Qualitative Use Case Requirements

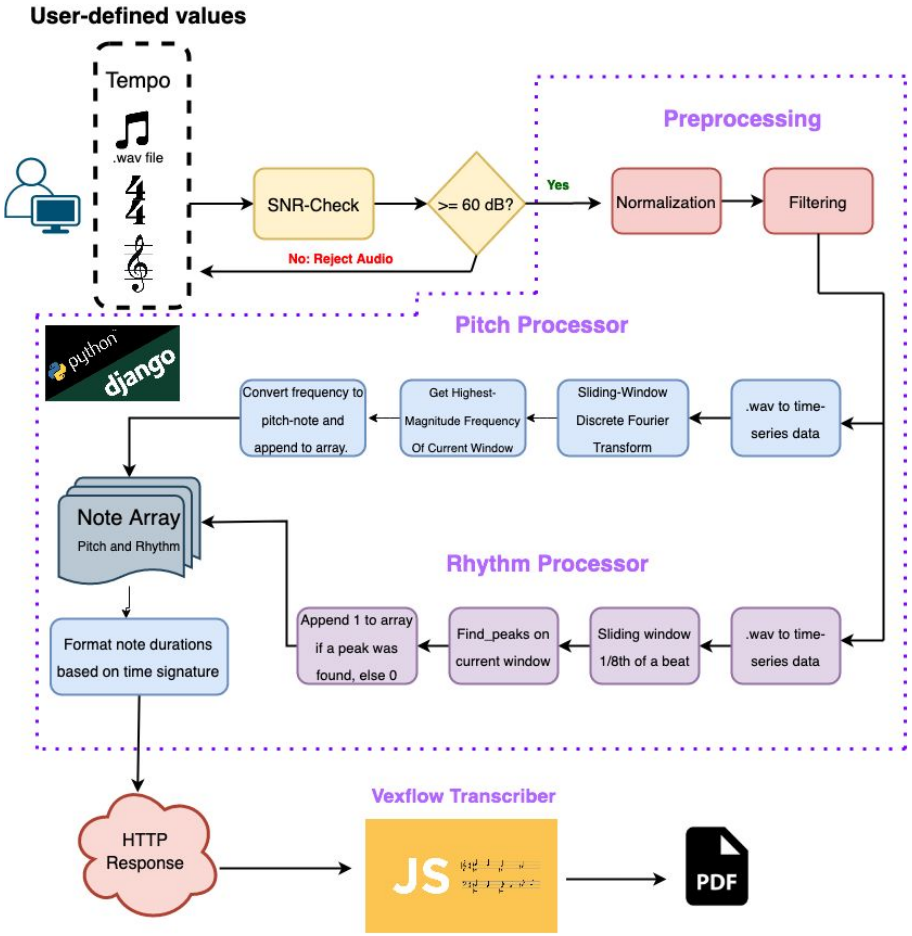
- Getting started in the music field, whether as a professional and amateur, requires money and time
- Inconvenient for younger low-resource people to write/find sheet music
- Added restrictions including:
  - Reject audios of low SNR
  - Monophonic Piano audio files
  - User-focused output requirements for pitch processor

Quantitative Use Case Requirements

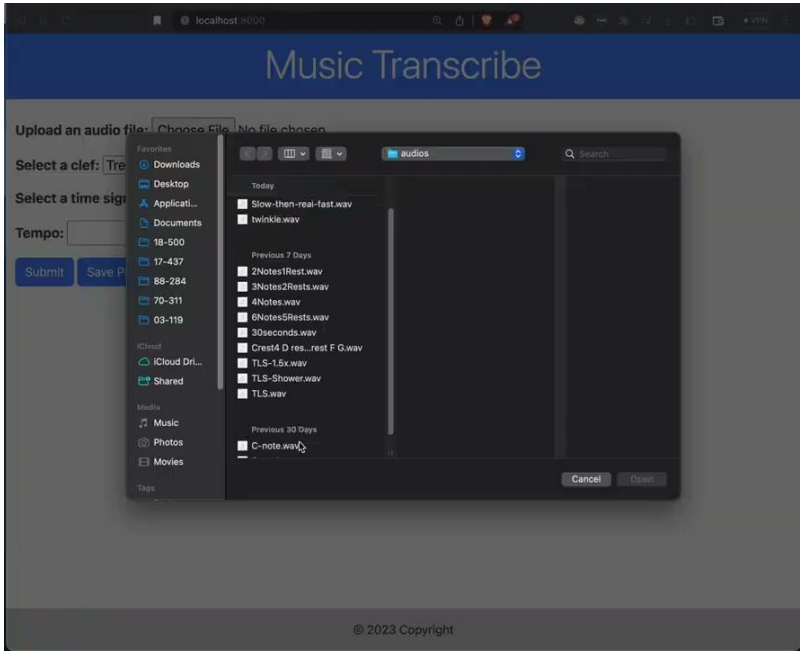
- User must input audio signal with SNR >= 60 dB
- Monophonic sound from a piano with reference A4 = 440 Hz
- Target 90% pitch accuracy
  - Based on user testing feedback
- Pieces of time signatures ¾, 4/4, 2/2, ⅜, 6/8
- Tempo range 60-100 bpm.
- Ensure each transcribed note is <= ⅛ of a beat of actual length

