# StudyBuddy — Full-Stack Prototype (Community + Personal)

This single document contains all the files you can copy into a repo to demo a working prototype. The app mimics a **project-management** UI (Kanban-style) for study rooms, supports **AI tutoring & quizzing**, **whiteboard/pen**, **PDF → flipbook**, **Google Meet/Zoom links**, **invites**, and **IntaSend payments** (KES 650/user/year). It has clear **frontend / backend separation**.

Folder layout

```
studybuddy/
  README.md
  TECH_STACK.md
  PROMPTS.md
  backend/
    app.py
    requirements.txt
    routers/
      auth.py
      rooms.py
      ai.py
      files.py
      payments.py
    models.py
    db.py
  frontend/
    index.html
    styles.css
    app.js
    whiteboard.js
    flipbook.js
```

---

## FILE: README.md

```
# StudyBuddy — AI Study + Community Rooms (SDG 4)

**Problem**: Students struggle with consistency, affordable personalized help,
isolation, and generic tools.

**Solution**: Project-style study rooms with AI tutor (Q&A, summaries,
```

~~~
flashcards, practice tests), community features, whiteboard, PDF-to-flipbook
viewer, Google Meet/Zoom links, invites, and IntaSend payments.

## Key Features
- Project-style **Kanban** (To Do / Doing / Done) for tasks per study room
- **AI Tutor**: Q&A, summaries, flashcards, quizzes (Hugging Face / OpenAI)
- **Whiteboard**: pen tool with export
- **PDF Upload → Flipbook** viewer
- **Meet Links**: attach Google Meet or Zoom links to sessions
- **Invite** participants via shareable link
- **Payments**: IntaSend (KES 650/user/year) — stubbed checkout endpoint
- Responsive: **Mobile + Desktop**

## Quickstart
1. **Backend**
   ```bash
   cd backend
   python -m venv venv && source venv/bin/activate
   pip install -r requirements.txt
   export DB_URL="sqlite:///studybuddy.db"  # quick demo DB
   export HF_API="<optional_token>"  # if using Hugging Face
   uvicorn app:app --reload
   ```
2. **Frontend**
   - Open `frontend/index.html` in a dev server (e.g., `npx serve frontend`)

## Notes
- Payments & AI calls are wired with safe fallbacks/mocks for demo.
- Replace stub tokens and implement real keys before production.
~~~

~~~
---

## FILE: TECH_STACK.md
```md
# Tech Stack & Versions
- **Backend**: FastAPI 0.110+, Uvicorn 0.29+, Pydantic 2+
- **DB**: SQLite for demo (swap to PostgreSQL easily)
- **ORM**: SQLAlchemy 2.0+
- **Auth**: JWT (python-jose 3.3+)
- **AI**: requests to Hugging Face Inference API or OpenAI (optional)
- **Frontend**: Vanilla JS + HTML/CSS (no build step) for portability
- **UI**: Kanban-style boards, modal dialogs
- **Payments**: IntaSend (REST) — stub endpoints
- **PDF**: PDF.js for parsing text (simplified) + custom CSS flipbook
~~~

```
> Reasoning: Avoid collisions by pinning major versions; SQLite for fast demo,
switch to Postgres in env.
```

~~~
---

## FILE: PROMPTS.md
```md
# Prompts used during creation & runtime

## Logo Design Prompt
A modern and sleek logo for an AI-powered branding agency. Minimalist design,
futuristic font, blue and purple gradient color scheme, with an abstract AI
brain + digital spark symbol.

## AI Tutor — Summarize Notes
"Summarize these notes into 5 bullet points and 5 flashcards (Q/A). Return JSON:
{ bullets: string[], flashcards: {question, answer}[] }.\nNOTES:\n{{notes}}"

## AI Tutor — Quiz Generator
"Create 8 multiple-choice questions (A–D) from these notes with one correct
answer each. Return JSON array of {question, options:[A,B,C,D], correctIndex}.
\nNOTES:\n{{notes}}"

## AI Tutor — Q&A
"Answer this question clearly in under 150 words and include one example at the
end.\nQUESTION: {{question}}"
~~~

~~~
---

## FILE: backend/requirements.txt
```txt
fastapi==0.111.0
uvicorn==0.30.1
sqlalchemy==2.0.30
pydantic==2.7.1
python-multipart==0.0.9
python-jose==3.3.0
passlib[bcrypt]==1.7.4
requests==2.32.3
pdfminer.six==20231228
~~~

## FILE: backend/db.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base
import os

DB_URL = os.getenv("DB_URL", "sqlite:///studybuddy.db")
engine = create_engine(DB_URL, connect_args={"check_same_thread": False} if
DB_URL.startswith("sqlite") else {})
SessionLocal = sessionmaker(bind=engine, autoflush=False, autocommit=False)
Base = declarative_base()

def get_db():
    from contextlib import contextmanager
    @contextmanager
    def _session_scope():
        db = SessionLocal()
        try:
            yield db
            db.commit()
        except Exception:
            db.rollback()
            raise
        finally:
            db.close()
    return _session_scope()
```

