# Frontend Integration Guide - AI Mock Interview (FastAPI + Supabase)

## Overview

This guide provides everything a frontend engineer (Next.js or any web frontend framework) needs to integrate with the AI Mock Interview backend API. The application allows users to upload a resume, participate in a video-based mock interview, and receive stress analysis feedback.

### Flow Pipeline

1. User logs in using Supabase (Google Sign-In).
2. User sees their previous interview sessions (/session-history).
3. User uploads a resume to start a new mock interview (/upload-resume).
4. Questions are generated dynamically from the resume (/generate-questions).
5. Interview begins: questions are displayed one at a time (/next-question), user records a video response (2-minute timer, no retakes), and stress is analyzed (/analyze-stress).
6. Session summary is displayed with stress scores and a downloadable PDF report (/session-summary).

## Authentication (Supabase)

1. Install SDK

   ```
   npm install @supabase/supabase-js
   ```

2. Initialize Supabase Client

   ```
   import { createClient } from '@supabase/supabase-js';
   export const supabase = createClient('https://pzqodlqmyfylolspvgxl.supabase.co', 'your-anon-key');
   ```

3. Google Sign-In

   ```
   await supabase.auth.signInWithOAuth({ provider: 'google' });
   ```

4. Get Access Token

```
const session = await supabase.auth.getSession();
const accessToken = session.data.session.access_token;
```

Use this token in every request as:
Authorization: Bearer <accessToken>

## Backend API Endpoints

### 1. Fetch Session History

GET /session-history?user_id=<id>
Authorization: Bearer <token>

Returns a list of past interview sessions:
[ ”session$_i d” : ”...”, ”date” : ”...”, ”stress_s core” : 0.75, ”stress_g raph_d ata” : ..., ...$]

### 2. Upload Resume

POST /upload-resume/<mock_user_id>
Content-Type: multipart/form-data
Authorization: Bearer <token>

Form fields:

- file: PDF file of the resume

Returns:
”status”: ”Resume uploaded”, ”data”: ”id”: ”...”, ”user$_i d” : ”...”, ”file_p ath” : ”...”, ”resume_t ext” :$
”...”

### 3. Generate Interview Questions

POST /generate-questions/<mock_user_id>/<resume_id>
Authorization: Bearer <token>

Generates questions dynamically from the resume. Returns:
”status”: ”Questions generated”, ”session$_i d” : ”uuid”, ”questions” : [”1.TellmeaboutyourexperiencewithPytho$

### 4. Fetch Next Question

GET /next-question/<session_id>/<question_number>
Authorization: Bearer <token>

Fetches the question at the specified number (e.g., 1, 2, ...). Returns:
”status”: ”Question retrieved”, ”question”: ”1. Tell me about your experience with Python”,
”category”: ”technical”, ”question$_n umber” : 1, ”total_q uestions” : 9$

### 5. Analyze Stress

POST /analyze-stress/<session_id>/<question_number>
Content-Type: multipart/form-data
Authorization: Bearer <token>

Form fields:

- video: Video file (mp4, avi, mov)

Returns:
”status”: ”Stress analysis completed”, ”stress$_s$*core*” : 0.75, ”*data*” : ”*pitch*” : 0.6, ”*pace*” : 0.8, ...

6. Session Summary

GET /session-summary/<session_id>
Authorization: Bearer <token>

Returns:
”questions”: [”question$_t$*ext*” : ”1.*TellmeaboutyourexperiencewithPython*”, ”*category*” : ”*technical*”, ”*stress$_s$*
0.75, ...]

## UI Tips

| Feature | UI Element Suggestion |
|---|---|
| Session History | Table with columns for date, stress score, and a mini stress graph |
| Resume Upload | Modal with file input and success message |
| Question Display | Card with question text, progress indicator (e.g., ”Question 1 of 9”), and video recorder |
| Video Recording | Video preview with Start/Stop buttons, 2-minute timer, and early submit option |
| Summary Page | Table of questions and stress scores, line graph of stress trends, PDF download button |

## Additional Notes

### Folder Outputs

- /mock.interview.resumes/<user$_i$*d* >→ where resumes are stored in Supabase Storage

- Data is also stored in Supabase tables: mock$_i$*nterview$_s$essions*, mock$_i$*nterview$_q$uestions*, mock$_i$*nterview$_s$tress*

### JWT Validation on Backend

JWT is decoded to get user$_i$*dinFastAPI* :

import jwt
user_id = jwt.decode(token, options={”verify_signature”: False})[”sub”]

### Ready to Go!

The backend is complete, documented, and supports all features. You can start building the frontend now. Need mock data or code snippets? Just ask!