Interface Contracts Setup - Everyone Do This First! ⚡

🎯 Goal: Define ALL interfaces so everyone can work independently

Timeline: Complete this in **Hour 0-1** before anyone writes real code!

📋 STEP 1: Create Shared Documents (Person 5 - 10 minutes)

Person 5: Create these files in a shared Google Doc or GitHub repo

1. `api-contracts.md` - All API endpoints
2. `data-models.md` - All data structures
3. `file-structure.md` - Project organization
4. `.env.example` - Environment variables template

Share the links in your team chat NOW!

🔧 STEP 2: Everyone Review & Agree on These Contracts

📊 DATA MODELS (Everyone must agree on these!)

```typescript
```

```
// ================================================
// CLASSROOM
// ================================================
interface Classroom {
  id: string;                    // UUID
  name: string;                  // "Physics Grade 8"
  subject: string;               // "Physics"
  grade_level: string;           // "8"
  story_theme: string;           // "Space Adventure"
  design_style: string;          // "manga" | "comic" | "cartoon"
  duration: string;              // "Fall 2024"
  created_at: string;            // ISO timestamp
}

// ================================================
// STUDENT
// ================================================
interface Student {
  id: string;                    // UUID
  classroom_id: string;          // Foreign key to classroom
  name: string;                  // "Alex Johnson"
  interests: string;             // "Football, robots, pizza"
  avatar_url: string | null;     // URL to generated avatar image
  photo_url: string | null;      // URL to uploaded photo (optional)
  created_at: string;            // ISO timestamp
}

// ================================================
// STORY
// ================================================
interface Story {
  id: string;                    // UUID
  classroom_id: string;          // Foreign key to classroom
  lesson_prompt: string;         // "Today we learned about Newton's Laws"
  title: string;                 // "The Playground Physics Mystery"
  status: "generating" | "completed" | "failed";
  progress: number;              // 0-100
  created_at: string;            // ISO timestamp
}

// ================================================
// PANEL
// ================================================
interface Panel {
  id: string;                    // UUID
  story_id: string;              // Foreign key to story
  panel_number: number;          // 1-20
  image_url: string;             // URL to generated panel image
  dialogue: string;              // "For every action, there's a reaction!"
  scene_description: string;     // For regeneration prompts
  created_at: string;            // ISO timestamp
}

// ============================================
// STORY OPTION (Temporary - before selection)
// ============================================
interface StoryOption {
  id: string;                    // Temp ID
  title: string;                 // "Space Station Physics"
  summary: string;               // "Students discover Newton's laws..."
  theme: string;                 // "space" | "playground" | "sports"
}
```

🌐 API ENDPOINTS (Backend implements, Frontend calls)

typescript

```
// ================================================
// CLASSROOM ENDPOINTS
// ================================================

// Create Classroom
POST /api/classrooms
Request: {
  name: string;
  subject: string;
  grade_level: string;
  story_theme: string;
  design_style: string;
  duration: string;
}
Response: Classroom

// Get Classroom
GET /api/classrooms/{classroom_id}
Response: Classroom

// Get All Classrooms (for teacher)
GET /api/classrooms
Response: Classroom[]

// ================================================
// STUDENT ENDPOINTS
// ================================================

// Create Student (from student signup)
POST /api/students
Request: {
  classroom_id: string;
  name: string;
  interests: string;
}
Response: Student

// Upload Student Photo
POST /api/students/{student_id}/photo
Request: FormData { photo: File }
Response: {
  photo_url: string;
}

// Generate Avatar (trigger FLUX)
POST /api/students/{student_id}/generate-avatar
```

```
Request: {
  use_photo: boolean;  // true if photo uploaded
}
Response: {
  avatar_url: string;
  status: "generating" | "completed";
}

// Get Students in Classroom
GET /api/classrooms/{classroom_id}/students
Response: Student[]

// Get Single Student
GET /api/students/{student_id}
Response: Student

// ============================================
// STORY GENERATION ENDPOINTS
// ============================================

// Generate 3 Story Options
POST /api/stories/generate-options
Request: {
  classroom_id: string;
  lesson_prompt: string;  // "Today we learned about Newton's Laws"
}
Response: {
  options: StoryOption[];  // Array of 3 options
}

// Generate Full Story (20 panels)
POST /api/stories/generate
Request: {
  classroom_id: string;
  selected_option_id: string;
  lesson_prompt: string;
}
Response: {
  story_id: string;
  status: "generating";
  progress: 0;
}

// Check Story Generation Progress
GET /api/stories/{story_id}/progress
Response: {
  story_id: string;
  status: "generating" | "completed" | "failed";
  progress: number;  // 0-100
  panels_completed: number;  // e.g., 5 out of 20
}

// Get Complete Story with Panels
GET /api/stories/{story_id}
Response: {
  story: Story;
  panels: Panel[];  // Sorted by panel_number
  students: Student[];  // Students featured in story
}

// Regenerate Specific Panels
POST /api/stories/{story_id}/regenerate
Request: {
  panel_numbers: number[];  // [3, 7, 12]
  correction_prompt: string;  // "Make panel 3 brighter"
}
Response: {
  status: "regenerating";
}

// Get All Stories for Classroom
GET /api/classrooms/{classroom_id}/stories
Response: Story[]

// ============================================
// EXPORT ENDPOINTS
// ============================================

// Export Story as PDF
GET /api/stories/{story_id}/export/pdf
Response: File (application/pdf)

// Get PDF URL (alternative)
POST /api/stories/{story_id}/export/pdf
Response: {
  pdf_url: string;
  expires_at: string;
}
```

