Title page – template will be provided

**! Acknowledgements**

**! Abstract / summary**

! One page maximum

! Should be a complete summary of your work

! Should make sense independently from rest of report

**Table of contents**

**!Introduction**

! Context for report

! Motivation for your work

**!What others have done chapter(s), e.g.**

! Background research and/or

! Literature review

**What you have done chapter(s**), e.g. might include some / all of

! Requirements capture / analysis – what your system should do

! Design – how you went about your work

! Implementation – practical techniques, problems, solutions

! Testing and/or evaluation – how well your solution worked

**! Discussion / conclusions**

! Critical analysis

! Honest appraisal

**! Further work**

! What more you would do if you had time

**References / bibliography**

**!Appendices**

! More detailed material that is not crucial to understanding of main message(s)

! Detailed experimental results

! Copy of questionnaire / interview script

Title page

Acknowledgements

Ppl to thank:

Supervisor

Friends that helped test

Father for passion in guitar & test

Abstract

The aim of this project is to create a learning aid for guitarists that use guitar tabs as a learning resource.

Table of Contents

Introduction 5

Background Research 6

About Guitar learning 6

Survey of similar apps 7

App 1 7

App 2 8

Background of tech 9

Requirements capture / Specification 10

Design 11

Guitar App 12

Chord Library 14

IMPLEMENTATION 15

Creating a web app 15

The home page 16

Guitar App: Inputting a Tab 17

Guitar App: Main app page 18

Component positioning 18

Arrow key functionality 18

Implementing Sound 18

Play button function 18

Chord Library 18

User Feedback 19

User 1 19

User 2 19

Improvements 19

Additional Features 19

# Introduction

Context

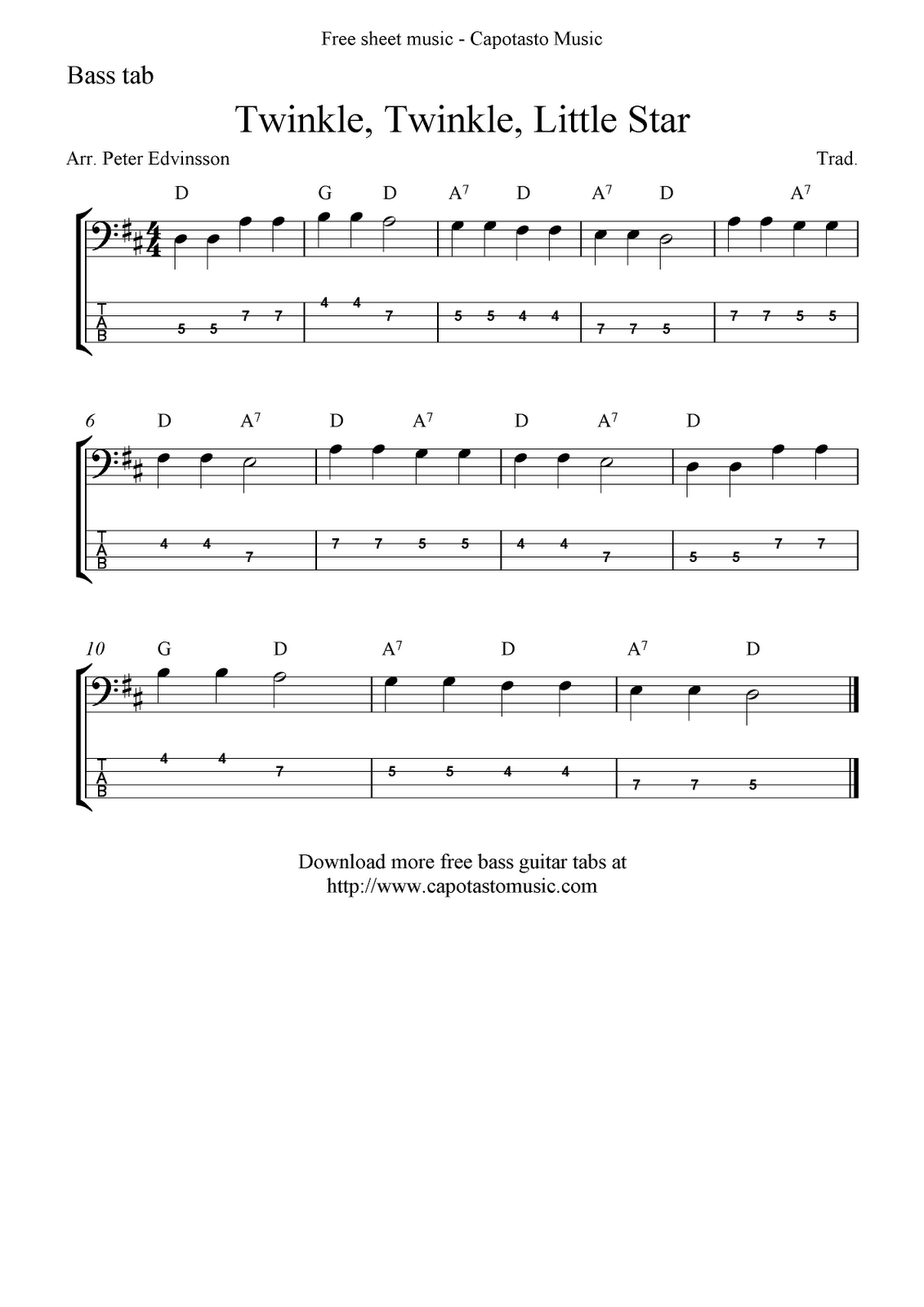
Motivation

# Background Research

## About Guitar learning

When learning to play the guitar, most beginners and even skilled players turn to “tabs” to learn new songs. This is because tabs are a more readable format for those that can’t read sheet music, allowing people to learn to play the guitar without having to learn a large amount of music theory beforehand.

An example comparison between the two formats are shown below:

  
Without knowledge of how to read sheet music, you would not be able to tell how to play the song through the sheet music version.

Tabs can show us what fret and on what strings we need to place our fingers on in order achieve the same sound.

Nowadays tabs are readily available online from multiple sites and generally appear in a text-based format such as:

e|-----0-------------------------0--------------0--------0------------------

B|--3-----3--1-------1-----1--3-----3--1------1----3--3----3--1-------1-----

G|--------------0h2--2--2----------------0h2---------------------0h2--2--2--

D|-------------------3----3-------------------------------------------3-----

A|--------------------------------------------0-----------------------------

E|----------------------------3--------------------3--3---------------------

However, tabs still don’t give us a good visual representation of how or where our fingers should be placed, as it is all text-based. This is what the app will be trying to improve upon.

