VedinkakSa (A Smart Classroom App)

A B. Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Nikhil Kumar & **Siddharth Sharma**
(160101047 & 160101071)

under the guidance of

Dr. Samit Bhattacharya

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**VedinkakSa (A Smart Classroom App)**" is a bonafide work of **Nikhil Kumar** & **Siddharth Sharma (Roll Nos. 160101047** & **160101071**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Samit Bhattacharya**

Associate Professor,

May, 2020 Department of Computer Science & Engineering,

Guwahati. Indian Institute of Technology Guwahati, Assam.

Acknowledgements

Special thanks to

Prof. Samit Bhattacharya, Subrata Tikadar and Ujjwal Biswas

for your Guidance and Support.

Contents

Chapter 1

Introduction

1.1 Background

[1] Affective Computing, (also known as Emotional Computing), is one of the important research areas in the field of Human-Computer Interaction (HCI). It is the study and development of systems and devices that can recognize, interpret, process, and simulate human affects.

It basically involves making the interaction more effective, by taking into consideration, the mental and emotional states of the user. The first and foremost step for designing such a system is to recognize the mental and emotional states of the user. This may be followed by the design and implementation of appropriate interface and interaction that complement and/or change the users' affective/emotional state.

1.2 Overview

[TB19] Using affective computing technology, computers can judge the learners' affection and learning state by recognizing their facial expressions.

In a classroom, the teacher can use the analysis result to understand the student's learning and accepting ability, and then formulate reasonable teaching plans. At the same time, they can pay attention to students' inner feelings, which is helpful to students' psychological health.

Especially in distance education, due to the separation of time and space, there is no emotional incentive between teachers and students for two-way communication. Without the atmosphere brought by traditional classroom learning, students are easily bored, and affect the learning effect. Applying affective computing in distance education system can effectively improve this situation.

1.3 End Goal

[TBT18] We want to make a system which identifies the mental and emotional states of the students, which can then be used by the teacher to tailor their teaching plans.

For this system, we are currently considering a target group of students using small handheld, touchscreen devices. These would include smartphones, and tablets.

We are working towards implementing a digital classroom system which aims to make the learning process more efficient, by adapting to the mental and emotional state of the students, while allowing the teachers to dynamically changing the learning strategies, hence enhancing interest and engagement.

1.4 Organization of The Report

We will start off with a quick introduction of the prior work that has already been done so far on the System. This is followed up by an analysis of the work that we did, followed by instructions for deploying the code.

Chapter 2

Review of Prior Works

So far a very bare bones android app has already been created. The app currently has different modes for both Students and Teachers.

The app initially supports:

- Login Service

  A simple login interface has already been implemented, and supports both teacher and Student Accounts.

- Slide Show Viewer

  So far, we just have a bare bones Slide Show viewer which only supports creating PDF, slides on the local device. As of now, there is no support for server side storage, and also no synchronisation amongst devices.

- State Visualisation Model

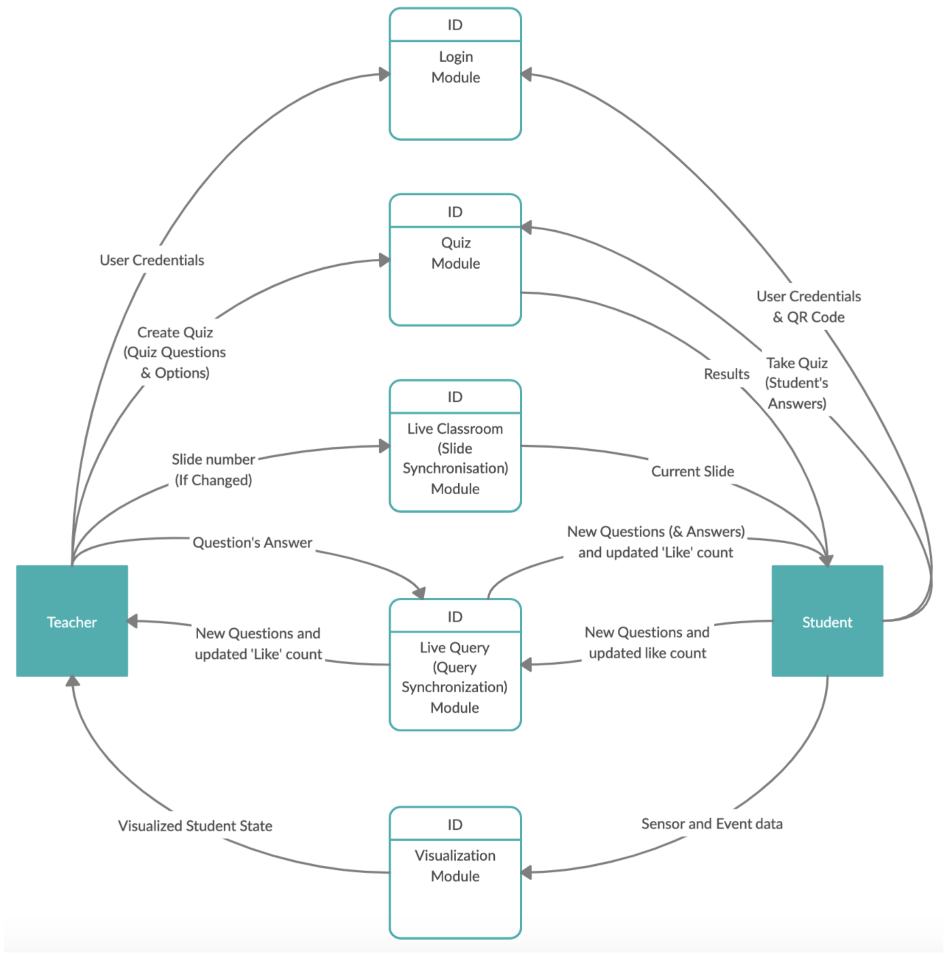
  The initial code also includes the model for evaluating the mental and emotional state (of the student) using the user event data (screen touches and other sensor values).

