Interview Prep AI

Complete Project Documentation

Week 1 Progress Report & Continuation Guide

■# Interview Prep AI - Complete Project Documentation

Progress Report & Continuation Guide

Project Name: Interview Prep AI
Developer: Solo Developer (Advanced Python/FastAPI, Beginner Flutter)
Start Date: February 2026
Current Status: Week 1 Complete (25% Complete)
Timeline: 20 weeks total (5 months)

■ PROJECT OVERVIEW

What We're Building
A complete AI-powered platform with three main features:
1. **Resume Management System** - Upload, parse, analyze, and optimize resumes
2. **AI Interview Simulator** - Practice interviews with AI using open-source models
3. **Skill Roadmaps** - Personalized learning paths and career development

Tech Stack
Backend:
- Framework: FastAPI (Python)
- Database: PostgreSQL (via Supabase)
- AI: Hugging Face Inference API (Mistral-7B, Llama-2-13B)
- File Processing: PyPDF2, python-docx, spaCy
- Auth: JWT with bcrypt

Frontend:
- Framework: Flutter (Web + Mobile)
- State Management: Riverpod
- Navigation: GoRouter
- HTTP Client: Dio
- Storage: Flutter Secure Storage

Deployment:
- Backend: Render.com / Railway.app
- Frontend: Vercel / Firebase
- Database: Supabase (Free tier)

■ WEEK 1 COMPLETED (Days 1-4)

Day 1: Backend Foundation ■
Completed:
- ■ GitHub repository created
- ■ Python virtual environment setup
- ■ FastAPI project structure created
- ■ All dependencies installed (fixed Python 3.13 compatibility issues)
- ■ Basic endpoints working (/, /health)
- ■ CORS middleware configured
- ■ Security utilities created (JWT, password hashing)

Files Created:
backend/
■■■ app/
■ ■■■ __init__.py
■ ■■■ main.py
■ ■■■ config.py
■ ■■■ database.py
■ ■■■ models/__init__.py
■ ■■■ schemas/__init__.py
■ ■■■ routers/__init__.py
■ ■■■ services/__init__.py
■ ■■■ utils/
■ ■■■ __init__.py
■ ■■■ security.py
■■■ requirements.txt
■■■ .env
■■■ venv/

Key Code Snippets:

requirements.txt:
fastapi==0.109.0
uvicorn[standard]==0.27.0
sqlalchemy==2.0.36
psycopg[binary]==3.1.18
python-multipart==0.0.6
python-dotenv==1.0.0
bcrypt==4.1.2
PyJWT==2.8.0
pydantic==2.9.2
pydantic-settings==2.6.1
email-validator==2.2.0
huggingface-hub==0.20.3
PyPDF2==3.0.1
python-docx==1.1.0

app/config.py:
from pydantic_settings import BaseSettings, SettingsConfigDict
from typing import Optional

class Settings(BaseSettings):
DATABASE_URL: str = "postgresql://user:password@localhost:5432/interview_prep"

SECRET_KEY: str = "dev-secret-key-change-in-production"
ALGORITHM: str = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES: int = 30
HUGGINGFACE_TOKEN: Optional[str] = None

model_config = SettingsConfigDict(
env_file=".env",
case_sensitive=True,
extra="ignore"
)

settings = Settings()

app/utils/security.py:
import bcrypt
import jwt
from datetime import datetime, timedelta
from typing import Optional
from app.config import settings

def hash_password(password: str) -> str:
salt = bcrypt.gensalt()
hashed = bcrypt.hashpw(password.encode("utf-8"), salt)
return hashed.decode("utf-8")

def verify_password(plain_password: str, hashed_password: str) -> bool:
try:
return bcrypt.checkpw(
plain_password.encode("utf-8"),
hashed_password.encode("utf-8")
)
except Exception:
return False

def create_access_token(data: dict, expires_delta: Optional[timedelta] = None) -> str:
to_encode = data.copy()
if expires_delta:
expire = datetime.utcnow() + expires_delta
else:
expire = datetime.utcnow() + timedelta(minutes=settings.ACCESS_TOKEN_EXPIRE_MINUTES)
to_encode.update({"exp": expire})
return jwt.encode(to_encode, settings.SECRET_KEY, algorithm=settings.ALGORITHM)

def decode_access_token(token: str) -> Optional[dict]:
try:
return jwt.decode(token, settings.SECRET_KEY, algorithms=[settings.ALGORITHM])
except:
return None