## Survey of similar apps

### App 1

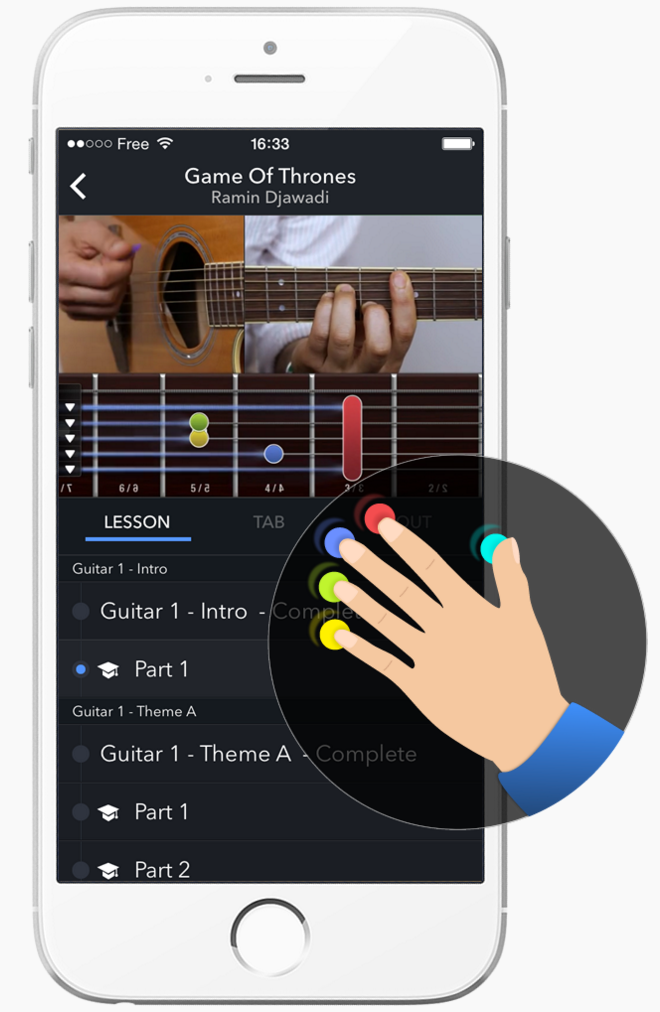
CouchGuitar is an iOS app that has a song-by-song tutorial for anyone trying to learn guitar without having to know any prior music theory.

They provide very intuitive colour coded methods of teaching coupled with the use of a video play through which provides a very simple and effective learning experience.

The app shows which fingers should be where on the fret board by colour coding each finger and then having a visual representation on screen coupled with a YouTube style video to guide you through the steps as if you had a real teacher in front of you teaching you how to play the song.

An upside to this is that it provides additional information that a virtually generated representation can’t provide, such as form or proper ways to place your fingers from the video guidance. Another upside of this app is that it will show you the correct timings and rhythm of the song, which a text-based tab may not show.

However the app doesn’t provide flexible song choices as they couple each tutorial with video, so they are limited to how many video tutorials the creator releases. This is similar to comparing it as a hard-coded project where it can only show songs that the creator has manually created.

A downside to this would be if support for the app ceases, then no new songs would be added. Additionally, if there was a newly released song you really want to learn, you must also wait for the content creator to manually create videos and their tab version which will most likely be slower than a community-based tab website such as ultimate-guitar.com where tabs are readily available for almost any song you can think of.

### App 2

<https://getinstinct.com/> is a web guitar-teaching app that provides very good structured lessons with very good visual aids. However all the lessons are set and you are unable to choose which songs you want to learn.

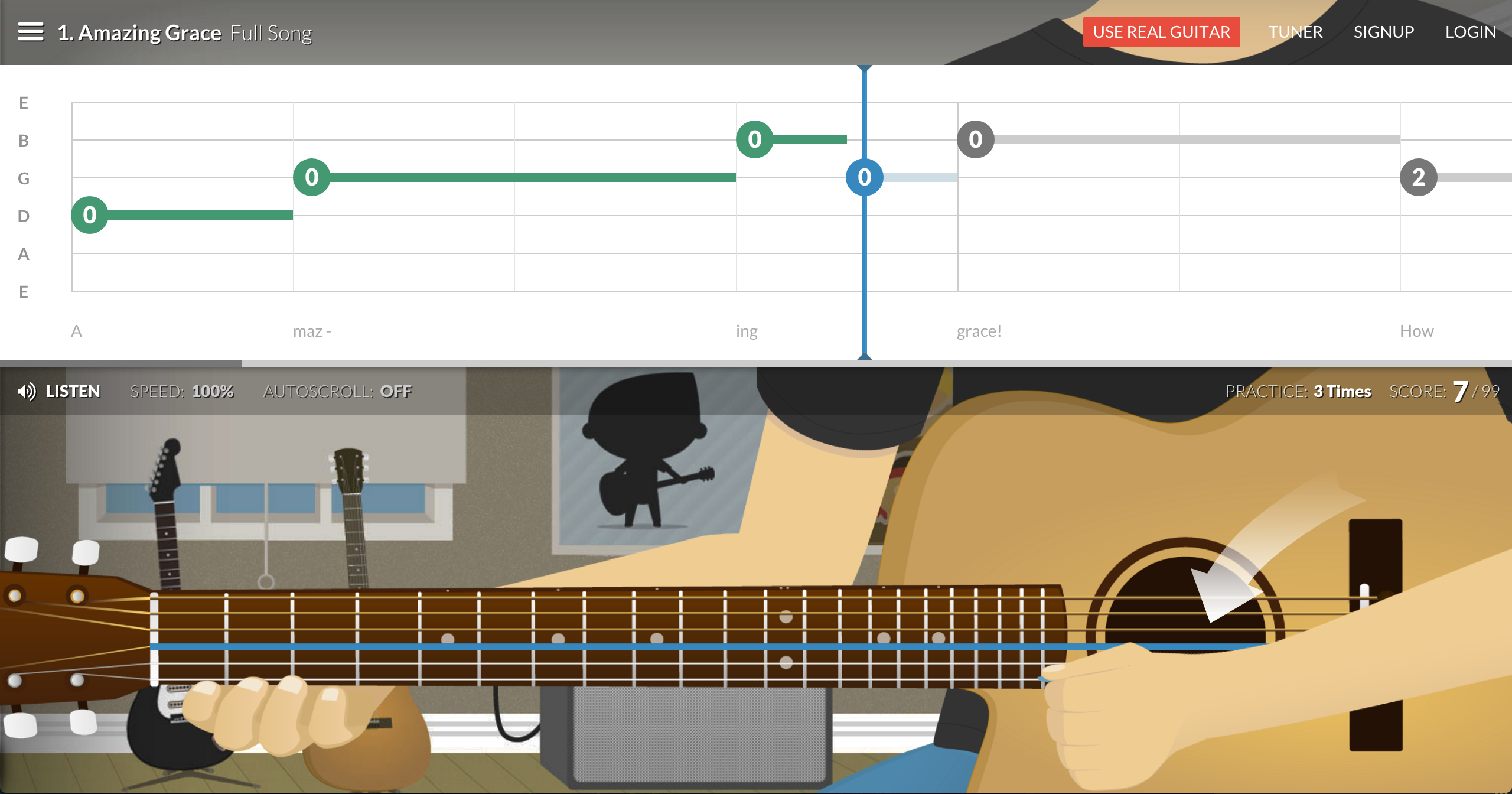
They do a very good job at providing an interactive, visual learning experience that is also very user friendly, providing 2 main ways of practicing the guitar.

The main method of practicing is when you have a microphone connected, the app will record what you are playing in real-time and give the user feedback on how accurate they are playing each part of the song. A great feature about this feedback-focused method is that the song progresses at the users own speed. This is because the song doesn’t progress until the app feels the user has played an accurate enough version of the current notes or chord the in that part of the song.

Another method provided is a completely computer generated virtual model that should be used when the user doesn’t have a guitar on them at that moment. The tabs would still be shown as normal, but the user can pretend to play the instrument by strumming the strings with your mouse as to simulate playing a real guitar.

