# Design Document

## *orator.io* by Team Gucci


Sung Kim (sunghyun)
William Bhot (willbhot)
Logan Girvin (lsgirvin)
Yina Zhu (yinazhu)
Hao Hu (hhu94)
Joey Li (joeyli96)


# System Architecture

## Modules

### Controllers

**Frontend controller**
The frontend controller will be in charge of communicating with the backend and updating the webpage view. It will also enable actions such as recording or playing audio.

**Backend controller**
The backend controller will listen for requests from the client and call the appropriate procedures. It will be able to create new users, speeches and recordings in the database. It will also use an external API to generate a transcript of a speech's audio and use Analyzer objects to analyze and obtain statistics about the speech.

**Analyzers**
This module is composed of four concrete analyzer classes that are implementing four interfaces, namely PaceAnalyzer, ToneAnalyzer, FrequencyAnalyzer and MLAnalyzer. Each of these analyzers contain an analyze function that will analyze the audio and transcript of a Recording object and populate the Statistics object in it. These four concrete analyzers will all live inside of a general Analyzer object to allow easy analysis of a Recording.

### Models

We will create a few classes to encapsulate our data:

- User - Stores information about a user and contains the speeches associated to that user.
- Speech - Represents a speech a user is practicing. It contains multiple recordings.
- Recording - Represents a single recording. It contains the directory of its audio file, a transcript and a list of statistics about the recording.
- Statistic - Class used to unify its subclasses. It enables its subclasses to be treated in as the same kind of object.
- Pace, Tone, WordFreqDist, MLStatistic - Representation of the different statistics about a recording.
- Word - Class used by WordFreqDist.

### Views

HTML and CSS

The project will be a webapp so the view is simply a combination of HTML and CSS. The view will be modified by the frontend controller.

## Stored data

The system will store data about each user, each speech and each recording. The schema will have the following tables:

```
CREATE TABLE users (
     PRIMARY KEY (user_id),
     user_id INT,
     email VARCHAR(256) UNIQUE,
     name VARCHAR(256),
);

CREATE TABLE speeches (
     PRIMARY KEY (speech_id),
     speech_id INT,
     user_id INT,
     name VARCHAR(256)
);
```

```
CREATE TABLE recordings (
     PRIMARY KEY (rec_id),
     rec_id INT,
     speech_id INT,
     audio_dir VARCHAR(256),
     audio_length INT,
     transcript VARCHAR(MAX),
     neutrality INT,
     happiness INT,
     sadness INT,
     anger INT,
     fear INT,
     ml_score INT
);
```