🎨 FRONTEND ROUTES (Person 3 & 4 implement)

typescript

```
// ============================================
// TEACHER ROUTES (Person 3)
// ============================================

/                          → Landing page
/teacher/dashboard            → Teacher home (list of classrooms)
/teacher/classroom/new        → Create new classroom form
/teacher/classroom/{id}       → Classroom detail (students, stories)
/teacher/classroom/{id}/invite → Student invite link page
/teacher/story/new            → New story generation wizard
/teacher/story/{id}           → View/edit story
/teacher/story/{id}/export     → Export options

// ============================================
// STUDENT ROUTES (Person 4)
// ============================================

/join/{classroom_id}          → Student signup page
/student/avatar/create        → Avatar creation wizard
/student/stories              → View all stories
/student/story/{id}           → Read specific story
```

PROJECT FILE STRUCTURE

```
hackathon-project/
├── backend/                  # Person 1 & 2
│   ├── main.py               # FastAPI app
│   ├── models.py             # Pydantic models
│   ├── database.py           # Supabase client
│   ├── routers/
│   │   ├── classrooms.py       # Classroom endpoints
│   │   ├── students.py         # Student endpoints
│   │   ├── stories.py          # Story endpoints
│   │   └── export.py           # Export endpoints
│   ├── services/
│   │   ├── openai_service.py   # GPT integration
│   │   ├── flux_service.py     # FLUX integration
│   │   └── avatar_service.py   # Avatar generation
│   ├── requirements.txt
│   └── .env
│
├── frontend/                 # Person 3 & 4
│   ├── src/
│   │   ├── components/
│   │   │   ├── teacher/        # Person 3
│   │   │   │   ├── ClassroomForm.tsx
│   │   │   │   ├── StoryGenerator.tsx
│   │   │   │   └── StoryViewer.tsx
│   │   │   └── student/        # Person 4
│   │   │       ├── AvatarCreator.tsx
│   │   │       ├── StudentSignup.tsx
│   │   │       └── StoryReader.tsx
│   │   ├── lib/
│   │   │   └── api.ts          # API client functions
│   │   ├── types/
│   │   │   └── index.ts        # TypeScript interfaces
│   │   └── App.tsx
│   └── package.json
│
└── docs/                       # Person 5
    ├── api-contracts.md
    ├── data-models.md
    └── deployment.md
```

✅ WHAT EACH PERSON DOES NOW (Next 30 minutes)

👤 Person 1 (Backend Lead)

python

```python
```

```python
# 1. Create backend/models.py with Pydantic models
from pydantic import BaseModel

class ClassroomCreate(BaseModel):
    name: str
    subject: str
    grade_level: str
    story_theme: str
    design_style: str = "manga"
    duration: str

class StudentCreate(BaseModel):
    classroom_id: str
    name: str
    interests: str

# ... add all other models

# 2. Create backend/main.py with route STUBS
from fastapi import FastAPI
app = FastAPI()

@app.post("/api/classrooms")
async def create_classroom(classroom: ClassroomCreate):
    # TODO: Implement
    return {"message": "stub"}

# ... add all other route stubs

# 3. Test that server runs: uvicorn main:app --reload
```