The audio-feedback system is a really good idea as it allows the user to actively practice with the app at their own pace without having to take their hands off of the guitar at any point in time when learning the lesson.

However this seems to have the same disadvantage as CoachGuitar app does in that all the lessons are pre-programmed and you do not have much selection of which songs you want to learn. This is also a more lesson/tutorial-based app where it focuses more on the basics such as chords and techniques rather than learning whole songs.



## Background of tech

Originally I planned to use swift to create and implement an iOS app similar to that of CouchGuitar. I wanted to expand and add to some of their design theme ideas in order to create an app with a smoother UI experience.

The main motivation for choosing to use swift was because me, my family and friends have all been avid uses of Apple’s iPhone devices for a long time and I wanted to have a go at creating an app for a device that I use daily.

The reasons why I chose swift over another language such as objective-C is because objective-C is a very hard language to learn and is not very intuitive. Objective-C has a massive learning curve and I believe it would’ve taken me a lot longer to learn the basics of the language, little lone what I want to eventually accomplish with it.

However a final decision was made where I decided to no longer create an iOS app using swift, but opted to go for a web-based solution instead.

Reasons for this after some researching were mainly because swift is still a relatively new programming language and there isn’t much documentation or libraries available that would be able to help me with self-learning of the language or to implement my ideas e.g. sound libraries.

As well as that, building a web application would also mean that the app will be available across all platforms compared to just iOS systems.

For the web app, I would need to use fundamental web technologies such as HTML, CSS and JavaScript for the front end, however instead of using PHP for the back end, I decided I wanted to use a Python web framework called Django.

Some reasons for choosing Django are:

* It is all coded in Python. This is a relatively easy language to learn as it is quite intuitive and cuts down a on a lot of syntax.
* Django can handle your database for you. This means it can create your database in SQLite as well as perform queries such as INSERT, UPDATE and DELETE.
* Defining a few fields and parameters, you can get Django to create HTML that includes error checking as well as formats the data.
* Django provides admin functionality that requires little or no work to deploy. It is also highly customizable.
* I will be taking a web-programming module next semester where we will also be learning to use Django. This may help to expand and better my web app.
* Django is very well documented and has very clear and concise tutorials.
* Focuses on fast development.

After having completed the Django tutorial, I am confident that I will be able to create a good app using Django as the backend as there was extensive documentation for all the basic features I had to apply.

As for the front-end technologies there is also extensive documentation available for all the front-end web technologies. In order to create a fluid and interactive application, the project will be using a lot of JavaScript and JQuery code. For example, JQuery can be used for smooth transitions of finger positioning images and JavaScript can be used to set a timed interval for playing out each tab.

# Requirements capture / Specification

The app should be able to:

* Be able to synthesize accurate sounding chords and notes based on the tabs provided
* Display a visual representation of finger positioning for the chords/notes
* Be able to play along with the song at various speeds e.g. 0.5x or 100bpm
* Be able to select variable sections of a tab and constantly repeat that section for practice.
* Contain a chord library containing visual representations of various chords.

Additional features that are desirable are:

* Refine time notations for the tabs. Allow for accurate time measurements for notes / chords (limited by the tab input).
* Contain an accessible library for all chord variations that can be visually and audibly played, replayed, and practiced.
* Search for the correct original BPM for the songs.
* Implement ways to handle and parse other actions from tabs such as ‘h’ hammer on and ‘/’ slides.

# Design

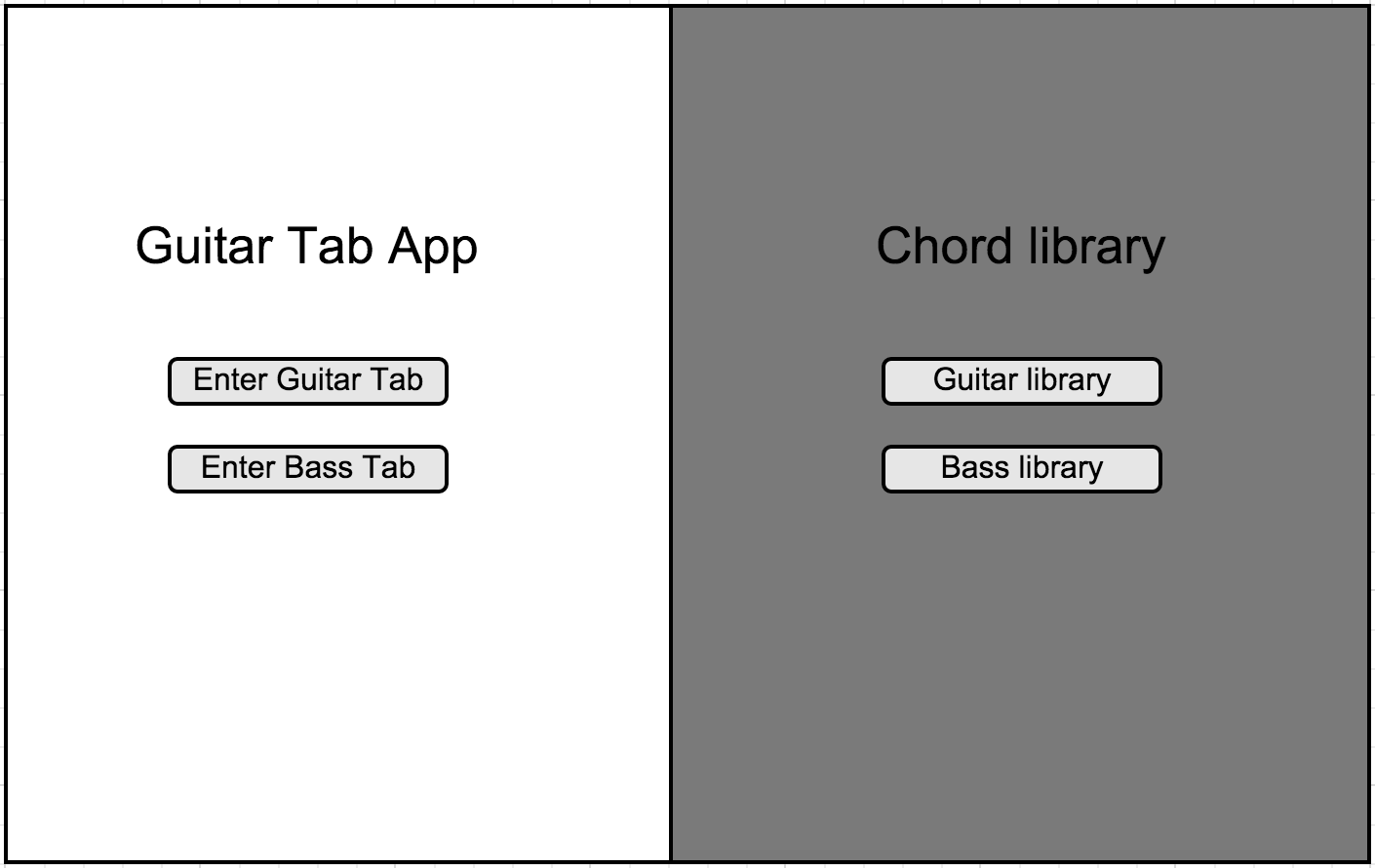


Figure 1. Home Page

Figure 1. Is the basic outline of the app. I wanted to provide a clean and sleek layout what allows the user to intuitively navigate between the main two features of the application.