## Alternative designs

### Analyzer

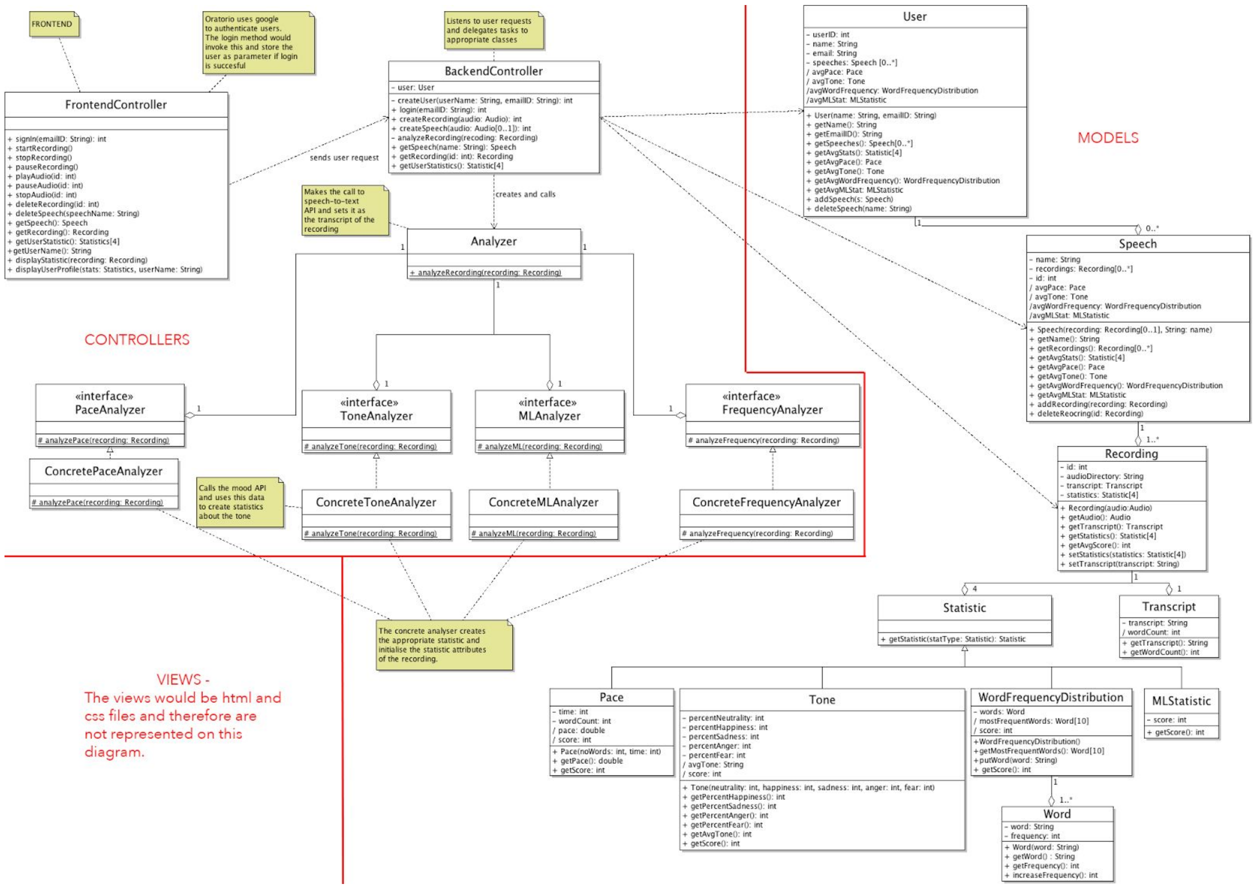
We considered an alternative to the current design of the Analyzer class and its current components. Instead having Analyzer contain four sub analyzer objects, we considered having a method for each analysis instead. We decided against this because the design with the four separate objects allows us to swap out implementations of the analyzers using the factory pattern.

