

Feature Licensing & Customization Points Analysis

Executive Summary

You currently have a **basic feature flag system**, but it's designed for **A/B testing and gradual rollouts**, not for **commercial license-based feature unlocking** (like Tesla's model).

To achieve your goal of selling features as upgrades that customers can unlock without software reinstallation, you need to implement a **comprehensive license management system**.

What You Want (Tesla Model)

Key Requirements:

1. **All features pre-installed** in the codebase
2. **Customer-specific feature activation** based on purchased licenses
3. **Dynamic unlocking** without code changes or redeployment
4. **Multiple tiers/packages** (Basic, Professional, Enterprise, etc.)
5. **Feature-level granularity** (à la carte purchases)
6. **License expiration & renewal** management
7. **Usage tracking & enforcement**
8. **Multi-tenant isolation** (each driving school has separate licenses)

Example Use Cases:

- **Driving School A** (Basic Package): Only has Lesson Management + Basic Calendar
 - **Driving School B** (Pro Package): Has everything except Advanced Reporting
 - **Driving School C** (Enterprise): Has all features unlocked
 - **Driving School D**: Buys Basic, then upgrades to add SMS Notifications (instant unlock)
-

What You Currently Have

1. Feature Flags System (`lib/config/features.ts`, `lib/config-utils.ts`)

Current Capabilities: - Toggle features on/off globally - Role-based access (e.g., only ADMIN sees certain features) - User-specific targeting (enable for specific user IDs) - Gradual rollout (rollout percentage) - Environment-based flags (production vs staging) - Expiration dates for temporary features - Audit trail (configuration history)

Current Limitations: - No customer/tenant-level licensing - No package/tier management - No license key validation - No payment/subscription integration - No usage limits or quotas - Features are hardcoded in `/lib/config/features.ts` (not database-driven for licensing)

2. Database Structure

Existing Tables:

```
model FeatureFlag {
  flagKey      String @unique
  isEnabled    Boolean
  enabledForRoles String[] // e.g., ["ADMIN", "INSTRUCTOR"]
  enabledForUsers String[] // e.g., ["user123", "user456"]
  rolloutPercent Int
  expiresAt     DateTime?
  // ... but NO customer/organization/license association
}

model SystemSetting {
  settingKey String @unique
  settingValue String
  // Global settings only
}

model UserPreference {
  userId      String @unique
  theme       String
  language    String
  // User-level preferences only
}
```

Missing Tables: - Organization (driving schools) - LicenseKey (purchased licenses) - SubscriptionPlan (pricing tiers) - PlanFeature (what features each plan includes) - FeatureLicense (active features per customer) - UsageTracking (feature usage limits)

Recommended Solution: Complete License Management System

Phase 1: Database Schema Enhancements

1.1 Organization Model (Multi-tenant support)

```
model Organization {
  id      String @id @default(cuid())
```

```

name          String    // "Conquistadora Driving School"
slug          String    @unique // "conquistadora"
subscriptionPlanId String?

// Billing
billingEmail   String?
billingAddress String?

// Status
isActive      Boolean   @default(true)
trialEndsAt   DateTime?

// Relations
subscriptionPlan SubscriptionPlan? @relation(fields: [subscriptionPlanId], references: [id])
users         User[]
licenses      FeatureLicense[]

createdAt     DateTime @default(now())
updatedAt     DateTime @updatedAt
}

```

1.2 Subscription Plans (Pricing tiers)

```

model SubscriptionPlan {
  id          String    @id @default(cuid())
  name        String    // "Basic", "Professional", "Enterprise"
  slug        String    @unique // "basic", "pro", "enterprise"
  displayName String    // "Professional Package"
  description String?

  // Pricing
  monthlyPrice Decimal @default(0)
  yearlyPrice   Decimal @default(0)
  currency      String  @default("EUR")

  // Limits
  maxUsers      Int?    // null = unlimited
  maxVehicles   Int?
  maxLessonsPerMonth Int?
  maxStorageGB  Int?

  // Features included
  includedFeatures PlanFeature[]

  // Display
  isPopular      Boolean @default(false)
}