👤 Person 2 (Database)

python

```
# 1. Set up Supabase account (10 min)
# 2. Create database schema (copy SQL from previous artifact)
# 3. Create backend/database.py

from supabase import create_client
import os

supabase = create_client(
    os.getenv("SUPABASE_URL"),
    os.getenv("SUPABASE_KEY")
)

# 4. Share .env file with team:
"""
SUPABASE_URL=https://xxx.supabase.co
SUPABASE_KEY=your_key_here
OPENAI_API_KEY=sk-...
FLUX_API_KEY=...
"""
```

👤 Person 3 (Teacher Frontend)

typescript

```
// 1. Create frontend/src/types/index.ts
export interface Classroom {
  id: string;
  name: string;
  subject: string;
  grade_level: string;
  story_theme: string;
  design_style: string;
  duration: string;
  created_at: string;
}

export interface Story { /* ... */ }
// ... add all interfaces

// 2. Create frontend/src/lib/api.ts with API client
const API_BASE = "http://localhost:8000/api";

export async function createClassroom(data: any) {
  const res = await fetch(`${API_BASE}/classrooms`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(data)
  });
  return res.json();
}

// ... add all API functions

// 3. Create component SKELETONS (empty UI)
// ClassroomForm.tsx, StoryGenerator.tsx, etc.
```

👤 Person 4 (Student Frontend)

```
typescript
```

```
// 1. Use the same types/index.ts from Person 3
// 2. Use the same api.ts from Person 3
// 3. Create component SKELETONS
// AvatarCreator.tsx, StudentSignup.tsx, StoryReader.tsx

// 4. Create mock data for development:
// frontend/src/lib/mockData.ts
export const mockStudent = {
  id: "1",
  name: "Alex",
  interests: "Soccer",
  avatar_url: "https://placeholder.com/150"
};
```

👤 Person 5 (DevOps)

```bash
# 1. Set up GitHub repo
git init
git add .
git commit -m "Initial structure"

# 2. Create .env.example
SUPABASE_URL=your_url_here
SUPABASE_KEY=your_key_here
OPENAI_API_KEY=your_key_here
FLUX_API_KEY=your_key_here

# 3. Set up deployment (Vercel + Railway)
# - Connect frontend to Vercel
# - Connect backend to Railway
# - Add environment variables

# 4. Create docs/api-contracts.md (copy from here)

# 5. Test that Person 1's stub API works
curl http://localhost:8000/api/classrooms
```

🎯 CHECKPOINT: After 1 Hour, Everyone Should Have:

✅ **Person 1**: FastAPI running with stub endpoints
✅ **Person 2**: Supabase database created, .env shared
✅ **Person 3**: Frontend skeleton with API client

✅ **Person 4**: Frontend skeleton with mock data
✅ **Person 5**: Deployment configured, docs created

🚦 NEXT STEP: Integration Test (Hour 1)

Everyone runs this test together:

```bash
# Person 1: Backend running on http://localhost:8000
# Person 3: Frontend running on http://localhost:3000

# Test 1: Person 3 calls Person 1's API from browser console
fetch('http://localhost:8000/api/classrooms', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    name: "Test Class",
    subject: "Physics",
    grade_level: "8",
    story_theme: "Space",
    design_style: "manga",
    duration: "Fall 2024"
  })
}).then(r => r.json()).then(console.log);

# If you see CORS error, Person 1 adds this to main.py:
from fastapi.middleware.cors import CORSMiddleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"]
)
```

📣 COMMUNICATION: Use This Format in Chat

```
[BLOCKER] Person 3: API endpoint /api/classrooms returning 500
[QUESTION] Person 4: Should avatar_url be nullable?
[DONE] Person 2: Database schema created ✅
[HELP] Person 1: Need help with CORS issue
[UPDATE] Person 5: Deployment is live at https://xxx
```

⚠️ CRITICAL RULES

1. **NO CODE CHANGES TO INTERFACES** without team approval
2. **ASK IN CHAT** before changing any data model
3. **USE MOCK DATA** in frontend until backend is ready
4. **COMMIT OFTEN** - every 30 minutes
5. **SYNC EVERY 2 HOURS** - quick standup

🎬 Once Everyone Confirms "READY" in Chat...

START BUILDING THE REAL FEATURES!

Person 1 & 2: Implement real API logic
Person 3 & 4: Build actual UI components
Person 5: Monitor integration & help blockers

Good luck! 🚀