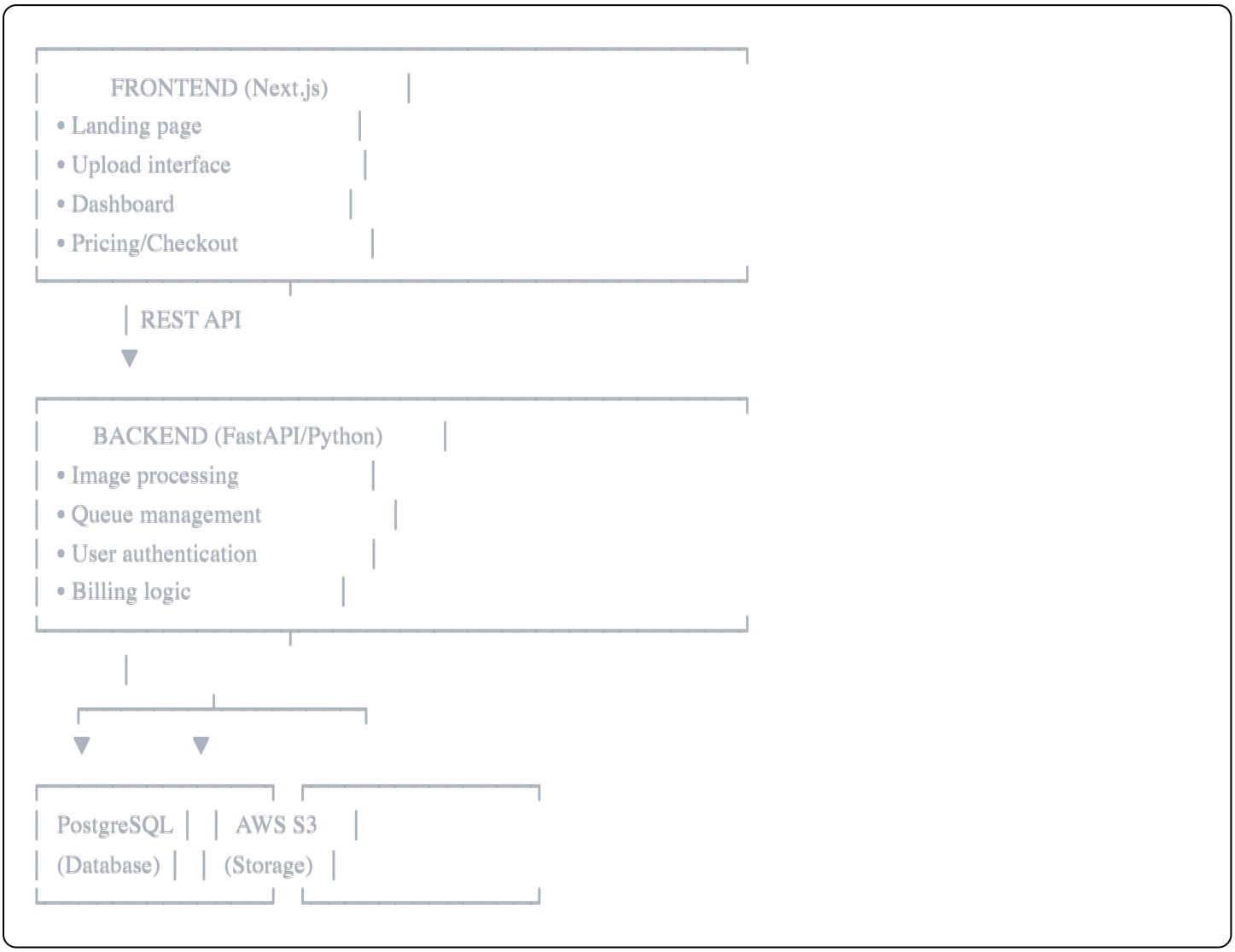
CENTYR - TECH STACK WEB MVP

🏗️ ARCHITETTURA COMPLETA



💻 FRONTEND STACK

Framework: Next.js 14 (App Router)