```

```

displayOrder    Int          @default(0)
isActive        Boolean      @default(true)

// Relations
organizations    Organization[]

createdAt        DateTime @default(now())
updatedAt        DateTime @updatedAt
}

```

1.3 Feature Catalog (All available features)

```

model Feature {
  id            String    @id @default(cuid())
  key           String    @unique // "lesson_management", "sms_notifications"
  name          String    // "Lesson Management"
  description    String?
  category       String   // "core", "advanced", "premium", "addon"

  // À la carte pricing (optional, for individual purchases)
  standalonePrice Decimal? @default(0)

  // Dependencies
  requiredFeatures String[] // Features that must be enabled first

  // Display
  displayOrder    Int          @default(0)
  icon            String?
  isAvailable     Boolean      @default(true)

  // Relations
  planFeatures    PlanFeature[]
  featureLicenses FeatureLicense[]

  createdAt        DateTime @default(now())
  updatedAt        DateTime @updatedAt
}

```

1.4 Plan-Feature Mapping

```

model PlanFeature {
  id            String    @id @default(cuid())
  planId        String
  featureId     String

  // Limits specific to this plan

```

```

usageLimit      Int?      // e.g., 100 SMS per month

plan            SubscriptionPlan @relation(fields: [planId], references: [id], onDelete: Cascade)
feature         Feature      @relation(fields: [featureId], references: [id], onDelete: Cascade)

@@unique([planId, featureId])
}

```

1.5 Active Feature Licenses (Per organization)

```

model FeatureLicense {
  id              String      @id @default(cuid())
  organizationId  String
  featureId       String

  // License details
  licenseKey      String?     @unique // Optional external key
  source          String      @default("plan") // "plan", "addon", "trial", "promo"

  // Status
  isActive        Boolean     @default(true)
  activatedAt     DateTime     @default(now())
  expiresAt       DateTime?    // null = permanent

  // Usage tracking
  usageLimit      Int?        // e.g., 500 SMS per month
  usageCount      Int          @default(0)
  lastResetAt     DateTime     @default(now())

  // Relations
  organization     Organization @relation(fields: [organizationId], references: [id], onDelete: Cascade)
  feature          Feature      @relation(fields: [featureId], references: [id], onDelete: Cascade)

  createdAt       DateTime     @default(now())
  updatedAt       DateTime     @updatedAt

  @@unique([organizationId, featureId])
  @@index([organizationId])
  @@index([featureId])
  @@index([expiresAt])
}

```

1.6 License History (Audit trail)

```

model LicenseHistory {
  id              String      @id @default(cuid())

```

```

organizationId String
featureId      String
action        String // "ACTIVATED", "DEACTIVATED", "RENEWED", "EXPIRED", "UPGRADED"

// Context
reason        String?
performedBy   String? // User ID
performedByRole String?

// Old vs New
oldExpiresAt  DateTime?
newExpiresAt  DateTime?

createdAt     DateTime @default(now())

@@index([organizationId])
@@index([createdAt])
}

```

Phase 2: Backend Implementation

2.1 License Manager Service (lib/license-manager.ts)

```

export class LicenseManager {
  // Check if organization has access to a feature
  static async hasFeatureAccess(
    organizationId: string,
    featureKey: string
  ): Promise<boolean>

  // Get all active features for an organization
  static async getOrganizationFeatures(
    organizationId: string
  ): Promise<Feature[]>

  // Activate a feature for an organization
  static async activateFeature(
    organizationId: string,
    featureKey: string,
    options?: {
      expiresAt?: Date;
      usageLimit?: number;
      source?: string;
    }
  ): Promise<FeatureLicense>
}