### Speeches and Recordings

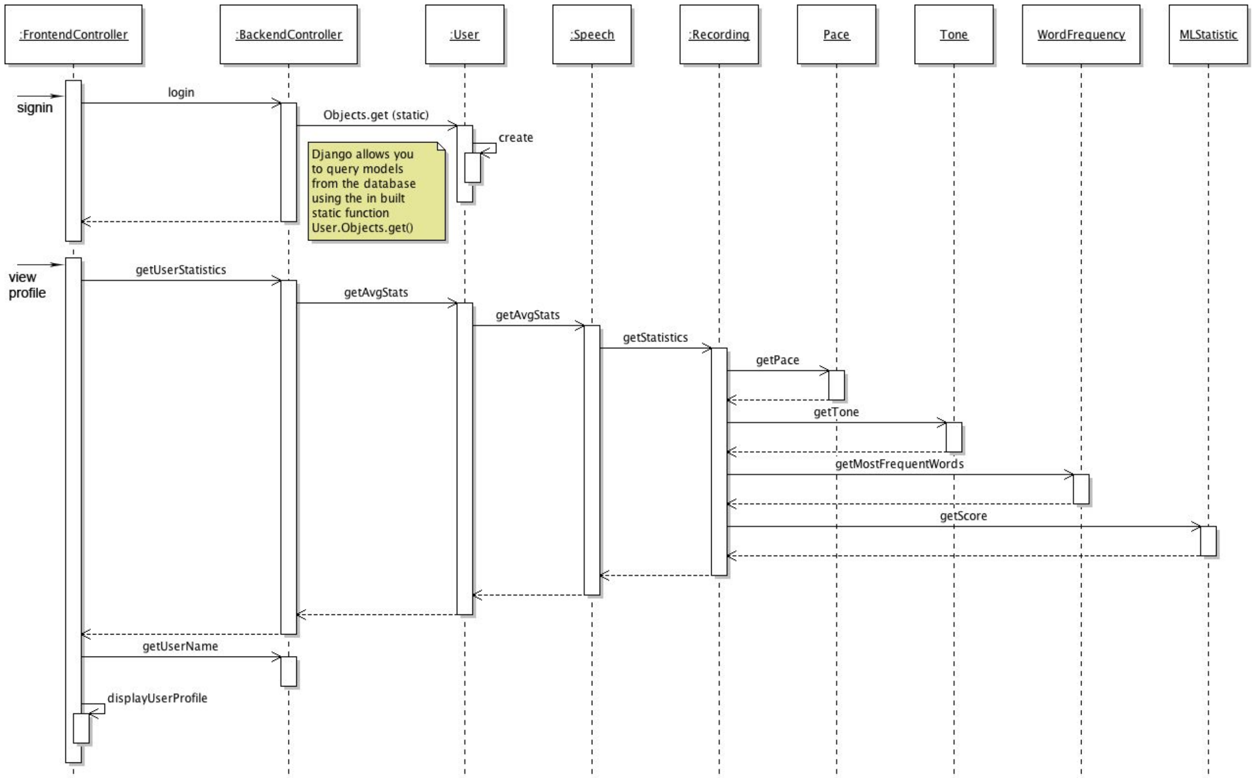
We originally had a design where both Speeches and Recordings would be subclasses of a class called Item. The idea was that this way we could create Recordings completely separately from Speeches. We decided against this as we eventually realized that Recordings is conceptually a level lower than Speeches, and that could lead to confusion.

