/test_blog.py::BlogTestCase::test_home_post
tests/test_blog.py::BlogTestCase::test_home_post
  /home/travis/virtualenv/python3.6.7/lib/python3.6/site-packages/arrow/factory.py:205: ArrowParseWarning: The .get()
parsing method without a format string will parse more strictly in version 0.15.0.See
https://github.com/crsmithdev/arrow/issues/612 for more details.
    ArrowParseWarning,

-- Docs: https://docs.pytest.org/en/latest/warnings.html
===================== 4 passed, 4 warnings in 0.59 seconds =====================
The command "pytest" exited with 0.
```

Figure 7. Test part in travis logs

9. Artifact

An artifact is one of many by-products produced during the software development.The artifact of our project was the docker image of the application itself which was pushed to DockerHub. We dockerized our application because then our application can run in any environment easily.
First, we install docker and run it locally. Then after it is running locally, we add a DockerFile. The DockerFile can be seen in Figure 8 below.

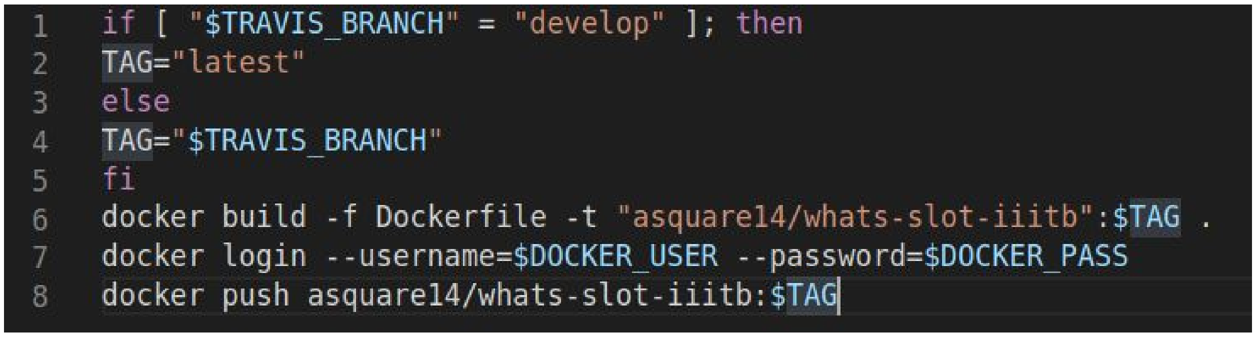


Dockerfile

```
FROM python:3.6-alpine
COPY . /app
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
ENTRYPOINT ["python"]
CMD ["app.py"]
```

Figure 8. DockerFile

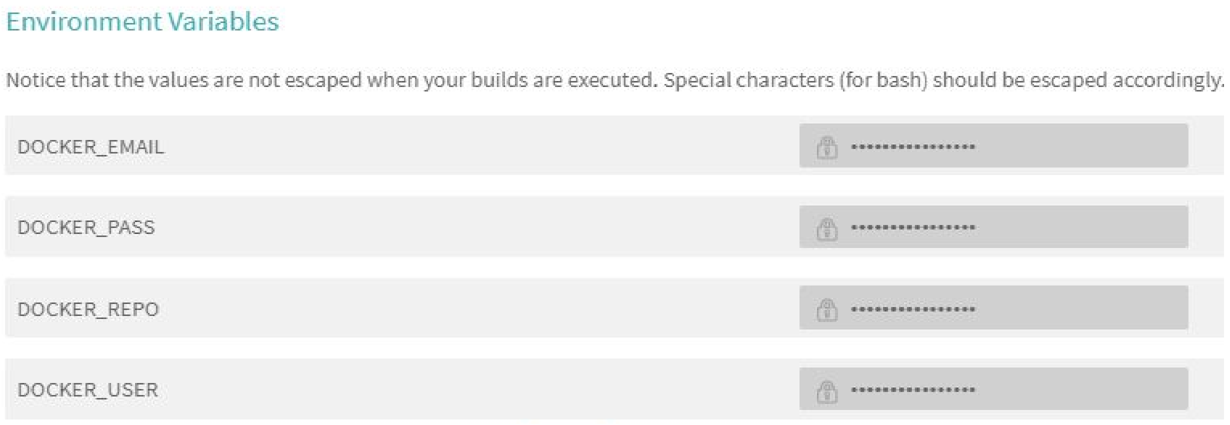
Docker pulls the latest alpine image, installs dependencies using requirements.txt generated earlier and then starts our Flask app when the container is ready. We push our docker image to DockerHub using Travis. We add a script called deploy_dockerhub.sh in the .travis directory.



```
if [ "$TRAVIS_BRANCH" = "develop" ]; then
TAG="latest"
else
TAG="$TRAVIS_BRANCH"
fi
docker build -f Dockerfile -t "asquare14/whats-slot-iiitb":$TAG .
docker login --username=$DOCKER_USER --password=$DOCKER_PASS
docker push asquare14/whats-slot-iiitb:$TAG
```

Figure 9. deploy_dockerhub.sh

We set environment variables in travis by going to settings as shown in figure 10 below.



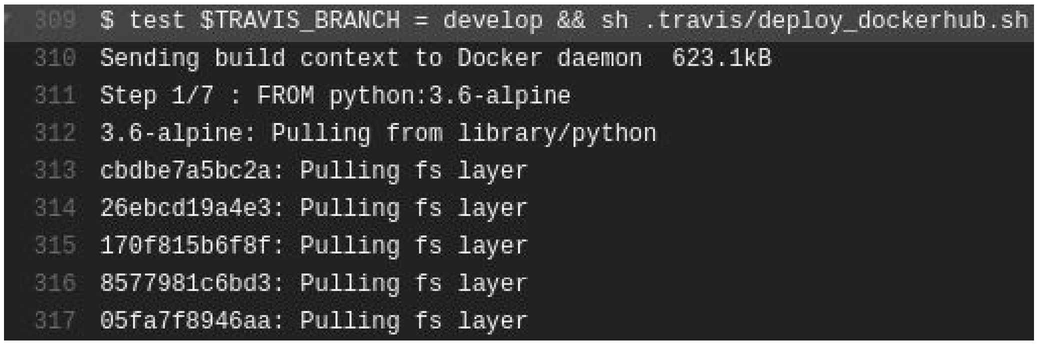
Environment Variables

Notice that the values are not escaped when your builds are executed. Special characters (for bash) should be escaped accordingly.

DOCKER_EMAIL	••••••••••••••••
DOCKER_PASS	••••••••••••••••
DOCKER_REPO	••••••••••••••••
DOCKER_USER	••••••••••••••••

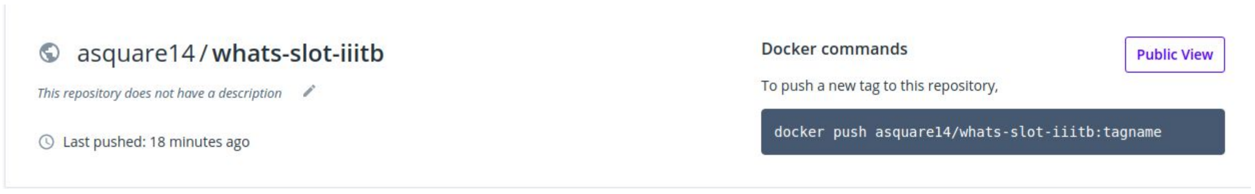
Figure 10. Settings Page of travis-ci.com

In our .travis.yml file (Figure 2) line 16 runs this script. FInally on merging a commit, we can now see Travis build our application, run tests, install docker and build image, and then deploy the image to DockerHub after the container is ready, Figure 11 below shows Travis building the docker image. Figure 12 is a screenshot of DockerHub repository with the latest image.



```
$ test $TRAVIS_BRANCH = develop && sh .travis/deploy_dockerhub.sh
Sending build context to Docker daemon  623.1kB
Step 1/7 : FROM python:3.6-alpine
3.6-alpine: Pulling from library/python
cbdbe7a5bc2a: Pulling fs layer
26ebcd19a4e3: Pulling fs layer
170f815b6f8f: Pulling fs layer
8577981c6bd3: Pulling fs layer
05fa7f8946aa: Pulling fs layer
```

Figure 11. Build of Docker Image by Travis



asquare14 / **whats-slot-iiitb**

This repository does not have a description

Last pushed: 18 minutes ago

Docker commands

To push a new tag to this repository,

```
docker push asquare14/whats-slot-iiitb:tagname
```

Figure 12. DockerHub Repository

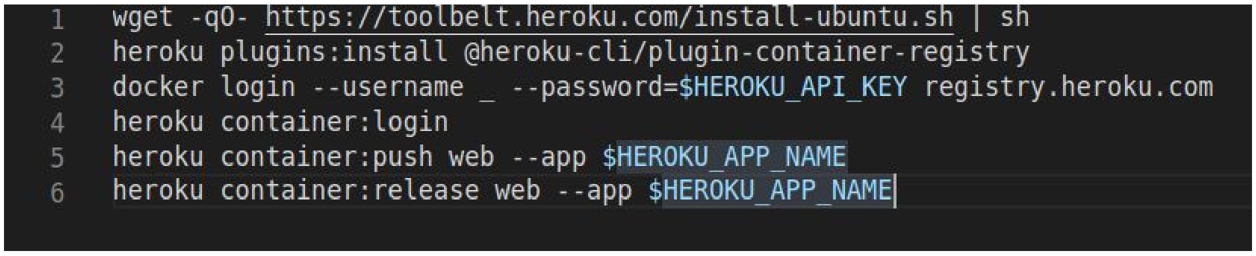
We run the docker image by pulling latest image from DockerHub:
docker run -p 5000:5000 -it asquare14/whats-slot-iiitb
Now, in the next part we will use travis to deploy our application to heroku.

10. Deploy

We used Heroku to deploy our project. We used the Heroku Command Line Tool to create our application. After installing Heroku CLI from Heroku Dev Centre type the following commands :

```
$ heroku login
$ heroku create
```

heroku create command creates our application and gives it a URL address as well as name. We can use this URL to go to our application on the web. We add a script deploy_heroku.sh. Travis uses this script to push our docker image to heroku. Figure 13 shows the contents of the file below.

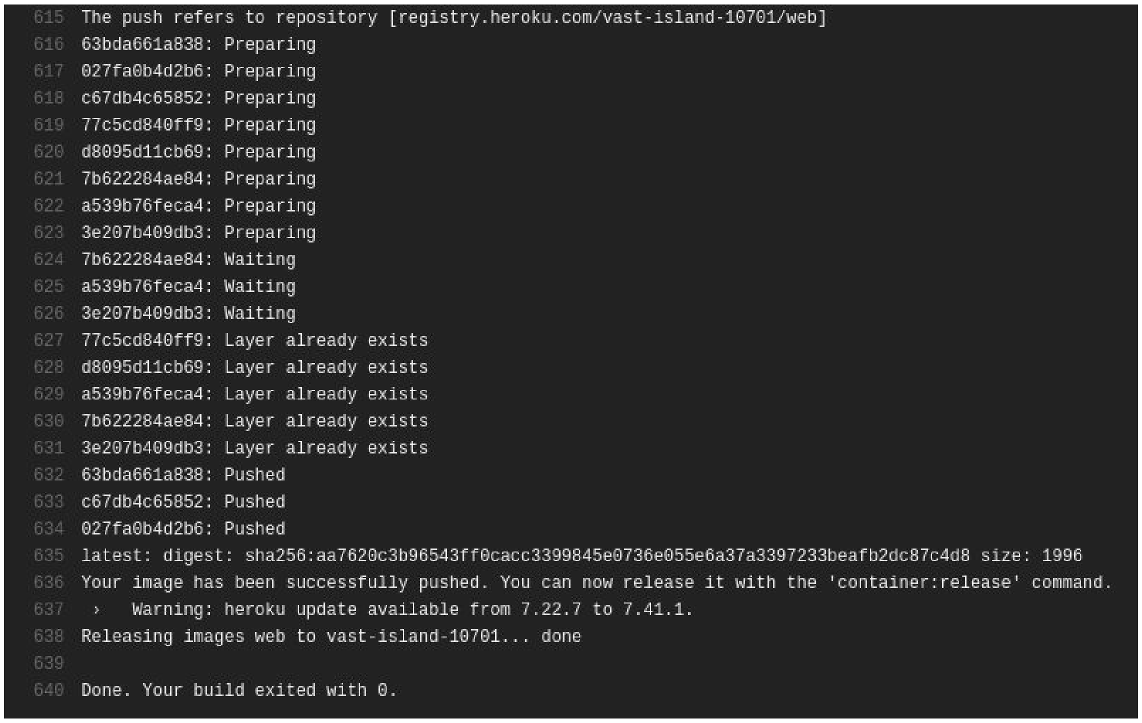


