TapTask - Technical Architecture & Implementation Guide

Version: 1.0

Date: October 25, 2025

Project Type: iOS Shortcuts Marketplace with Apple Pay Integration

---

Table of Contents

Executive Summary

TapTask is a full-stack marketplace application for buying and selling iOS Shortcuts. The platform enables creators to monetize their automation workflows while providing users with instant access to productivity-enhancing shortcuts through a seamless Apple Pay checkout experience.

Core Capabilities

- **Marketplace**: Browse, search, and discover iOS Shortcuts by category

- **E-commerce**: Secure payments via Stripe with Apple Pay support
- **Creator Platform**: Submit shortcuts, track earnings, and manage listings
- **Admin Dashboard**: Content moderation and marketplace management
- **User Library**: Personal collection of purchased shortcuts

---

Technology Stack

Frontend

- **Framework**: React 19 with TypeScript
- **Build Tool**: Vite 7.x
- **Routing**: Wouter (lightweight React router)
- **Styling**: Tailwind CSS 4.x
- **UI Components**: shadcn/ui (Radix UI primitives)
- **State Management**: tRPC React Query integration
- **Form Handling**: React Hook Form + Zod validation

Backend

- **Runtime**: Node.js 22.x
- **Framework**: Express 4.x
- **API Layer**: tRPC 11.x (type-safe RPC)
- **Database ORM**: Drizzle ORM
- **Database**: MySQL (compatible with PlanetScale, Railway, AWS RDS)
- **Authentication**: JWT-based session management
- **Payment Processing**: Stripe SDK

Development Tools

- **Package Manager**: pnpm (workspace support)
- **TypeScript**: 5.x (strict mode)
- **Hot Reload**: tsx watch (backend), Vite HMR (frontend)
- **Database Migrations**: Drizzle Kit