```

```

// Deactivate a feature
static async deactivateFeature(
  organizationId: string,
  featureKey: string
): Promise<void>

// Check usage limits
static async checkUsageLimit(
  organizationId: string,
  featureKey: string
): Promise<{ allowed: boolean; remaining: number }>

// Track feature usage
static async trackUsage(
  organizationId: string,
  featureKey: string,
  amount?: number
): Promise<void>

// Verify license key
static async verifyLicenseKey(
  licenseKey: string
): Promise<{ valid: boolean; features: string[] }>
}

```

2.2 Middleware for Feature Access Control (middleware/feature-access.ts)

```

export function requireFeature(featureKey: string) {
  return async (req, res, next) => {
    const user = req.user;
    const orgId = user.organizationId;

    const hasAccess = await LicenseManager.hasFeatureAccess(orgId, featureKey);

    if (!hasAccess) {
      return res.status(403).json({
        error: 'Feature not available',
        featureKey,
        upgradeUrl: '/pricing'
      });
    }

    next();
  };
}

```

2.3 API Routes

```
// GET /api/organization/features
// Returns all active features for current org

// POST /api/organization/features/activate
// Activate a feature (admin/billing only)

// POST /api/organization/licenses/validate
// Validate and activate license key

// GET /api/features/catalog
// Get all available features with pricing

// GET /api/subscription-plans
// Get all available plans
```

Phase 3: Frontend Implementation

3.1 Feature Access Hook (hooks/use-feature-access.ts)

```
export function useFeatureAccess(featureKey: string) {
  const { data: session } = useSession();
  const orgId = session?.user?.organizationId;

  const { data, isLoading } = useSWR(
    orgId ? `/api/organization/features` : null,
    fetcher
  );

  const hasAccess = data?.features?.includes(featureKey) || false;

  return {
    hasAccess,
    isLoading,
    showUpgradePrompt: !hasAccess && !isLoading,
  };
}

// Usage in components:
const { hasAccess, showUpgradePrompt } = useFeatureAccess('sms_notifications');

if (!hasAccess) {
  return <UpgradePrompt feature="SMS Notifications" />;
}
```

3.2 Feature Gate Component (components/feature-gate.tsx)

```

export function FeatureGate({
  feature,
  fallback,
  children
}): {
  feature: string;
  fallback?: React.ReactNode;
  children: React.ReactNode;
}) {
  const { hasAccess, showUpgradePrompt } = useFeatureAccess(feature);

  if (!hasAccess) {
    return fallback || <UpgradePrompt feature={feature} />;
  }

  return <>{children}</>;
}

```

```

// Usage:
<FeatureGate feature="sms_notifications">
  <SMSSettingsPanel />
</FeatureGate>

```

3.3 Upgrade Prompts

```

export function UpgradePrompt({ feature }: { feature: string }) {
  return (
    <Card className="border-2 border-dashed border-blue-300 bg-blue-50">
      <CardContent className="flex items-center justify-between p-6">
        <div>
          <h3 className="font-semibold text-lg">
            Unlock {feature}
          </h3>
          <p className="text-sm text-gray-600">
            This feature is available in our Professional plan
          </p>
        </div>
        <Button asChild>
          <Link href="/pricing">
            <Lock className="w-4 h-4 mr-2" />
            Upgrade Now
          </Link>
        </Button>
      </CardContent>
    </Card>
  );
}

```

}

Phase 4: Admin Interface

4.1 License Management Dashboard

- View all organizations and their active features
- Manually activate/deactivate features
- Grant promotional/trial access
- View usage statistics
- Manage license keys

4.2 Customer Self-Service Portal

- View current plan and features
- See available upgrades
- Activate license keys
- View usage statistics
- Manage billing

Naming Recommendations

Your current terminology is confusing. Here's what I recommend:

Current Term	Better Term	Purpose
"Configuration Points"	"Feature Licensing System"	Overall system
"Feature Flags"	"Feature Flags" (keep)	Internal A/B testing
N/A	"Feature Licenses"	Purchased/active features per org
N/A	"Subscription Plans"	Pricing tiers
N/A	"Feature Catalog"	All available features