```
wget -qO- https://toolbelt.heroku.com/install-ubuntu.sh | sh
heroku plugins:install @heroku-cli/plugin-container-registry
docker login --username _ --password=$HEROKU_API_KEY registry.heroku.com
heroku container:login
heroku container:push web --app $HEROKU_APP_NAME
heroku container:release web --app $HEROKU_APP_NAME
```

Figure 13. deploy_heroku.sh

We set environment variables in travis by going to settings as shown in figure 10 above and set the variables HEROKU_API_KEY and HEROKU_APP_NAME.

The command in line 17, Figure 2, checks whether the repository's develop branch is active. In the Figure 14 below, we can see that travis has pushed our docker image to heroku. (lines 615-638). On every successful commit we can now see as Travis successfully pushes the docker container to DockerHub. After the pipeline finishes successfully, we can view our live application using the URL generated by heroku.



```
615 The push refers to repository [registry.heroku.com/vast-island-10701/web]
616 63bda661a838: Preparing
617 027fa0b4d2b6: Preparing
618 c67db4c65852: Preparing
619 77c5cd840ff9: Preparing
620 d8095d11cb69: Preparing
621 7b622284ae84: Preparing
622 a539b76feca4: Preparing
623 3e207b409db3: Preparing
624 7b622284ae84: Waiting
625 a539b76feca4: Waiting
626 3e207b409db3: Waiting
627 77c5cd840ff9: Layer already exists
628 d8095d11cb69: Layer already exists
629 a539b76feca4: Layer already exists
630 7b622284ae84: Layer already exists
631 3e207b409db3: Layer already exists
632 63bda661a838: Pushed
633 c67db4c65852: Pushed
634 027fa0b4d2b6: Pushed
635 latest: digest: sha256:aa7620c3b96543ff0cacc3399845e0736e055e6a37a3397233beafb2dc87c4d8 size: 1996
636 Your image has been successfully pushed. You can now release it with the 'container:release' command.
637  ›   Warning: heroku update available from 7.22.7 to 7.41.1.
638 Releasing images web to vast-island-10701... done
639
640 Done. Your build exited with 0.
```

Figure 14. Releasing Image to Heroku

11. Monitoring

We used ELK for monitoring. ELK stack stands for Elasticsearch, Logstash and Kibana. Elasticsearch is built on Apache Lucene library which makes it much faster than the traditional SQL. It is a schema-flexible, REST and JSON based distributed search engine. It provides the ability to search, analyze as well as store large amounts of data. Elastic search provides flexibility and scalability. It also operates in real time.

Logstash is used for centralized logging as well as parsing. It is an extract, transfer and load tool that modifies, parses and stores logs with ElasticSearch. Logstash can also collect data from multiple systems into a central system.

Kibana is a web-based tool through which Elastic Search databases visualize and analyze stored data. Kibana offers powerful and easy to use features such as histograms, line graphs, pie charts, heat maps, etc.

Figure 15 below explains how ELK interact with each other.

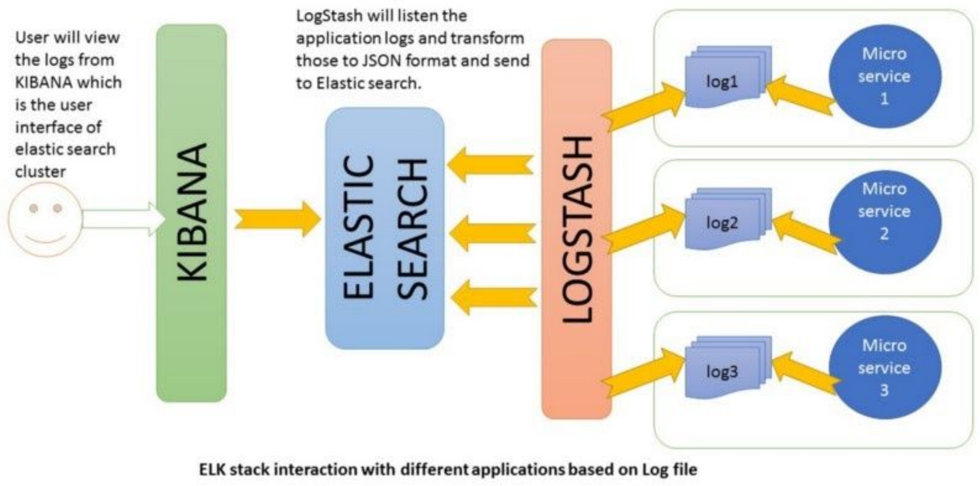
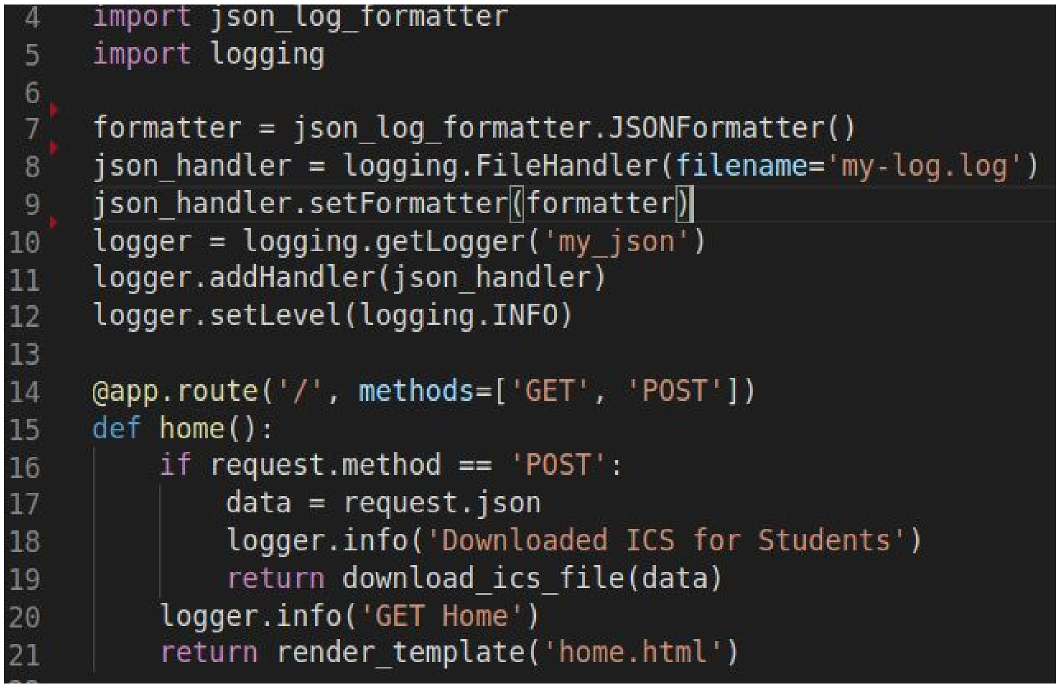

User will view the logs from KIBANA which is the user interface of elastic search cluster
LogStash will listen the application logs and transform those to JSON format and send to Elastic search.
KIBANA
ELASTIC SEARCH
LOGSTASH
log1
log2
log3
Micro service 1
Micro service 2
Micro service 3
ELK stack interaction with different applications based on Log file

Figure 15. ELK stack interaction with different applications (Taken from FreeCodeCamp)

11.1 Implementation Overview of ELK

For our application we used the logging library (line 5, Figure 16) in Python. We wanted our logs to be in JSON format and for this, we used the JSON-log-formatter(line 4, Figure 16) library in python. This library helps us to store logs in JSON format and also allows for easy integration with logstash. The Figure 16 located below shows the implementation.



```python
import json_log_formatter
import logging

formatter = json_log_formatter.JSONFormatter()
json_handler = logging.FileHandler(filename='my-log.log')
json_handler.setFormatter(formatter)
logger = logging.getLogger('my_json')
logger.addHandler(json_handler)
logger.setLevel(logging.INFO)

@app.route('/', methods=['GET', 'POST'])
def home():
    if request.method == 'POST':
        data = request.json
        logger.info('Downloaded ICS for Students')
        return download_ics_file(data)
    logger.info('GET Home')
    return render_template('home.html')
```

Figure 16. Logging

11.2 Setting up ELK

1. To set up ElasticSearch, we download it and unzip the package. Then, we

```
$ cd elasticsearch
$ cd bin/  &&  ./elasticsearch
```

2. To set up Logstash, download it and unzip it. Then,

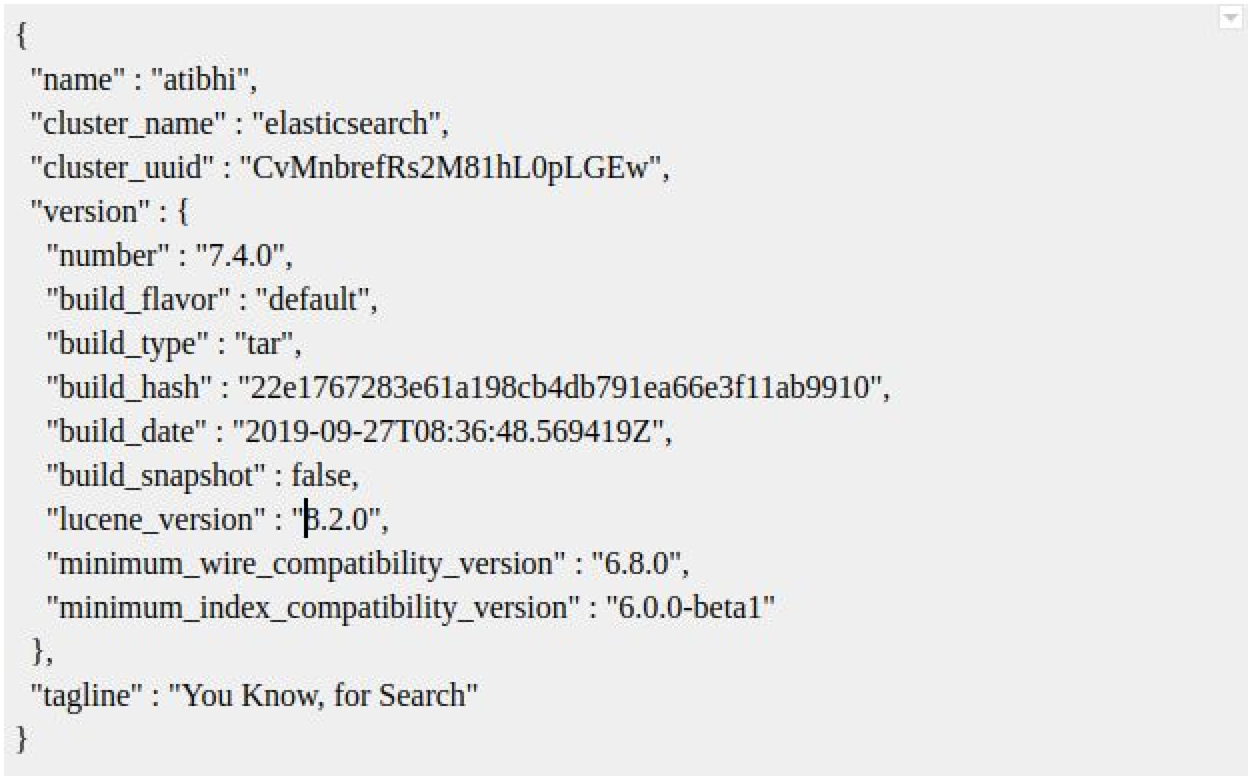
```
$ cd logstash
$ cd bin/ &&  ./logstash -f "<file_path_to_logstash.conf>"
```

3. To set up Kibana, download it and unzip it. Then,

