Student Attendance Tracker

Technical Specifications

Contents

1. Introduction

Student Attendance Tracker aims to track attendance of the students studying at ITU. SAT uses GPS coordinates to track the location of the student and voice authentication to establish the identity of the student. This document aims to explain the technical aspects of the system.

2. Requirements

The requirements of the system from a technical stand point is as follows:

- User Interface – Front End
- Centralized Database
- Voice Authentication System
- Location Tracker
- Middleware

These requirements are explored in depth in the following sections.

2.1 User Interface – Front End:

We needed a user interface which is portable for the students and professors. Hence we chose the mobile platform as the front end. Since every one as a android or I-phone these days, we have chosen android and ios platform devices as our platform and have developed SAT application. This SAT application is our front end.

2.2 Centralized Database:

The mobile devices has to store all the data in a centralized database. We have chosen MongoDb[1] has our data store. Mongo DB is a schema-less nosql database that is flexible and easier to scale. Mongo Db is open source and is free to be used in our project.

2.3 Voice Authentication System:

Since our SAT application uses voice authentication to establish user identity, we need a voice authentication library. We used ALIZE[2] library which is a Open Source Toolkit for State-of-the-Art Speaker Recognition. Alize library has three major components

1. Registration
2. Training
3. Testing

2.3.1 Registration: This registers a student's name and voice to the system.

2.3.2 Training: This phase uses voice samples of students along with the Universal Background Model to model each student's voice.

2.3.3 Testing: This phase uses the knowledge from the training phase to establish the identity of the student from a given voice sample.

2.4 Location Tracker:

We track the GPS coordinates of the center of the class room and radius of the classroom. This lets

us create a circular region around within which we enable students to to give their attendance. We use the GPS system inbuilt with in the phone to decide if the students are in the class room or not. The location tracking system is accurate to 15 meters.

2.5 Middleware:

In order to abstract the details of the database and the voice authentication library, we have a middleware layer which is used by the front end. The middleware layer is a REST api to make it language agnostic and for a cleaner simpler interface. The middleware is implemented using Django Rest Framework[3].