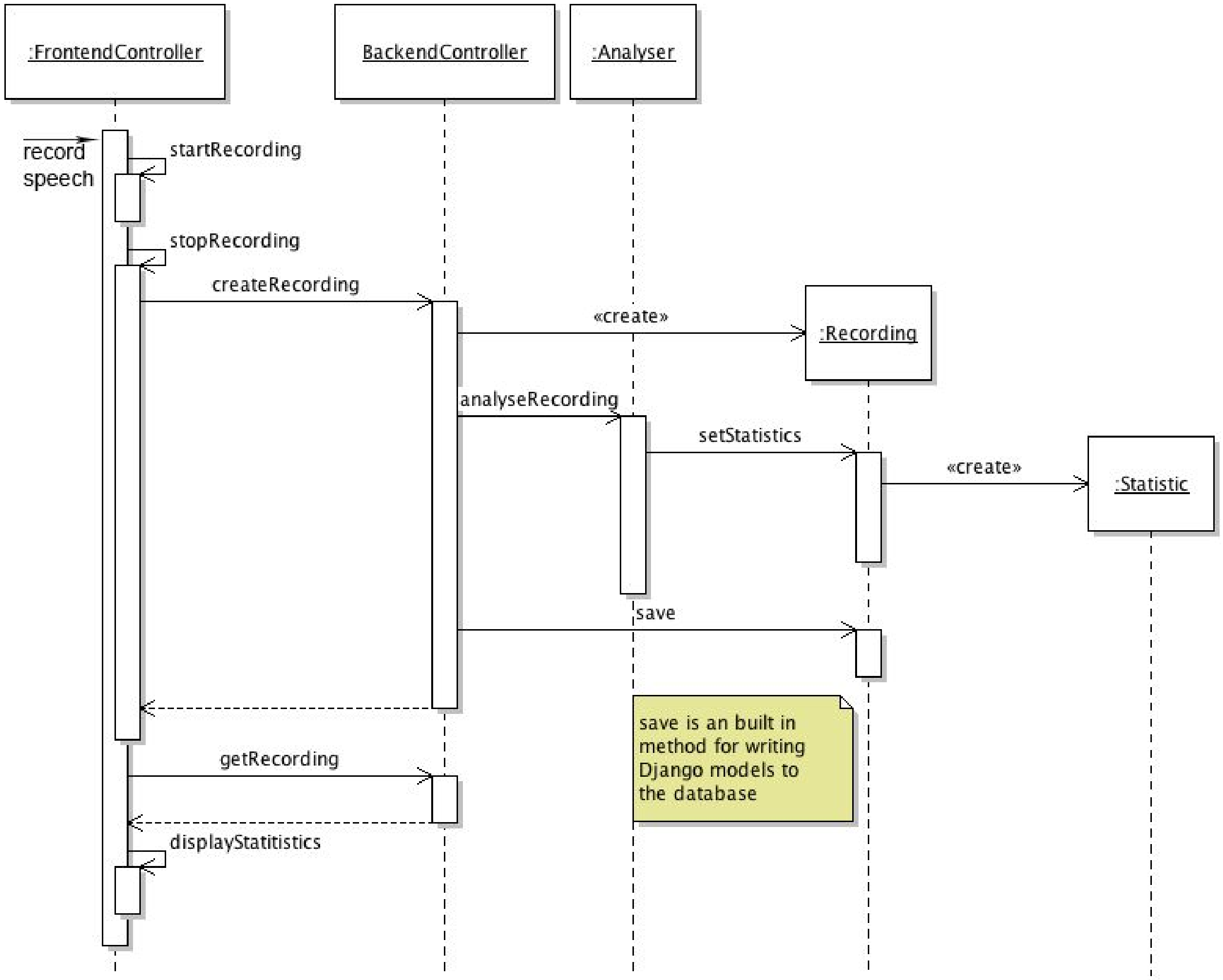
# Diagrams

## Class Diagram



# Sequence Diagrams




:FrontendController
BackendController
:Analyser
record speech
startRecording
stopRecording
createRecording
«create»
:Recording
analyseRecording
setStatistics
«create»
:Statistic
save
save is an built in method for writing Django models to the database
getRecording
displayStatitistics

# Process

## Risk Assessment

This section identifies the top risks to the successful completion of our project:

1.

| | |
|---|---|
| **Summary of Risk** | Speech-to-text may not be accurate in giving the transcript of the user's recording. Since many of our features depend on this transcript (e.g. pace analysis, text analysis), we need to make sure that our method of speech to text is reliable. |
| **Likelihood of occurring (low to high)** | Low |
| **Impact if it occurs (low to high)** | High |
| **Research done on risk** | We have noticed that some speech to text APIs like IBM Watson are not as reliable as others; however, there are many alternatives that seem to be able to reliably retrieve a mostly accurate transcript of the user's audio such as Google's Cloud Speech API and Microsoft's Bing Speech API. |
| **Plan for detecting problem** | We plan to test our speech API with examples of many speeches and their transcripts to make sure that it is robust and reliable. |
| **Mitigation plan should it occur** | We have designed our system architecture in such a way that the speech to text is a modular part of our design. If one of the speech APIs fail, it will be fairly simple to switch it out with another one that works better instead of having to redesign our whole project. |

This differs from our software requirements process as we have a plan for mitigation. Instead of just saying that we will use a different speech API if it fails, we have structured our software architecture to make this method of mitigation easy to implement.

2.

| | |
|---|---|
| **Summary of Risk** | We are attempting to give feedback on a user's speech without any manual process. This means that our project may be subject to giving wrong feedback even if our project is implemented correctly. We want to minimize the risk of giving user's wrong or hurtful feedback. |

| | |
|---|---|
| **Likelihood of occurring (low to high)** | Medium |
| **Impact if it occurs (low to high)** | Medium |
| **Research done on risk** | We looked at the average speech speed and other aspects of a good speech to find out how to give feedback. Even if we do this, we have a moderate chance that the feedback that we give may not be universally useful for all speeches. They may change depending on the tone that the user wants to set for their speech or the content on the speech. |
| **Plan for detecting problem** | We can detect this problem by running our analysis algorithms through a whole bunch of pre-categorized speeches to make sure that our algorithm has less false positives on negative feedback. |
| **Mitigation plan should it occur** | We have made sure that a lot of the feedback given is as objective as possible. We want to tell people how fast they are talking and which words they have repeatedly used. We also have our algorithm less dependant on more volatile parts of our program like mood APIs and more dependant on parts like pace and word analyzer. |

This differs from our software requirements process since instead of being looser on the specifications where it will have more false negatives on giving negative feedback, we are planning to use many speeches to test our algorithm to figure out what the more specific requirements are.