The home page will consist mainly of 2 buttons. One button is to enter a guitar tab for the main functionality and the other button is to view a chord library for the guitar. If time permits, this app should also accommodate bass tabs and chords functionality.

When the user pressed the “Enter Guitar Tab” button they will be presented with the page shown in Figure 2.

## Guitar App

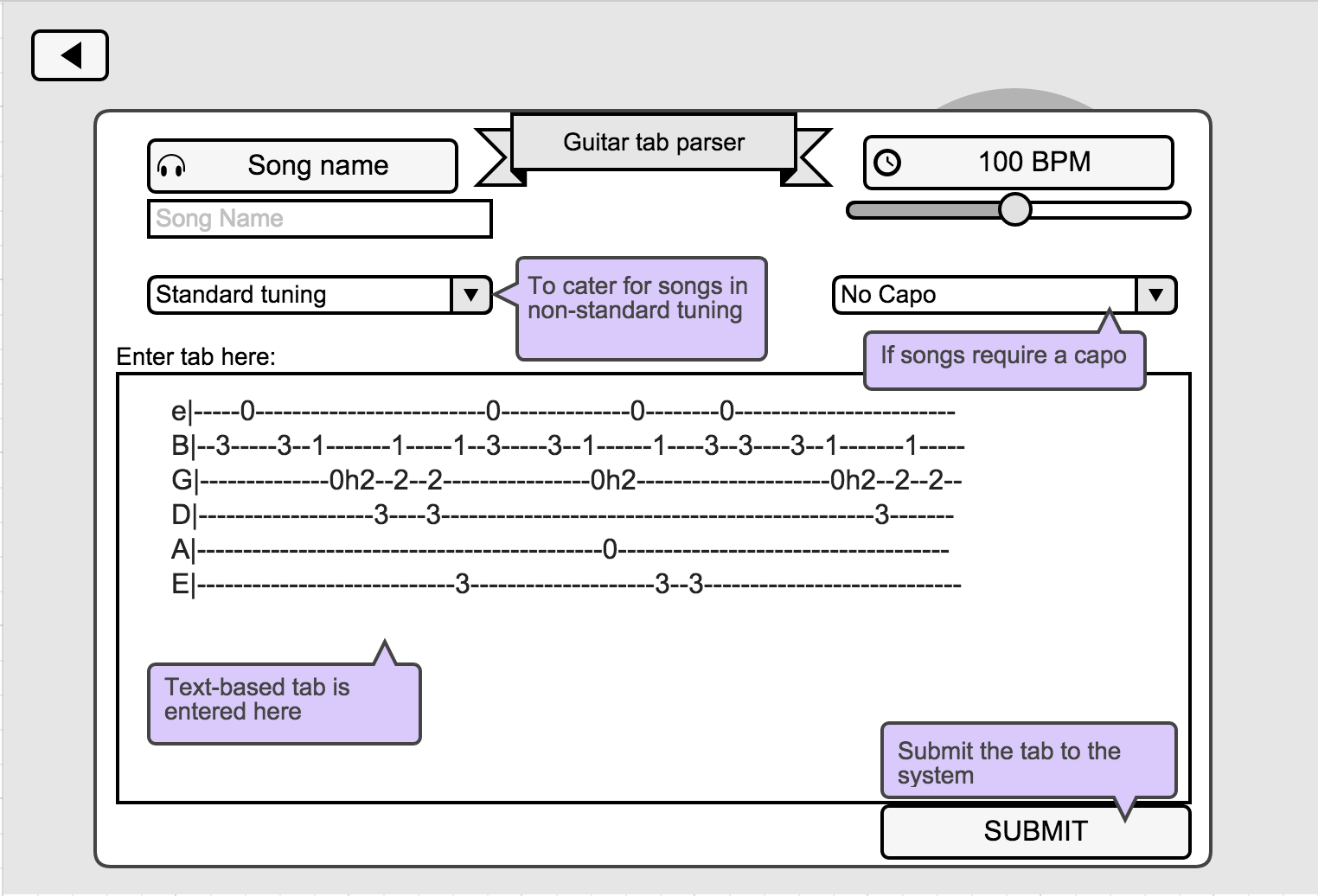


Figure 2. Tab entry Page

The page shown in Fig. 2 will function as a configuration and data entry page. Here the user will be able to enter their tabs into the application and enter some information about how the tab should work.

Some songs require different guitar tuning to play and some require you to use a capo. The user can tell the app this and the necessary transpositions will be made.

The user should be able to select a speed that they want the song to be played at. This should also be able to be changed later on.

After all the configurations have been completed, the tab has been entered and the user presses submit, they will be presented with the main feature of the app as shown in Figure 3.