---

## FILE: backend/models.py

```
from sqlalchemy import Column, Integer, String, Text, ForeignKey, DateTime,
Boolean
from sqlalchemy.orm import relationship
from datetime import datetime
from db import Base

class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True)
    email = Column(String, unique=True, index=True)
    name = Column(String)
    password_hash = Column(String)
    plan = Column(String, default="free")
```

```
class Room(Base):
    __tablename__ = "rooms"
    id = Column(Integer, primary_key=True)
    owner_id = Column(Integer, ForeignKey("users.id"))
    title = Column(String)
    meet_link = Column(String)  # Google Meet or Zoom URL
    invite_code = Column(String, index=True)
    owner = relationship("User")

class Task(Base):
    __tablename__ = "tasks"
    id = Column(Integer, primary_key=True)
    room_id = Column(Integer, ForeignKey("rooms.id"))
    title = Column(String)
    column = Column(String, default="todo")  # todo, doing, done

class Message(Base):
    __tablename__ = "messages"
    id = Column(Integer, primary_key=True)
    room_id = Column(Integer, ForeignKey("rooms.id"))
    user_id = Column(Integer, ForeignKey("users.id"))
    content = Column(Text)
    created_at = Column(DateTime, default=datetime.utcnow)

class FileAsset(Base):
    __tablename__ = "files"
    id = Column(Integer, primary_key=True)
    room_id = Column(Integer, ForeignKey("rooms.id"))
    name = Column(String)
    path = Column(String)
    kind = Column(String)  # pdf, image, etc.
```

## FILE: backend/routers/auth.py

```
from fastapi import APIRouter, Depends, HTTPException
from pydantic import BaseModel
from sqlalchemy.orm import Session
from db import get_db
from models import User
from passlib.hash import bcrypt

router = APIRouter(prefix="/auth", tags=["auth"])

class RegisterIn(BaseModel):
    email: str
    name: str
    password: str

@router.post('/register')
def register(payload: RegisterIn):
    with get_db() as db:
        if db.query(User).filter(User.email==payload.email).first():
            raise HTTPException(400, "Email already exists")
        u = User(email=payload.email, name=payload.name,
password_hash=bcrypt.hash(payload.password))
        db.add(u)
        db.flush()
        return {"id": u.id, "email": u.email, "name": u.name}
```

