API Keys Integration Guide for ANOINT Array

🔐 Environment Variables Setup

File Structure

```
/app
├── .env.example          # Template file (committed to git)
├── .env.local           # Your actual secrets (git-ignored)
└── lib/env.ts           # Type-safe environment access
```

Best Practices

✅ DO:

- Use `.env.local` for local development secrets
- Use `.env.example` as a template for team members
- Keep API keys server-side only (no `NEXT_PUBLIC_` prefix)
- Use TypeScript for environment variable validation
- Validate required environment variables on startup

❌ DON'T:

- Never commit actual API keys to git
- Never use `NEXT_PUBLIC_` prefix for secrets
- Don't hardcode API keys in your code
- Don't expose sensitive data to the client-side

📚 Usage Examples

1. In API Routes (Server-side)

```
// app/api/payment/route.ts
import { serverEnv } from '@/lib/env';
import Stripe from 'stripe';

const stripe = new Stripe(serverEnv.STRIPE_SECRET_KEY!, {
  apiVersion: '2023-10-16',
});

export async function POST(request: Request) {
  try {
    // Use your API key safely on the server
    const paymentIntent = await stripe.paymentIntents.create({
      amount: 2000,
      currency: 'usd',
    });

    return Response.json({ client_secret: paymentIntent.client_secret });
  } catch (error) {
    return Response.json({ error: 'Payment failed' }, { status: 500 });
  }
}
```

2. Email Service Integration

```
// app/api/send-email/route.ts
import { serverEnv } from '@/lib/env';

export async function POST(request: Request) {
  const { to, subject, content } = await request.json();

  // Example with SendGrid
  const response = await fetch('https://api.sendgrid.v3/mail/send', {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${serverEnv.EMAIL_API_KEY}`,
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      personalizations: [{ to: [{ email: to }] }],
      from: { email: serverEnv.EMAIL_FROM },
      subject,
      content: [{ type: 'text/html', value: content }],
    }),
  });

  if (response.ok) {
    return Response.json({ success: true });
  } else {
    return Response.json({ error: 'Email failed to send' }, { status: 500 });
  }
}
```

3. File Upload to Cloud Storage

```
// app/api/upload-to-cloud/route.ts
import { serverEnv } from '@/lib/env';
import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3';

const s3Client = new S3Client({
  region: serverEnv.AWS_REGION,
  credentials: {
    accessKeyId: serverEnv.AWS_ACCESS_KEY_ID!,
    secretAccessKey: serverEnv.AWS_SECRET_ACCESS_KEY!,
  },
});

export async function POST(request: Request) {
  const formData = await request.formData();
  const file = formData.get('file') as File;

  if (!file) {
    return Response.json({ error: 'No file provided' }, { status: 400 });
  }

  const bytes = await file.arrayBuffer();
  const buffer = Buffer.from(bytes);

  const uploadCommand = new PutObjectCommand({
    Bucket: serverEnv.AWS_BUCKET_NAME,
    Key: `products/${Date.now()}-${file.name}`,
    Body: buffer,
    ContentType: file.type,
  });

  try {
    await s3Client.send(uploadCommand);
    return Response.json({
      success: true,
      url: `https://d2908q01vomqb2.cloudfront.net/e1822db470e60d090affd0956d743cb0e7cd-
f113/2024/03/20/1_architecture.png
    });
  } catch (error) {
    return Response.json({ error: 'Upload failed' }, { status: 500 });
  }
}
```

4. Client-side Usage (Non-sensitive data only)

```
// components/analytics.tsx
import { clientEnv } from '@/lib/env';

export function Analytics() {
  // Only use NEXT_PUBLIC_ prefixed variables on client-side
  const gaId = clientEnv.NEXT_PUBLIC_GOOGLE_ANALYTICS_ID;

  if (!gaId || clientEnv.NEXT_PUBLIC_APP_ENV !== 'production') {
    return null;
  }

  return (
    <script
      async
      src={`https://www.googletagmanager.com/gtag/js?id=${gaId}`}
    />
  );
}
```

🚀 Adding Your API Keys

Step 1: Add to .env.local

```
# Open your .env.local file and add:
STRIPE_SECRET_KEY="sk_test_your_actual_stripe_key_here"
OPENAI_API_KEY="sk-your_actual_openai_key_here"
EMAIL_API_KEY="your_actual_sendgrid_key_here"
```

Step 2: Update lib/env.ts if needed

```
// Add new environment variables to the serverEnv object
export const serverEnv = {
  // ... existing variables
  YOUR_NEW_API_KEY: process.env.YOUR_NEW_API_KEY,
} as const;
```

Step 3: Use in your API routes

```
import { serverEnv } from '@/lib/env';

// Now you can safely use serverEnv.YOUR_NEW_API_KEY
```

🔒 Security Checklist

- [ ] All sensitive API keys are in `.env.local`
- [ ] `.env.local` is in `.gitignore`
- [ ] No API keys have `NEXT_PUBLIC_` prefix unless they need to be client-side
- [ ] Environment variables are validated on startup
- [ ] API keys are only used in server-side code (API routes, server components)
- [ ] Production environment variables are set in your deployment platform

📝 Production Deployment

When deploying to production, set these environment variables in your hosting platform:

Vercel:

```
vercel env add STRIPE_SECRET_KEY
vercel env add EMAIL_API_KEY
# ... add all your secret keys
```

Other platforms:

Add environment variables through your platform's dashboard or CLI.

🆘 Troubleshooting

Common Issues:

1. **"Environment variable not found"**
   - Check spelling in `.env.local`
   - Restart your development server after adding new variables

2. **API key not working**
   - Verify the key is correct in your provider's dashboard
   - Check if the key has the required permissions/scopes

3. **Client-side access to server variables**
   - Never try to access server-side env vars on the client
   - Use `NEXT_PUBLIC_` prefix only for non-sensitive client data

This setup provides enterprise-level security for your API keys while maintaining developer-friendly access patterns.