- Visualisation Dashboard

  A prototype dashboard, containing the algorithms for grid formation and average state calculation is already present.

Chapter 3

System Architecture



ID
Login Module
ID
Quiz Module
ID
Live Classroom (Slide Synchronisation) Module
ID
Live Query (Query Synchronization) Module
ID
Visualization Module
Teacher
Student
User Credentials
Create Quiz (Quiz Questions & Options)
Slide number (If Changed)
Question's Answer
New Questions and updated 'Like' count
Visualized Student State
User Credentials & QR Code
Take Quiz (Student's Answers)
Results
Current Slide
New Questions (& Answers) and updated 'Like' count
New Questions and updated like count
Sensor and Event data

Fig. 3.1 Data Flow for the whole System

The system comprises of a server component (with embedded models), with the internal Database, and Android Clients for both teachers and students.

The whole system can be divided into five different modules:

- Login Module
- Live Classroom Module
- Live Query Module
- Quiz Module
- Visualisation Module

The above diagram illustrates the Data Flow between all these modules and the entities, as they work together in sync.

Now, we will look into the internal structure (data flow) for each of these modules.

3.1 Login Module

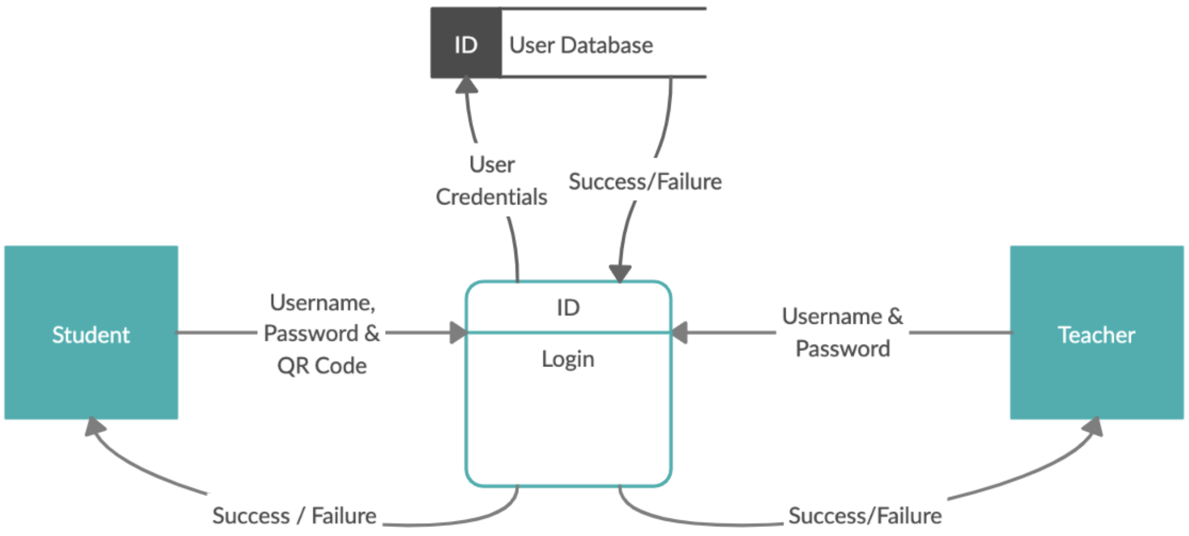


Fig. 3.2 Data Flow for Login Module

The user is required to enter the credentials (and a valid QR code, corresponding to the desk coordinates, in case of students). The app then verifies the user credentials, by sending a request to the user database, and then authenticates the user.

In case of students, the User Database would also get the desk coordinates for the corresponding user.

3.2 Live Classroom (Slide Synchronization) Module

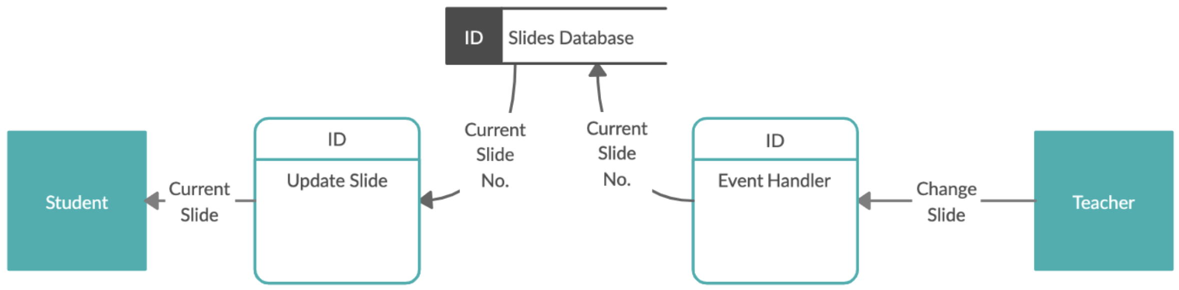


Fig. 3.3 Data Flow for Slide Synchronization module

The Slide Synchronization module basically works to keep the slide on the student devices in sync with that on the teacher device.