---

## FILE: backend/routers/rooms.py

```
import secrets
from fastapi import APIRouter, UploadFile, File, HTTPException
from pydantic import BaseModel
from db import get_db
from models import Room, Task, Message, FileAsset
from sqlalchemy.orm import Session
import os

router = APIRouter(prefix="/rooms", tags=["rooms"])
UPLOAD_DIR = os.getenv("UPLOAD_DIR", "uploads")
os.makedirs(UPLOAD_DIR, exist_ok=True)

class RoomIn(BaseModel):
    title: str
    meet_link: str | None = None

@router.post('/')
def create_room(data: RoomIn):
    with get_db() as db:
        r = Room(title=data.title, meet_link=data.meet_link or "",
invite_code=secrets.token_hex(4))
        db.add(r); db.flush()
        return {"id": r.id, "title": r.title, "invite": r.invite_code,
"meet_link": r.meet_link}

@router.get('/invite/{code}')
def get_room_by_code(code: str):
    with get_db() as db:
        r = db.query(Room).filter(Room.invite_code==code).first()
        if not r:
            raise HTTPException(404, "Room not found")
        return {"id": r.id, "title": r.title, "meet_link": r.meet_link}

class TaskIn(BaseModel):
    title: str
    column: str = "todo"

@router.post('/{room_id}/tasks')
def add_task(room_id: int, t: TaskIn):
    with get_db() as db:
        task = Task(room_id=room_id, title=t.title, column=t.column)
        db.add(task); db.flush()
        return {"id": task.id, "title": task.title, "column": task.column}

@router.get('/{room_id}/tasks')
def list_tasks(room_id: int):
    with get_db() as db:
        q = db.query(Task).filter(Task.room_id==room_id).all()
        return [{"id": i.id, "title": i.title, "column": i.column} for i in q]

@router.post('/{room_id}/upload')
async def upload_pdf(room_id: int, f: UploadFile = File(...)):
    if not f.filename.lower().endswith('.pdf'):
        raise HTTPException(400, "Only PDF supported")
    path = os.path.join(UPLOAD_DIR, f"room{room_id}_{f.filename}")
    with open(path, 'wb') as out:
        out.write(await f.read())
    with get_db() as db:
        fa = FileAsset(room_id=room_id, name=f.filename, path=path, kind='pdf')
        db.add(fa)
    return {"status":"ok","name":f.filename,"path":path}
```

## FILE: backend/routers/ai.py

```
from fastapi import APIRouter, HTTPException
from pydantic import BaseModel
import os, requests, json

router = APIRouter(prefix="/ai", tags=["ai"])
HF_API = os.getenv("HF_API")
HF_MODEL = os.getenv("HF_MODEL", "google/flan-t5-small")
```

```python
HF_URL = f"https://api-inference.huggingface.co/models/{HF_MODEL}"

class NotesIn(BaseModel):
    notes: str

@router.post('/summarize')
def summarize(data: NotesIn):
    if not data.notes.strip():
        return {"bullets":[], "flashcards":[]}
    if not HF_API:
        # fallback mock
        sents = [s.strip() for s in data.notes.split('.') if s.strip()][:5]
        fc = [{"question": f"Key point {i+1}?", "answer": s} for i, s in
enumerate(sents)]
        return {"bullets": sents, "flashcards": fc}
    prompt = (
        "Summarize these notes into 5 bullets and 5 flashcards (Q/A). Return
JSON with keys 'bullets' and 'flashcards'.\n\n" + data.notes
    )
    r = requests.post(HF_URL, headers={"Authorization": f"Bearer {HF_API}"},
json={"inputs": prompt}, timeout=25)
    try:
        resp = r.json()
        text = resp[0].get('generated_text', '') if isinstance(resp, list) else
json.dumps(resp)
        return json.loads(text)
    except Exception:
        raise HTTPException(502, "AI provider error")

class QuestionIn(BaseModel):
    question: str

@router.post('/qa')
def qa(data: QuestionIn):
    if not HF_API:
        return {"answer": "This is a mock answer. Add HF_API for real. Example:
Break the concept into definitions + example."}
    prompt = f"Answer in under 150 words with one example.\nQUESTION:
{data.question}"
    r = requests.post(HF_URL, headers={"Authorization": f"Bearer {HF_API}"},
json={"inputs": prompt}, timeout=20)
    try:
        resp = r.json()
        text = resp[0].get('generated_text', '') if isinstance(resp, list) else
str(resp)
        return {"answer": text}
    except Exception:
        return {"answer": "AI response unavailable."}
```

## FILE: backend/routers/files.py

```python
from fastapi import APIRouter, HTTPException
from pydantic import BaseModel
from pdfminer.high_level import extract_text
import os

router = APIRouter(prefix="/files", tags=["files"])

class ReadIn(BaseModel):
    path: str

@router.post('/read')
def read_pdf(data: ReadIn):
    if not os.path.exists(data.path):
        raise HTTPException(404, "File not found")
    try:
        text = extract_text(data.path)
        return {"text": text[:5000]}  # cap for demo
    except Exception:
        raise HTTPException(400, "Unable to parse PDF")
```