```
$ cd kibana
$ cd bin/ && ./kibana
```

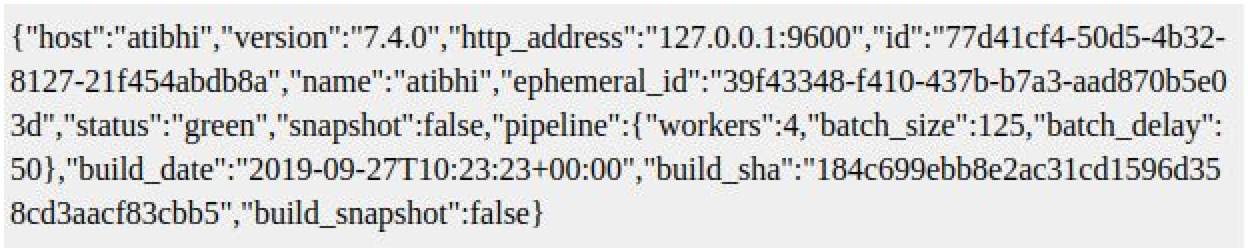
11.3 To check if the installation is working properly

For ElasticSearch, go to http://localhost:9200



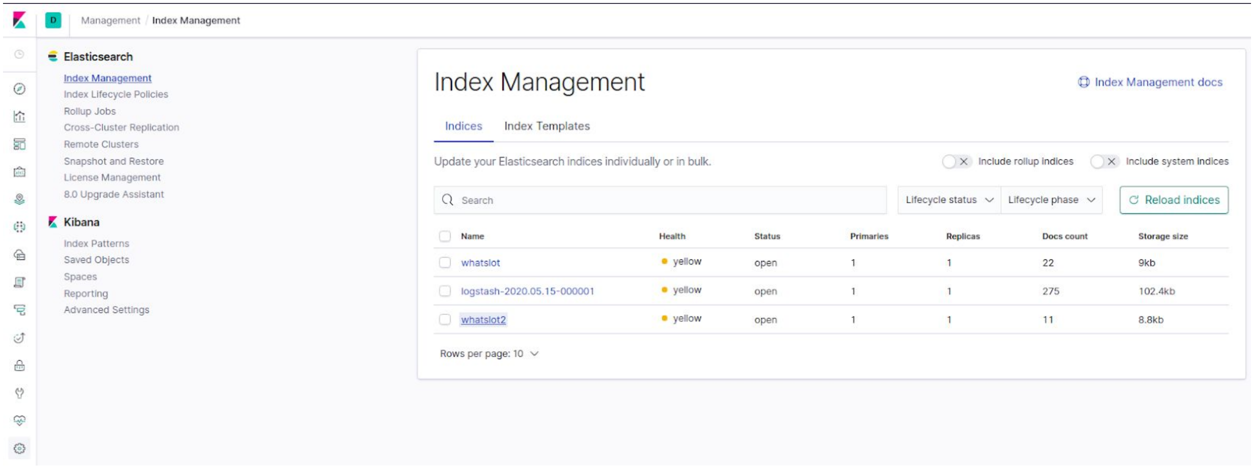
```
{
  "name" : "atibhi",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "CvMnbrefRs2M81hL0pLGEw",
  "version" : {
    "number" : "7.4.0",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "22e1767283e61a198cb4db791ea66e3f11ab9910",
    "build_date" : "2019-09-27T08:36:48.569419Z",
    "build_snapshot" : false,
    "lucene_version" : "8.2.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

For Logstash, go to http://localhost:9600



```
{"host":"atibhi","version":"7.4.0","http_address":"127.0.0.1:9600","id":"77d41cf4-50d5-4b32-
8127-21f454abdb8a","name":"atibhi","ephemeral_id":"39f43348-f410-437b-b7a3-aad870b5e0
3d","status":"green","snapshot":false,"pipeline":{"workers":4,"batch_size":125,"batch_delay":
50},"build_date":"2019-09-27T10:23:23+00:00","build_sha":"184c699ebb8e2ac31cd1596d35
8cd3aacf83cbb5","build_snapshot":false}
```

The Kibana dashboard is at http://localhost:5601



Index Management

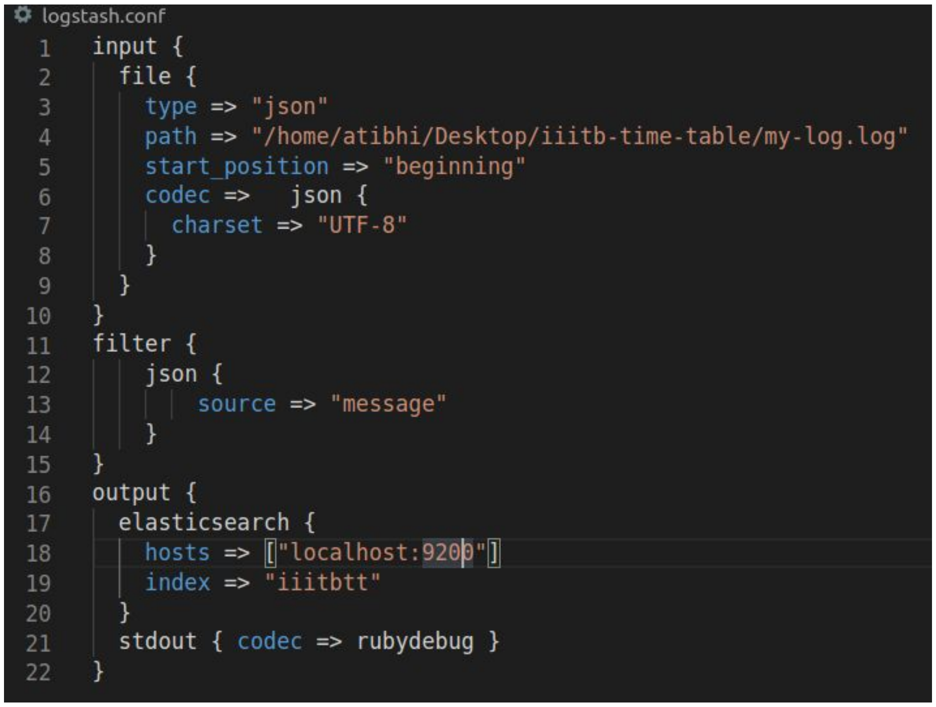
Update your Elasticsearch indices individually or in bulk.

Name	Health	Status	Primaries	Replicas	Docs count	Storage size
whatslot	● yellow	open	1	1	22	9kb
logstash-2020.05.15-000001	● yellow	open	1	1	275	102.4kb
whatslot2	● yellow	open	1	1	11	8.8kb

Figure 17. Kibana Up and Running

11.4 Configuring Logstash

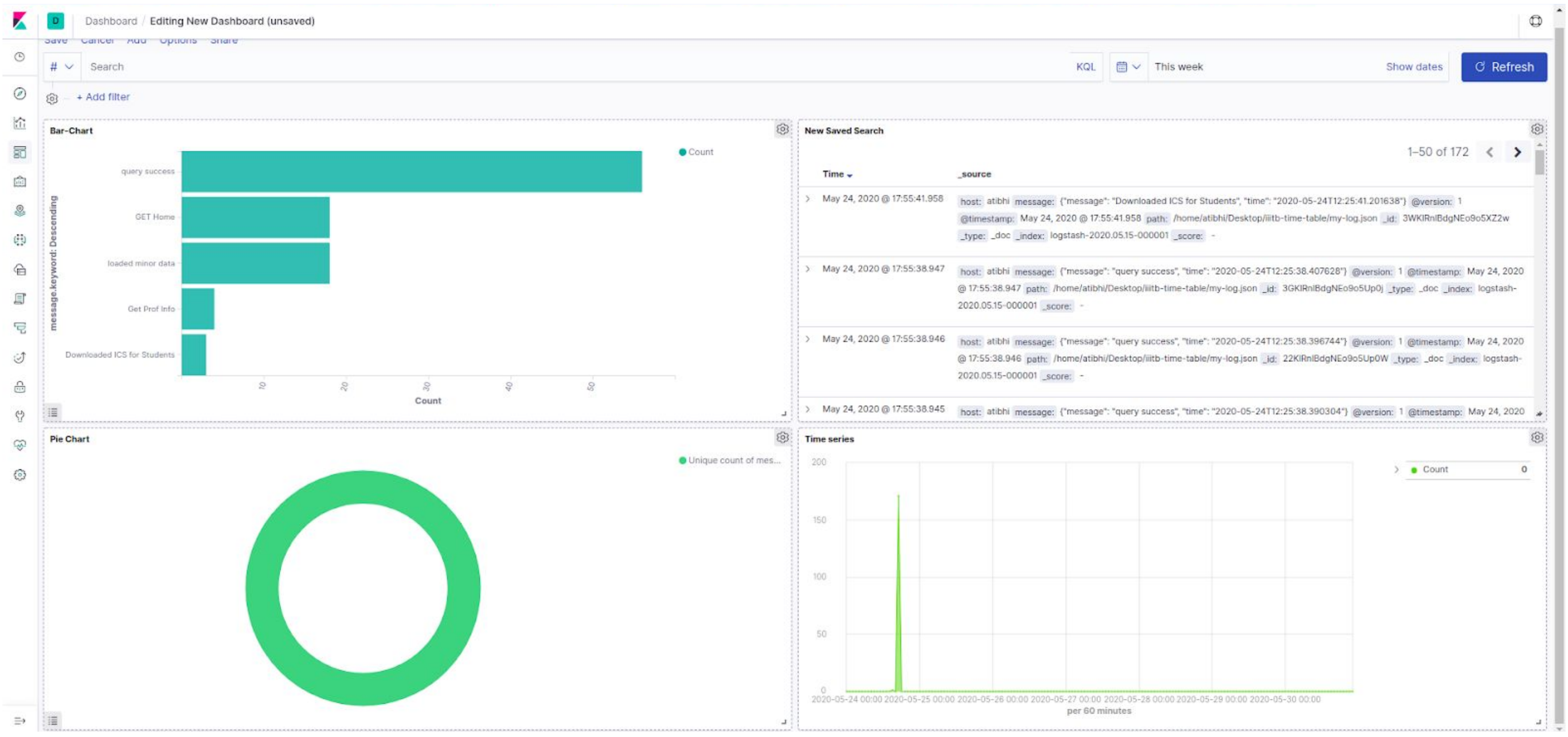
Figure 18 below shows the logstash.conf file which is important for logstash to send logs to kibana.



```
logstash.conf
input {
  file {
    type => "json"
    path => "/home/atibhi/Desktop/iiitb-time-table/my-log.log"
    start_position => "beginning"
    codec =>   json {
      charset => "UTF-8"
    }
  }
}
filter {
    json {
      source => "message"
    }
}
output {
  elasticsearch {
    hosts => ["localhost:9200"]
    index => "iiitbtt"
  }
  stdout { codec => rubydebug }
}
```