This is done with the help of a 'onPageChange' handler which is called whenever the teacher swipes left or right, in order to change the page. This would send the updated page number to the Slides Database, from where it can be 'polled' by each of the student devices. The student devices poll the server after regular intervals, and update the slide on the corresponding devices.

3.3 Live Query (Query Synchronization) Module

The Query Synchronization module basically creates a query dashboard where the students can ask questions (during the class), and the teacher can answer them. The Questions are grouped according to the slide corresponding to them.

The students also have the ability to 'Like' any question in order to have it displayed more prominently. The questions are ranked based on these Likes, with the most liked

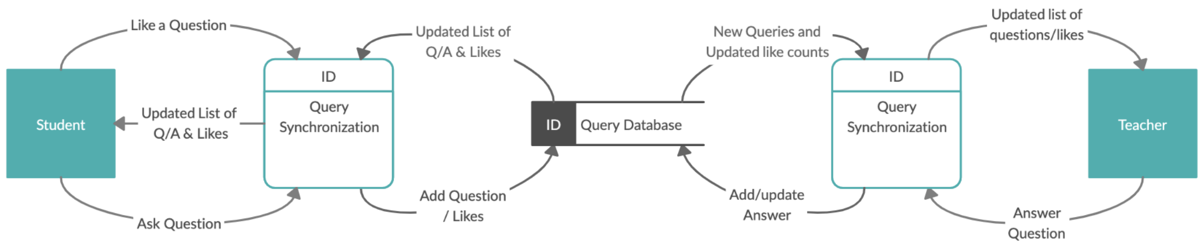


Fig. 3.4 Data Flow for Query Synchronization module

questions (which would correspond to the most common doubts) displayed on the top.

The questions, answers, and the corresponding 'Like' counts are stored in the query database, from where they can be synchronised across the student and teacher devices.

In case of any activity (this includes posting a new question, 'Liking' a question, or answering a question), the device would push the updates to the query database. These updates would be 'polled' at regular intervals by all the other student and teacher devices, thus synchronizing the queries across all the devices.

3.4 Quiz Module

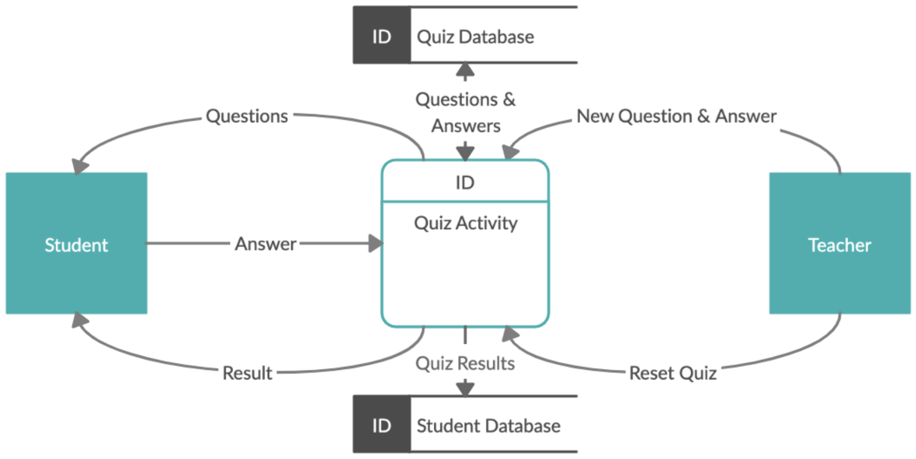


Fig. 3.5 Data Flow for Quiz Module

The Quiz module allows the teacher to make quizzes. The teacher has an option to add questions to the quiz, or create a new quiz from scratch. Once the quiz has been created, the students can then take the quiz, and get their results.