Challenges Faced:
1. ■ psycopg2-binary compilation error → ■ Fixed by using psycopg[binary]
2. ■ SQLAlchemy Python 3.13 compatibility → ■ Fixed by upgrading to 2.0.36
3. ■ Pydantic settings configuration → ■ Fixed with model_config
4. ■ Database URL format → ■ Changed to postgresql+psycopg://

Day 2: Database & Authentication ■
Completed:
- ■ Supabase PostgreSQL database connected

- ■ User model created with SQLAlchemy
- ■ Database tables created
- ■ Pydantic schemas for request/response validation
- ■ Registration endpoint working
- ■ Login endpoint (returns JWT token)
- ■ Protected endpoints with authentication
- ■ Get/update user profile endpoints

Database Schema:
CREATE TABLE users (
id SERIAL PRIMARY KEY,
email VARCHAR(255) UNIQUE NOT NULL,
hashed_password VARCHAR(255) NOT NULL,
full_name VARCHAR(255) NOT NULL,
is_active BOOLEAN DEFAULT TRUE,
is_verified BOOLEAN DEFAULT FALSE,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

API Endpoints Created:
POST /api/v1/auth/register - Register new user
POST /api/v1/auth/login - Login (get JWT token)
POST /api/v1/auth/logout - Logout
GET /api/v1/users/me - Get current user profile
PUT /api/v1/users/me - Update user profile
DELETE /api/v1/users/me - Delete user account

Key Files:

app/models/user.py:
from sqlalchemy import Column, Integer, String, DateTime, Boolean
from app.database import Base
import datetime

class User(Base):
__tablename__ = "users"

id = Column(Integer, primary_key=True, index=True)
email = Column(String(255), unique=True, index=True, nullable=False)
hashed_password = Column(String(255), nullable=False)
full_name = Column(String(255), nullable=False)
is_active = Column(Boolean, default=True)
is_verified = Column(Boolean, default=False)
created_at = Column(DateTime, default=datetime.datetime.utcnow)
updated_at = Column(DateTime, default=datetime.datetime.utcnow,
onupdate=datetime.datetime.utcnow)

app/routers/auth.py (key functions):
@router.post("/register", response_model=UserResponse, status_code=201)
def register(user_data: UserCreate, db: Session = Depends(get_db)):
# Check if email exists
existing_user = db.query(User).filter(User.email == user_data.email).first()
if existing_user:
raise HTTPException(status_code=400, detail="Email already registered")

# Create new user
new_user = User(
email=user_data.email,

```
hashed_password=hash_password(user_data.password),
full_name=user_data.full_name
)
db.add(new_user)
db.commit()
db.refresh(new_user)
return new_user

@router.post("/login", response_model=Token)
def login(form_data: OAuth2PasswordRequestForm = Depends(), db: Session = Depends(get_db)):
user = db.query(User).filter(User.email == form_data.username).first()
if not user or not verify_password(form_data.password, user.hashed_password):
raise HTTPException(status_code=401, detail="Incorrect email or password")

access_token = create_access_token(data={"sub": user.email})
return {"access_token": access_token, "token_type": "bearer"}
```

Testing Done:
- ■ User registration via Swagger UI
- ■ Login and receive JWT token
- ■ Access protected endpoints with token
- ■ Token validation working
- ■ Error handling for duplicate emails
- ■ Password validation working

Days 3-4: Flutter Frontend ■
Completed:
- ■ Flutter project created
- ■ Dependencies installed (Riverpod, GoRouter, Dio)
- ■ Project structure organized
- ■ Theme system (light/dark)
- ■ API service with Dio
- ■ Auth service with token management
- ■ State management with Riverpod
- ■ Login screen UI
- ■ Registration screen UI
- ■ Home screen UI
- ■ Navigation with GoRouter
- ■ Form validation
- ■ Loading states
- ■ Error handling

Flutter Project Structure:
```
frontend/
■■■ lib/
■ ■■■ main.dart
■ ■■■ core/
■ ■ ■■■ constants/
■ ■ ■ ■■■ api_constants.dart
■ ■ ■■■ theme/
■ ■ ■ ■■■ app_theme.dart
■ ■ ■■■ router/
■ ■ ■■■ app_router.dart
■ ■■■ features/
■ ■ ■■■ auth/
■ ■ ■ ■■■ models/
```