3.

| | |
|---|---|
| **Summary of Risk** | Machine learning data might be hard to find. To integrate machine learning into our service, we need a lot of good (and bad) speeches to train our model on. |
| **Likelihood of occurring (low to high)** | High |
| **Impact if it occurs (low to high)** | Low |
| **Research done on risk** | We have found many statistical machine learning algorithms that we want to try. K-means and linear regression being some of them. K-means might be a good method to cluster similar groups of speeches together and linear regression might be a good method for just figuring out whether the speech is good or bad. |

| | |
|---|---|
| **Plan for detecting problem** | We are going to test our model with a few manually categorized speeches and see how accurate our model is. |
| **Mitigation plan should it occur** | If the model is not reliable, we can either reduce the feedback that the model gives to just "good" or "bad" to reduce the complexity or simply just cut the feature out of the final service. |

We have researched many methods of implementing the machine learning aspect of the project (e.g. k-means, linear regression) from the software requirements specification.

4.

| | |
|---|---|
| **Summary of Risk** | We may need to implement a method of storing many audio files. |
| **Likelihood of occurring (low to high)** | Low |
| **Impact if it occurs (low to high)** | Medium |
| **Research done on risk** | Currently, we are going to stream the audio that we get from the user and directly analyze and send back the feedback. We may run into problems later on if there's a lot of users where there is not enough memory to process many users at once. Since we do not expect these many number of users during our project development, the likelihood of this happening is low, but if our service started to lose data, it would be a bad experience for the user. |
| **Plan for detecting problem** | We should track our memory usage and if it passes a certain threshold while running, it will send out a notification to the group. |
| **Mitigation plan should it occur** | If there is not enough local memory to hold the audio files while we process it, we can have an intermediate stage where the audio files are stored in a database like SQL. We can retrieve the audio file from there when we have space for it and then input the feedback into the same table. |

This differs from our software requirements process as we had not planned out our software architecture back then, but now that it has been designed, we can look into different methods of storing audio and the flow of the data and data analysis.

5.

| | |
|---|---|
| **Summary of Risk** | If we implement the subjective APIs wrong, we might not be able to tell through unit tests whether it is actually wrong. |
| **Likelihood of** | Medium |

| | |
|---|---|
| **occurring (low to high)** | |
| **Impact if it occurs (low to high)** | High |
| **Research done on risk** | Some of the APIs that we are using do not have a “correct” output that we can test using unit cases. We won’t know for sure whether we accidentally negated some mood values since we do not know the exact mood value it would have produced in the first place. |
| **Plan for detecting problem** | We can use integration testing to make sure that outputs are consistent with each use. We can also test end-to-end and test it directly using our own speeches, for example, we call our algorithms on extreme examples of happiness during the speech that uses positive language and sees if our algorithm detects happiness. |
| **Mitigation plan should it occur** | We can weigh our algorithm so that during its mood and tone calculations, it not only uses the subjective API to come to the conclusion but also uses the language in the transcript to see if it lines up with its original prediction. |

This differs from our software requirements process since we had not thought about what how we would test the correctness of a subjective API and creating this risk mitigation tactic made us discuss different methods of analysis.

## Project Schedule

The Project schedule (on the next page) roughly correlates to the tasks needed to complete, and number of developers, per week. It is assumed that each task includes the unit tests for that specific implementation of a class or method. Quality Assurance is used as a catch-all, representing both assurance that people are testing and that the tests are catching bugs, and additional help with bugs or features that may be causing problems, as deadline slipping will occur without developer assistance. Machine Learning represents research, algorithm writing, and training of a model to correctly identify good speeches from bad ones.

| Week 4 | Week 5 | Zero Feature Release | Week 6 | Week 7 | Beta Release | Week 8 | Feature Complete Release | Week 9 | Week 10 | Release Candidate |
|---|---|---|---|---|---|---|---|---|---|---|
| Front End Template Data | | | Speech to Text | | | Text Analsyis | | Text Analysis | | |
| Front End Mic Behavior | | | Manage Users | | | Pace Analysis | | Pace Analysis | | |
| | Data Model | | Manage Recording | | | Tone Analysis | | Tone Analysis | | |
| Front End Menu Behavior | | | Machine Learning | | | Machine Learning | | Machine Learning | | |
| | Front End - Back End Stubs | | Quality Assurance | | | Quality Assurance | | Quality Assurance | | |
| | Automation | | Machine Learning | | | Machine Learning | | Machine Learning | | |

## Team Structure

The team is mostly uniformly structured, with divisions of sub teams for specific modules and roles. Due to the size of the modules, team members will be switching from completed modules to other sub teams. Sub teams primarily communicate through their sub team channel on slack, and branch of the same sub team branch. The Project Manager will be monitoring the overall team progress, and modify the composition or prioritization of each sub team. Regardless of sub team, each member is to document and write tests for their own code. The team also communicates through a team wide slack channel, and meets weekly with a minimum of Tuesdays at 3:30pm.