Perché Next.js:

- ✅ Server-Side Rendering (SEO)
- ✅ API routes built-in
- ✅ Image optimization
- ✅ TypeScript support
- ✅ Vercel deployment (free tier)
- ✅ Fast refresh (dev experience)

Setup Iniziale

bash

```
# Create Next.js app
npx create-next-app@latest centyr-web
# Durante setup:
# ✅ TypeScript: Yes
# ✅ ESLint: Yes
# ✅ Tailwind CSS: Yes
# ✅ App Router: Yes
# ✅ Import alias: Yes (@/*)

cd centyr-web

# Install dependencies
npm install \
  @radix-ui/react-dialog \
  @radix-ui/react-dropdown-menu \
  @radix-ui/react-select \
  @radix-ui/react-toast \
  lucide-react \
  clsx \
  tailwind-merge \
  next-themes \
  react-dropzone \
  zustand \
  axios \
  stripe \
  @stripe/stripe-js \
  react-hot-toast

# Dev dependencies
npm install -D \
  @types/node \
  @types/react \
  prettier \
  prettier-plugin-tailwindcss
```

Project Structure

```
centyr-web/
├── app/
│   ├── (marketing)/
│   │   ├── page.tsx              # Landing page
│   │   ├── pricing/page.tsx      # Pricing page
```

```
│   │   └── about/page.tsx        # About page
│   ├── (auth)/
│   │   ├── login/page.tsx
│   │   └── signup/page.tsx
│   ├── (dashboard)/
│   │   ├── dashboard/page.tsx    # Main dashboard
│   │   ├── upload/page.tsx       # Upload interface
│   │   ├── history/page.tsx      # Processing history
│   │   └── settings/page.tsx     # User settings
│   ├── api/
│   │   ├── auth/[...nextauth]/route.ts
│   │   ├── upload/route.ts
│   │   ├── process/route.ts
│   │   └── webhooks/stripe/route.ts
│   ├── layout.tsx                # Root layout
│   └── globals.css               # Global styles
├── components/
│   ├── ui/                       # Shadcn components
│   ├── marketing/
│   │   ├── Hero.tsx
│   │   ├── Features.tsx
│   │   ├── Pricing.tsx
│   │   ├── Testimonials.tsx
│   │   └── FAQ.tsx
│   ├── dashboard/
│   │   ├── UploadZone.tsx
│   │   ├── ImagePreview.tsx
│   │   ├── ProcessingQueue.tsx
│   │   └── StatsCards.tsx
│   └── shared/
│       ├── Header.tsx
│       ├── Footer.tsx
│       └── Navbar.tsx
├── lib/
│   ├── api.ts                    # API client
│   ├── auth.ts                   # Auth helpers
│   ├── stripe.ts                 # Stripe client
│   └── utils.ts                  # Utilities
├── hooks/
│   ├── useUpload.ts
│   ├── useAuth.ts
│   └── useSubscription.ts
├── types/
│   └── index.ts                  # TypeScript types
├── public/
```

```
│   └── images/
└── .env.local              # Environment variables
```