Figure 18. Logstash.conf

11.5 Visualizing Logs in Kibana



```
May 24, 2020 @ 17:55:41.958  host: atibhi message: {"message": "Downloaded ICS for Students", "time": "2020-05-24T12:25:41.201638"} @version: 1 @timestamp: May 24, 2020 @ 17:55:41.958 path: /home/atibhi/Desktop/iiitb-time-table/my-log.json _id: 3WKIRnlBdgNEo9o5XZ2w _type: _doc _index: logstash-2020.05.15-000001 _score: -
May 24, 2020 @ 17:55:38.947  host: atibhi message: {"message": "query success", "time": "2020-05-24T12:25:38.407628"} @version: 1 @timestamp: May 24, 2020 @ 17:55:38.947 path: /home/atibhi/Desktop/iiitb-time-table/my-log.json _id: 3GKIRnlBdgNEo9o5Up0j _type: _doc _index: logstash-2020.05.15-000001 _score: -
May 24, 2020 @ 17:55:38.946  host: atibhi message: {"message": "query success", "time": "2020-05-24T12:25:38.396744"} @version: 1 @timestamp: May 24, 2020 @ 17:55:38.946 path: /home/atibhi/Desktop/iiitb-time-table/my-log.json _id: 22KIRnlBdgNEo9o5Up0W _type: _doc _index: logstash-2020.05.15-000001 _score: -
May 24, 2020 @ 17:55:38.945  host: atibhi message: {"message": "query success", "time": "2020-05-24T12:25:38.390304"} @version: 1 @timestamp: May 24, 2020
```

Figure 19. Kibana Dashboard

In the above Figure 19, we can see the Kibana dashboard where I have visualized the logs of our application. In the top left corner is a bar graph that shows how many times which part of our website was visited. In the top right corner is the timestamp and other information of logs that were coming to kibana. In the bottom left corner, the pie chart shows the time at which logs came. Since, our logfile had only one time at the time of visualization, it is on colour. The bottom right corner, shows that the traffic peaked at the time of the spike.

On generation of new logs, the graphs in kibana are updated automatically as long as ELK is running successfully. In situations where a huge amount of data is generated every single second, the data-visualisation is necessary. ELK is one of the most effective ways of organizing raw-data.

12. Functional and Non-functional Requirements

Functional requirements define the functions a system or software must perform, i.e. what input it takes, the behaviour of the software and output generated by it. Functional requirements also help us know in advance the expected behaviour of the system.

Functional requirements for our software are:

- Students can select courses and visualize their respective course timings.
- In case of clash, the slot turns red.
- Students can clear the visualizer, i.e. all the slots selected become empty.
- Students can download the ics file of their timetable, and can import it to google, apple or outlook calendar.
- Students can directly import to the calendar, without downloading an ics file.
- Students and faculty can view faculty time-table.

A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system.They are essential to ensure the usability and effectiveness of the entire software system. If we fail to meet the non-functional requirements, software is not likely to satisfy the needs of the user.

Some of the non-functional requirements for our application are:

- The website is capable of handling the required load .
- The software is portable as we have a dockerized version of it on dockerhub and anyone can pull it and run irrespective of which environment they are using.
- Users don't have any access to the database which contains the course and faculty timings.

13. Results

Our website has two main pages, the student time-table page, and the faculty time-table page. The landing page provides a common area to navigate between both these pages. Figure 20 below shows the homepage of our application which provides a navigation option to the student time table page and the faculty time table page.

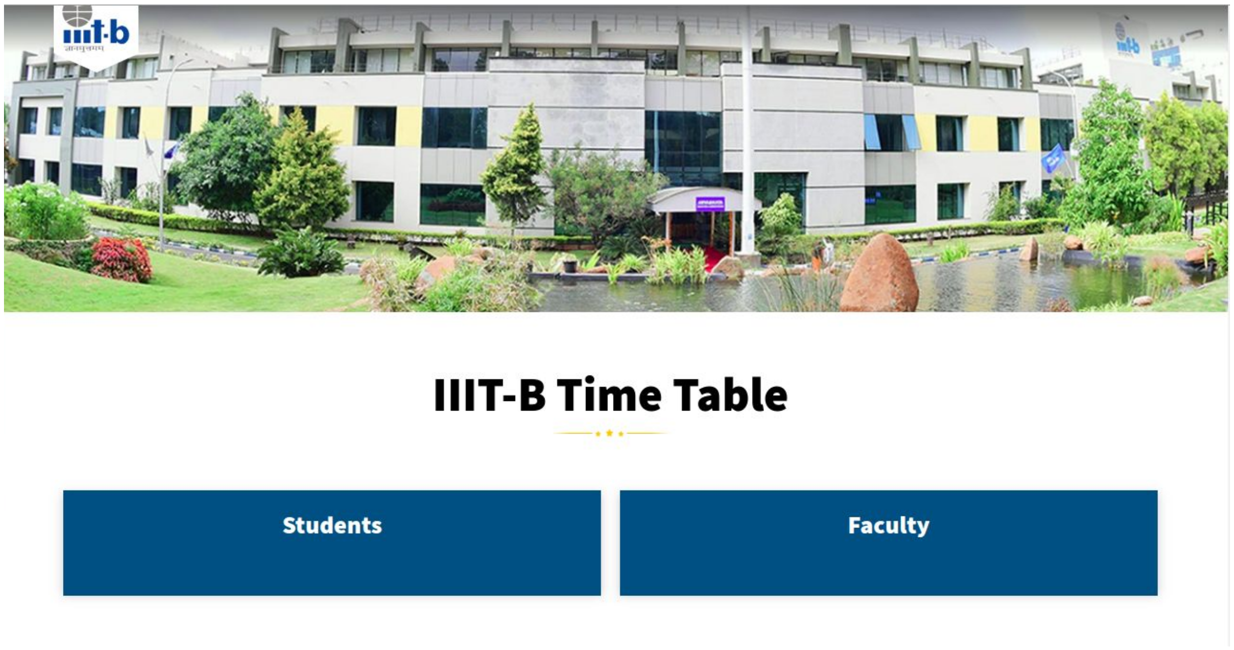
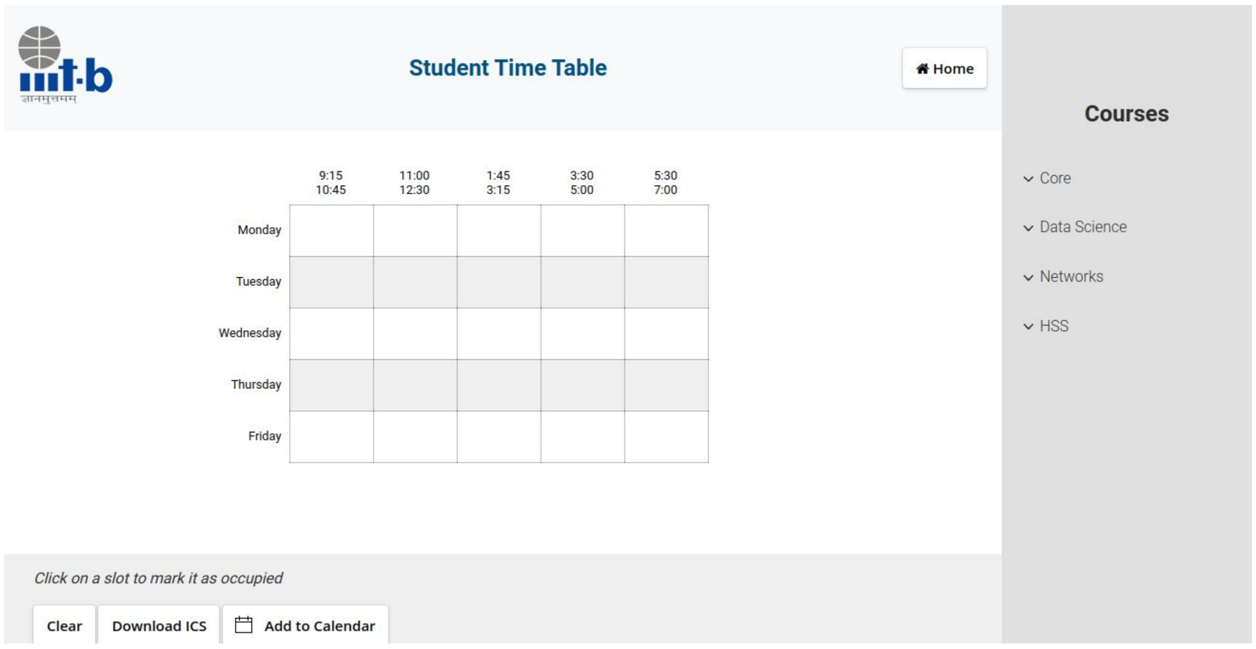


Figure 20 : Landing Page/Home Page

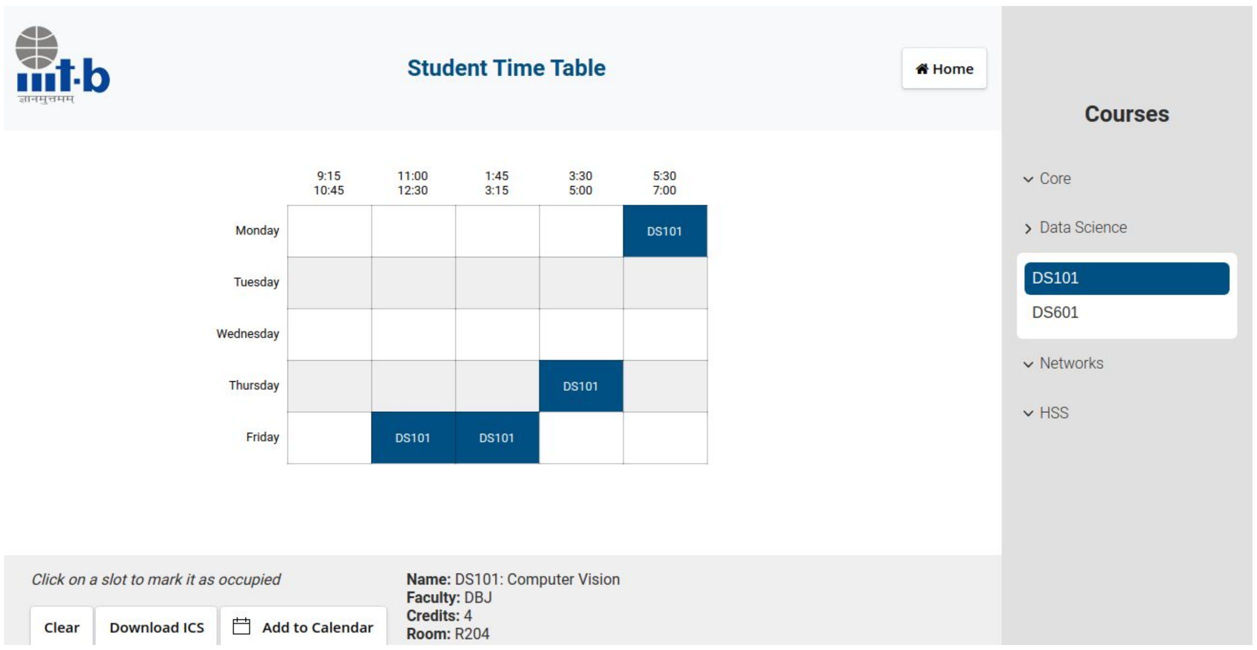
Figure 21 located below shows the student time table page.