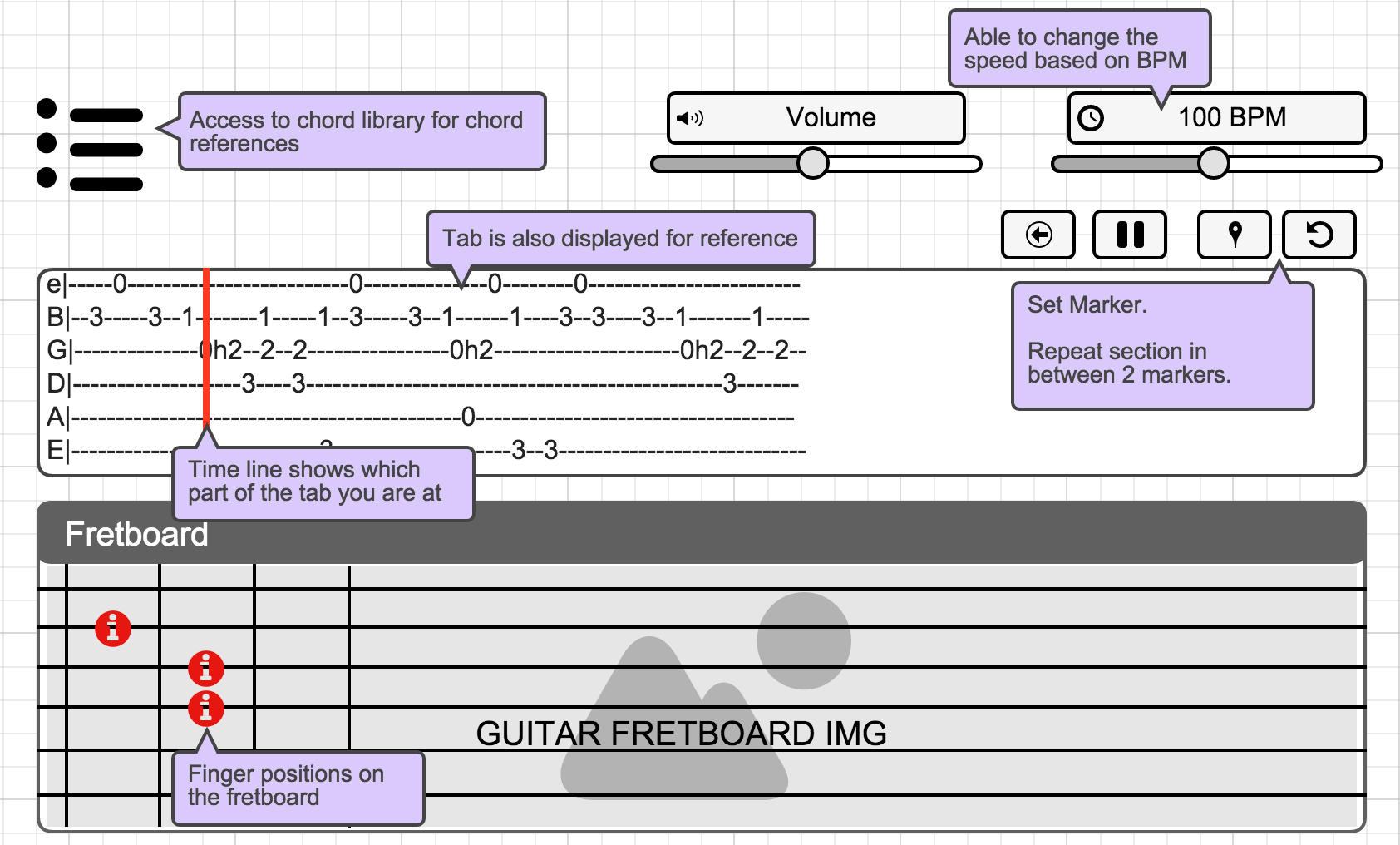


Figure 3. Guitar Page

Figure 3. Shows us how the main features of the app will play out. The tab entered by the user should be displayed in a properly formatted structure where a red bar will follow along with it to indicate what part of the song we are currently at.

The red “timeline” bar should be able to be moved by the mouse to any part in the song and the fretboard should change dynamically according to the timeline position. The timeline should also be able to be controlled with a play/pause button as well as a button to go back a few steps.

For the repetition feature, there will be 2 buttons on the left that will allow you to set a marker at the current timeline position. Doing this twice will create a section in the tab. If a section exists then when you press the repeat button, this section should constantly repeat until the user presses pause or sets a new marker. This functionality is for when the user needs to practice a specific part of the song without having to let go of the guitar.

The fretboard is where the visual help function will be shown. At every point in the song, this should show positions on the fretboard for where your fingers should be placed. To start off with, just use the same colour button for each position.

The app should be able to play sounds of the notes as an audio output so that the user can also verify if what they are playing is correct using their ears.

There should also be a shortcut to search for certain chords on this page so the user doesn’t have to go through the home page again to access this.

## Chord Library

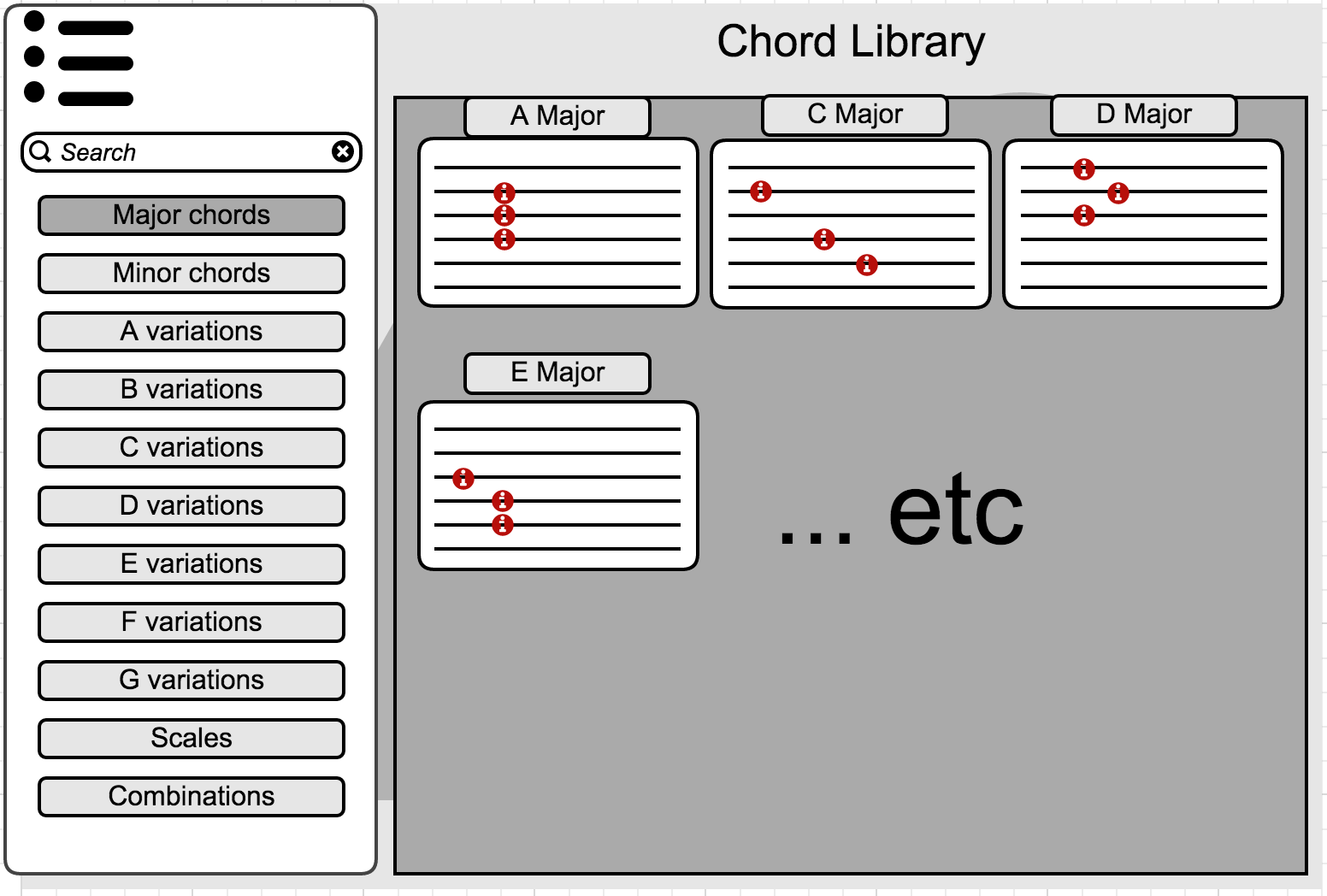


Figure 4. Chord library Page

On the flip side of the application when the user presses the “Guitar library” in Fig. 1 they will be presented with the chord library page shown in Figure 4.

This will just be for reference to the user and contain all their guitar chord needs all in one place.

There should be a search function on this page for when the user needs to find a specific chord. There should also be a shortcut menu that contains easily navigable options for different chords and their variations.

The main part of the screen should be used to show the different chords in a visual representation.

An extra feature could be to allow the user to manually arrange their chords in their own orders e.g. if they are learn a chord sequence they can arrange them so they are next to each other.

# IMPLEMENTATION

## Creating a web app

To start off the project, I firstly needed to build a basic web app where I can navigate between a few different pages with the ability to send data between each redirect. To do this I initially started using Django by running through the tutorial as planned, which is fairly straightforward and very easy to learn from.

Reference Django <https://docs.djangoproject.com/en/1.9/> here

Django provides us with 3 main layers. The first layer – the model layer provides an abstraction layer for structuring and manipulating the data (from django). This allows us to create and manipulate an SQLite database through python code straight from the Django framework. The guitar app however doesn’t really have a need for a database at the moment unless I want to store previously searched tabs, or perhaps create a login page.

The second layer is the view layer which is responsible for processing a user’s POST and GET requests through python code that then sends back a response. This enables us to easily use python for the server-side code instead of having to use PHP.