## FILE: backend/routers/payments.py

```python
from fastapi import APIRouter
from pydantic import BaseModel

router = APIRouter(prefix="/payments", tags=["payments"])

PRICE_KES = 650

class CheckoutIn(BaseModel):
    user_id: int

@router.post('/checkout')
def checkout(data: CheckoutIn):
    # Stub response. Replace with real IntaSend REST call.
    return {
        "amount_kes": PRICE_KES,
        "provider": "IntaSend",
        "status": "initialized",
        "payment_link": "https://pay.intasend.com/checkout/example"
    }
```

## FILE: backend/app.py

```python
from fastapi import FastAPI
from db import Base, engine
import models
from routers import auth, rooms, ai, files, payments

app = FastAPI(title="StudyBuddy API")

# Create tables
Base.metadata.create_all(bind=engine)

# Routers
app.include_router(auth.router)
app.include_router(rooms.router)
app.include_router(ai.router)
app.include_router(files.router)
app.include_router(payments.router)

@app.get('/')
def root():
    return {"status": "ok", "service": "StudyBuddy"}
```

## FILE: frontend/index.html

```html
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>StudyBuddy — Project Rooms</title>
  <link rel="stylesheet" href="styles.css" />
</head>
<body>
  <header class="topbar">
    <div class="logo">🤖👆🏽 StudyBuddy</div>
    <div class="actions">
      <button id="newRoomBtn">New Room</button>
      <button id="checkoutBtn">Upgrade (KES 650/yr)</button>
    </div>
  </header>

  <main class="grid">
    <aside class="sidebar">
      <h3>Rooms</h3>
      <ul id="roomList"></ul>
      <div class="invite">
        <input id="inviteCode" placeholder="Invite code" />
        <button id="joinBtn">Join</button>
      </div>
    </aside>

    <section class="content">
      <div class="roomHeader">
        <input id="roomTitle" placeholder="Room title" />
        <input id="meetLink" placeholder="Google Meet / Zoom link" />
        <button id="saveRoom">Save</button>
      </div>

      <div class="kanban">
        <div class="col" data-col="todo">
          <h4>To Do</h4>
          <ul id="todo"></ul>
          <div class="addTask">
            <input id="newTaskTitle" placeholder="New task" />
            <button id="addTaskBtn">Add</button>
          </div>
        </div>
        <div class="col" data-col="doing">
          <h4>Doing</h4>
          <ul id="doing"></ul>
        </div>
        <div class="col" data-col="done">
          <h4>Done</h4>
          <ul id="done"></ul>
        </div>
      </div>

      <div class="panels">
        <section class="panel">
          <h4>AI Tutor</h4>
          <textarea id="notes" placeholder="Paste notes to summarize & make
flashcards"></textarea>
          <button id="summarizeBtn">Summarize + Flashcards</button>
          <textarea id="question" placeholder="Ask a question"></textarea>
          <button id="qaBtn">Ask AI</button>
          <pre id="aiOut"></pre>
        </section>

        <section class="panel">
          <h4>Whiteboard</h4>
          <canvas id="board" width="640" height="360"></canvas>
          <div>
            <button id="pen">Pen</button>
            <button id="eraser">Eraser</button>
            <button id="clear">Clear</button>
            <button id="exportPNG">Export PNG</button>
          </div>
        </section>

        <section class="panel">
          <h4>PDF → Flipbook</h4>
          <input type="file" id="pdfInput" accept="application/pdf" />
          <button id="uploadPdf">Upload</button>
          <div id="flipbook" class="flipbook"></div>
        </section>
      </div>
    </section>
  </main>

  <script src="app.js"></script>
  <script src="whiteboard.js"></script>
  <script src="flipbook.js"></script>
</body>
</html>
```