	9:15 10:45	11:00 12:30	1:45 3:15	3:30 5:00	5:30 7:00
Monday					
Tuesday					
Wednesday					
Thursday					
Friday					

Figure 21. Student Homepage

As shown below in Figure 22, students can select the relevant courses from the side menu. Upon selection the visualizer will display the respective slots for courses in blue.



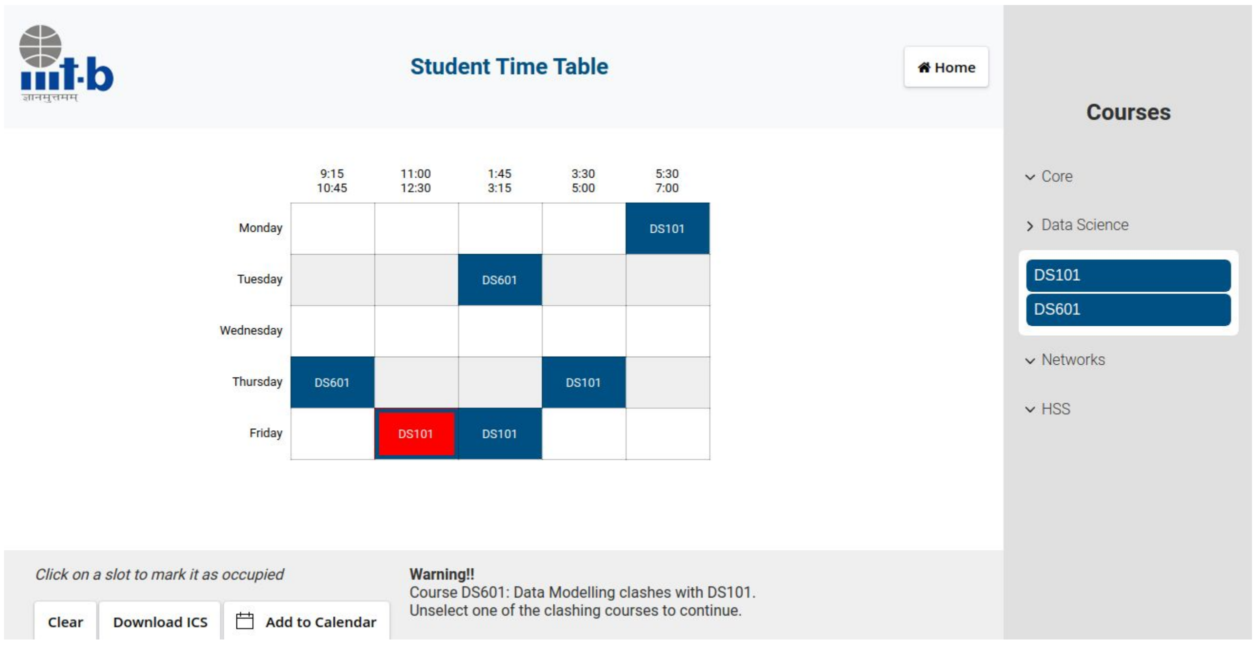
	9:15 10:45	11:00 12:30	1:45 3:15	3:30 5:00	5:30 7:00
Monday					DS101
Tuesday					
Wednesday					
Thursday				DS101	
Friday		DS101	DS101		

Click on a slot to mark it as occupied

Name: DS101: Computer Vision
Faculty: DBJ
Credits: 4
Room: R204

Figure 22. Student selecting courses

In case the selected courses clash, the visualizer turns the slot red in colour as shown in Figure 23.



	9:15 10:45	11:00 12:30	1:45 3:15	3:30 5:00	5:30 7:00
Monday					DS101
Tuesday			DS601		
Wednesday					
Thursday	DS601			DS101	
Friday		DS101	DS101		

Click on a slot to mark it as occupied

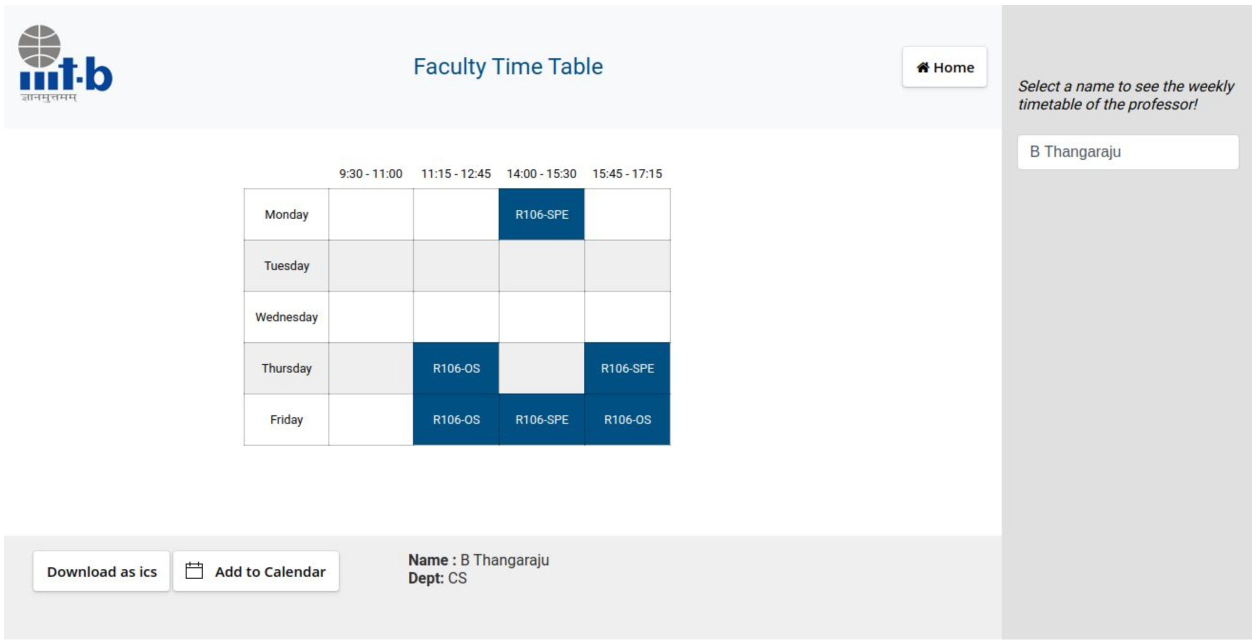
Warning!!
Course DS601: Data Modelling clashes with DS101.
Unselect one of the clashing courses to continue.

Figure 23. When courses clash

Some other features of the student section are as follows :

- Students can unselect one of the clashing courses to resolve the clash.
- Students can even mark a cell as occupied by clicking on it, in case they don't want any courses at that particular time.
- Students can download the ics file by clicking on the download ics button, and import it to their calendars.

The Faculty section of our website is shown in Figure 24 below. Students can view the timetable of faculties too, and their respective venues at that time, Faculties can also import their time table to their google calendar.



	9:30 - 11:00	11:15 - 12:45	14:00 - 15:30	15:45 - 17:15
Monday			R106-SPE	
Tuesday				
Wednesday				
Thursday		R106-OS		R106-SPE
Friday		R106-OS	R106-SPE	R106-OS

Name : B Thangaraju
Dept: CS

Figure 24. Faculty Page

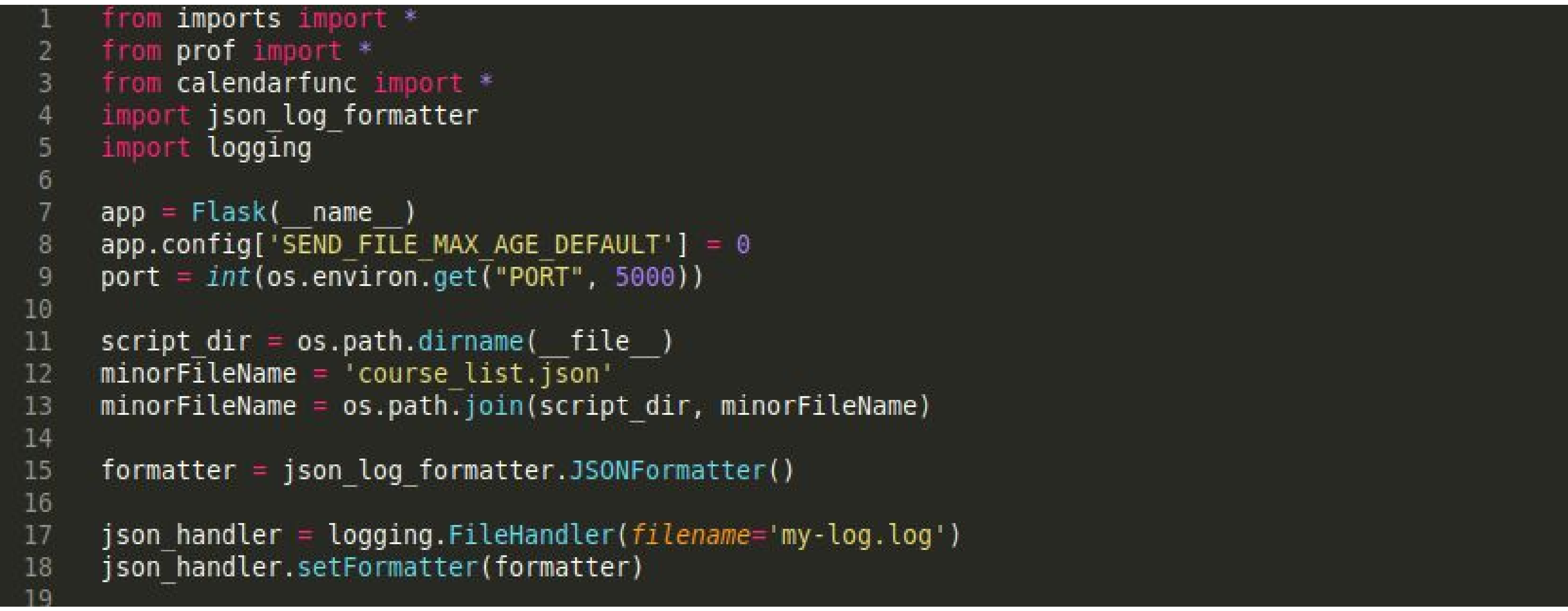
Compared to what-slot, the inspiration behind this whole idea, below are the few extra features we added.

- What-slot does not provide any clash indicator or any provision to unselect the courses. They just show the slots for a selected course, and you have to manually click on each individual slots in order to book it for that course.
- We added a feature to clear the whole visualizer in one click.
- We added a feature where students can download the ICS file of their time table and import it to their respective calendars(google/outlook/apple)
- We integrated the faculty time-table with this application. Students can view the time table of different faculties, and faculties can download ics files of their timetable.