Key Components

1. Upload Zone Component

typescript

```tsx
// components/dashboard/UploadZone.tsx
'use client';

import { useCallback, useState } from 'react';
import { useDropzone } from 'react-dropzone';
import { Upload, X } from 'lucide-react';
import { toast } from 'react-hot-toast';

interface UploadZoneProps {
  onUpload: (files: File[]) => void;
  maxFiles?: number;
}

export function UploadZone({ onUpload, maxFiles = 10 }: UploadZoneProps) {
  const [files, setFiles] = useState<File[]>([]);

  const onDrop = useCallback((acceptedFiles: File[]) => {
    if (acceptedFiles.length > maxFiles) {
      toast.error(`Maximum ${maxFiles} files allowed`);
      return;
    }

    setFiles(acceptedFiles);
    onUpload(acceptedFiles);
  }, [maxFiles, onUpload]);

  const { getRootProps, getInputProps, isDragActive } = useDropzone({
    onDrop,
    accept: {
      'image/jpeg': ['.jpg', '.jpeg'],
      'image/png': ['.png'],
      'image/webp': ['.webp']
    },
    maxSize: 50 * 1024 * 1024, // 50MB
  });

  return (
    <div
      {...getRootProps()}
      className={`
        border-2 border-dashed rounded-lg p-12
        transition-colors cursor-pointer
        ${isDragActive
          ? 'border-blue-500 bg-blue-50'
          : 'border-gray-300 hover:border-gray-400'
        }
```

```
    `}
  >
    <input {...getInputProps()} />
    <div className="flex flex-col items-center justify-center text-center">
      <Upload className="w-12 h-12 text-gray-400 mb-4" />
      <p className="text-lg font-semibold mb-2">
        {isDragActive ? 'Drop files here' : 'Drag & drop product images'}
      </p>
      <p className="text-sm text-gray-500">
        or click to browse (max {maxFiles} files, 50MB each)
      </p>
      <p className="text-xs text-gray-400 mt-2">
        Supports: JPG, PNG, WebP
      </p>
    </div>

    {files.length > 0 && (
      <div className="mt-6 grid grid-cols-3 gap-4">
        {files.map((file, i) => (
          <div key={i} className="relative">
            <img
              src={URL.createObjectURL(file)}
              alt={file.name}
              className="w-full h-32 object-cover rounded"
            />
            <button
              onClick={(e) => {
                e.stopPropagation();
                setFiles(files.filter((_, idx) => idx !== i));
              }}
              className="absolute top-2 right-2 p-1 bg-red-500 rounded-full"
            >
              <X className="w-4 h-4 text-white" />
            </button>
          </div>
        ))}
      </div>
    )}
  </div>
 );
}
```

2. API Client

typescript

```typescript
// lib/api.ts
import axios from 'axios';

const API_BASE_URL = process.env.NEXT_PUBLIC_API_URL || 'http://localhost:8000';

export const api = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});

// Add auth token to requests
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('auth_token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Image processing API
export async function processImages(files: File[]) {
  const formData = new FormData();
  files.forEach((file) => {
    formData.append('images', file);
  });

  const response = await api.post('/process', formData, {
    headers: {
      'Content-Type': 'multipart/form-data',
    },
  });

  return response.data;
}

// Get processing status
export async function getProcessingStatus(jobId: string) {
  const response = await api.get(`/status/${jobId}`);
  return response.data;
}

// Download processed images
export async function downloadProcessedImages(jobId: string) {
  const response = await api.get(`/download/${jobId}`, {
```

```
    responseType: 'blob',
  });

  // Create download link
  const url = window.URL.createObjectURL(new Blob([response.data]));
  const link = document.createElement('a');
  link.href = url;
  link.setAttribute('download', `centyr-processed-${jobId}.zip`);
  document.body.appendChild(link);
  link.click();
  link.remove();
}
```

Styling: Tailwind CSS

javascript

```
// tailwind.config.js
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    './app/**/*.{js,ts,jsx,tsx,mdx}',
    './components/**/*.{js,ts,jsx,tsx,mdx}',
  ],
  theme: {
    extend: {
      colors: {
        primary: {
          50: '#eff6ff',
          500: '#2563eb',
          600: '#1d4ed8',
          700: '#1e40af',
        },
        success: {
          500: '#10b981',
        },
        warning: {
          500: '#f59e0b',
        },
      },
      animation: {
        'fade-in': 'fadeIn 0.5s ease-in-out',
        'slide-up': 'slideUp 0.3s ease-out',
      },
      keyframes: {
        fadeIn: {
          '0%': { opacity: '0' },
          '100%': { opacity: '1' },
        },
        slideUp: {
          '0%': { transform: 'translateY(10px)', opacity: '0' },
          '100%': { transform: 'translateY(0)', opacity: '1' },
        },
      },
    },
  },
  plugins: [
    require('@tailwindcss/forms'),
    require('@tailwindcss/typography'),
  ],
}
```