■ ■ ■ ■ ■■■ user_model.dart
■ ■ ■ ■■■ providers/
■ ■ ■ ■ ■■■ auth_provider.dart
■ ■ ■ ■■■ screens/
■ ■ ■ ■■■ login_screen.dart
■ ■ ■ ■■■ register_screen.dart
■ ■ ■■■ home/
■ ■ ■■■ screens/
■ ■ ■■■ home_screen.dart
■ ■■■ services/
■ ■ ■■■ api_service.dart
■ ■ ■■■ auth_service.dart
■ ■■■ shared/
■ ■■■ widgets/
■ ■■■ custom_button.dart
■ ■■■ custom_text_field.dart
■■■ pubspec.yaml

Key Dependencies:
dependencies:
flutter_riverpod: ^2.4.9 # State management
go_router: ^13.0.0 # Navigation
dio: ^5.4.0 # HTTP client
shared_preferences: ^2.2.2 # Local storage
flutter_secure_storage: ^9.0.0 # Secure token storage

State Management Pattern:
// Auth State
class AuthState {
final bool isLoading;
final bool isAuthenticated;
final String? error;
final User? user;
}

// Auth Notifier
class AuthNotifier extends StateNotifier {
final AuthService _authService;

// Check auth status on app start
Future checkAuthStatus() async { }

// Register new user
Future register({required String email, required String password, required String fullName}) async { }

// Login user
Future login({required String email, required String password}) async { }

// Logout user
Future logout() async { }
}

Navigation Routes:
/login → Login Screen
/register → Registration Screen
/home → Home Screen (requires auth)

UI Features:
- Material Design 3

- Custom theme with primary blue color
- Form validation (email format, password length, matching passwords)
- Loading indicators during API calls
- Error messages via SnackBar
- Secure password fields with show/hide toggle
- Responsive design

Testing Done:
- ■ Complete registration flow
- ■ Login flow with token storage
- ■ Navigation between screens
- ■ Token persistence (stays logged in)
- ■ Logout functionality
- ■ Protected routes
- ■ API integration working
- ■ Error handling (duplicate email, wrong password)

■ IMPORTANT TECHNICAL NOTES

Python 3.13 Compatibility Issues Fixed
1. **psycopg2-binary** doesn't work → Use `psycopg[binary]==3.1.18`
2. **SQLAlchemy 2.0.25** has issues → Use `sqlalchemy==2.0.36`
3. **Pydantic 2.x** config syntax changed:
# Old way (doesn't work)
class Config:
env_file = ".env"

# New way (works)
model_config = SettingsConfigDict(env_file=".env", extra="ignore")

Database Connection String Format

Correct format with psycopg3

postgresql+psycopg://postgres:password@host:5432/database

Wrong format (will fail)

postgresql://postgres:password@host:5432/database

JWT Token Flow
1. User logs in → Backend creates JWT token
2. Frontend stores token in Flutter Secure Storage
3. All API requests include: `Authorization: Bearer `
4. Backend validates token with `get_current_user` dependency
5. Logout → Delete token from storage

Environment Variables (.env)
DATABASE_URL=postgresql+psycopg://postgres:YOUR_PASSWORD@db.xxx.supabase.co:5432/postgres
SECRET_KEY=your-super-secret-key-min-32-characters
HUGGINGFACE_TOKEN=hf_your_token_here
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=30

■ REMAINING WORK (Weeks 2-5)

Week 2: Resume Module - Backend (Days 5-9)
Phase 2 from original plan

Day 5: Resume Upload System
- [ ] Create Resume model (SQLAlchemy)
- [ ] Create Resume schemas (Pydantic)
- [ ] File upload endpoint (PDF/DOCX)
- [ ] File validation (size, type)
- [ ] Store files securely
- [ ] List/get/delete resume endpoints

Day 6: Resume Parsing
- [ ] Integrate PyPDF2 for PDF parsing
- [ ] Integrate python-docx for DOCX parsing
- [ ] Extract contact information (email, phone, LinkedIn)
- [ ] Extract education section
- [ ] Extract work experience
- [ ] Extract skills
- [ ] Handle different resume formats

Day 7: Skill Extraction with NLP
- [ ] Install and configure spaCy
- [ ] Create comprehensive skills database (200+ skills)
- [ ] Extract technical skills
- [ ] Extract soft skills
- [ ] Categorize skills (programming, tools, frameworks, etc.)
- [ ] Rate proficiency levels