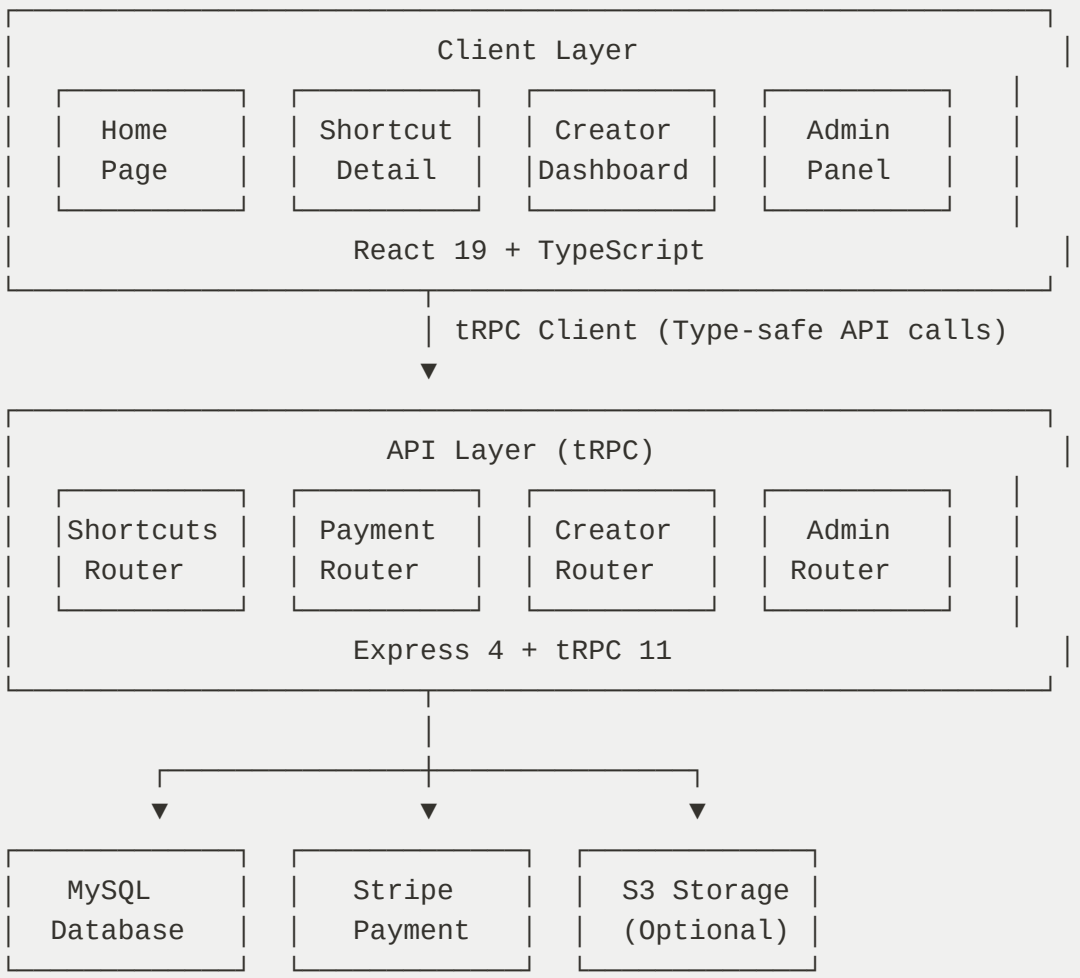
Infrastructure

- **Deployment**: Vercel, Railway, or any Node.js platform
- **Database Hosting**: PlanetScale, Railway, or AWS RDS
- **File Storage**: S3-compatible storage (optional for images)
- **CDN**: Cloudflare or Vercel Edge Network

System Architecture

High-Level Architecture

Plain Text



Request Flow

1. **Client Request**: User interacts with React UI
2. **tRPC Call**: Type-safe API call via tRPC client
3. **Authentication**: JWT token validated in middleware

4. **Business Logic**: Router procedures execute database queries
5. **Database**: Drizzle ORM translates to SQL queries
6. **Response**: Type-safe data returned to client
7. **UI Update**: React Query caches and updates UI