🐍 BACKEND STACK

Framework: FastAPI (Python)

Perché FastAPI:

- ✅ Async support (fast processing)
- ✅ Auto API documentation (Swagger)
- ✅ Type hints (fewer bugs)
- ✅ Easy integration con tuo script
- ✅ WebSocket support (real-time updates)

Setup Iniziale

bash

```
# Create project
mkdir centyr-api
cd centyr-api

# Create virtual environment
python -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate

# Install dependencies
pip install \
  fastapi \
  uvicorn[standard] \
  python-multipart \
  pillow \
  opencv-python \
  numpy \
  python-jose[cryptography] \
  passlib[bcrypt] \
  python-dotenv \
  sqlalchemy \
  psycopg2-binary \
  alembic \
  boto3 \
  stripe \
  celery \
  redis

# Create requirements.txt
pip freeze > requirements.txt
```

Project Structure

```
centyr-api/
├── app/
│   ├── main.py              # FastAPI app
│   ├── config.py            # Configuration
│   ├── database.py          # DB connection
│   ├── models/
│   │   ├── user.py
│   │   ├── image.py
│   │   └── subscription.py
│   ├── schemas/
│   │   ├── user.py
│   │   ├── image.py
│   │   └── auth.py
```

```
│   ├── routers/
│   │   ├── auth.py
│   │   ├── images.py
│   │   ├── users.py
│   │   └── webhooks.py
│   ├── services/
│   │   ├── image_processor.py   # Your alignment script!
│   │   ├── storage.py           # S3 operations
│   │   ├── auth.py              # JWT handling
│   │   └── stripe_service.py    # Billing
│   ├── utils/
│   │   ├── security.py
│   │   └── helpers.py
│   └── tasks/
│       └── process_images.py    # Celery tasks
├── alembic/                     # DB migrations
├── tests/
├── .env
└── requirements.txt
```

Core API Code

main.py

python

```python
# app/main.py
from fastapi import FastAPI, File, UploadFile, Depends, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from typing import List
import uvicorn

from app.routers import auth, images, users, webhooks
from app.config import settings

app = FastAPI(
    title="Centyr API",
    description="AI-powered product image alignment",
    version="1.0.0"
)

# CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:3000", "https://centyr.com"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Include routers
app.include_router(auth.router, prefix="/auth", tags=["auth"])
app.include_router(images.router, prefix="/images", tags=["images"])
app.include_router(users.router, prefix="/users", tags=["users"])
app.include_router(webhooks.router, prefix="/webhooks", tags=["webhooks"])

@app.get("/")
def read_root():
    return {"message": "Centyr API v1.0.0", "status": "operational"}

@app.get("/health")
def health_check():
    return {"status": "healthy"}

if __name__ == "__main__":
    uvicorn.run("app.main:app", host="0.0.0.0", port=8000, reload=True)
```