14. Code Walk-through

We have used flask for the back-end section of our project. Flask is a web application framework written in python, which uses Werkzeug WSGI toolkit and Jinja2 template engine. Some of the reasons why we chose Flask are because it is Scalable. Development in Flask is simple. It is Flexible and there are very few parts of Flask that cannot be easily and safely altered because of its simplicity and minimality. It has superior performance as there are fewer levels of abstraction between us and the database, the requests, the cache, etc. So performance is inherently better from the start. It is also modular. Flask allows more efficiency, better testability, and better performance.

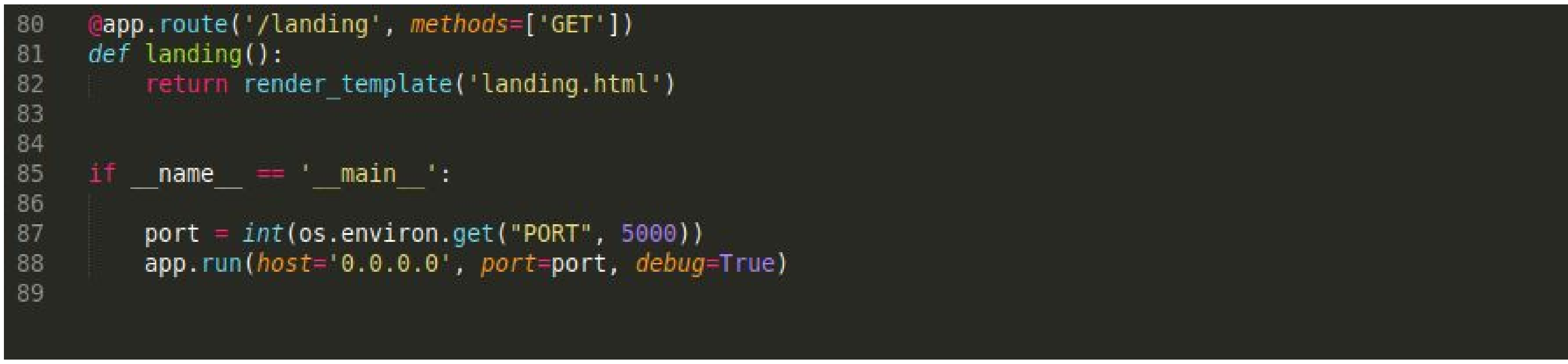
Our main code where we import the flask module and define different routes is in the file *app.py*.



```
1    from imports import *
2    from prof import *
3    from calendarfunc import *
4    import json_log_formatter
5    import logging
6
7    app = Flask(__name__)
8    app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0
9    port = int(os.environ.get("PORT", 5000))
10
11   script_dir = os.path.dirname(__file__)
12   minorFileName = 'course_list.json'
13   minorFileName = os.path.join(script_dir, minorFileName)
14
15   formatter = json_log_formatter.JSONFormatter()
16
17   json_handler = logging.FileHandler(filename='my-log.log')
18   json_handler.setFormatter(formatter)
19
```

Figure 25. app.py

We have imported the flask module in imports.py, and created an object of flask class in **line 7,** which is our WSGI application. Then we have defined several routes using app.route, which is a decorator of flask call and helps us define which function should be called with the given url.

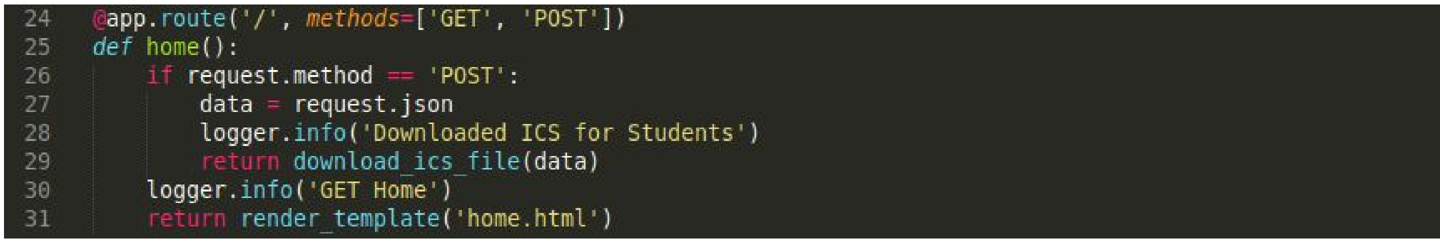


```
80  @app.route('/landing', methods=['GET'])
81  def landing():
82      return render_template('landing.html')
83
84
85  if __name__ == '__main__':
86
87      port = int(os.environ.get("PORT", 5000))
88      app.run(host='0.0.0.0', port=port, debug=True)
89
```

Figure 26. app.py-landing function

As seen in Figure 26 line 81 the home page of our application is the landing page, when the url "/landing" is loaded, the function landing() gets called, and through it the html template "landing.html" defined in the templates folder is rendered, which defines the structure and content of the landing page.

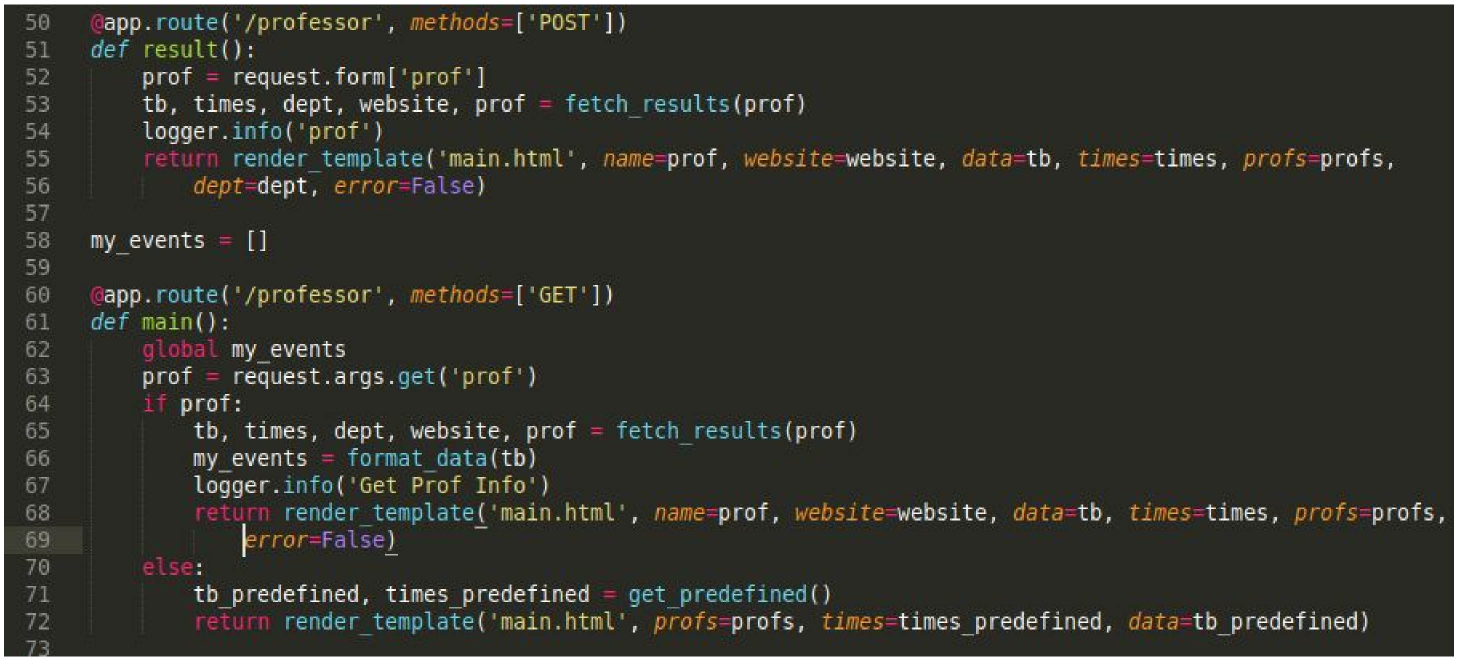
When the user clicks on the student button, to get the student time table page, the "/" url gets called. It can be seen in Figure 27, lines 25-31



```
24  @app.route('/', methods=['GET', 'POST'])
25  def home():
26      if request.method == 'POST':
27          data = request.json
28          logger.info('Downloaded ICS for Students')
29          return download_ics_file(data)
30      logger.info('GET Home')
31      return render_template('home.html')
```

Figure 27. app.py-home()

When the http request sent is "GET", the template "home.html" defined in the templates folder is rendered, which defines the structure and web content of the student time table page. When the request is a "POST" request, which happens when the user clicks on the "Download ICS button", the dowload_ics_file function defined in calendarfunc.py is called.(lines 26-31) When the user clicks on the professor button in the landing page, the "/professor" url is called, and when the method is "Get" the main() function gets called and result() gets called in case of "Post". This can be seen in Figure 28 below.

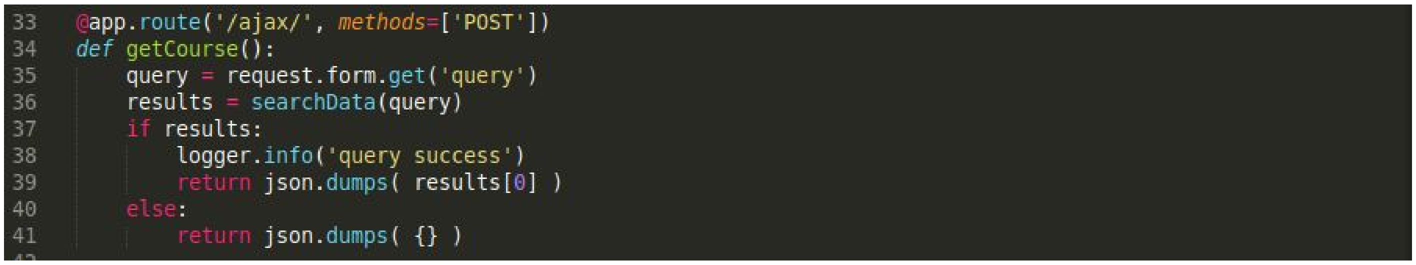


```
50 @app.route('/professor', methods=['POST'])
51 def result():
52     prof = request.form['prof']
53     tb, times, dept, website, prof = fetch_results(prof)
54     logger.info('prof')
55     return render_template('main.html', name=prof, website=website, data=tb, times=times, profs=profs,
56         dept=dept, error=False)
57 
58 my_events = []
59 
60 @app.route('/professor', methods=['GET'])
61 def main():
62     global my_events
63     prof = request.args.get('prof')
64     if prof:
65         tb, times, dept, website, prof = fetch_results(prof)
66         my_events = format_data(tb)
67         logger.info('Get Prof Info')
68         return render_template('main.html', name=prof, website=website, data=tb, times=times, profs=profs,
69             error=False)
70     else:
71         tb_predefined, times_predefined = get_predefined()
72         return render_template('main.html', profs=profs, times=times_predefined, data=tb_predefined)
```

Figure 28. app.py

In case a professor name is searched for and provided as an argument in the http request **(line 63)**, various details of the prof is fetched through functions defined in prof.py, and the template main.html is rendered with the various details as arguments.