3.5 Visualisation Module

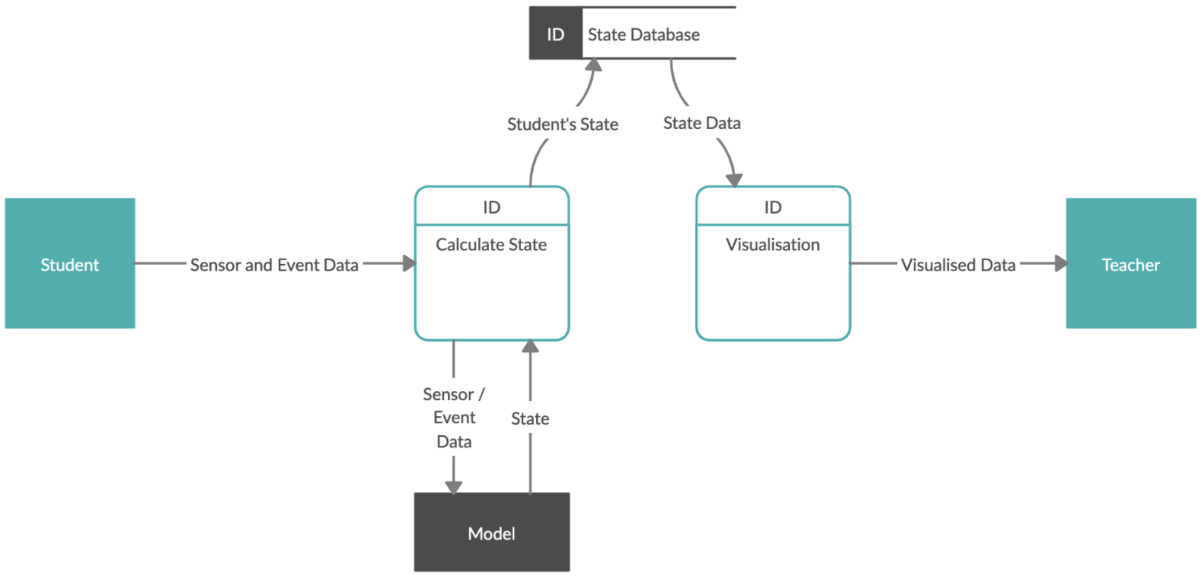


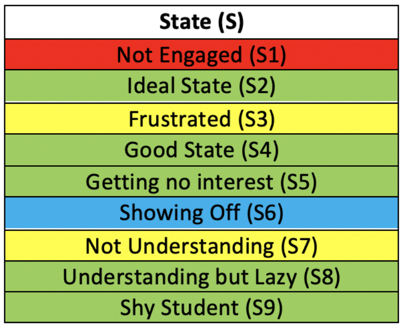
Fig. 3.6 Visualisation Module

The Visualisation module provides the teacher device with a dashboard for viewing the mental states of all the students. This would give the teacher a good idea about the students' sates (mental states) based on their their emotion, involvement, and level of classroom activities.

The student device logs all the event and sensor data (required for calculating the mental state), while the app is active. This includes the values from the accelerometer, Gyroscope, battery, and memory utilization. It also includes events like screen touches, typing speed, and shake frequency, and also activities like asking questions in the class, 'Liking' questions, answering surprise popups, taking classnotes, etc.

[2] We use predefined models to calculate the mental state of the student using these

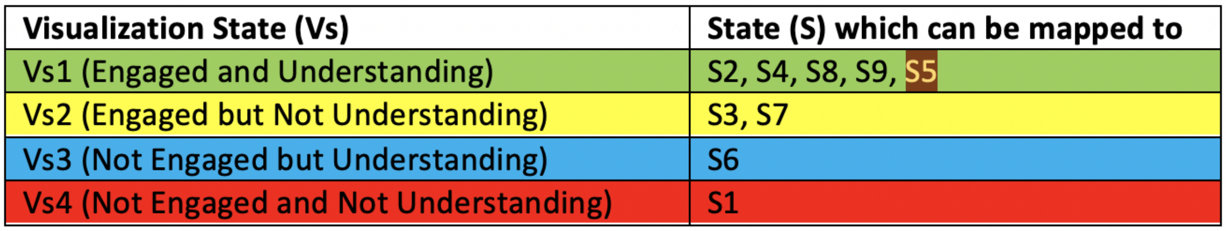
parameters. The mental states are divided into 9 categories:



State (S)
Not Engaged (S1)
Ideal State (S2)
Frustrated (S3)
Good State (S4)
Getting no interest (S5)
Showing Off (S6)
Not Understanding (S7)
Understanding but Lazy (S8)
Shy Student (S9)

Fig. 3.7 Mental states

We then map these 9 states to 4 states, so as to make it easier to observe, each corresponding to different colors in the visualisation dashboard on the teacher device. The mapping to different colors, as shown in Fig 3.8 is: vs1 -> Green; vs2 -> Yellow; vs3 -> Blue; vs4 -> Red.



Visualization State (Vs)	State (S) which can be mapped to
Vs1 (Engaged and Understanding)	S2, S4, S8, S9, S5
Vs2 (Engaged but Not Understanding)	S3, S7
Vs3 (Not Engaged but Understanding)	S6
Vs4 (Not Engaged and Not Understanding)	S1

Fig. 3.8 Visualisation states

These states are then stored in the state database, which is then utilised by the visualisation module to give the teacher a real time view of the student states.

Chapter 4

Analysis of work done

4.1 Introduction

We included the following features into the app over the course of these two semesters:

- Synchronise Slides across teacher and Student Devices
- Create a Live query Dashboard with the following functionality:
  - Allow students to post live queries on each slide
  - Allow teachers to answer the queries live in class
  - Create a 'Like' based ranking system for queries
- Prerequisites for Visualisation
  - Add Classnotes functionality
  - Add Quiz functionality
  - Surprise Popups
- Create a Visualisation Dashboard showing the mental and emotional states of the students in the class, to the teacher.

4.2 Synchronising Slides across devices

To synchronise slides across the teacher and student devices, we came with a client-server model, with the server acting as an intermediary. To maintain this synchronisation, both the teacher and student devices are polls the main server. The teacher device keeps updating the current slide information on the server, and the student devices keep polling the server, at regular intervals, for the same. On receiving the result from the server, the student device updates the current slide to match the teacher slide.

Initially, we tried rerendering the slide after every polling interval, however, after testing it on our test device, we noticed that the screen was flashing constantly (at every rerender), which basically led to a very bad experience for the user.

To fix this issue we kept a record of the current active slide on the student devices and made it a point to rerender only when the current slide is being updated on the teacher's device.