The template layer is the last layer that aids the front-end by providing Django syntax for rendering information into HTML components as well as allowing us to use the data for any JavaScript functions that we may need.

After finishing the tutorial and learning python as well as the Django framework, I then adjusted and adapted it to meet the requirements for the guitar app.

This involved creating new HTML and CSS templates that contain roughly all the necessary components in roughly the right layout so that I can clearly see how each component interacts with each other.

Creating all the components such as buttons in HTML and placing them first before coding any functionality into them allows me to ensure that all the required features will be present. I can then work eventually work through getting each component working before working on any additional features.

Refactoring the views and templates from the tutorial proved fairly simple as for now I just had to change the response logic within views.py to redirect to the new templates I created for my guitar app.

## The home page

The home page for the web app is by design very simplistic. The purpose of this page is only to direct the user into the area of the app that they are interested in at the moment. Because of this, I have opted for a very simple layout that only contains a few buttons leading to different functions of the app.

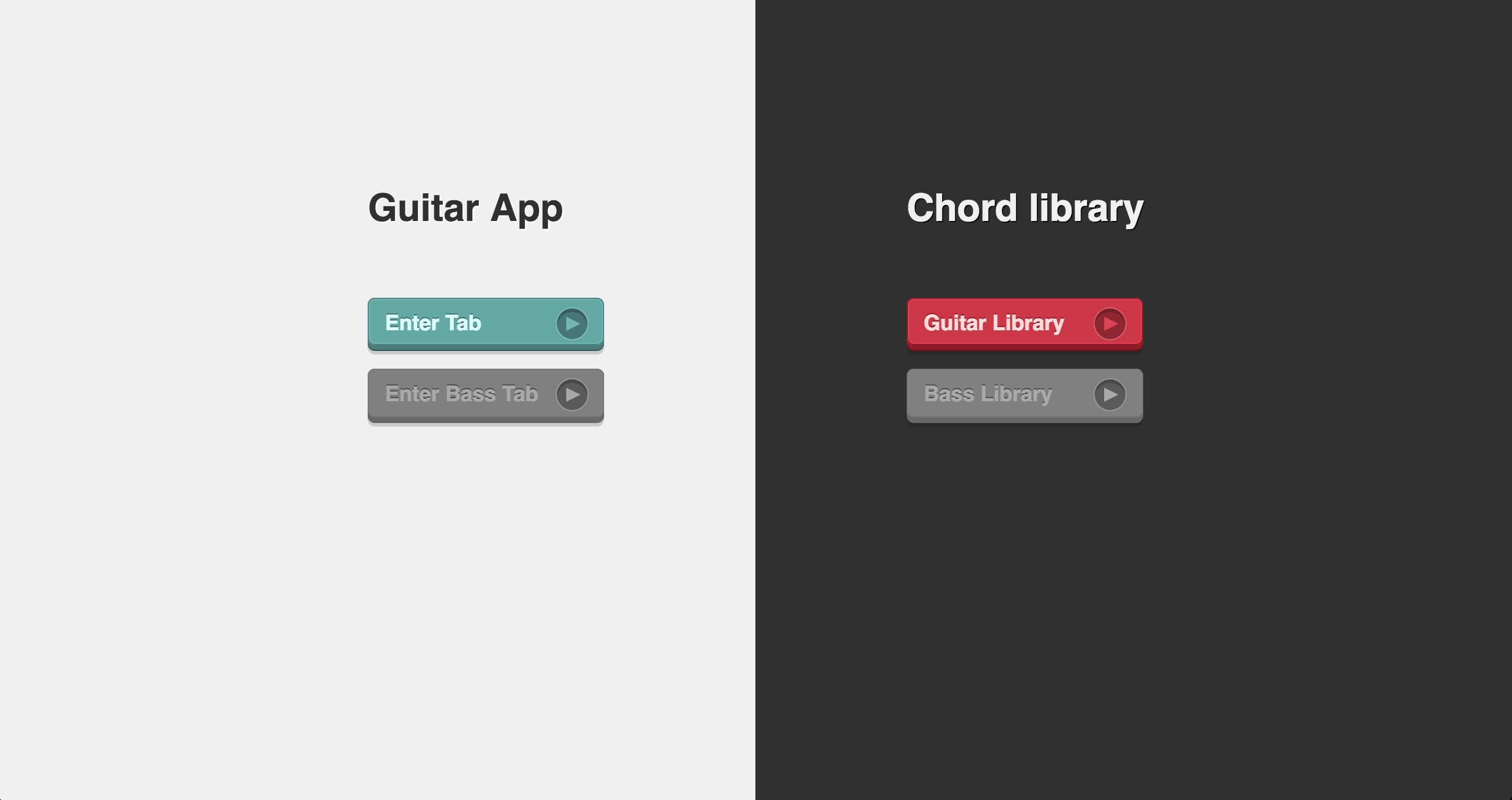


Figure 1. Home Page

Splitting the home page into 2 contrasting segments allows the user to very intuitively notice that there is a difference between the two options presented.

Functionality wise, the buttons utilize both the template and view layers of Django to redirect us to a new page.

<**a** href=**"{% url 'guitar:chords' %}"** class=**"button btn-pink clearfix"**><**span**>Guitar Library</**span**>

The template layer allows us to use the {%url%} syntax to access different pages within the application. We can then use python in views.py to render the new page as easily as:

*def* chords(*request*):  
 *return* render(*request*, 'guitar/chords.html')

This is done for each of the buttons that we have with the only difference being that they redirect to different segments of the app.

## Guitar App: Inputting a Tab

The purpose of this page is to allow the user to pre-configure their song choice and enter their tab into the app from any other sources. The tab should be entered into the large text area provided and this will be send through a POST request to the next view where we will then do computations and read the tab data in the app’s main view later on.