Image Processing Router

```python
```

```python
# app/routers/images.py
from fastapi import APIRouter, File, UploadFile, Depends, BackgroundTasks
from typing import List
from app.services.image_processor import align_images
from app.services.storage import upload_to_s3, generate_download_url
from app.models.user import User
from app.routers.auth import get_current_user
import uuid

router = APIRouter()

@router.post("/process")
async def process_images(
    background_tasks: BackgroundTasks,
    files: List[UploadFile] = File(...),
    current_user: User = Depends(get_current_user)
):
    """
    Process uploaded images with AI alignment
    """
    # Check subscription limits
    if len(files) > current_user.monthly_limit:
        raise HTTPException(400, "Monthly limit exceeded")

    # Generate job ID
    job_id = str(uuid.uuid4())

    # Save original images to S3
    original_urls = []
    for file in files:
        contents = await file.read()
        url = await upload_to_s3(contents, f"originals/{job_id}/{file.filename}")
        original_urls.append(url)

    # Queue processing job
    background_tasks.add_task(
        align_images,
        job_id=job_id,
        image_urls=original_urls,
        user_id=current_user.id
    )

    return {
        "job_id": job_id,
        "status": "processing",
        "images_count": len(files),
```

```
        "estimated_time": len(files) * 3  # 3 seconds per image
    }

@router.get("/status/{job_id}")
async def get_status(
    job_id: str,
    current_user: User = Depends(get_current_user)
):
    """
    Get processing status
    """
    # Check job status from database/Redis
    status = await get_job_status(job_id)

    return {
        "job_id": job_id,
        "status": status.get("status"),  # "processing", "completed", "failed"
        "progress": status.get("progress"),  # 0-100
        "processed_count": status.get("processed", 0),
        "total_count": status.get("total", 0),
    }

@router.get("/download/{job_id}")
async def download_images(
    job_id: str,
    current_user: User = Depends(get_current_user)
):
    """
    Download processed images as ZIP
    """
    # Generate signed download URL
    download_url = await generate_download_url(job_id)

    return {
        "download_url": download_url,
        "expires_in": 3600  # 1 hour
    }
```

Image Processor Service (YOUR SCRIPT!)

python

```python
# app/services/image_processor.py
import cv2
import numpy as np
from PIL import Image
import io

# Import your alignment script functions
from your_script import rimuovi_ombre_e_trova_prodotto, crea_immagine_allineata

async def align_images(job_id: str, image_urls: List[str], user_id: int):
    """
    Process images with alignment algorithm
    """
    processed_urls = []

    for i, url in enumerate(image_urls):
        try:
            # Download image
            image_data = await download_from_s3(url)
            img = Image.open(io.BytesIO(image_data))

            # YOUR ALGORITHM HERE!
            processed_img, info = crea_immagine_allineata(
                img,
                larghezza=1000,
                altezza=1000,
                margine_sup=50,
                margine_inf=50,
                margine_lat=50
            )

            # Save processed image
            buffer = io.BytesIO()
            processed_img.save(buffer, format='PNG', quality=95)
            buffer.seek(0)

            # Upload to S3
            processed_url = await upload_to_s3(
                buffer.getvalue(),
                f"processed/{job_id}/{i}.png"
            )
            processed_urls.append(processed_url)

            # Update progress
            await update_job_progress(job_id, (i + 1) / len(image_urls) * 100)
```

```
        except Exception as e:
            # Log error
            print(f"Error processing image {i}: {e}")
            continue

    # Mark job as complete
    await mark_job_complete(job_id, processed_urls)

    # Send email notification
    await send_completion_email(user_id, job_id)

    return processed_urls
```

🗄️ DATABASE

PostgreSQL con SQLAlchemy

Models

python

```python
# 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, unique=True, index=True)
    hashed_password = Column(String)
    full_name = Column(String, nullable=True)
    is_active = Column(Boolean, default=True)
    is_verified = Column(Boolean, default=False)

    # Subscription
    plan = Column(String, default="free")  # free, basic, growth, scale
    monthly_limit = Column(Integer, default=10)
    images_used_this_month = Column(Integer, default=0)
    stripe_customer_id = Column(String, nullable=True)
    stripe_subscription_id = Column(String, nullable=True)

    created_at = Column(DateTime, default=datetime.datetime.utcnow)
    updated_at = Column(DateTime, onupdate=datetime.datetime.utcnow)
```