Day 8: AI Resume Analysis
- [ ] Setup Hugging Face Inference API

- [ ] Choose model: Mistral-7B-Instruct-v0.2 (recommended)
- [ ] Create analysis prompts
- [ ] Overall resume score (1-10)
- [ ] Identify strengths (3-5 points)
- [ ] Identify weaknesses (3-5 points)
- [ ] ATS compatibility check
- [ ] Keyword recommendations

Day 9: Resume Generation & Templates
- [ ] Create resume templates (Professional, Modern, Creative)
- [ ] Template database/JSON structure
- [ ] Resume generation service (python-docx)
- [ ] Apply formatting (headers, bullets, fonts)
- [ ] Generate downloadable DOCX files
- [ ] Test with multiple templates

Models to Create:
class Resume(Base):
__tablename__ = "resumes"
id = Column(Integer, primary_key=True)
user_id = Column(Integer, ForeignKey("users.id"))
title = Column(String(255))
file_path = Column(String(500))
file_type = Column(String(10))
parsed_content = Column(Text)
contact_info = Column(JSON)
education = Column(JSON)
experience = Column(JSON)
skills = Column(JSON)
analysis_score = Column(Float)
analysis_feedback = Column(JSON)
created_at = Column(DateTime)
updated_at = Column(DateTime)

API Endpoints to Create:
POST /api/v1/resumes/upload - Upload resume file
GET /api/v1/resumes - List user's resumes
GET /api/v1/resumes/{id} - Get resume details
POST /api/v1/resumes/{id}/parse - Parse resume
POST /api/v1/resumes/{id}/analyze - AI analysis
POST /api/v1/resumes/{id}/generate - Generate formatted resume
DELETE /api/v1/resumes/{id} - Delete resume

Hugging Face Integration:
from huggingface_hub import InferenceClient

client = InferenceClient(token=settings.HUGGINGFACE_TOKEN)

def analyze_resume(resume_text: str) -> dict:
prompt = f"""You are an expert resume reviewer. Analyze this resume:

{resume_text}

Provide analysis in JSON format:
{{
"overall_score": <1-10>,
"strengths": [],
"weaknesses": [],
"ats_score": <1-10>,

"keywords": []
}}"""

response = client.text_generation(
"mistralai/Mistral-7B-Instruct-v0.2",
prompt=prompt,
max_new_tokens=1000
)
return parse_json_response(response)

Week 3: Resume Module - Frontend (Days 10-12)

Day 10: Resume UI Components
- [ ] Create Resume model (Dart)
- [ ] Create Resume provider (Riverpod)
- [ ] Resume list screen
- [ ] File picker integration
- [ ] Upload progress indicator
- [ ] Resume card widget

Day 11: Resume Detail & Display
- [ ] Resume detail screen
- [ ] Display parsed information
- [ ] Contact info section
- [ ] Skills section
- [ ] Education section
- [ ] Experience section
- [ ] Analysis results display

Day 12: Resume Actions
- [ ] Parse button
- [ ] Analyze button
- [ ] Generate button
- [ ] Download functionality
- [ ] Delete confirmation dialog
- [ ] Edit resume metadata

Flutter Screens:
/resumes → Resume List Screen
/resumes/upload → Upload Resume Screen
/resumes/:id → Resume Detail Screen
/resumes/:id/analyze → Analysis Results Screen

Week 4-5: AI Interview Module (Days 13-20)

Backend Tasks:
- [ ] Interview model (session, messages, scores)
- [ ] Start interview endpoint
- [ ] Send message endpoint
- [ ] End interview endpoint
- [ ] Interview evaluation logic
- [ ] Question generation with AI
- [ ] Response evaluation with AI
- [ ] Interview history storage

Frontend Tasks:

- [ ] Interview setup screen (role, difficulty)
- [ ] Chat interface
- [ ] Real-time message display
- [ ] Speech-to-text (optional)
- [ ] Interview feedback screen
- [ ] Interview history list
- [ ] Replay past interviews

AI Models to Use:
- Llama-2-13b-chat-hf (conversation)
- Mistral-7B-Instruct (evaluation)

Week 6-7: Skill Roadmaps (Days 21-28)

Backend Tasks:
- [ ] Roadmap model
- [ ] Skill gap analysis
- [ ] Learning path generation
- [ ] Resource recommendations
- [ ] Progress tracking
- [ ] Milestone system