Database Design

Entity Relationship Diagram

Plain Text

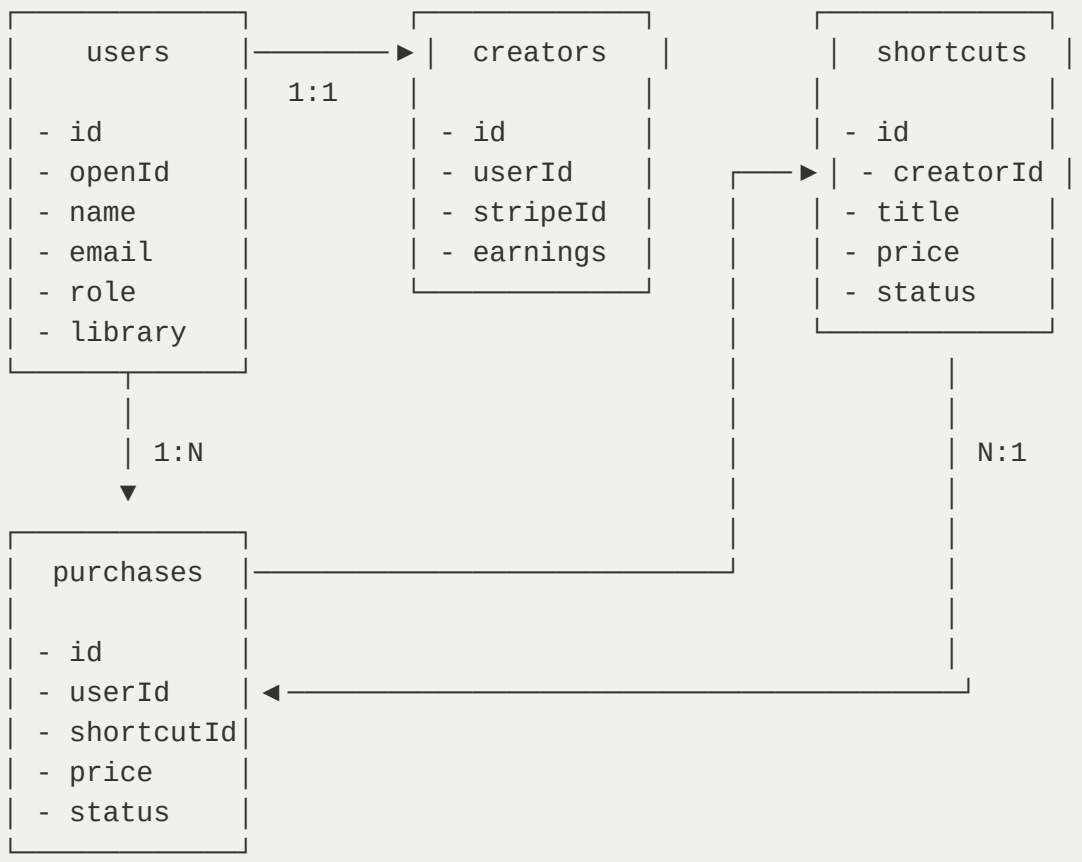


Table Schemas

users

SQL

```sql
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
```

```
  openId VARCHAR(64) UNIQUE NOT NULL,
  name TEXT,
  email VARCHAR(320),
  loginMethod VARCHAR(64),
  role ENUM('user', 'creator', 'admin') DEFAULT 'user',
  stripeCustomerId VARCHAR(255),
  library TEXT,  -- JSON array of shortcut IDs
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  lastSignedIn TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

shortcuts

SQL

```
CREATE TABLE shortcuts (
  id INT AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(255) NOT NULL,
  slug VARCHAR(255) UNIQUE NOT NULL,
  description TEXT NOT NULL,
  category VARCHAR(100) NOT NULL,
  tags TEXT,  -- JSON array
  price INT DEFAULT 0,  -- In cents (e.g., 299 = $2.99)
  iCloudLink TEXT NOT NULL,
  previewImage TEXT,
  previewMedia TEXT,
  creatorId INT NOT NULL,
  creatorName VARCHAR(255) NOT NULL,
  creatorAvatar TEXT,
  status ENUM('pending', 'approved', 'rejected') DEFAULT 'pending',
  featured INT DEFAULT 0,  -- Boolean (0 or 1)
  trending INT DEFAULT 0,  -- Boolean (0 or 1)
  downloads INT DEFAULT 0,
  purchases INT DEFAULT 0,
  requiredIOSVersion VARCHAR(50),
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  FOREIGN KEY (creatorId) REFERENCES users(id)
);
```

creators

SQL

```sql
CREATE TABLE creators (
  id INT AUTO_INCREMENT PRIMARY KEY,
  userId INT UNIQUE NOT NULL,
  stripeAccountId VARCHAR(255),
  stripeAccountStatus ENUM('pending', 'active', 'restricted') DEFAULT
'pending',
  totalEarnings INT DEFAULT 0,  -- In cents
  pendingEarnings INT DEFAULT 0,
  shortcutsSubmitted INT DEFAULT 0,
  shortcutsApproved INT DEFAULT 0,
  shortcutsSold INT DEFAULT 0,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  FOREIGN KEY (userId) REFERENCES users(id)
);
```

purchases

SQL

```sql
CREATE TABLE purchases (
  id INT AUTO_INCREMENT PRIMARY KEY,
  userId INT NOT NULL,
  shortcutId INT NOT NULL,
  price INT NOT NULL,  -- In cents
  stripePaymentIntentId VARCHAR(255),
  status ENUM('pending', 'completed', 'refunded') DEFAULT 'pending',
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (userId) REFERENCES users(id),
  FOREIGN KEY (shortcutId) REFERENCES shortcuts(id)
);
```

reports

SQL

```sql
CREATE TABLE reports (
  id INT AUTO_INCREMENT PRIMARY KEY,
  shortcutId INT NOT NULL,
  reportedBy INT NOT NULL,
  reason TEXT NOT NULL,
  status ENUM('pending', 'resolved', 'dismissed') DEFAULT 'pending',
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (shortcutId) REFERENCES shortcuts(id),
```

```
  FOREIGN KEY (reportedBy) REFERENCES users(id)
);
```

Indexing Strategy

SQL

```
-- Performance indexes
CREATE INDEX idx_shortcuts_status ON shortcuts(status);
CREATE INDEX idx_shortcuts_category ON shortcuts(category);
CREATE INDEX idx_shortcuts_creator ON shortcuts(creatorId);
CREATE INDEX idx_shortcuts_featured ON shortcuts(featured);
CREATE INDEX idx_shortcuts_trending ON shortcuts(trending);
CREATE INDEX idx_purchases_user ON purchases(userId);
CREATE INDEX idx_purchases_shortcut ON purchases(shortcutId);
```

API Architecture

tRPC Router Structure

TypeScript