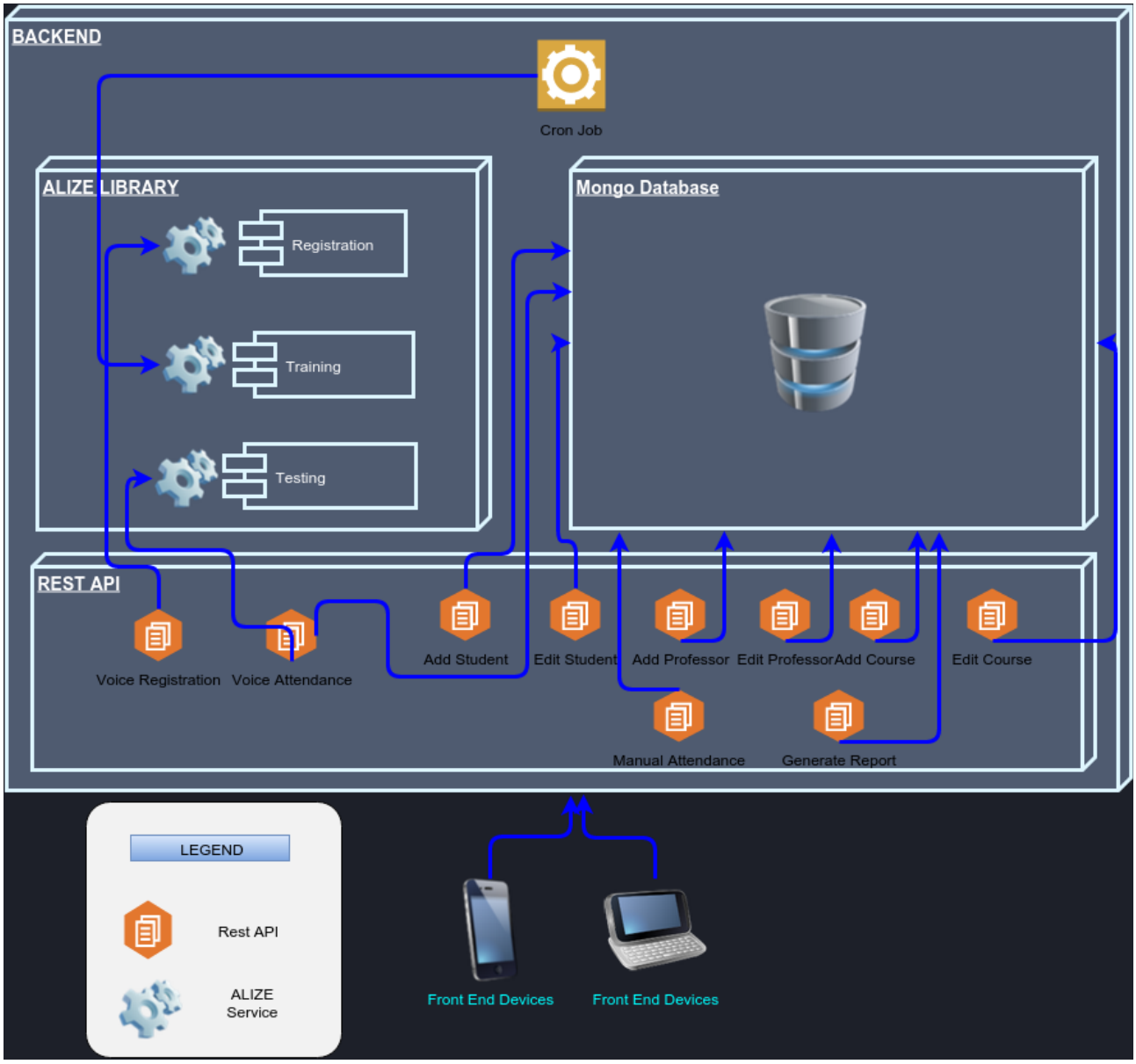

Figure 1: Architecure of the System

3. Architecture:

Please refer to figure 1 for architecture of the system. The Mongo DB, Alize library and Django REST framework together constitute the back end. The REST framework exposes a set of API

which is specified in appendix.

The cron job is used to train ALIZE periodically. The training phase can take a while and hence it is made as an offline process. The front end devices (Android and IOS) can communicate with the back end using a standardized REST API.

4. Voice Authentication:

This section explains in depth the voice authentication process used in our system. As described in section 2.3, the voice authentication is a three stepped process

1. Registration
2. Training
3. Testing

The steps involved in the registration process are

4.1 Registration:

1. System receives a voice sample for a registered student.
2. The voice sample is saved in the file system and the details of the voice sample and the student information is saved in the *training* file.

4.2 Training:

1. All the voice samples are read from the *training* file
2. The voice sample is reconstructed into MFCC coefficients.[4] MFCC coefficients capture the features of a voice and is used in voice recognition.
3. After a series of transformations the MFCC coefficients are transformed into i-vectors[5].
4. The i-vectors of all the voice samples are stored.

4.3 Testing:

1. We receive a voice sample from the student
2. We construct the i-vectors for the voice sample as explained in the testing phase.
3. We compare the i-vectors of the student's voice sample from the training phase to detect if the current received voice sample is from the same speaker or not.
4. We compute a score between 0 to 1, where 1 is the complete match and 0 is complete mismatch.
5. If the score is above the threshold value of 0.7, we authenticate the student else we reject the student's attendance.

5. **Data Model:**

Please refer to figure 2 for the data model. The figure 2 gives all the entities, attributes and the relationships between them.

5.1 Professor:
All information about the professors go in the professors entity.

5.2 Courses:
All information pertaining to courses go in this entity. The attribute *Is weekend* determines if the course is a weekend course or a week day course. If it's a week day course, *day of week* contains the day of the course and if it's a weekend course *specific dates* contain the different dates course is available. The *duration start* and *duration end* gives the start date and end date of the course while

time start and *time end* gives the duration of each course.

5.3 Student:
Information about all the students and the courses they have enrolled is present in students entity

5.4 Attendance Record:
Attendance information of all the students to all the courses they have enrolled is present in the Attendance Record entity. Given any specific duration, the attendance information of any student towards any course can be obtained from this entity.