Frontend Tasks:
- [ ] Roadmap visualization
- [ ] Progress dashboard
- [ ] Resource browser
- [ ] Goal setting interface
- [ ] Achievement badges

Week 8: Testing & Bug Fixes (Days 29-32)
- [ ] Integration testing
- [ ] Performance testing
- [ ] Security testing
- [ ] Bug fixing
- [ ] Code optimization
- [ ] Documentation

Week 9: Deployment (Days 33-36)
- [ ] Backend deployment (Render/Railway)
- [ ] Frontend deployment (Vercel/Firebase)
- [ ] Database migration to production
- [ ] Environment variables setup
- [ ] SSL/HTTPS configuration
- [ ] Monitoring setup (Sentry)
- [ ] Analytics integration

Week 10: Polish & Launch (Days 37-40)
- [ ] User acceptance testing
- [ ] UI/UX improvements
- [ ] Performance optimization
- [ ] Documentation completion
- [ ] Demo video creation

- [ ] Launch preparation

■ CURRENT STATE SUMMARY

What's Working ■
1. **Backend:**
- FastAPI server running on port 8000
- PostgreSQL database connected (Supabase)
- User authentication with JWT
- Protected endpoints
- CORS configured

2. **Frontend:**
- Flutter app running on Chrome/mobile
- Login/Register screens functional
- Home screen with user profile
- API integration working
- Token storage and management
- Navigation working

3. **Git Repository:**
- All code committed
- .gitignore configured
- README.md exists

What's NOT Done Yet ■
1. **Resume Module** (0% complete)
2. **Interview Module** (0% complete)
3. **Roadmap Module** (0% complete)
4. **Deployment** (0% complete)

Overall Progress: 25% (Week 1 of 4 core weeks)

■ NEXT STEPS (Week 2 - Resume Module)

Immediate Tasks (Day 5):

Backend:
1. Create Resume model in `app/models/resume.py`
2. Create Resume schemas in `app/schemas/resume.py`
3. Create file upload utilities
4. Create resume router with upload endpoint
5. Test file upload with Postman

Frontend:
1. Create Resume model in Dart
2. Create Resume provider
3. Create file picker service
4. Create Resume list screen UI

5. Create upload button

Estimated Time: 8 hours

■ IMPORTANT COMMANDS REFERENCE

Backend Commands

Activate virtual environment

```
cd backend
.\venv\Scripts\Activate # Windows
source venv/bin/activate # Mac/Linux
```

Install dependencies

pip install -r requirements.txt

Run server

uvicorn app.main:app --reload

Run on specific port

```
uvicorn app.main:app --reload --port 8000
```

Create database tables

```
python init_db.py
```

Test database connection

python test_db.py

Frontend Commands
cd frontend

Install dependencies

flutter pub get

Run on Chrome

flutter run -d chrome

Run on Android emulator

flutter run

List devices

flutter devices

Clean build

flutter clean
flutter pub get

Build for production

flutter build web
flutter build apk

Git Commands

Check status

git status

Add all files

git add .

Commit

git commit -m "message"

Push

git push origin main

Pull latest

git pull origin main

■ COMMON ISSUES & SOLUTIONS

Backend Issues

Issue 1: ModuleNotFoundError
Solution: Make sure virtual environment is activated
.\venv\Scripts\Activate

Issue 2: Database connection failed
Solution: Check DATABASE_URL in .env file
Ensure it starts with: postgresql+psycopg://

Issue 3: Import errors
Solution: Restart server after creating new files
Ctrl+C then uvicorn app.main:app --reload

Frontend Issues

Issue 1: Connection refused (localhost:8000)
Solution:
- Backend must be running
- Check baseUrl in api_constants.dart
- Use http://10.0.2.2:8000 for Android emulator

Issue 2: DioException [bad response]
Solution: Check backend logs for actual error
Print response in Flutter: print(e.response?.data)

Issue 3: Token not persisting
Solution: Check FlutterSecureStorage permissions
Verify token is being saved: await _storage.read(key: 'access_token')

■ PROJECT FILE STRUCTURE (Complete)

interview-prep-ai-1/
■■■ backend/
■ ■■■ app/
■ ■ ■■■ __init__.py
■ ■ ■■■ main.py
■ ■ ■■■ config.py
■ ■ ■■■ database.py
■ ■ ■■■ models/
■ ■ ■ ■■■ __init__.py
■ ■ ■ ■■■ user.py