System Implementation & Changes

- Front End
  - Option for user to select tempo, otherwise system auto-detects (*librosa* to detect the tempo)
- Audio Input Validation System
- Frequency Processor System
- Rhythm Processor System
- Transcriber System
- Note Formatting System
  - make data readable by front-end


User-defined values
Tempo
.wav file
SNR-Check
>= 60 dB?
Yes
No: Reject Audio
Preprocessing
Normalization
Filtering
Pitch Processor
python
django
Convert frequency to pitch-note and append to array.
Get Highest-Magnitude Frequency Of Current Window
Sliding-Window Discrete Fourier Transform
.wav to time-series data
Note Array
Pitch and Rhythm
Rhythm Processor
Format note durations based on time signature
Append 1 to array if a peak was found, else 0
Find_peaks on current window
Sliding window 1/8th of a beat
.wav to time-series data
Vexflow Transcriber
HTTP Response
JS
PDF

Complete Solution



Test, Verification & Validation - Pitch Processor

- Ensure pitch processor of a known piece (e.g. C scale) contains ONLY expected notes
- Run on large inputs sets to ensure pitch detection is time-invariant
- Passing: 95% accuracy

Test, Verification & Validation - Rhythm Processor

- Make sure that it outputs the right amount of notes and rests.
  - Consecutive ones would represent a note being played
  - Any zero would represent no note being played at that instant.
- Test cases use regular patterns, e.g. (note, rest, note) or (note, note, note, rest) which can be automatically detected
- Passing test: 90% accuracy

Test, Verification & Validation - User Feedback

- Survey musicians with audio signal and musical transcription.
- Musicians assign score 1-10 based on how well transcription represents audio, in terms of pitch and note duration.
- Low-Scoring/Failed Cases
  - Ask musicians for specifics on why
  - Manually examine signal and transcription

