**CodeCrucible - Complete Feature Implementation & Paywall Enforcement**

**Critical Mission**

Implement EVERY feature shown in the subscription tiers and enforce proper paywall restrictions. All features must be fully functional with proper tier-based access controls and graceful upgrade prompts.

**Subscription Tier Features (MUST BE IMPLEMENTED)**

**FREE TIER (Dev Mode)**

* ✅ **Basic 2-voice combinations** (Explorer + Maintainer only)
* ✅ **3 generations per day** (hard limit with reset at midnight UTC)
* ✅ **Basic analytics** (simple usage stats, last 7 days only)
* ✅ **Community support** (help documentation and FAQ)

**PRO TIER ($19/month)**

* ✅ **Unlimited code generations** (remove daily limits)
* ✅ **Advanced synthesis engine** (multi-voice code combination algorithm)
* ✅ **Analytics dashboard** (detailed VFSP tracking, 90 days history)
* ✅ **Priority voice recommendations** (AI suggests optimal voice combinations)
* ✅ **Export generated code** (download as files, GitHub integration)
* ✅ **Advanced customization** (custom voice profiles, themes, preferences)

**TEAM TIER ($49/month)**

* ✅ **Everything in Pro** (inherit all Pro features)
* ✅ **Team collaboration** (shared sessions, real-time multi-user editing)
* ✅ **Shared voice profiles** (team can create and share custom voices)
* ✅ **Advanced analytics** (team performance, collaborative insights)
* ✅ **Team management** (invite members, role permissions, usage tracking)
* ✅ **Priority support** (email support with 24h response time)

**ENTERPRISE TIER ($99/month)**

* ✅ **Everything in Team** (inherit all Team features)
* ✅ **Custom AI training** (train voices on company codebases)
* ✅ **On-premise deployment** (self-hosted instance options)
* ✅ **SSO integration** (Google Workspace, Microsoft 365, Okta)
* ✅ **Dedicated support** (phone support, dedicated success manager)
* ✅ **Custom integrations** (API access, webhooks, enterprise tools)
* ✅ **SLA guarantees** (99.9% uptime, performance commitments)
* ✅ **Compliance features** (SOC2, GDPR, audit logs, data residency)

**Technical Implementation Requirements**

**Subscription State Management**

interface UserSubscription {

userId: string;

tier: 'free' | 'pro' | 'team' | 'enterprise';

status: 'active' | 'canceled' | 'past\_due' | 'trialing';

currentPeriodStart: Date;

currentPeriodEnd: Date;

stripeCustomerId: string;

stripeSubscriptionId: string;

usageThisMonth: {

generations: number;

voicesUsed: string[];

synthesisAttempts: number;

teamMembers?: number;

};

}

// Middleware for EVERY protected route

const enforceSubscriptionLimits = async (req, res, next) => {

const user = await getUserWithSubscription(req.userId);

const feature = getRequiredFeature(req.path);

if (!hasFeatureAccess(user.subscription.tier, feature)) {

return res.status(403).json({

error: 'Feature requires upgrade',

requiredTier: getMinimumTier(feature),

upgradeUrl: `/upgrade?feature=${feature}`

});

}

// Check usage limits

if (feature === 'generation' && user.subscription.tier === 'free') {

if (user.subscription.usageThisMonth.generations >= 3) {

return res.status(429).json({

error: 'Daily generation limit reached',

upgradeUrl: '/upgrade?reason=limit\_reached'

});

}

}

next();

};

**Feature Access Control System**

const FEATURE\_MATRIX = {

'unlimited\_generations': ['pro', 'team', 'enterprise'],

'synthesis\_engine': ['pro', 'team', 'enterprise'],

'analytics\_dashboard': ['pro', 'team', 'enterprise'],

'voice\_recommendations': ['pro', 'team', 'enterprise'],

'code\_export': ['pro', 'team', 'enterprise'],

'custom\_voices': ['pro', 'team', 'enterprise'],

'team\_collaboration': ['team', 'enterprise'],

'shared\_profiles': ['team', 'enterprise'],

'team\_management': ['team', 'enterprise'],

'priority\_support': ['team', 'enterprise'],

'custom\_ai\_training': ['enterprise'],

'sso\_integration': ['enterprise'],

'api\_access': ['enterprise'],

'compliance\_features': ['enterprise']

};

const hasFeatureAccess = (userTier: string, feature: string): boolean => {

return FEATURE\_MATRIX[feature]?.includes(userTier) || false;

};

**Frontend Paywall Components**

const FeatureGate: React.FC<{

feature: string;

children: React.ReactNode;

fallback?: React.ReactNode;

}> = ({ feature, children, fallback }) => {

const { user } = useAuth();

const hasAccess = hasFeatureAccess(user.subscription.tier, feature);

if (!hasAccess) {

return fallback || <UpgradePrompt feature={feature} />;

}

return <>{children}</>;

};

const UpgradePrompt: React.FC<{ feature: string }> = ({ feature }) => {

const requiredTier = getMinimumTier(feature);

const pricing = TIER\_PRICING[requiredTier];

return (

<div className="upgrade-prompt">

<h3>Unlock {feature.replace('\_', ' ')}</h3>

<p>This feature requires {requiredTier} subscription (${pricing}/month)</p>

<button onClick={() => redirectToUpgrade(requiredTier)}>

Upgrade to {requiredTier}

</button>

</div>

);

};

**API Route Protection Examples**

// Voice generation with limits

app.post('/api/generate', enforceSubscriptionLimits, async (req, res) => {

const { voices, prompt } = req.body;

const user = await getUser(req.userId);

