Professional musicians wish to collect and log data of practice sessions efficiently.

Many practice apps don’t have:

- Audio recording
- Tuner
- Metronome
- Practice portal with logs
Clients

Dr. John Irrera

Dr. Annie Stevens
Deliverables

- A mobile app designed as an all-encompassing practice hub for musicians.
- Back-end conversion from Firebase to Supabase.
- Implementation of metronome feature.
Conversion: Firebase ➔ Supabase

- Converted files relating to the API.
- Integrated Supabase API for user registration and authentication.
Supabase

- Set up tables in Supabase for practice data.
- Display user and practice data on app.
- Update data in backend when user edits fields.
## Supabase Table: User Profile

<table>
<thead>
<tr>
<th>Columns</th>
<th>Type</th>
<th>Default Value</th>
<th>Primary</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>uuid</td>
<td>NULL</td>
<td>✔️</td>
</tr>
<tr>
<td>email</td>
<td>text</td>
<td>NULL</td>
<td>×</td>
</tr>
<tr>
<td>profilePicture</td>
<td>text</td>
<td>NULL</td>
<td>×</td>
</tr>
<tr>
<td>name</td>
<td>text</td>
<td>NULL</td>
<td>×</td>
</tr>
<tr>
<td>dateOfBirth</td>
<td>date</td>
<td>NULL</td>
<td>×</td>
</tr>
<tr>
<td>instruments</td>
<td>jsonb</td>
<td>NULL</td>
<td>×</td>
</tr>
<tr>
<td>level</td>
<td>text</td>
<td>NULL</td>
<td>×</td>
</tr>
</tbody>
</table>
Supabase Table: Practice Data

<table>
<thead>
<tr>
<th>id</th>
<th>uuid</th>
<th>name</th>
<th>email</th>
<th>instrument</th>
<th>level</th>
<th>profile</th>
<th>birth_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>904849d3-eb03-4d</td>
<td>Jillian Y</td>
<td><a href="mailto:jylagan@vt.edu">jylagan@vt.edu</a></td>
<td>[&quot;Bass Guitar&quot;, &quot;Drum&quot;]</td>
<td>Pre-collegiate</td>
<td>EMPTY</td>
<td>2024-04-02</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>user_id</th>
<th>uuid</th>
<th>title</th>
<th>piece</th>
<th>composer</th>
<th>notes</th>
<th>instrument</th>
</tr>
</thead>
<tbody>
<tr>
<td>904849d3-eb03-4d</td>
<td>fd53a16d-f538-4d</td>
<td>Recital Practice</td>
<td>Für Elise</td>
<td>Beethoven</td>
<td>Notes</td>
<td>Piano</td>
</tr>
</tbody>
</table>
Redux

- Redux is an open source JavaScript library for managing global application state.
- We use Redux in our project for updating the backend when changes are made by the user on the frontend side of the application, through dispatching.
Redux Challenges

In the original code, we directly took practice data from Supabase and manually set their fields to different values according to the user’s input in the UI when updating a practice plan.

```javascript
try {
  const practiceId = await DataManagementAPI.addPracticeData(plan.title, plan.piece, plan.composer, plan.instrument, selectedDate, plan.notes);
  currentPracticeData.weeklyPracticeData.push({ id: practiceId, title: plan.title, piece: plan.piece, composer: plan.composer, instrument: plan.instrument, practiceDate: weekDay, duration: 0, status: STATUS[0], notes: plan.notes } as IPPracticeDataProps);
  const existing = currentMusicPieces.musicPieces.find(item => item.title === musicPiece.title && item.piece === musicPiece.piece &&
    item.composer === musicPiece.composer && item.instrument === musicPiece.instrument &&
    item.notes === musicPiece.notes);

  if (!existing) {
    currentMusicPieces.musicPieces.push(musicPiece);
  }
```
In our solution, we dispatch to update the store by using a reducer to return the new state of the practice data in the application and save it.

```javascript
async function handleSave(plan: any) {
  try {
    const practiceId = await DataManagementAPI.addPracticeData(plan.title, plan.piece, plan.composer, plan.instrument, selected);

    // Dispatch actions to update Redux store
    dispatch(addPracticeData({
      id: practiceId,
      title: plan.title,
      piece: plan.piece,
      composer: plan.composer,
      instrument: plan.instrument,
      notes: plan.notes,
      practiceDate: weekDay,
      duration: 0,
      status: STATUS[0]
    }));
  }

  const existing = currentMusicPieces.musicPieces.find(item => item.title === musicPiece.title && item.piece === musicPiece.piece && item.composer === musicPiece.composer && item.instrument === musicPiece.instrument && item.notes === musicPiece.notes);

  if (!existing) {
    dispatch(addMusicPiece(musicPiece));
  }
```
Metronome Integration