python

```python
# app/models/image.py
from sqlalchemy import Column, Integer, String, DateTime, ForeignKey, Enum
from app.database import Base
import datetime
import enum

class JobStatus(str, enum.Enum):
    PENDING = "pending"
    PROCESSING = "processing"
    COMPLETED = "completed"
    FAILED = "failed"

class ProcessingJob(Base):
    __tablename__ = "processing_jobs"

    id = Column(Integer, primary_key=True)
    job_id = Column(String, unique=True, index=True)
    user_id = Column(Integer, ForeignKey("users.id"))

    status = Column(Enum(JobStatus), default=JobStatus.PENDING)
    total_images = Column(Integer)
    processed_images = Column(Integer, default=0)
    progress = Column(Integer, default=0)  # 0-100

    original_urls = Column(String)  # JSON string
    processed_urls = Column(String, nullable=True)  # JSON string

    created_at = Column(DateTime, default=datetime.datetime.utcnow)
    completed_at = Column(DateTime, nullable=True)
```

Migrations

```bash
# Initialize Alembic
alembic init alembic

# Create first migration
alembic revision --autogenerate -m "Initial tables"

# Run migrations
alembic upgrade head
```

☁️ STORAGE: AWS S3

S3 Service

```python
```

```python
# app/services/storage.py
import boto3
from app.config import settings
from botocore.exceptions import ClientError

s3_client = boto3.client(
    's3',
    aws_access_key_id=settings.AWS_ACCESS_KEY,
    aws_secret_access_key=settings.AWS_SECRET_KEY,
    region_name=settings.AWS_REGION
)

async def upload_to_s3(file_data: bytes, key: str) -> str:
    """
    Upload file to S3 and return URL
    """
    try:
        s3_client.put_object(
            Bucket=settings.S3_BUCKET,
            Key=key,
            Body=file_data,
            ContentType='image/png'
        )

        url = f"https://{settings.S3_BUCKET}.s3.{settings.AWS_REGION}.amazonaws.com/{key}"
        return url

    except ClientError as e:
        print(f"Error uploading to S3: {e}")
        raise

async def generate_download_url(job_id: str, expires_in: int = 3600) -> str:
    """
    Generate pre-signed URL for downloading processed images
    """
    key = f"processed/{job_id}.zip"

    url = s3_client.generate_presigned_url(
        'get_object',
        Params={'Bucket': settings.S3_BUCKET, 'Key': key},
        ExpiresIn=expires_in
    )

    return url
```

PAYMENTS: Stripe

Stripe Service

python

```python
# app/services/stripe_service.py
import stripe
from app.config import settings

stripe.api_key = settings.STRIPE_SECRET_KEY

PLANS = {
    "basic": {"price_id": "price_xxx", "limit": 100},
    "growth": {"price_id": "price_yyy", "limit": 500},
    "scale": {"price_id": "price_zzz", "limit": 999999},
}

async def create_checkout_session(user_id: int, plan: str, success_url: str, cancel_url: str):
    """
    Create Stripe checkout session
    """
    session = stripe.checkout.Session.create(
        customer_email=user.email,
        payment_method_types=['card'],
        line_items=[{
            'price': PLANS[plan]["price_id"],
            'quantity': 1,
        }],
        mode='subscription',
        success_url=success_url,
        cancel_url=cancel_url,
        metadata={
            'user_id': user_id,
            'plan': plan
        }
    )

    return session.url

async def handle_webhook(payload, sig_header):
    """
    Handle Stripe webhooks
    """
    event = stripe.Webhook.construct_event(
        payload, sig_header, settings.STRIPE_WEBHOOK_SECRET
    )

    if event.type == 'checkout.session.completed':
        session = event.data.object
        user_id = session.metadata.user_id
        plan = session.metadata.plan
```

```
        # Update user subscription
        await update_user_subscription(user_id, plan)

    elif event.type == 'customer.subscription.deleted':
        # Handle cancellation
        pass

    return {"status": "success"}
```

🚀 DEPLOYMENT

Frontend: Vercel

bash

```
# Install Vercel CLI
npm i -g vercel

# Login
vercel login

# Deploy
vercel --prod

# Set environment variables in Vercel dashboard
# NEXT_PUBLIC_API_URL=https://api.centyr.com
# STRIPE_PUBLIC_KEY=pk_live_xxx
```