However, as of now, our system assumes that the file exists locally on both the teacher as well as on the student devices. So, for the system to work, the students need to pre-download the files manually on their devices, (in the specific app directory). This, however, would lead to problems, in case the teacher needs to start a new topic, in the middle of a lecture, and the students do not have the file already present in their devices. Also, some of the Android Devices do not ship with a built-in File manager, and so storing the slides in the app directory would require a lot of effort.

To fix this, we resorted to having server-side storage for all the slides. The Slide gets uploaded to the server from the Teacher device. This file is then "streamed" to all the connected student devices. This would also allow students coming late (or in the middle of the class) to get the current slide up and running in no time.

However, since we are trying to look for a scalable solution (for handling large class-rooms), we need to figure out a trade-off between small and long polling time intervals. The small interval would make the synchronisation faster, but at the cost of increased server load, while a longer time interval would decrease the server load, but the synchronisation would now take much longer. We ended up setting the polling interval to 5 seconds.

4.3 Creating the Live Query Dashboard

We wanted to create a platform for students to ask questions and doubts while the lecture is in progress. This would be a community based feed of questions asked by the students. Each student would be able to ask questions, and the teacher would be able to answer them in real time, right on the dashboard itself. The questions and answers would be visible to all the students as well as the teacher.

The first step was to synchronise the list of questions and answers amongst all the students as well the teacher devices. We again used the client-server model, as done during the slide synchronisation, with the server acting as an intermediary.

We started off by creating a centralised database, on the server side, containing all the questions and answers. So basically, whenever a student asks a question (or the teacher answers one), it is pushed to the server, and inserted into the database, from where it can be sent to all the devices as and when required.

To keep an updated list of questions and answers, both the teacher and student devices keep polling the main server for all the entries in the database.

However, since the database can be large, the recurrent data transfer due to polling would require a lot of bandwidth. Thus it would be extremely painful to scale the app to larger classroom sizes. So we needed to figure out a way to optimise our polling requests so as to decrease the response size, and thus decreasing the bandwidth requirement, especially for larger classrooms.

To overcome this, we needed to come up with an alternative way to stream the database entries to each of the devices. So we introduced caching by using a local database inside each device. We maintained a local database in each device, storing the details of all the questions (and answers), along with the corresponding timestamps. Now, only the questions (and answers) asked after the last local timestamp are to be streamed to the each of the devices. This would ensure that each question (and answer) is going to be streamed only once to each device, thus optimising both the bandwidth utilisation, and the server load.

Next, we wanted to implement a community-based ranking system for the questions feed. Initially, we just had a timestamp-based ranking. However, we wanted to somehow prop up the most important (or the most common doubts) so that the teacher can answer them first. So we devised a 'Like' based ranking system, which would allow students to upvote questions they feel are most important. The questions in the feed would be shown on the dashboard according to their rank. This would also gamify the process for the students, which would increase their involvement in the class.

Here again, we need to synchronise the 'Likes' for each question. To do this we annotated the centralised question database with a 'Like count' for each question. The students can now like any question. Liking a question would update its 'Like count' in the main database. Both the teacher and student devices would keep polling the server for the updated 'Like count' at regular intervals, and update the questions dashboard with the new rankings.

The screenshots below show how the revamped slide viewer works together in sync with the Query Dashboard in the student and teacher devices.