- Metronome in JUCE framework (C++).
- App in React (JS).
- Integration:
  - iOS: C++ → Objective-C → JS
  - Android: C++ → Java → JS
- Goal: Show JUCE component as View within React.
Integration Challenges/Steps

- Not a well-documented process.
  - Lack of resources / examples.
- iOS steps
  - Define ReactJuceView, subclass RCTViewManager.
- Expo Go
  - Expo Go to create the host app for metronome instead of JUCE.
  - CMake to link JUCE and Expo Go.
Metronome Implementation

Features:

- BPM display
- Play/Stop button
- << and >> increment by 10
- < and > increment by 1
- Hold down buttons to rapidly increment
Metronome Demo

Clicking Play to start the metronome.
Clicking > to increment the metronome by 1.
Clicking >> to increment the metronome by 10.

*A video demonstration is included in our project files: “MetronomeDemoVideo.mp4”.*
App Screenshots: Login + Registration
App Screenshots: Home, Progress, Profile

**Home**
- Overall Hours: 10,000 Remaining
- Goal: 10,000 hours
- Pieces: 0

**Progress**
- Activity
  - Daily Hours: 2 Remaining
  - Goal: 2 hours
  - Pieces: 0

- Distribution
  - No distribution available.

**Profile**
- Jillian Y
- Change Profile Picture
- Email: jylagan@vt.edu
- Date of Birth: 2024-04-02
- Level: Pre-collegiate
- Instrument(s): Bass Guitar, Drums/Percussion, Piano
- Logout
- Delete Account
App Screenshots: Journal + Practice
Timeline

Meet with lead programmer and clients

Complete basic metronome implementation

Finish preliminary research

Finish additional metronome features (+/- buttons, UI)

Finish Supabase integration and tables

Meet with clients for final approval

Finish all manual testing
Manual Testing: 7 students completed a questionnaire after using the app or metronome.

**App Testing**
- Is the layout of the app intuitive to navigate?
- Is it easy to tell if user actions have succeeded/failed?

**Metronome Testing**
- Is the beep pleasant to listen to?
- When holding the button, is the timing acceptable?
Testing Results

- Layout of the app was intuitive and easy to navigate.
- Registration process and adding a practice plan straightforward.
- Metronome UI easy and feels pleasant.
- Suggestions:
  - Ability to change the daily hours of practice for their goals.
  - Adding a Scales/Technique component to a practice plan.
  - Separate tracking for rehearsals vs. personal practice.
  - Different/customizable sounds to differentiate metronome downbeat.
Future Work

- Gamification aspects.
- Tuner, audio recorder.
- Generate weekly logs of practice sessions with checklists and notifications.
Acknowledgements

Clients
- Dr. John Irrera
- Dr. Annie Stevens

Professor
- Dr. Fox

Lead Programmer
- Ben Winters

Past Capstone:
- Alexander Georgiev
- Bailey McKelway
- Claire Holmes
- Mahlet Zemui
- Rahul Aneja
References

- See report for other references