Backend: Railway.app

bash

```
# Install Railway CLI
npm i -g @railway/cli

# Login
railway login

# Initialize
railway init

# Deploy
railway up

# Set environment variables
railway variables set \
  DATABASE_URL=postgresql://... \
  AWS_ACCESS_KEY=xxx \
  AWS_SECRET_KEY=xxx \
  STRIPE_SECRET_KEY=sk_live_xxx
```

Or use Fly.io:

bash

```
# Install Fly CLI
curl -L https://fly.io/install.sh | sh

# Login
fly auth login

# Launch
fly launch

# Deploy
fly deploy
```

🔒 ENVIRONMENT VARIABLES

.env (Backend)

bash

```
# Database
DATABASE_URL=postgresql://user:password@localhost:5432/centyr

# AWS
AWS_ACCESS_KEY=AKIAIOSFODNN7EXAMPLE
AWS_SECRET_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
AWS_REGION=us-east-1
S3_BUCKET=centyr-images

# Stripe
STRIPE_SECRET_KEY=sk_test_xxx
STRIPE_WEBHOOK_SECRET=whsec_xxx

# JWT
SECRET_KEY=your-secret-key-min-32-chars
ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_MINUTES=30

# Email (SendGrid)
SENDGRID_API_KEY=SG.xxx
FROM_EMAIL=noreply@centyr.com

# Redis (for Celery)
REDIS_URL=redis://localhost:6379
```

.env.local (Frontend)

bash

```
NEXT_PUBLIC_API_URL=http://localhost:8000
NEXT_PUBLIC_STRIPE_PUBLIC_KEY=pk_test_xxx
NEXTAUTH_SECRET=your-nextauth-secret
NEXTAUTH_URL=http://localhost:3000
```

🧪 TESTING

Backend Tests

python

```python
# tests/test_images.py
import pytest
from httpx import AsyncClient
from app.main import app

@pytest.mark.asyncio
async def test_process_images():
    async with AsyncClient(app=app, base_url="http://test") as client:
        # Upload test image
        files = {'files': ('test.jpg', open('test.jpg', 'rb'), 'image/jpeg')}
        response = await client.post("/images/process", files=files)

        assert response.status_code == 200
        assert "job_id" in response.json()

@pytest.mark.asyncio
async def test_get_status():
    async with AsyncClient(app=app, base_url="http://test") as client:
        response = await client.get("/images/status/test-job-id")

        assert response.status_code == 200
        assert "status" in response.json()
```

Run tests:

```bash
pytest tests/ -v
```

MONITORING

Sentry (Error Tracking)

```python
# Add to main.py
import sentry_sdk

sentry_sdk.init(
    dsn="https://xxx@sentry.io/xxx",
    traces_sample_rate=1.0,
)
```

Logging

```python
# app/config.py
import logging

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)

logger = logging.getLogger(__name__)
```

✅ DEVELOPMENT WORKFLOW

Day-to-day:

```bash
# Terminal 1: Frontend
cd centyr-web
npm run dev
# → http://localhost:3000

# Terminal 2: Backend
cd centyr-api
source venv/bin/activate
uvicorn app.main:app --reload
# → http://localhost:8000

# Terminal 3: Database
docker run --name centyr-postgres \
  -e POSTGRES_PASSWORD=password \
  -p 5432:5432 -d postgres

# Terminal 4: Redis (for Celery)
docker run --name centyr-redis -p 6379:6379 -d redis
```

🎯 MVP FEATURE CHECKLIST

Week 1-2:

- [ ] Next.js setup + landing page
- [ ] Upload interface

- [ ] FastAPI setup
- [ ] Basic image processing (your script)
- [ ] S3 integration

Week 3-4:

- [ ] User authentication
- [ ] Stripe checkout
- [ ] Job queue (Celery)
- [ ] Download processed images
- [ ] Email notifications

Ready to deploy!

⚡ QUICK