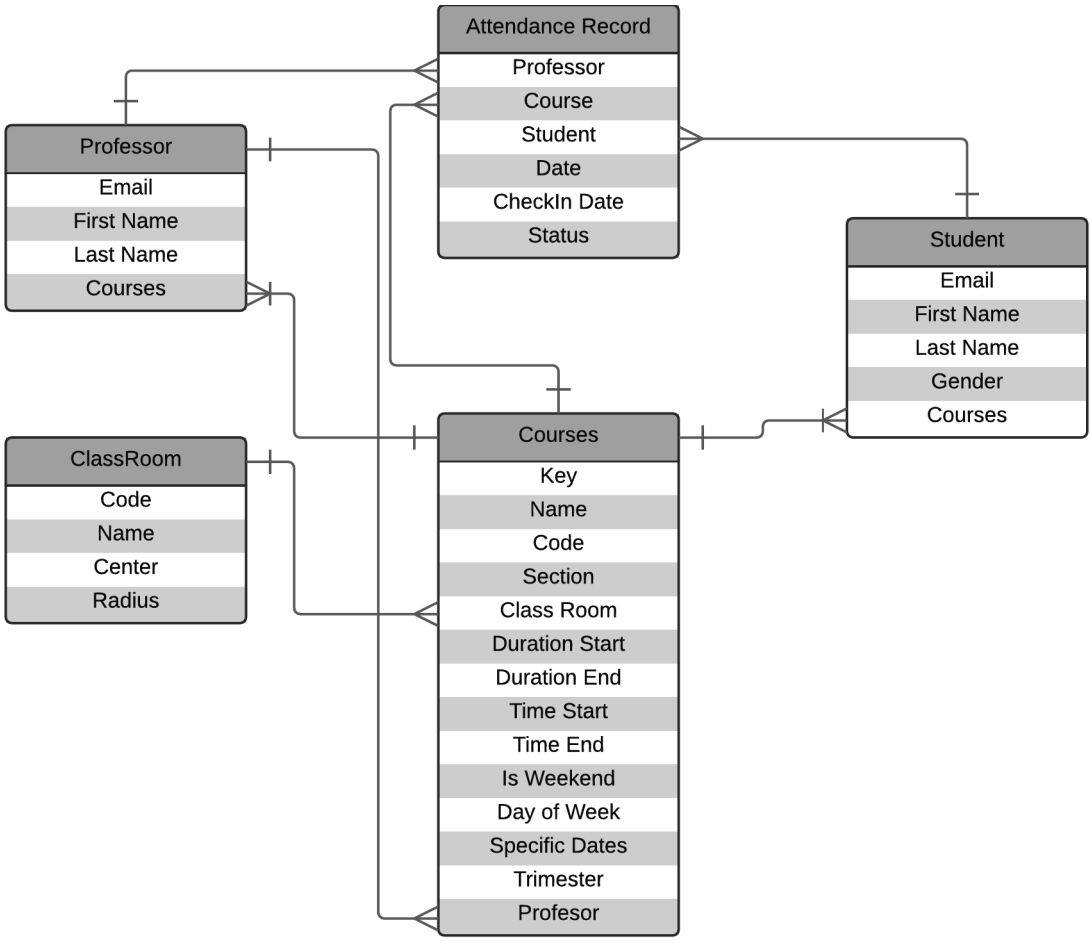

Figure 2: Entity Relationship Diagram

6. Software Development Model:

The Software Development Model adopted in SAT project is hybrid of Rapid Application Development (RAD) and Scrum.

6.1 Rationale:
The three components of our application are

1. Back end system [REST + Mongo + ALIZE]
2. Android Platform
3. IOS Platform.

One can perceive the Android and Ios modules are independent of each other while both of them

depend on the back end for the REST end point. The back-end has no dependencies on Android and Ios platforms.

This made us to wonder if we can parallelize the development between the three modules as there are minimal dependencies. We purposefully had the front end begin the development with UI screens which had no dependencies with the back end.

RAD is a incremental software development model which modularizes the different components of the application and each of the component is developed in parallel. This fit our requirements perfectly but we wanted to make it more flexible. We combined the sprint backlog from the Scrum methodology to make RAD flexible.

Traditional RAD involves developing prototypes from requirements and incrementally refining and fixing issues to achieve at the final product. We used RAD to identify development check points where each team can individually develop their functions which can be tested by QA and we had three such RAD cycles. In each of the RAD cycle, we had a weekly sprint with a common product backlog and a sprint backlog for each module of the RAD cycle. On Completion of a RAD cycle we had a partial deliverable which can be tested by QA.

We wanted to make the development modules as independent as we can because, we were working remotely with different schedules. So in order to increase the efficiency and reduce cost of development we opted for this hybrid model.

The Development Check points are given in table 1.

No	Android	Ios	Backend
1	UI Screens	UI Screens	Set up backend with database and database related REST apis
2	Db Related REST Integration	Db Related REST Integration	Set up ALIZE and voice authentication REST apis
3	ALIZE related REST Integration and other UI issues	ALIZE related REST Integration and other UI issues	Set up Report Generation, cron jobs and fix backend issues.

Table 1: Development Checkpoints

6.2 Challenges and Solutions:

The challenges faced and the solutions used to tackled them are listed in table 2.

No	Challenges	Solutions
1	Not all teams could complete their RAD cycle at the decided time.	We worked around it by prioritizing tasks which blocked other teams so the development never stalled.
2	Some of the back end features developed in RAD cycle 2 had bugs (code/design) which were apparent only in RAD cycle 3, so we won't have early detection of problems.	Fortunately there weren't many bugs and QA did a thorough testing at the end of each cycle, very few bugs slipped through and they were easily fixable.

3	The last RAD cycle had dependencies	We had to communicate more frequently because we had left over features with dependencies in the last RAD cycle.

Table 2: Challenges and Solutions

7. Configuration Instructions:

Please visit https://github.com/sriramrathinavelu/attendanceSWE2015/wiki/Setting-up-Server for instructions on setting up the server.

8. Bibliography

1. https://www.mongodb.org/
2. http://mistral.univ-avignon.fr/mediawiki/index.php/Main_Page
3. http://www.django-rest-framework.org/
4. https://en.wikipedia.org/wiki/Mel-frequency_cepstrum
5. I-Vector Tutorial

9. Appendix

Please visit http://docs.sat.apiary.io/# for viewing the REST documentation