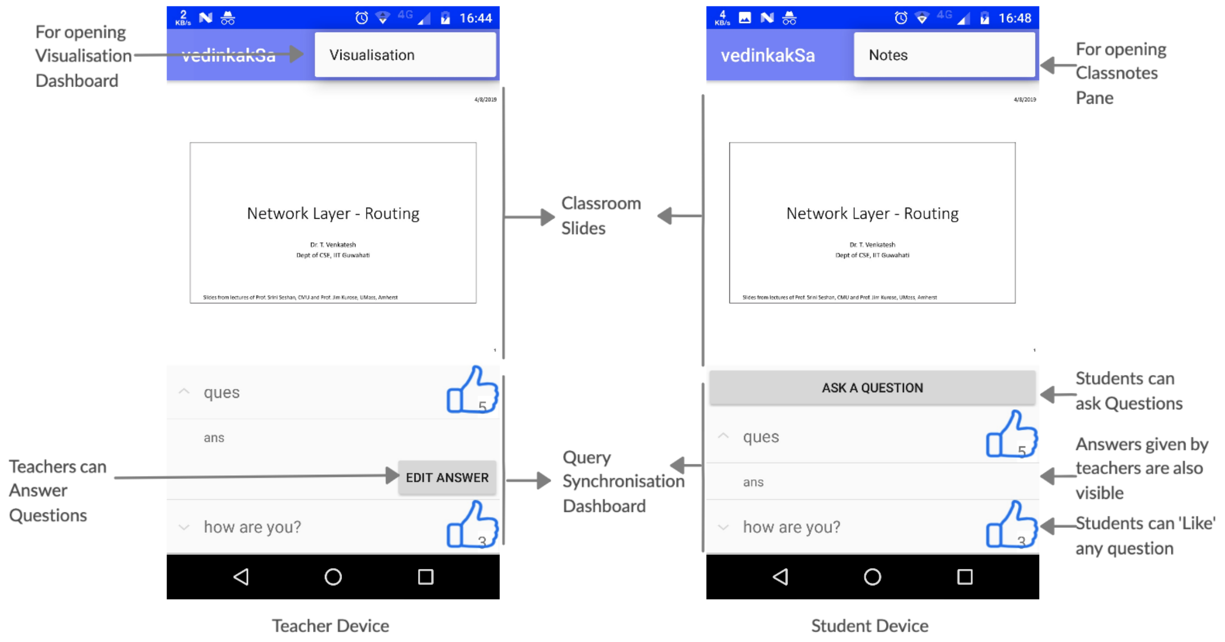

For opening Visualisation Dashboard
vedinkakSa
Visualisation
16:44
Network Layer - Routing
Dr. T. Venkatesh
Dept of CSE, IIT Guwahati
ques
ans
EDIT ANSWER
how are you?
Teachers can Answer Questions
Teacher Device
Classroom Slides
Query Synchronisation Dashboard
16:48
vedinkakSa
Notes
For opening Classnotes Pane
Network Layer - Routing
ASK A QUESTION
Students can ask Questions
Answers given by teachers are also visible
Students can 'Like' any question
Student Device

Fig. 4.1 Live Classroom Demo

4.4 Creating the Visualisation Dashboard

Next up, we wanted to create a dashboard showing the mental state of the students in the class. We already have a model (developed by Mr. Subrata Tikadar) for evaluating the user's mental state based on the user behavior and resource utilization pattern.

This includes the values from the accelerometer, gyroscope, battery, and memory utilization. It also includes events like screen touches, typing speed, and shake frequency, and also activities like asking questions in the class, 'Liking' questions, answering surprise popups, adding classnotes, etc.

For this, we had to first implement all the prerequisite features in the app, which would then be used to log student activity. These features include: Classnotes, Quiz and Surprise Popups.

The Classnotes feature is basically just a scratchpad which allows the students to take notes during the lecture itself. The Classnotes are saved offline in the student's device, and organised according to the slides. These notes can thus be viewed as and when required.

The Quiz module allows the teacher to make quizzes. The teacher has an option to add questions to the quiz, or create a new quiz from scratch. Once the quiz has been created, the students can then take the quiz, and get their results.

The Surprise Popups are a set of dialog boxes that appear at random intervals of time, and stay on screen for 5 seconds. These are used to detect whether the student is paying attention or not.

Once these prerequisites are done, we move on to collection of event and sensor data from the student devices.

We started by logging the sensor values. Once that was done, we moved on to capturing all the touch events. For this, we initially tried to override the Android SDK [2] methods, for motion and touches, for each activity in the app, and added the code for logging inside these methods. However, while the logging seemed to work well, while the app was actively being used, it completely stopped when the active app on the screen changes, or the user opens another app.

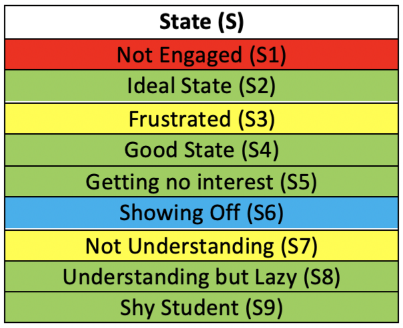
To overcome this issue, we need to somehow use a persistent logging method. So we would need a background service which is constantly running, and logging the data, even when the app is not in use. We tried moving forward with this approach, and created a background service for the same. To log the touch events by a background service [4] , we created an overlay on the device.

Next, for logging keyboard touches, we moved forward with a slightly different approach. We extended the Keyboard class from the Android SDK [3], and added custom support for Logging.

However, we faced a major bug while implementing this. Android actually responds to events in a recursive manner. So in case a user taps on the keyboard, Android first calls an OnTouch event (in the Screen Overlay) followed by an OnKeyListener Method (for the Keyboard). Now if we are logging both the touch events and the keypresses, the overlay tends to "consume" the touch events, due to which the keypress is never generated. So to fix this issue, we need to explicitly call the OnKeyPressListener method in our overlay.

We then used the custom keyboard to extract the typing speed and the maximum length of characters typed without breaks. This was pretty straight forward. We just logged the length of the string typed, the time taken to type it, and then calculated the typing speed based on that.

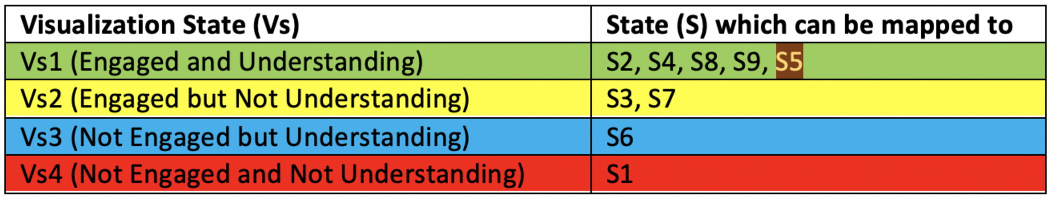
Now that all the logging of sensor and event data was complete, we used predefined models to calculate the mental state of the student using these parameters. The mental states are divided into 9 categories:



State (S)
Not Engaged (S1)
Ideal State (S2)
Frustrated (S3)
Good State (S4)
Getting no interest (S5)
Showing Off (S6)
Not Understanding (S7)
Understanding but Lazy (S8)
Shy Student (S9)

Fig. 4.2 Mental states

We then map these 9 states to 4 states, so as to make it easier to observe, each corresponding to different colors in the visualisation dashboard on the teacher device. The mapping to different colors, as shown in Fig 3.8 is: vs1 -> Green; vs2 -> Yellow, vs3 -> Blue; vs4 -> Red.