Suggested File Structure:

```
lib/  
  licensing/  
    license-manager.ts    # Core license logic  
    feature-catalog.ts    # Feature definitions  
    plan-manager.ts       # Subscription plan logic  
    usage-tracker.ts      # Usage limits & tracking  
  config/
```

<code>features.ts</code>	<code># Internal feature flags (A/B testing)</code>
<code>system-settings.ts</code>	<code># Global system settings</code>

Implementation Checklist

Immediate Actions (Phase 1)

- ☐ Add `Organization` model to Prisma schema
- ☐ Add `SubscriptionPlan` model
- ☐ Add `Feature` catalog model
- ☐ Add `PlanFeature` mapping model
- ☐ Add `FeatureLicense` model
- ☐ Add `LicenseHistory` model
- ☐ Run migrations

Core Implementation (Phase 2)

- ☐ Create `LicenseManager` service
- ☐ Create API routes for license management
- ☐ Add feature access middleware
- ☐ Seed initial feature catalog
- ☐ Seed subscription plans (Basic, Pro, Enterprise)

Frontend Integration (Phase 3)

- ☐ Create `useFeatureAccess` hook
- ☐ Create `FeatureGate` component
- ☐ Create `UpgradePrompt` component
- ☐ Update all feature-gated pages/components
- ☐ Create pricing page

Admin Tools (Phase 4)

- ☐ License management dashboard
- ☐ Customer portal for self-service
- ☐ Usage analytics dashboard

Testing & Validation

- ☐ Test feature activation/deactivation
 - ☐ Test usage limits
 - ☐ Test license expiration
 - ☐ Test upgrade flows
 - ☐ Test multi-tenant isolation
-

Quick Win: Minimal Implementation

If you want to start small, here's a **simplified version** using your existing structure:

Option A: Extend Current Feature Flags

Add organizationId to existing FeatureFlag model:

```
model FeatureFlag {
  // ... existing fields
  organizationId String? // null = global, specific ID = org-specific

  @@index([organizationId])
}
```

Update isFeatureEnabled() to check organization:

```
export async function isFeatureEnabled(
  flagKey: string,
  userId?: string,
  organizationId?: string // NEW
): Promise<boolean> {
  // Check org-specific flag first
  if (organizationId) {
    const orgFlag = await prisma.featureFlag.findFirst({
      where: { flagKey, organizationId }
    });
    if (orgFlag) return orgFlag.isEnabled;
  }

  // Fall back to global flag
  const globalFlag = await prisma.featureFlag.findFirst({
    where: { flagKey, organizationId: null }
  });
  return globalFlag?.isEnabled || false;
}
```

Pros: Quick to implement, uses existing infrastructure **Cons:** Not scalable, no license key support, no subscription management

Recommendations

For Your Use Case (Selling to Multiple Driving Schools):

I strongly recommend implementing the full system (Phases 1-4) because:

1. **Scalability:** You'll likely have 10-100+ driving schools, each with different licenses
2. **Revenue Management:** Proper subscription/license tracking is essential for billing
3. **Customer Experience:** Self-service upgrades improve satisfaction
4. **Legal Compliance:** License keys and audit trails may be required
5. **Support Efficiency:** Admins need visibility into who has what

Timeline Estimate:

- **Phase 1 (Schema):** 4-6 hours
- **Phase 2 (Backend):** 8-12 hours
- **Phase 3 (Frontend):** 8-12 hours
- **Phase 4 (Admin):** 6-8 hours
- **Total:** ~30-40 hours for complete system

Quick Start (2-3 hours):

If you want to test the concept first, implement **Option A** (extend feature flags) with `organizationId`, then migrate to full system later.

Next Steps

Would you like me to: 1. **Implement the complete system** (Phases 1-4)? 2. **Start with the quick win** (Option A) and expand later? 3. **Create a prototype** with just 2-3 features to test the concept? 4. **Focus on specific aspect** (e.g., just the licensing logic, just the UI)?

Let me know your preference and timeline!