---

## FILE: frontend/styles.css

```
*{box-sizing:border-box} body{margin:0;font-family:Inter,Arial,sans-
serif;background:#0f1115;color:#eaeef6}
.topbar{display:flex;justify-content:space-between;align-items:center;padding:
12px 16px;background:#121826;border-bottom:1px solid #1f2636}
.logo{font-weight:800}
.actions button{margin-left:8px}
.grid{display:grid;grid-template-columns:260px 1fr;gap:12px;padding:12px}
.sidebar{background:#121826;border:1px solid #1f2636;border-radius:12px;padding:
12px}
.content{display:flex;flex-direction:column;gap:12px}
.roomHeader{display:flex;gap:8px}
.roomHeader input{flex:1}
```

```
.kanban{display:grid;grid-template-columns:repeat(3,1fr);gap:12px}
.col{background:#121826;border:1px solid #1f2636;border-radius:12px;padding:
12px}
.col h4{margin-top:0}
.addTask{display:flex;gap:8px;margin-top:8px}
.panels{display:grid;grid-template-columns:repeat(3,1fr);gap:12px}
.panel{background:#121826;border:1px solid #1f2636;border-radius:12px;padding:
12px}
textarea,input,button{background:#0b1320;color:#eaeef6;border:1px solid
#253150;border-radius:8px;padding:8px}
button{cursor:pointer}
.flipbook{height:360px;overflow:auto;border:1px dashed #253150;border-radius:
8px;padding:8px}
.page{background:#0b1320;margin:8px 0;padding:12px;border-radius:6px}
ul{list-style:none;margin:0;padding:0} li{margin:6px 0;padding:
8px;background:#0b1320;border:1px solid #253150;border-radius:6px}
```

---

## FILE: frontend/app.js

```
const API = "http://localhost:8000";
let currentRoom = null;

// Rooms sidebar
async function createRoom(){
  const title = prompt("Room title?") || "My Study Room";
  const meet_link = document.getElementById('meetLink').value || '';
  const r = await fetch(`${API}/rooms/`, {method:'POST', headers:{'Content-
Type':'application/json'}, body: JSON.stringify({title,
meet_link})}).then(r=>r.json());
  addRoomToList(r);
  selectRoom(r.id, r.title, r.meet_link);
  alert(`Invite code: ${r.invite}`);
}

function addRoomToList(r){
  const li = document.createElement('li');
  li.textContent = r.title || `Room #${r.id}`;
  li.onclick = ()=> selectRoom(r.id, r.title, r.meet_link);
  document.getElementById('roomList').appendChild(li);
}

function selectRoom(id, title, meet){
  currentRoom = id;
  document.getElementById('roomTitle').value = title || '';
  document.getElementById('meetLink').value = meet || '';
  loadTasks();
}

// Tasks
async function addTask(){
  if(!currentRoom) return alert('Create or select a room first');
  const title = document.getElementById('newTaskTitle').value.trim();
  if(!title) return;
  const t = await fetch(`${API}/rooms/${currentRoom}/tasks`, {method:'POST',
headers:{'Content-Type':'application/json'}, body:
JSON.stringify({title})}).then(r=>r.json());
  renderTask(t);
  document.getElementById('newTaskTitle').value='';
}

async function loadTasks(){
  const list = await fetch(`${API}/rooms/${currentRoom}/
tasks`).then(r=>r.json());
  ['todo','doing','done'].forEach(c=> document.getElementById(c).innerHTML='');
  list.forEach(renderTask);
}

function renderTask(t){
  const li = document.createElement('li');
  li.textContent = t.title;
  document.getElementById(t.column).appendChild(li);
}

// AI Tutor
async function summarize(){
  const notes = document.getElementById('notes').value;
  const data = await fetch(`${API}/ai/summarize`, {method:'POST', headers:
{'Content-Type':'application/json'}, body:
JSON.stringify({notes})}).then(r=>r.json());
  document.getElementById('aiOut').textContent = JSON.stringify(data, null, 2);
}

async function ask(){
  const question = document.getElementById('question').value;
  const data = await fetch(`${API}/ai/qa`, {method:'POST', headers:{'Content-
Type':'application/json'}, body: JSON.stringify({question})}).then(r=>r.json());
  document.getElementById('aiOut').textContent = data.answer || 'No answer';
}

// PDF upload
async function uploadPdf(){
  if(!currentRoom) return alert('Select a room');
  const f = document.getElementById('pdfInput').files[0];
  if(!f) return alert('Pick a PDF');
  const fd = new FormData(); fd.append('f', f);
  const res = await fetch(`${API}/rooms/${currentRoom}/upload`, {method:'POST',
body: fd}).then(r=>r.json());
  // request text
  const txt = await fetch(`${API}/files/read`, {method:'POST', headers:
{'Content-Type':'application/json'}, body: JSON.stringify({path:
res.path})}).then(r=>r.json());
  // build flipbook pages (mock pagination)
  buildFlipbook(txt.text || '');
}

function buildFlipbook(text){
  const fb = document.getElementById('flipbook');
  fb.innerHTML = '';
  const pages = text.match(/.{1,800}/gs) || [];
  pages.slice(0,10).forEach((chunk, i)=>{
    const div = document.createElement('div');
    div.className='page';
    div.innerHTML = `<h5>Page ${i+1}</h5><p>${chunk.replace(/\n/g,'<br/>')}</
p>`;
    fb.appendChild(div);
  });
}

// Payments
async function checkout(){
  const resp = await fetch(`${API}/payments/checkout`, {method:'POST', headers:
{'Content-Type':'application/json'}, body: JSON.stringify({user_id:
1})}).then(r=>r.json());
  alert(`Proceed to payment: ${resp.payment_link}`);
  window.open(resp.payment_link, '_blank');
}

// Invite join
async function joinByCode(){
  const code = document.getElementById('inviteCode').value.trim();
  if(!code) return;
  const r = await fetch(`${API}/rooms/invite/${code}`).then(r=>r.json());
  addRoomToList(r); selectRoom(r.id, r.title, r.meet_link);
}

// Events
document.getElementById('newRoomBtn').onclick = createRoom;
document.getElementById('addTaskBtn').onclick = addTask;
document.getElementById('summarizeBtn').onclick = summarize;
document.getElementById('qaBtn').onclick = ask;
document.getElementById('uploadPdf').onclick = uploadPdf;
document.getElementById('checkoutBtn').onclick = checkout;
document.getElementById('joinBtn').onclick = joinByCode;
```