Visualization State (Vs)	State (S) which can be mapped to
Vs1 (Engaged and Understanding)	S2, S4, S8, S9, S5
Vs2 (Engaged but Not Understanding)	S3, S7
Vs3 (Not Engaged but Understanding)	S6
Vs4 (Not Engaged and Not Understanding)	S1

Fig. 4.3 Visualisation states

Next up, we moved on to creating the visualisation dashboard for the teacher device.

We have two levels of visualisation. The level 1 visualisation gives an overview of the whole class, while the level 2 visualisation would give a much more detailed description for a small subset of the class. Thus, we first require an algorithms to convert the whole classroom (of any given dimensions) into smaller grids, each of which would have a separate level 2 visualisation.

The level 1 visualisation would basically be a collection of all these smaller grids, with each of these grids showing the average state state of all the students that they contain. For this, we would need to figure out how to average out the states of all the students in the smaller level 2 grids.

A prototype dashboard, containing the algorithms for grid formation and average state calculation was already present. So we started off by polling the data from the state database, and then used the prebuilt algorithms in order to create and render the classroom.

Now that the bare bones dashboard was ready, we moved on to adding basic features and some visual polish. We started off by adding the student images in the level 2 visualisation. We also made the dashboard rerender itself with new data every few seconds so as to provide real time feedback.

We also fixed a few bugs which were impacting how the student images were being rendered.

This is how the visualisation dashboard finally looks like:

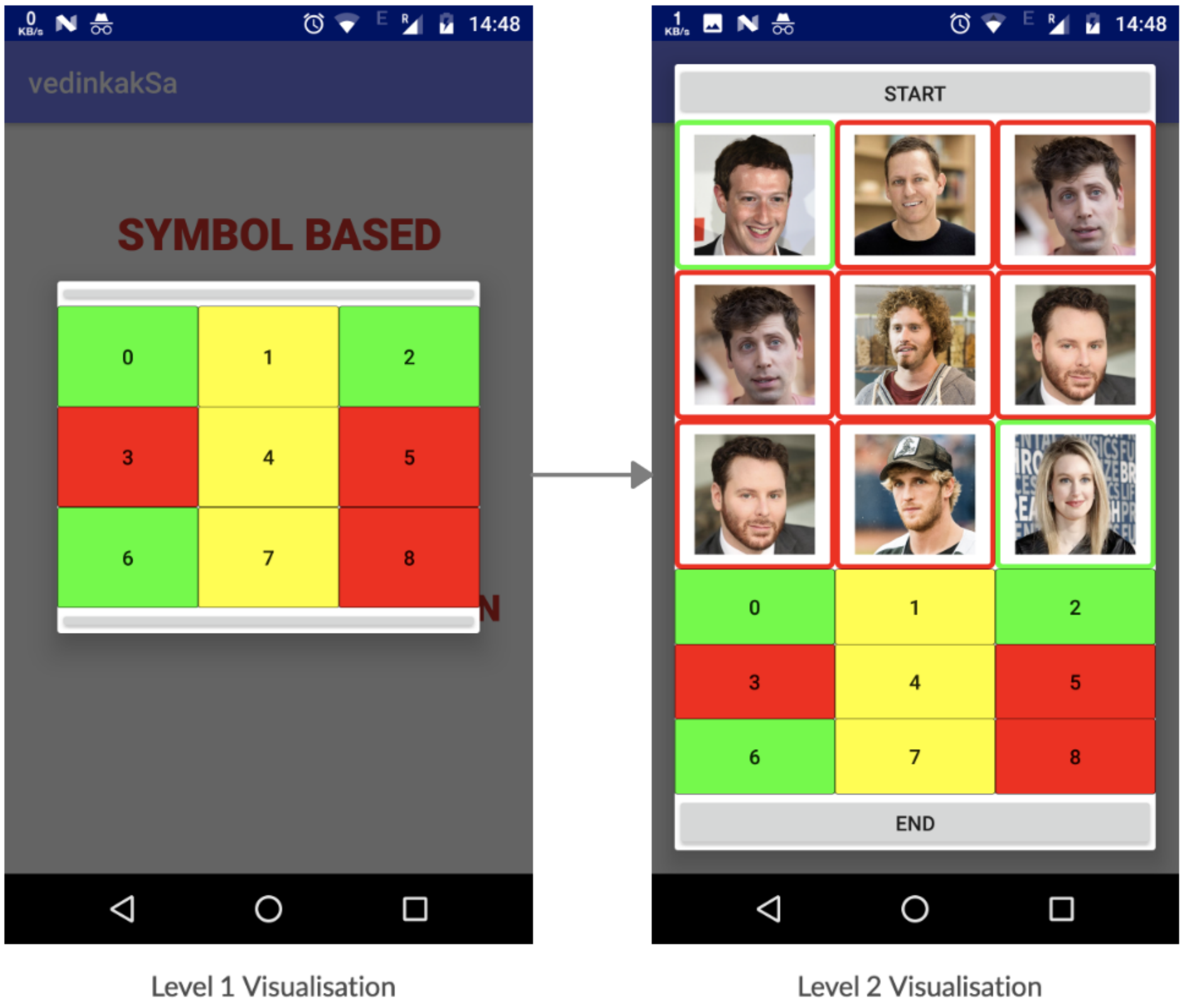


Fig. 4.4 Visualisation Dashboard Demo

Chapter 5

Instructions for Deployment

5.1 Software stack requirements

- PHP 7
- MySQL 8.0.20
- PHPMyAdmin 4.8.5
- Python 3
- Android Studio version 3.4.1
- Java JDK 8
- IDE / Code Editor of choice for working on PHP and Python Code

5.2 Build Instructions

Before you begin, please ensure that both PHP and MySQL have been installed and are running.