■ ■ ■■■ schemas/
■ ■ ■ ■■■ __init__.py
■ ■ ■ ■■■ user.py
■ ■ ■■■ routers/
■ ■ ■ ■■■ __init__.py
■ ■ ■ ■■■ auth.py
■ ■ ■ ■■■ users.py
■ ■ ■■■ services/
■ ■ ■ ■■■ __init__.py
■ ■ ■■■ utils/
■ ■ ■■■ __init__.py
■ ■ ■■■ security.py
■ ■■■ venv/
■ ■■■ requirements.txt
■ ■■■ .env
■ ■■■ init_db.py
■ ■■■ test_db.py
■■■ frontend/
■ ■■■ lib/
■ ■ ■■■ main.dart
■ ■ ■■■ core/
■ ■ ■ ■■■ constants/
■ ■ ■ ■ ■■■ api_constants.dart
■ ■ ■ ■■■ theme/
■ ■ ■ ■ ■■■ app_theme.dart
■ ■ ■ ■■■ router/
■ ■ ■ ■■■ app_router.dart
■ ■ ■■■ features/
■ ■ ■ ■■■ auth/
■ ■ ■ ■ ■■■ models/
■ ■ ■ ■ ■ ■■■ user_model.dart
■ ■ ■ ■ ■■■ providers/
■ ■ ■ ■ ■ ■■■ auth_provider.dart
■ ■ ■ ■ ■■■ screens/
■ ■ ■ ■ ■■■ login_screen.dart
■ ■ ■ ■ ■■■ register_screen.dart
■ ■ ■ ■■■ home/
■ ■ ■ ■■■ screens/
■ ■ ■ ■■■ home_screen.dart
■ ■ ■■■ services/
■ ■ ■ ■■■ api_service.dart
■ ■ ■ ■■■ auth_service.dart
■ ■ ■■■ shared/
■ ■ ■■■ widgets/
■ ■ ■■■ custom_button.dart
■ ■ ■■■ custom_text_field.dart
■ ■■■ pubspec.yaml
■ ■■■ android/ios/web/
■■■ docs/
■ ■■■ PROJECT_DOCUMENTATION.md (this file)
■■■ .gitignore
■■■ README.md

■ API ENDPOINT REFERENCE

Authentication Endpoints
POST /api/v1/auth/register
Body: {
"email": "user@example.com",
"password": "password123",
"full_name": "John Doe"
}
Response: 201 Created
{
"id": 1,
"email": "user@example.com",
"full_name": "John Doe",
"is_active": true,
"is_verified": false,
"created_at": "2026-02-09T12:00:00",
"updated_at": "2026-02-09T12:00:00"
}

POST /api/v1/auth/login
Body (form-data): {
"username": "user@example.com",
"password": "password123"
}
Response: 200 OK
{
"access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
"token_type": "bearer"
}

GET /api/v1/users/me
Headers: {
"Authorization": "Bearer "
}
Response: 200 OK
{
"id": 1,
"email": "user@example.com",
"full_name": "John Doe",
...
}

■ PROGRESS METRICS

Time Spent: ~16 hours
- Day 1: 4 hours (setup, debugging)
- Day 2: 6 hours (database, auth)
- Days 3-4: 6 hours (Flutter UI)

Lines of Code Written: ~2,500

- Backend: ~1,200 lines
- Frontend: ~1,300 lines

Files Created: 28
- Backend: 15 files
- Frontend: 13 files

Git Commits: 4
1. "Day 1 Complete: Backend foundation"
2. "Day 2 Complete: User authentication"
3. "Days 3-4 Complete: Flutter frontend"
4. Current documentation

■ LESSONS LEARNED

Technical Lessons
1. **Python 3.13 is bleeding edge** - Had compatibility issues, but solved them
2. **Always use `postgresql+psycopg://`** with psycopg3
3. **Pydantic 2.x has breaking changes** from 1.x
4. **Flutter Riverpod is powerful** but requires understanding state management
5. **JWT tokens need secure storage** - Use Flutter Secure Storage

Process Lessons
1. **Start with backend first** - Frontend depends on it
2. **Test each component immediately** - Don't wait until the end
3. **Document as you go** - This file would be harder to create later
4. **Commit frequently** - Small, focused commits are better
5. **One feature at a time** - Don't try to build everything at once

---

■ AI MODEL STRATEGY

Models to Use (Hugging Face)