The Front End sub team is responsible for the webapp interface the client will use. Since this task is front facing, it will have to be mostly complete by the zero feature release. Thus, This sub team will mostly write templates and javascript that calls stubs in the back end, to be implemented later.  After the zero feature release, this team will be released to other teams, with occasional fixes to protocol or data-model changes.

The Back end sub team will be active throughout the lifecycle of this app, working on data analysis and storage. They will implement the main server functionalities of creation and retrieval of users and recordings, as well as analysis functionalities such as basic speech analysis, and external API calls.

Lastly, the sub team of Machine Learning will oversee the specific speech analysis module utilizing machine learning. This includes research into current working speech models, writing the train and predict functionality, and training a model to critique speeches.

A list of team members, and their sub teams are shown below:

| Name | Sub Teams |
|---|---|
| Joey Li | Project Manager, Front End, Back End |
| William Bhot | Front End, Back End |
| Hao Hu | Back End |
| Logan Girvin | Front End, Machine Learning |
| Yina Zhu | Back End |
| Sung Kim | Back End, Machine Learning |

# Test plan

## Unit test strategy:

Coverage/purpose:

We use unit testing in each module (or class) to check if the code is doing what it is expected to do, and can handle edge cases and exceptions as expected. The test for integration between modules will be covered in system test strategy.

How to develop test:

We would like to take advantage of both white box and black box testing. Since we are going to work on separate tasks, we can have a person who have written the code and a person working on other tasks to each come up with test cases for a module. In this way we are utilizing both testing strategies to avoid bugs maximally.

Frequency:

We use unit testing after any changes of the code in the module to ensure the correctness. We have not discussed about utilizing a continuous integration system yet, but we might decide to use one after further discussion.

## System test strategy:

System testing is divided into two parts: integration and performance.

### 1) Integration test:

Test coverage/purpose:

Since we are using multiple APIs (Google API, Speech-to-text API, Tone analyzing API) in our project, it is essential to test integration. Beside utilizing APIs, we can also use integration test to check other features requiring integration, such as internet, database and log error handling.

How to develop test:

We are using stubs (input) and mocks (output) to communicate with the API. For instance, we can create a stub that feeds an audio recording to Tone analyzing API, and use a mock to receive the output, check if it is as expected and assert the result to a test.

Frequency:

We test integration after changes of code between modules to ensure the correctness of integration between modules.

### 2) Performance test:

Target coverage/purpose:

For performance, we are mainly testing the runtime usage and the memory usage.

How to develop test:

Runtime usage:
We can compare the performance based on times to load a web page and to respond to a request. We can also test which parts of the code cost the most time and what paths are used to reach them, and discuss whether we can optimize them.
Memory usage:
We can test what objects cost the most space and their usage to see if it is possible for optimization.
Frequency:
We test the performance after we have finished the basic implementation and can look into optimization.

**Usability test strategy:**
Test coverage/purpose:
Usability test helps to test if the the website is usable as described.
How to develop test:
In this test, we will have users who have no knowledge of our implementation to use the website according to the user guides we provide to them.
Frequency:
We test usability whenever we finish our development for a single release to check if there is anything else that we have not taken into consideration.

## Documentation plan

Most of our implementations are in the back end and our UI is quite simple and clear, so it would be enough to use integrated help text (such as a "?" button) throughout the UI to inform users with the functions of the product. If we have a chance to add more stretch features to the basic ones and the UI becomes more complicated, we will also use a written manual (a user guide) to walk users through the web pages.

## Coding style guidelines

We are using Python and JavaScript for our project, so we will follow PEP8 and Google JavaScript Style Guide.
PEP8 Descriptions:
https://www.python.org/dev/peps/pep-0008/
PEP8 Checker Installation:
https://pypi.python.org/pypi/pep8
Google JavaScript Style Guide:
https://google.github.io/styleguide/javascriptguide.xml