Note:

- The specific commands for starting/stopping the MySQL services depends upon the host OS and method of installation. For further info, please refer to the official documentation.
- Before starting the server, please make sure that you have a mysql user with an username ”root” (and no password). Please refer to mysql documentation for information on how to create a new user.

1. Copy the server directory and place it in the PHP site folder in your computer. Note: the site folder path depends upon your OS. Here is the list of default paths according to your OS:
   Linux: `/var/www/`
   MacOS: `/Library/WebServer/Documents/`

2. Now, open `http://localhost/test.php` on any web browser. If it displays “Success!”, then the files have been copied correctly. Else, go back to step 1.

3. Open PHPMyAdmin (`http://localhost/phpmyadmin`), and import (Import -> Choose File) the `database.mysql` file, in the database directory while leaving all options set as default.

4. Next, Create a virtual environment for Python3, using the `virtualenv` or `venv` packages, for creating an isolated environment for running the models Note: Here are the list of commands for creating the virtual environment:
   Using virtualenv:

   ```
   Virtualenv -p python3 venv
   source venv/bin/activate
   ```

Using venv:

```
python3 -m venv myenv
source myenv/bin/activate
```

5. Install all the python dependencies mentioned in the requirements.txt file using the command:

```
pip3 install -r models/requirements.txt
```

6. Run the models using the command:

```
cd models
python3 main.py
```

7. Now, open the project in Android Studio, and allow Android Studio to Index all the files (this may take a while).

8. Try to build the APK. This installs all the required dependencies and Gradle Libraries. You will be prompted to install the missing SDKs and Build Tools along the way. Please follow the prompts.

9. Check IP of your server (the computer having the php files, the database as well as the python models) Note: you can use the `ifconfig` command or GUI (Settings -> Networks) to get the IP.

10. Now replace the IP in the server field with this IP, in the `Constants.java` file (java/iitg/vedinkaksa/Constants.java) in the Android Project.

11. Next, build and install (run -> run `app`) the APK on all the test devices. Note: Do not disconnect the device from the computer in case you want to check the logs (containing all the sensor data and other events)

12. Now, run the app on all the devices, and log in using the credentials provided in the Credentials folder. In case QR Code is required, use the ones provided in the QR codes folder

13. Now, open the `Logcat` pane on Android Studio to view all the logs while the app is running

5.3 Iterating upon the Models

The python State detection models are independent of the Android App. Thus, iterating upon the model would not require you to recompile the Android APK (unless the server IP is changed). However, you are required to restart the Python server.
There are different functions corresponding to each model in the Code (in models/classifier/classifier/main.py). In order to update any model, please change code in the function corresponding to that model.
Before restarting the server, please verify that the path to the training data for each model has been specified correctly in the corresponding functions.
Finally, interrupt the server (`Ctrl + C`), in case it's already running, and then restart the server using the commands given in Step 6.

Chapter 6

Conclusion

We have definitely come a long way, and we have prepared a full fledged working prototype.

The Live Classroom, Query Dashboard and the quiz functionality seems to be working working smoothly. We tested this on 7 devices simultaneously and every module seems to be working as expected, i.e. the slides are being synchronised between the student and teacher devices, and the queries are also being properly synchronised and displayed on all devices, along with their corresponding like counts.

For the state visualisation, we were only able to test this module on one device. In order to test this app, we need data from various student devices. However, since we were not having access to more than one device, we put mock data into the state Database, and confirmed that the visualisation module was showing it properly on the teacher device.

We have also verified that the event and sensor data is being reported correctly. We have also made sure that the mental state received from the model is being mapped to the correct visualisation state.

However, that being said, we were not able to test the complete system on multiple devices simultaneously.

We have added the deployment instructions so that anyone can easily get the whole system up and running.

References

[1] Affective computing: https://en.wikipedia.org/wiki/Affective_computing.

[2] Api reference: https://developer.android.com/reference.

[3] Android documentation: https://developer.android.com/docs.

[4] Services overview: https://developer.android.com/guide/components/services.

[TB19] Subrata Tikadar and Samit Bhattacharya. A novel method to build and validate an affective state prediction model from touch-typing. In David Lamas, Fernando Loizides, Lennart Nacke, Helen Petrie, Marco Winckler, and Panayiotis Zaphiris, editors, *Human-Computer Interaction – INTERACT 2019*, pages 99–119, Cham, 2019. Springer International Publishing.

[TBT18] S. Tikadar, S. Bhattacharya, and V. Tamarapalli. A blended learning platform to improve teaching-learning experience. In *2018 IEEE 18th International Conference on Advanced Learning Technologies (ICALT)*, pages 87–89, 2018.