In case no professor is searched for and the argument in the request does not exist, details associated with a blank time table are fetched using functions defined in prof.py, and main.html is rendered with those details as arguments.



```
33 @app.route('/ajax/', methods=['POST'])
34 def getCourse():
35     query = request.form.get('query')
36     results = searchData(query)
37     if results:
38         logger.info('query success')
39         return json.dumps( results[0] )
40     else:
41         return json.dumps( {} )
```

Figure 29. app.py - getCourse()

When a course is selected on the students page, "/ajax" request is called, with the course as argument, results are fetched using functions defined in search.py, and json of those results is returned, which is then used in "mainscript.js" for further use. Figure 29 above, lines 33-41

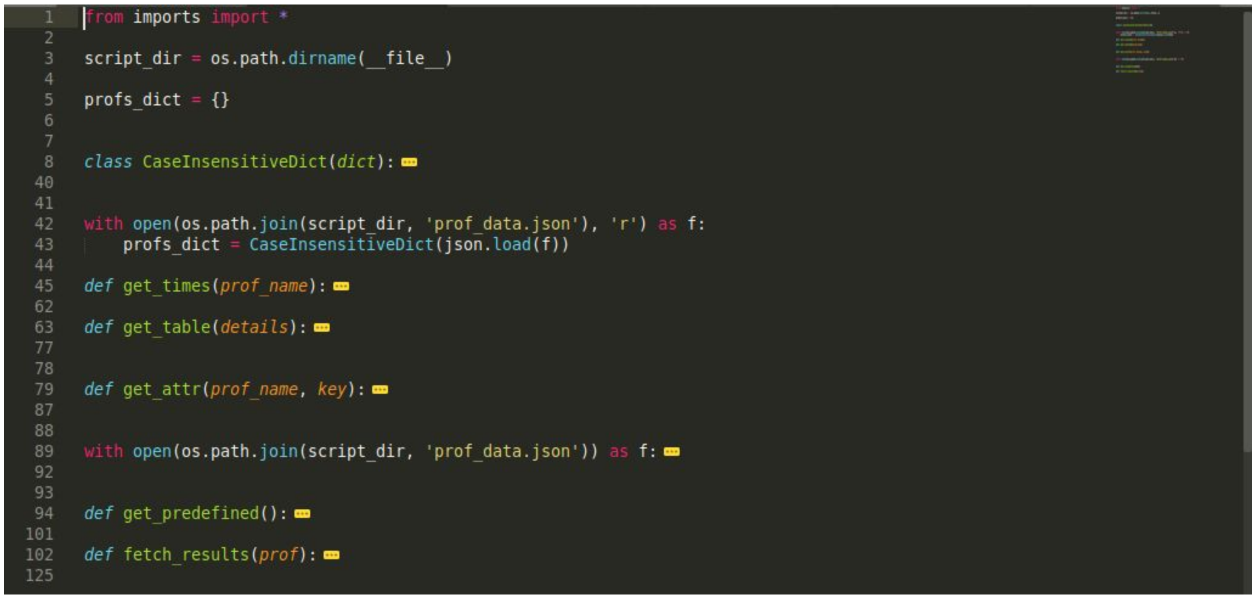

```
  1  from imports import *
  2
  3  script_dir = os.path.dirname(__file__)
  4
  5  profs_dict = {}
  6
  7
  8  class CaseInsensitiveDict(dict): ⋯
 40
 41
 42  with open(os.path.join(script_dir, 'prof_data.json'), 'r') as f:
 43      profs_dict = CaseInsensitiveDict(json.load(f))
 44
 45  def get_times(prof_name): ⋯
 62
 63  def get_table(details): ⋯
 77
 78
 79  def get_attr(prof_name, key): ⋯
 87
 88
 89  with open(os.path.join(script_dir, 'prof_data.json')) as f: ⋯
 92
 93
 94  def get_predefined(): ⋯
101
102  def fetch_results(prof): ⋯
125
```

Figure 30. prof.py

The prof.py file as shown above in Figure 30 defines various functions related to fetching details of a particular professor when a request is called, and loading the data initially when the app is started for each user once. We have used a Case Insensitive dictionary to implement the search, to make sure that case difference of letters does not affect the search. (line 8)

The calendarfunc.py file as shown in Figure 31 below defines all functions related to downloading of ics files.

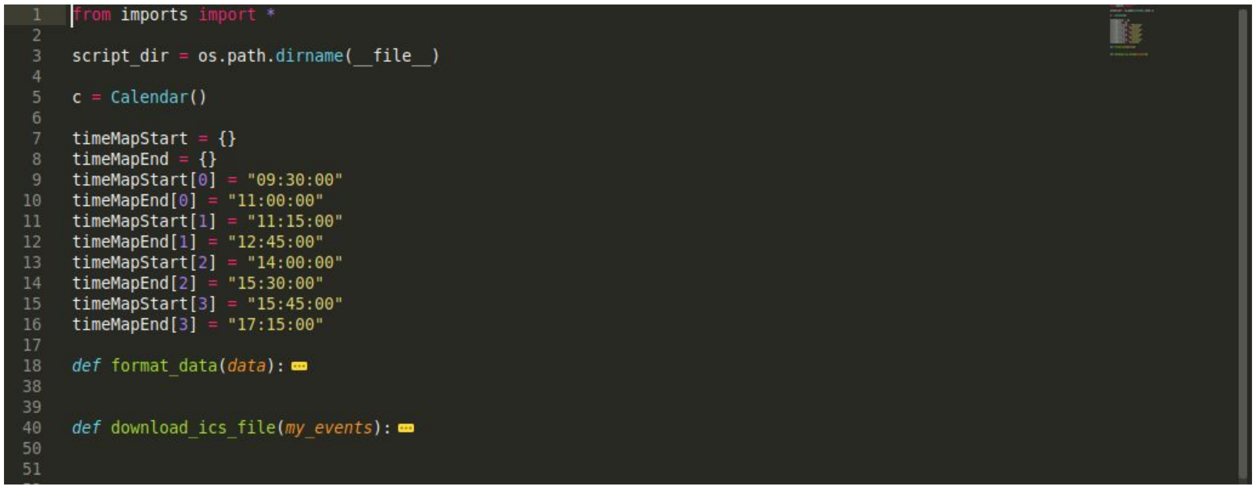
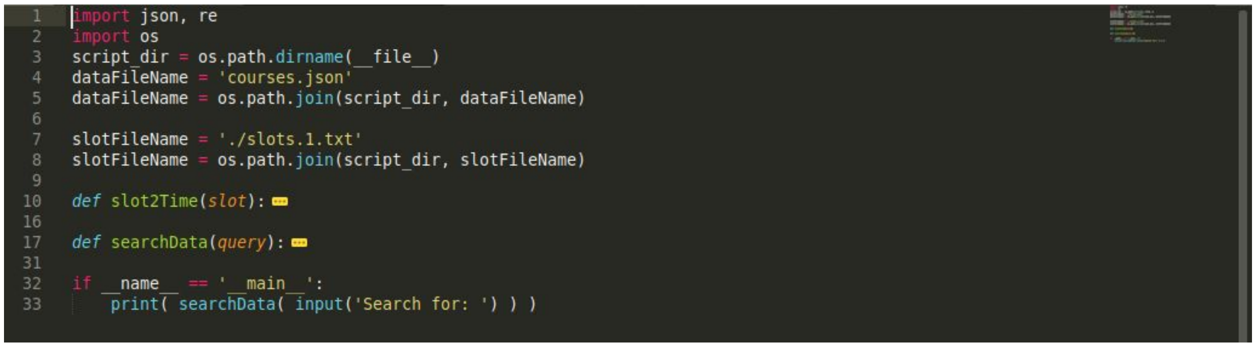

```
 1  from imports import *
 2
 3  script_dir = os.path.dirname(__file__)
 4
 5  c = Calendar()
 6
 7  timeMapStart = {}
 8  timeMapEnd = {}
 9  timeMapStart[0] = "09:30:00"
10  timeMapEnd[0] = "11:00:00"
11  timeMapStart[1] = "11:15:00"
12  timeMapEnd[1] = "12:45:00"
13  timeMapStart[2] = "14:00:00"
14  timeMapEnd[2] = "15:30:00"
15  timeMapStart[3] = "15:45:00"
16  timeMapEnd[3] = "17:15:00"
17
18  def format_data(data): ⋯
38
39
40  def download_ics_file(my_events): ⋯
50
51
```

Figure 31. calendarfunc.py

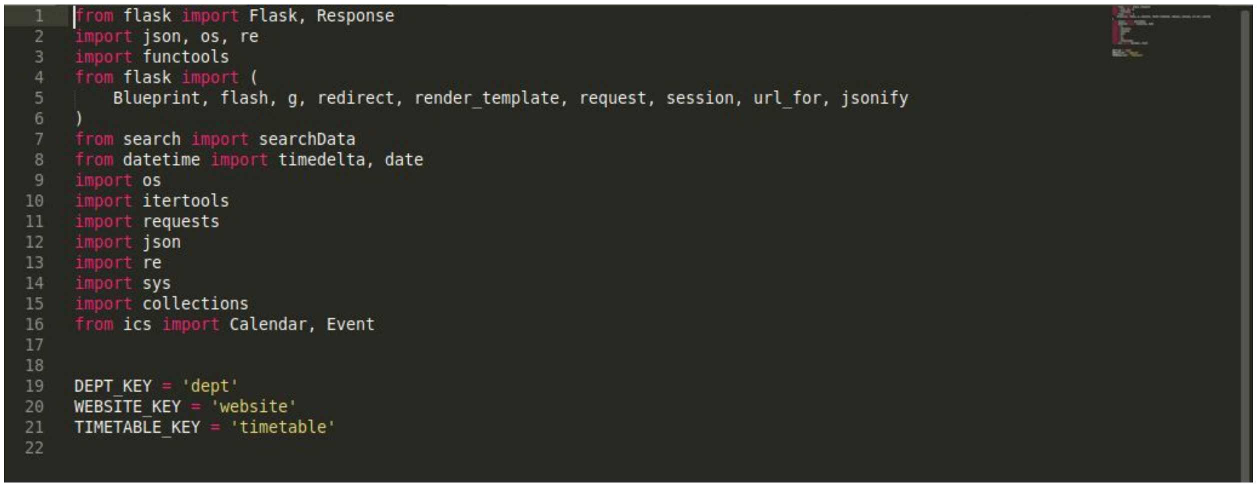
format_data changes data to a form which is suitable for download_ics_file function, which calls an api for ics file. (line 18)



```
import json, re
import os
script_dir = os.path.dirname(__file__)
dataFileName = 'courses.json'
dataFileName = os.path.join(script_dir, dataFileName)

slotFileName = './slots.1.txt'
slotFileName = os.path.join(script_dir, slotFileName)

def slot2Time(slot): ...

def searchData(query): ...

if __name__ == '__main__':
    print( searchData( input('Search for: ') ) )
```

Figure 32. search.py

search.py file shown above in Figure 32 defines functions related with fetching details of the course the user asked for, in the student time table visualizer.