// Free tier: max 2 voices

if (user.subscription.tier === 'free' && voices.length > 2) {

return res.status(403).json({

error: 'Free tier limited to 2 voices',

upgradeUrl: '/upgrade?feature=unlimited\_voices'

});

}

// Increment usage counter

await incrementUsage(user.id, 'generations');

const results = await generateMultiVoiceCode(voices, prompt);

res.json(results);

});

// Synthesis engine (Pro+)

app.post('/api/synthesize', enforceSubscriptionLimits, async (req, res) => {

// This route automatically blocked for free users by middleware

const synthesis = await synthesizeVoiceOutputs(req.body.outputs);

res.json(synthesis);

});

// Team collaboration (Team+)

app.post('/api/teams/invite', enforceSubscriptionLimits, async (req, res) => {

const invitation = await createTeamInvitation(req.body);

res.json(invitation);

});

**Usage Tracking & Analytics**

const trackFeatureUsage = async (userId: string, feature: string, metadata?: any) => {

await database.insert('feature\_usage').values({

userId,

feature,

timestamp: new Date(),

metadata: JSON.stringify(metadata)

});

// Update monthly usage counters

await updateMonthlyUsage(userId, feature);

};

const generateAnalyticsDashboard = async (userId: string, tier: string) => {

const baseAnalytics = await getBasicUsageStats(userId);

if (tier === 'free') {

return {

...baseAnalytics,

upgradePrompt: 'Unlock detailed analytics with Pro subscription'

};

}

const advancedAnalytics = await getAdvancedAnalytics(userId);

return { ...baseAnalytics, ...advancedAnalytics };

};

**Stripe Integration (CRITICAL)**

**Webhook Handling for Real-time Updates**

app.post('/api/stripe/webhook', express.raw({type: 'application/json'}), async (req, res) => {

const sig = req.headers['stripe-signature'];

const event = stripe.webhooks.constructEvent(req.body, sig, process.env.STRIPE\_WEBHOOK\_SECRET);

switch (event.type) {

case 'checkout.session.completed':

const session = event.data.object;

await updateUserSubscription(session.customer, session.metadata.tier);

break;

case 'customer.subscription.updated':

const subscription = event.data.object;

await syncSubscriptionStatus(subscription);

break;

case 'customer.subscription.deleted':

await downgradeToFree(event.data.object.customer);

break;

case 'invoice.payment\_failed':

await handlePaymentFailure(event.data.object);

break;

}

res.json({received: true});

});

**Upgrade Flow Implementation**

const createUpgradeCheckout = async (userId: string, targetTier: string) => {

const user = await getUser(userId);

const priceId = STRIPE\_PRICE\_IDS[targetTier];

const session = await stripe.checkout.sessions.create({

customer: user.stripeCustomerId,

payment\_method\_types: ['card'],

line\_items: [{ price: priceId, quantity: 1 }],

mode: 'subscription',

success\_url: `${process.env.BASE\_URL}/dashboard?upgrade=success`,

cancel\_url: `${process.env.BASE\_URL}/upgrade?cancelled=true`,

metadata: {

userId: user.id,

tier: targetTier

}

});

return session.url;

};

**UI Implementation Requirements**

**Feature-Specific UI States**

1. **Voice Selection**: Show locked voices with upgrade prompts
2. **Analytics Dashboard**: Display tier-appropriate charts and data
3. **Team Features**: Hide team management for individual plans
4. **Export Options**: Disable export buttons for free users
5. **Custom Voices**: Show "Pro Feature" badges on locked functionality

**Upgrade Prompts Strategy**

* **Contextual**: Show relevant upgrade prompts when users hit limits
* **Non-intrusive**: Don't block workflow, just suggest upgrades
* **Value-focused**: Highlight specific benefits of upgrading
* **One-click**: Direct links to Stripe checkout for each tier

**Usage Indicators**

const UsageMeter: React.FC = () => {

const { user } = useAuth();

const usage = user.subscription.usageThisMonth;

const limits = TIER\_LIMITS[user.subscription.tier];

return (

<div className="usage-meter">

<div className="usage-bar">

<div

className="usage-fill"

style={{ width: `${(usage.generations / limits.generations) \* 100}%` }}

/>

</div>

<span>{usage.generations}/{limits.generations} generations used</span>

{usage.generations >= limits.generations \* 0.8 && (

<UpgradePrompt feature="unlimited\_generations" />

)}

</div>

);

};

**Testing Requirements**

**Feature Access Tests**

* ✅ Free users cannot access Pro/Team/Enterprise features
* ✅ Pro users can access Pro features but not Team/Enterprise
* ✅ Upgrade flows work correctly from each tier
* ✅ Usage limits are enforced and reset properly
* ✅ Stripe webhooks update subscription status in real-time

**Error Handling**

* ✅ Graceful degradation when payment fails
* ✅ Clear error messages for feature access denials
* ✅ Automatic retry for failed webhook deliveries
* ✅ Rollback mechanisms for subscription changes

**Success Criteria**

* **Every feature in the subscription modal must be implemented and working**
* **Proper paywall enforcement on all protected features**
* **Seamless upgrade flows with immediate feature access**
* **Real-time usage tracking and limit enforcement**
* **Professional error handling and user feedback**
* **Stripe webhook integration for subscription management**

**Performance Requirements**

* **API response times < 200ms for protected routes**
* **Real-time subscription status updates**
* **Efficient feature access checking (cached permissions)**
* **Scalable usage tracking system**

Build this as a production-ready SaaS platform that could handle thousands of paying customers from day one. Every feature must be bulletproof and every paywall must be properly implemented.