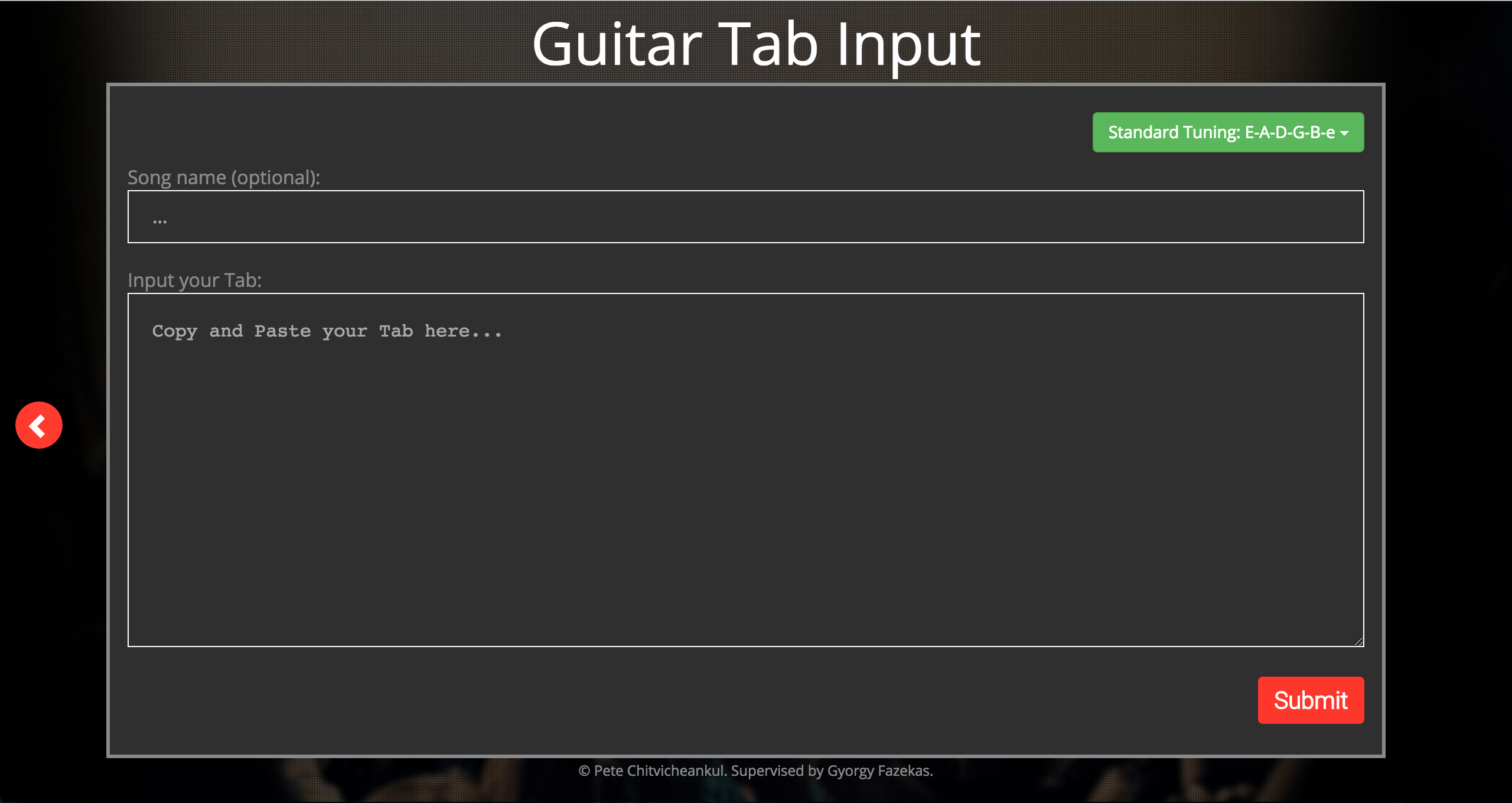


Figure 2. Tab entry page

For this page we needed to use a form in the HTML, as we will have some data that we need to send on towards the next view. I then had to put all the components that we require a value from within the form so that the data can be retained. This included the song name, the tab and the tuning for the tab.

The song tuning for the tab was an extra addition and will be what decides what format (string identifiers) the tab should be in so the program can detect and pre-process the tab into 6 individual lines for each string. However, I was unable to add additional tunings yet, but I will include this within the future work section. For now other tunings have been put as placeholders but have been disabled.

<**form** action=**"{% url 'guitar:visualiser' %}"** method=**"post"** id = **"tabform"**>

Using Django’s syntax we can again call the visualiser view when the form is submitted allowing us to preprocess and handle the data. Since data needs to be sent through a request this time, we first check that the request method is a POST within views.py. If a valid POST request was sent, we can then access all the data that was within that form by accessing it through “*request*.POST.get()” where we can put the name of the different components into the .get() method and retrieve their raw values. E.g. *request*.POST.get('songname')

[TODO]

The next step after getting access to the user inputted data is to pre-process it. Since there is no real format for tabs, and no way to limit what the user will input I will use the selected tuning to help identify which strings belong where based on a standard format that most tabs start with a string identifier in front of each line e.g. e|----- or A|----- where ‘e’ and ‘A’ are the string identifiers.

The first implementation required the user’s to manually append multi-line tabs into one 6-string tab when some tabs can end up being over 100 lines long. Since this was a big hassle I created a way that allowed the user to copy and paste in the whole page. Using the string identifier’s we are able to tell which line should be appended to each other automatically as well as ignoring all other lines that’s are irrelevant to the tab that may have been copy and pasted in by the user.

*for* str *in* strings:  
 stringlen = len(str)  
 *if* len(str) > 1:  
 *if* str[0] == 'e' *and* (str[1] == '|' *or* str[1] == '-'):  
 str = str[1:]  
 *for* s *in* str:  
 *if* s != '-':  
 str = str[1:]  
 *else*:  
 *break* finalStrings[0] += str  
 *continue*

The above snippet shows how this is done by firstly checking for each string identifier, which in this case is ‘e’. We then trim the string so that it is only that tab part that we are interested in and we append this to the final tab array.

Finally once we have finished pre-processing, we then send the form data to the next page with:

*return* render(*request*, 'guitar/visualiser.html', {'form' : *request*.POST})