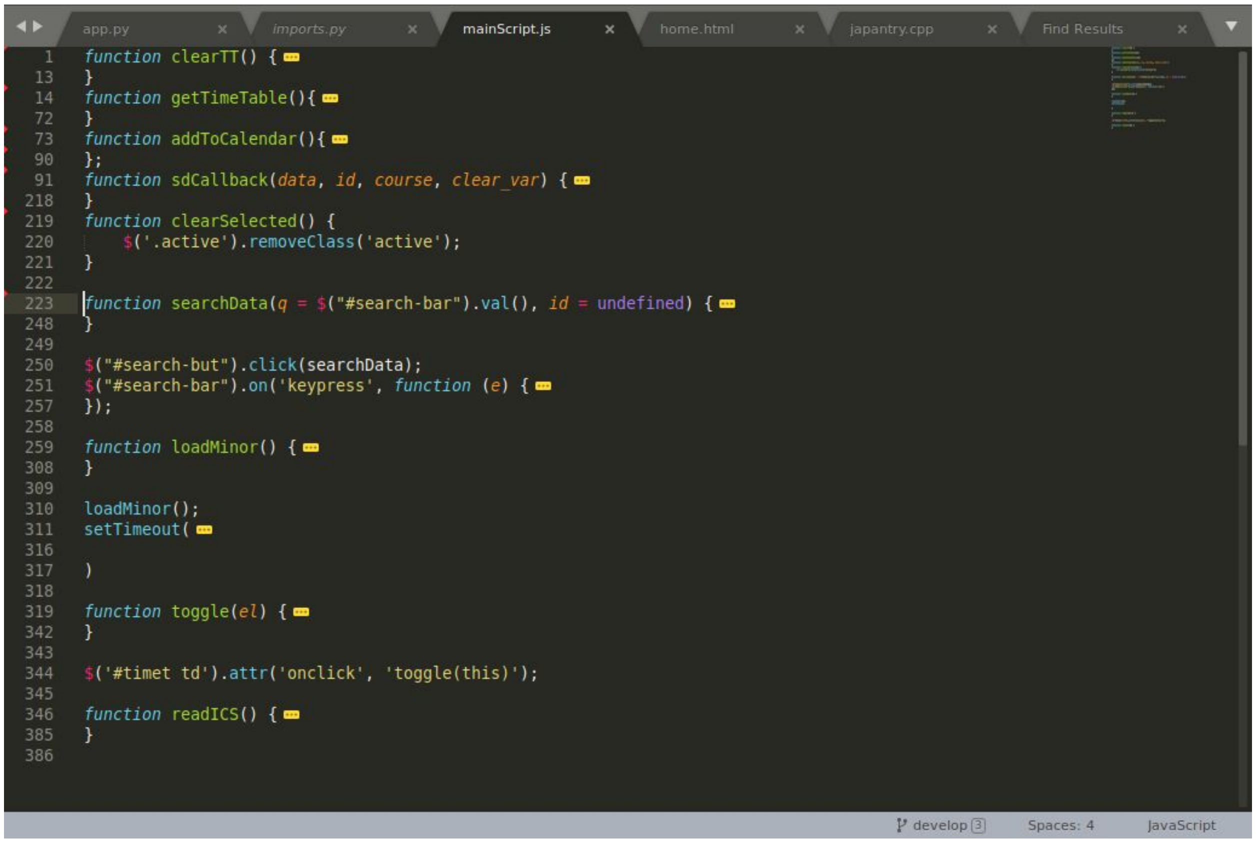


```
from flask import Flask, Response
import json, os, re
import functools
from flask import (
    Blueprint, flash, g, redirect, render_template, request, session, url_for, jsonify
)
from search import searchData
from datetime import timedelta, date
import os
import itertools
import requests
import json
import re
import sys
import collections
from ics import Calendar, Event


DEPT_KEY = 'dept'
WEBSITE_KEY = 'website'
TIMETABLE_KEY = 'timetable'

```

Figure 33. Imports.py

In imports.py as shown above in Figure 33, we have imported different modules we needed for our application, and have imported this file in all other .py files.

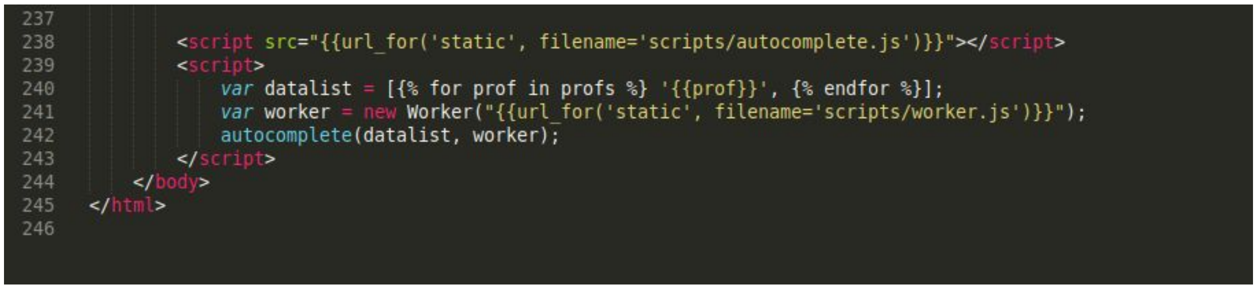
In mainscript.js (Figure 34 below), we have defined various javascript functions which helps us define the behaviour of the student time table page.
Function load_minor is called initially when the page is loaded, which fetches all the courses in the sidebar (line 310).
Function searchData (line 223) is called when the course is clicked on, which checks if the course is already active or not, and sends appropriate argument to the sdCallBack function (line 91) which depending on the argument received, either marks or unmarks the slots of the respective course, and also checks for clash and notifies accordingly.



```
function clearTT() {
}
function getTimeTable(){
}
function addToCalendar(){
};
function sdCallback(data, id, course, clear_var) {
}
function clearSelected() {
    $('.active').removeClass('active');
}

function searchData(q = $("#search-bar").val(), id = undefined) {
}

$("#search-but").click(searchData);
$("#search-bar").on('keypress', function (e) {
});

function loadMinor() {
}

loadMinor();
setTimeout(

)

function toggle(el) {
}

$('#timet td').attr('onclick', 'toggle(this)');

function readICS() {
}
```

Figure 34. Mainscript.js

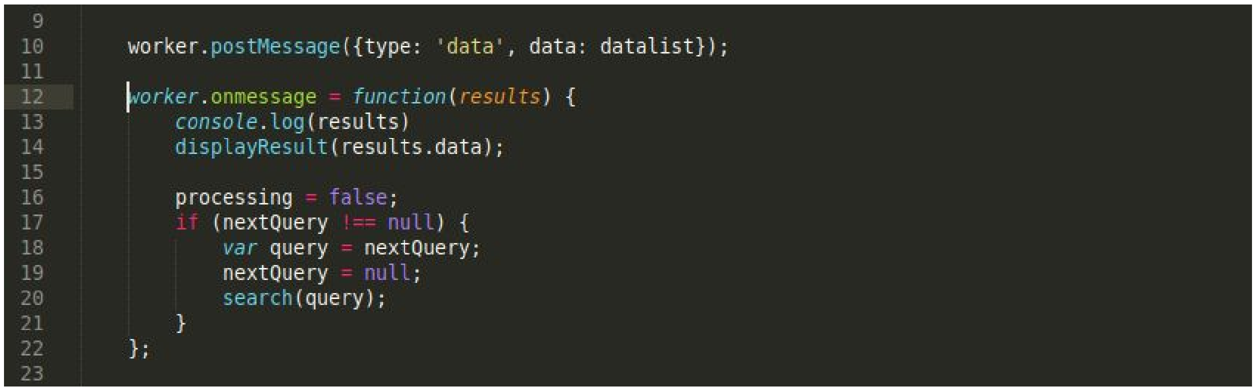
For the faculty time table page, we have two javascript files, *autocomplete.js* and *worker.js*.
As we know, js is a single threaded programming language hence multiple scripts can not run at the same time. HTML5 provides **web workers**, which is a javascript that runs in the background, independently from other scripts.
Web Workers allows us to execute long-running scripts to handle intensive tasks, but without any effect on the UI or on any other scripts that handle how users interact with our application. In our case, while searching for a professor's name, we are assigning scores to strings on the basis of how much they match with the search query, which is being run as a worker script defined in "worker.js" which interacts with the user input handling events.



```

            <script src="{{url_for('static', filename='scripts/autocomplete.js')}}"></script>
            <script>
                var datalist = [{% for prof in profs %} '{{prof}}', {% endfor %}];
                var worker = new Worker("{{url_for('static', filename='scripts/worker.js')}}");
                autocomplete(datalist, worker);
            </script>
        </body>
</html>
```

Figure 35. Main.html

In Line 241 in the above figure, we created a new worker object which runs in an isolated io thread.
PostMessage () is used to communicate between worker and it's parent page.(Line 10, figure 36)



```
9
10      worker.postMessage({type: 'data', data: datalist});
11
12      worker.onmessage = function(results) {
13          console.log(results)
14          displayResult(results.data);
15
16          processing = false;
17          if (nextQuery !== null) {
18              var query = nextQuery;
19              nextQuery = null;
20              search(query);
21          }
22      };
23
```

Figure 36.Worker.js

15. Future Work

1. We can extend this project so that students can register for electives directly from IIIT-B Timetable manager. For this user login and admin dashboard can be created.
2. We can also integrate mess schedules which can be visualized on this platform.
3. Right now, our application has two main pages: student and faculty page. We are thinking of creating a "Rooms" page, which will show empty and occupied rooms at any time by extracting information from student and faculty data. Here, admin can also see the occupied rooms before allocating any room for meetings.

16. Conclusion

This project allowed us to learn a lot of new tools and technologies. We successfully created the IIIT-B Timetable Manager application and deployed it using DevOps tools. We used GitHub, Travis, DockerHub, ELK Stack. We created the pipeline and integrated using the above mentioned tools. We configured the logstash and visualized the logs using Kibana. By using this approach we were able to monitor our project easily.

Our application is a platform which will aggregate the full college timetable of each and every student.

17. Acknowledgement

We would like to thank Prof. Thangaraju for giving us an opportunity to work on this project and our TA Ansh Goyal for his continuous help and guidance throughout the project.

18. References

[1] “Testing Flask Applications: Flask Documentation (1.1.x).
Available at: https://flask.palletsprojects.com/en/1.1.x/testing/”

[2] “Samule Atule. CI/CD: Flask App + GitHub, Travis, & Heroku. (2020).
Available at: https://sweetcode.io/flask-app-github-travis-heroku/”

[3] “Get started with Docker Compose. Docker Documentation.
Available at: https://docs.docker.com/compose/gettingstarted/”

[4] “Daniel Berman. The Complete Guide to the ELK Stack. (June 24, 2019)
Available at: https://logz.io/learn/complete-guide-elk-stack/#kibana”

[5] “Kayan Azimov. Putting Jenkins Build Logs Into Dockerized ELK Stack. (October 19, 2017). Available at: https://dzone.com/articles/putting-jenkins-build-logs-into-elk-stack-filebeat"

[6] “Ben at Codementor. Building a logging system using the ELK stack (Elasticsearch, Logstash, Kibana) | Codementor.io (February 5, 2018). Available at: Building a logging system using the ELK stack (Elasticsearch, Logstash, Kibana) | Codementor”

[7] “Joe Millor. Heroku Log Drains into Logstash | joemillor.me (January 31, 2014). Available at: https://joemiller.me/2014/01/heroku-log-drains-into-logstash/”

[8] “Mwpreston. A newbie’s guide to ELK - Part 3 - Logstash structure and conditionals | blog.mwpreston.net (January 4, 2018). Available at:
https://blog.mwpreston.net/2018/01/04/a-newbies-guide-to-elk-part-4-logstash-conditionals/”

[9] “Google Developers | Google Calendar API. Available at:
https://developers.google.com/calendar/quickstart/python”

[10] “Eric Bidelman. The Basics of Web Workers | html5rocks.com (July 26th, 2010). Available at: https://www.html5rocks.com/en/tutorials/workers/basics/”

[11] “What Slot| Available at : https://whatslot.metakgp.org/.
(https://github.com/metakgp/what-slot)”