Test, Verification & Validation - Results

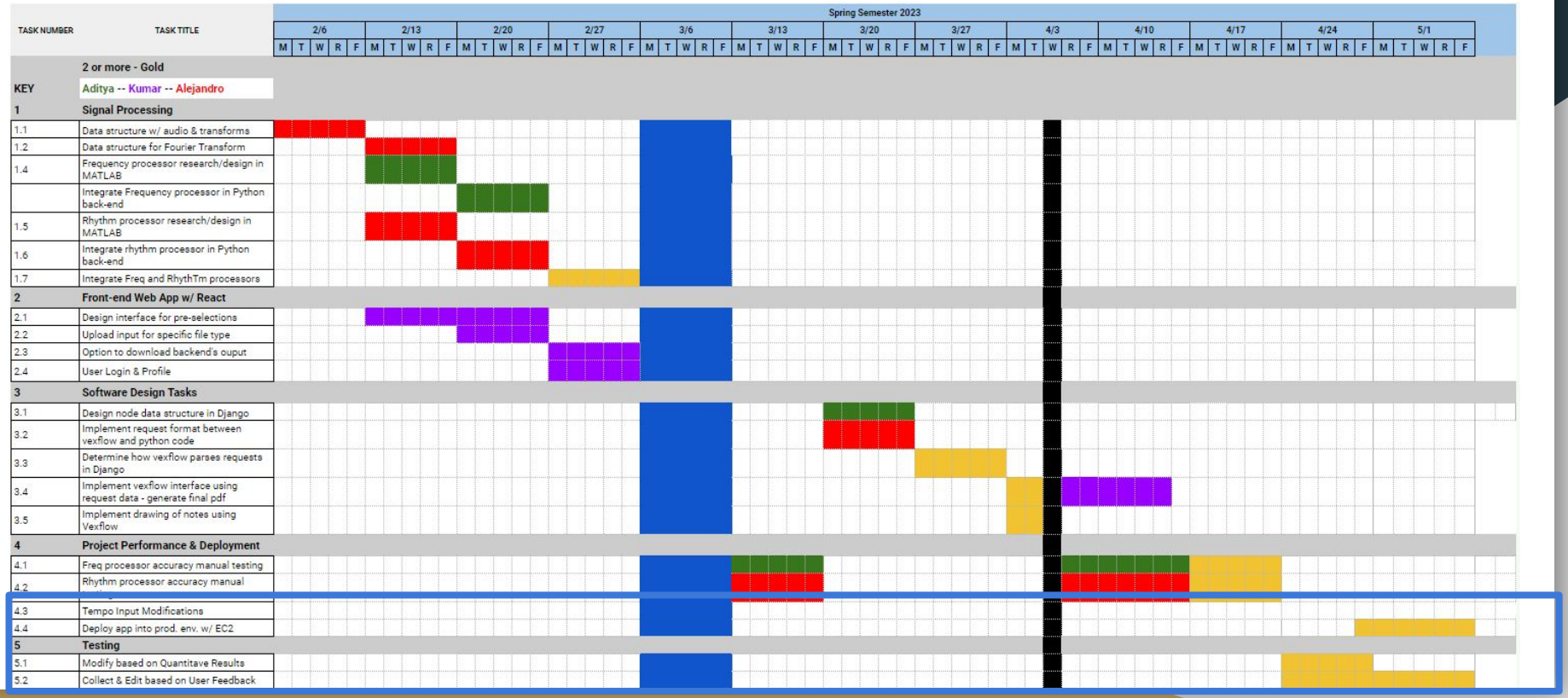
Song	Speed	Pitch accuracy	Onset accuracy	
C scale	60 bpm, user-defined	**100%**	**100%**	
Ascending/Descending C scale	Varying	**75%**	**73%**	
Mary had a little lamb	Undefined, moderate	**100%**	**72%**	

- **Target pitch accuracy: 95%**
- **Target onset accuracy: 90%**

Design Trade-offs

- **Requiring consistent tempo**
  - If audio tempo exceeds 50% faster than slowest tempo, notes are not detected.
  - Set requirement of consistent tempo for accuracy.
- **Limiting to ½ beat**
  - Attempting to determine quarter-beat or lower results in failed tests in pitch and rhythm processors
- **SNR detection**
  - Reject audios with SNR < 60dB for better accuracy of transcription
  - May be frustrating for users who have audios with low SNRs.

Project Management



TASK NUMBER	TASK TITLE
	2 or more - Gold
KEY	Aditya -- Kumar -- Alejandro
1	**Signal Processing**
1.1	Data structure w/ audio & transforms
1.2	Data structure for Fourier Transform
1.4	Frequency processor research/design in MATLAB
	Integrate Frequency processor in Python back-end
1.5	Rhythm processor research/design in MATLAB
1.6	Integrate rhythm processor in Python back-end
1.7	Integrate Freq and RhythTm processors
2	**Front-end Web App w/ React**
2.1	Design interface for pre-selections
2.2	Upload input for specific file type
2.3	Option to download backend's ouput
2.4	User Login & Profile
3	**Software Design Tasks**
3.1	Design node data structure in Django
3.2	Implement request format between vexflow and python code
3.3	Determine how vexflow parses requests in Django
3.4	Implement vexflow interface using request data - generate final pdf
3.5	Implement drawing of notes using Vexflow
4	**Project Performance & Deployment**
4.1	Freq processor accuracy manual testing
4.2	Rhythm processor accuracy manual
4.3	Tempo Input Modifications
4.4	Deploy app into prod. env. w/ EC2
5	**Testing**
5.1	Modify based on Quantitave Results
5.2	Collect & Edit based on User Feedback