For Resume Analysis:
- **Primary:** `mistralai/Mistral-7B-Instruct-v0.2`
- **Why:** Best instruction-following, great for structured outputs
- **Backup:** `meta-llama/Llama-2-13b-chat-hf`

For Interview Conversation:
- **Primary:** `meta-llama/Llama-2-13b-chat-hf`
- **Why:** Strong conversational abilities
- **Backup:** `mistralai/Mixtral-8x7B-Instruct-v0.1` (slower but higher quality)

For Skill Gap Analysis:
- **Primary:** `mistralai/Mistral-7B-Instruct-v0.2`
- **Why:** Good at analysis and recommendations

API Usage Tips
1. Use Hugging Face Inference API (free tier)

2. Cache common responses to save costs
3. Set reasonable max_tokens (500-1000)
4. Handle rate limits gracefully
5. Always have fallback responses

■ BEST PRACTICES ESTABLISHED

Backend
- Always use dependency injection (`Depends()`)
- Validate all inputs with Pydantic schemas
- Use HTTPException for errors
- Return appropriate status codes
- Document endpoints with docstrings
- Keep routes, models, and services separate

Frontend
- Use Riverpod for state management
- Keep widgets small and reusable
- Handle loading and error states
- Use secure storage for tokens
- Validate forms before submission
- Show user feedback (SnackBar)

Database
- Use migrations for schema changes
- Index frequently queried fields
- Use foreign keys for relationships
- Set cascade delete appropriately
- Always use timestamps

Security
- Hash passwords with bcrypt
- Use JWT with expiration
- Validate tokens on every protected endpoint
- Use HTTPS in production
- Never commit .env files
- Rotate secrets regularly

■ SUCCESS CRITERIA

Minimum Viable Product (MVP) - End of Week 10
- [ ] User can register and login
- [ ] User can upload and parse resume
- [ ] User can get AI feedback on resume
- [ ] User can practice interview with AI (3-5 questions)
- [ ] User can see basic skill gaps
- [ ] App is deployed and accessible online

Full Product - End of Week 20
- [ ] All MVP features +
- [ ] Multiple resume templates
- [ ] Full interview sessions (10-15 questions)
- [ ] Detailed interview analytics
- [ ] Complete skill roadmaps
- [ ] Progress tracking
- [ ] Mobile app (Android/iOS)
- [ ] Production-grade deployment

■ CONTACT & REPOSITORY INFO

GitHub Repository: https://github.com/YOUR_USERNAME/interview-prep-ai-1
Developer: Solo Developer
Project Type: Capstone Project
Duration: 20 weeks (February - July 2026)

■ CRITICAL REMINDERS FOR CONTINUATION

Before Starting Next Session:

1. **Activate Backend:**
cd backend
.\venv\Scripts\Activate
uvicorn app.main:app --reload

2. **Check Backend is Running:**
- Open http://localhost:8000/docs
- Should see Swagger UI

3. **Start Frontend (if needed):**
cd frontend
flutter run -d chrome

4. **Pull Latest Code:**
git pull origin main

Week 2 First Task:
Create Resume model in backend → Test with Postman → Create frontend UI

■ ROADMAP VISUALIZATION

Week 1 ■ DONE
■■■ Day 1: Backend Setup ■
■■■ Day 2: Authentication ■

■■■ Days 3-4: Flutter UI ■

Week 2-3 ■ NEXT
■■■ Day 5-9: Resume Backend
■■■ Day 10-12: Resume Frontend

Week 4-5 ■ FUTURE
■■■ AI Interview Backend
■■■ AI Interview Frontend

Week 6-7 ■ FUTURE
■■■ Skill Roadmaps Backend
■■■ Skill Roadmaps Frontend

Week 8-10 ■ FINAL
■■■ Testing & Bug Fixes
■■■ Deployment
■■■ Launch

■ CONCLUSION

Week 1 Status: SUCCESSFULLY COMPLETED ■

You have built:
- A production-ready authentication system
- A beautiful Flutter mobile app
- A solid foundation for the remaining features

Next Milestone: Resume Module (Week 2-3)

Confidence Level: HIGH - Your advanced Python skills showed, and you've overcome initial Flutter challenges

Motivation: 25% done, 75% to go. The hard part (foundation) is done. Now we build features! ■

Document Version: 1.0
Last Updated: February 9, 2026
Status: Week 1 Complete, Ready for Week 2

This document contains everything needed to continue the project in a new chat session. Read this first, then ask: "Let's continue with Week 2, Day 5!"