---

## FILE: frontend/whiteboard.js

```
const canvas = document.getElementById('board');
const ctx = canvas.getContext('2d');
let drawing = false, erasing = false, last = null;

canvas.addEventListener('mousedown', e=>{ drawing = true; last = [e.offsetX,
e.offsetY]; });
canvas.addEventListener('mouseup', ()=> drawing=false);
canvas.addEventListener('mouseleave', ()=> drawing=false);
canvas.addEventListener('mousemove', e=>{
  if(!drawing) return;
  ctx.lineWidth = erasing ? 16 : 3;
  ctx.strokeStyle = erasing ? '#0b1320' : '#eaeef6';
  ctx.lineCap='round';
  ctx.beginPath();
  ctx.moveTo(last[0], last[1]);
  ctx.lineTo(e.offsetX, e.offsetY);
  ctx.stroke();
  last = [e.offsetX, e.offsetY];
});

document.getElementById('pen').onclick = ()=> erasing=false;
document.getElementById('eraser').onclick = ()=> erasing=true;
document.getElementById('clear').onclick = ()=> ctx.clearRect(0,
0,canvas.width,canvas.height);
document.getElementById('exportPNG').onclick = ()=>{
  const url = canvas.toDataURL('image/png');
  const a = document.createElement('a'); a.href=url;
a.download='whiteboard.png'; a.click();
};
```

---

## FILE: frontend/flipbook.js

```
// Basic CSS flipbook substitute: vertical pages with next/prev via scroll
// Already built in app.js buildFlipbook(). This file can hold future
enhancements.
```

---

## How to demo

1. Start backend: `uvicorn app:app --reload` (in /backend)
2. Open `frontend/index.html` in a local server
3. Create a room, copy the **invite code**, paste it into the Join box
4. Add tasks to the Kanban
5. Paste notes → Summarize + Flashcards, ask a question
6. Draw on the whiteboard, export PNG
7. Upload a PDF; it becomes a flipbook-like paged viewer
8. Click Upgrade to open the IntaSend stub checkout link

---

## Notes on production

- Replace SQLite with PostgreSQL via `DB_URL`
- Secure auth (JWT) and CORS
- Replace AI and IntaSend stubs with real keys and error handling
- Use PDF.js canvas rendering for true flipbook visuals or embed FlipHTML5 viewer if you have an account/API ```