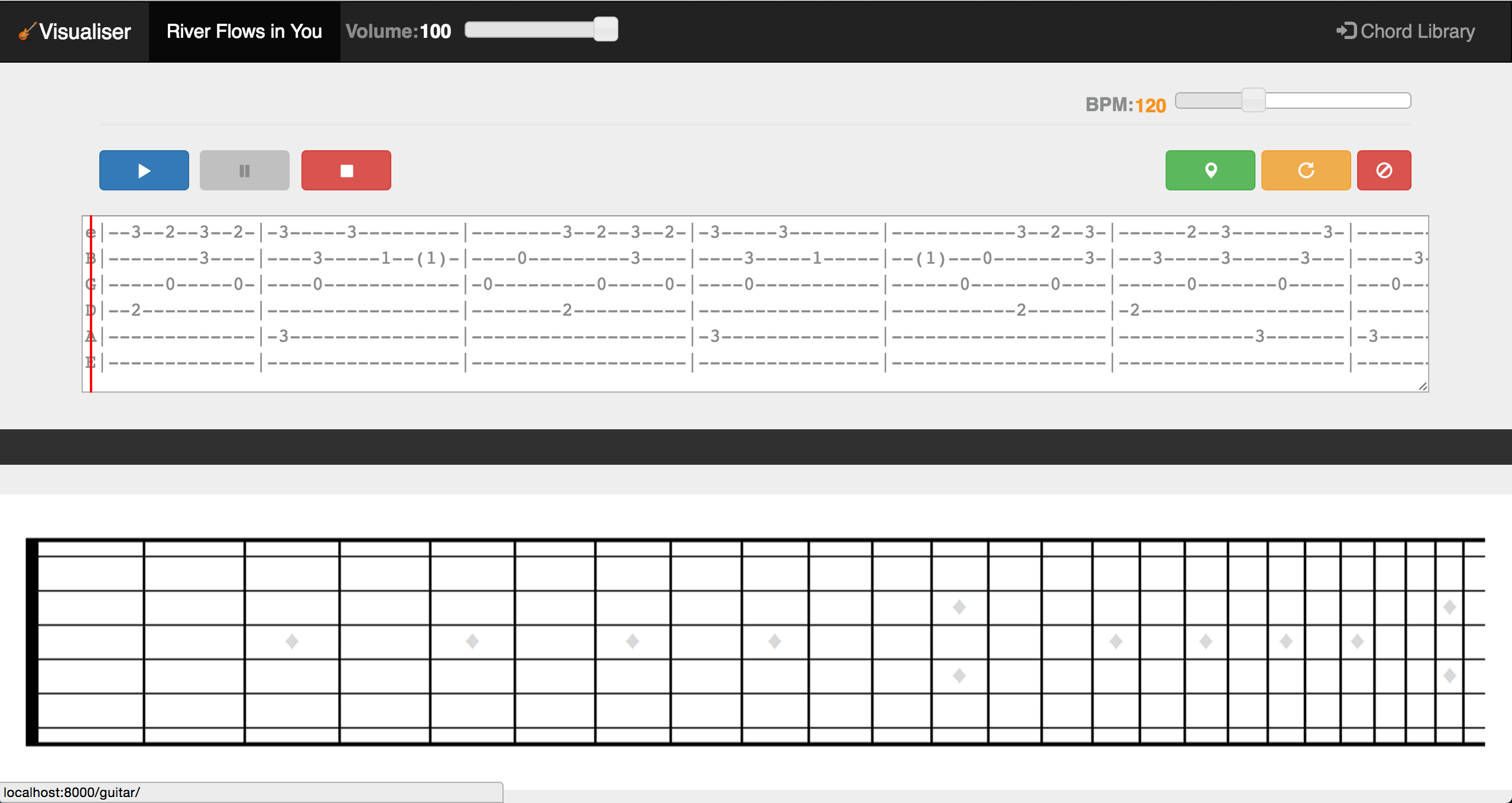
## Guitar App: Visualiser

This page is where all the functionality happens. The HTML elements will consist of 3 main parts with the first part will be the menu and configuration options, filling the top segment of page with a BPM slider, volume bar and play/pause/stop buttons.

The central segment of the page is where a fully formatted representation of the tab is placed. This section should also contain a red bar that follows along with the tab as it plays to indicate what part of the song is currently playing.

The bottom segment contains an image of a guitar fretboard where finger positions will be shown using an image of a red dot on each string as finger representations. The red dots will dynamically change their positions based on the current position of the tab being played.

Difficulties on this page include reading multiple text lines at the same time, getting sounds to play synchronously, getting the tab to automatically play in changing time intervals as well as getting all the elements on the page to align with the tab such as the position line and finger positions.



### Component positioning

Styling on this page is very important, as there are many components as well as components that require accurate positioning e.g. finger positions.

When playing the guitar, you can only press a string on one fret at a time only. Because of this I will only need one red dot (finger positioning dot) per string, allowing me to hardcode each dot’s top position. The left position will be the only attribute that needs to dynamically change.

The left position however will only be moving at set intervals as the frets across the fretboard are at fixed intervals. Therefore I store the left position of each fret into an array called notePositions. Whenever a note is then played, it will then search the same position in the array for the left attribute value of the red dot image and update it in real time as the tab is played.

The majority of the buttons and page sections are styled with bootstrap. Bootstrap allowed me to easily manipulate the page in a way I wanted it with a set of buttons/functions on separate sides of the page.

[TODO]

position other components: slider, play/pause/stop, ADD volume (howler does this)

write about window resizing here

### Arrow key step functionality

I have added an extra function to step through the tab one character (or note) at a time using the arrow keys as I initially had trouble getting a fluid play function to work. This was to originally there to test if the logic for reading the tabs worked or not, but it is a nice addition as you can truly go at your own learning pace using this feature.

For this function I have a global counter variable to tell me what character/note is currently being read on the tab. JQuery is then used to listen for whenever a key is pressed on the window and checks whether the key pressed is either the left or right arrow. When the event is fired, firstly increment the note counter and then call a JavaScript function that analyses the character for each of the 6 guitar strings at the position of the counter.

<insert code snippets?>

The function then checks whether the character read is a legible note or not mainly by checking whether the character is a valid integer or not. When a valid note is read from the tab, it then goes through the notePositions array to find out the new left attribute of the dot on that string and updates the CSS for that image.

Additionally, the function will also play the sound of the relevant note using an external library that will be explained in the next section. Simply using the playSound() function we are able to pass information of what string and what fret we are currently on in order to locate the correct sound to play.

### Implementing Sound

Initially when trying to implement sound I attempted to use the standard HTML <audio> element tags in order to load and play guitar notes. I had to figure out which note was being played and load each individual note to be played one at a time. Originally I recorded each individual guitar note played for 2 seconds and saved them as an mp3 file. This ended up with me containing a large folder of sound files within the project.

This was not ideal as whenever I had to play a sound, I would need to send a get request each time for each different sound to be played which is very bad practice as there could be high volumes of get requests when multiple users and all requesting for multiple sounds. Another reason this didn’t work well was that the delays in each get request caused delays to playing sounds, which was quite noticeable when 3 or more notes needed to be played at the same time e.g. chords.

In order to reduce the amount of get requests and number of sound files, I instead used a music software to record each note at 2 second intervals into a big 1 minute long sound file. Further research led me to finding a sound library called Howler.js that allows us to load sound files, split them into sections and play them synchronously.

I was able to split the sound file using howler sprites where I was able to name a sound, specify its starting point and duration in milliseconds. The naming convention I use for this is s1n0 where the number after ‘s’ is the string number and the number after ‘n’ is the fret denoted by the tab.

sound = **new** Howl({  
 urls: [**"../static/guitar/sounds/Guitar12frets2s.mp3"**],  
 sprite: {  
 s6n0: [0, 1800],  
 s6n1: [2000, 1800],  
 s6n2: [4000, 1800],  
 s6n3: [6000, 1800],

To play the sound I then simple call the howl.play() method, passing in the name of the sprite (sound segment) and the correct sound will play. As I will have more than one place in the code that I created a new function specifically to play sounds as shown below:

**function** playSound(soundSprite){sound.play(soundSprite);  
}

### Play button function

Initially when I attempted to implement the play function, I tried to use JQuery animation-delays as a timer and then within the animation I would change the CSS of the corresponding fingering position dots as well as play the corresponding sound. This however did not work as expected as the finger component never changed at each interval despite the delay happening.

I then ventured to find an alternate solution and found out about the JavaScript setInterval() function. This allows me to perform my tab reading function at specified intervals. Using a JQuery event listener for the play button, I then clear all existing intervals and then begin to read the tab in the playTab() function, iterating through the character positions by 1 each time.

$(**'.play'**).click(**function**(){  
 clearInterval(interval);  
 interval = setInterval(**function**(){  
 playTab(strings, charposition);  
 charposition++}, speed);  
});

Eventually I added a speed slider that determines the rate the program reads the tab, which is dynamically changed whenever the user moves the slider. As we are creating an app for music, it only made sense to use BPM (beats per minute) as the time signature measurement. However setInterval() uses milliseconds as the time measurement so some time conversion calculations was required for this as well.

The playTab() method is where the main functionality is calculated and processed. An array containing each separated string from the tab and the current character position along the tab is passed to the method where a For loop is used to iterate through each of the strings and searches for the character at ‘charposition’.

Each character is then checked to see whether it is actually a note or not by using! isNaN() to check if the character is an integer or not. If the character is a plausible note, then I have used JQuery to dynamically change the CSS left-position of the matching red dot to the correct position on the guitar fretboard. As well as that, this is also where we call the playSound() method and form a string to represent the correct sprite we will need to play the correct sound.

### Pause and Stop buttons

clearInterval

### Setting Markers

### repeating a segment

[TODO]

not yet completed.

## Chord Library

# User Feedback

## User 1

## User 2

# Improvements

CAPO

HELP MENUS

# Additional Features

Critical Analysis

Honest Appraisal

Further work (if there was more time)

# Conclusion