```
appRouter
├── auth
│   ├── me (query) - Get current user
│   └── logout (mutation) - Sign out
├── shortcuts
│   ├── list (query) - Browse shortcuts with filters
│   ├── getBySlug (query) - Get single shortcut
│   ├── download (mutation) - Track download
│   └── myLibrary (query) - User's purchased shortcuts
├── payment
│   ├── createCheckoutSession (mutation) - Stripe checkout
│   ├── createPaymentIntent (mutation) - Apple Pay
│   └── confirmPurchase (mutation) - Verify and record purchase
├── creator
│   ├── getProfile (query) - Creator stats
│   ├── myShortcuts (query) - Creator's submissions
│   └── submitShortcut (mutation) - Submit new shortcut
└── admin
    ├── pendingShortcuts (query) - Submissions awaiting review
    ├── approveShortcut (mutation) - Approve submission
```

```
    ├── rejectShortcut (mutation) - Reject submission
    └── toggleFeatured (mutation) - Mark as featured
```

Example API Implementation

File: `server/routers.ts`

TypeScript

```typescript
import { router, publicProcedure, protectedProcedure } from "./_core/trpc";
import { z } from "zod";
import * as db from "./db";
import * as stripe from "./stripe";

export const appRouter = router({
  shortcuts: router({
    list: publicProcedure
      .input(z.object({
        category: z.string().optional(),
        featured: z.boolean().optional(),
        limit: z.number().optional(),
      }).optional())
      .query(async ({ input }) => {
        return await db.getShortcuts({
          ...input,
          status: "approved",
        });
      }),

    getBySlug: publicProcedure
      .input(z.object({ slug: z.string() }))
      .query(async ({ input }) => {
        return await db.getShortcutBySlug(input.slug);
      }),
  }),

  payment: router({
    createCheckoutSession: protectedProcedure
      .input(z.object({ shortcutId: z.number() }))
      .mutation(async ({ ctx, input }) => {
        const shortcut = await db.getShortcutById(input.shortcutId);
        if (!shortcut) throw new Error("Shortcut not found");

        const session = await stripe.createCheckoutSession({
          priceAmount: shortcut.price,
          currency: "usd",
          shortcutId: shortcut.id,
```

```
          shortcutTitle: shortcut.title,
          successUrl: `${process.env.APP_URL}/shortcut/${shortcut.slug}?
payment=success`,
          cancelUrl: `${process.env.APP_URL}/shortcut/${shortcut.slug}`,
          customerEmail: ctx.user.email,
        });

        return { sessionId: session.id, url: session.url };
      }),
  }),
});

export type AppRouter = typeof appRouter;
```

Database Query Layer

File: `server/db.ts`

TypeScript

```
import { drizzle } from "drizzle-orm/mysql2";
import { eq } from "drizzle-orm";
import { shortcuts, users, purchases } from "../drizzle/schema";

const db = drizzle(process.env.DATABASE_URL!);

export async function getShortcuts(filters?: {
  category?: string;
  featured?: boolean;
  status?: "pending" | "approved" | "rejected";
  limit?: number;
}) {
  let query = db.select().from(shortcuts);

  if (filters?.category) {
    query = query.where(eq(shortcuts.category, filters.category));
  }
  if (filters?.featured) {
    query = query.where(eq(shortcuts.featured, 1));
  }
  if (filters?.status) {
    query = query.where(eq(shortcuts.status, filters.status));
  }
  if (filters?.limit) {
    query = query.limit(filters.limit);
  }
```

```
  return await query;
}

export async function createPurchase(purchase: {
  userId: number;
  shortcutId: number;
  price: number;
  stripePaymentIntentId: string;
}) {
  await db.insert(purchases).values({
    ...purchase,
    status: "completed",
  });
}
```

Frontend Architecture

Component Hierarchy

Plain Text

```
App
├── Router
│   ├── Home (Landing Page)
│   │   ├── Hero Section
│   │   ├── Category Grid
│   │   ├── Featured Shortcuts
│   │   └── Trending Section
│   ├── ShortcutDetail
│   │   ├── Preview Media
│   │   ├── Description
│   │   └── Purchase CTA
│   ├── Checkout
│   │   ├── Order Summary
│   │   └── Stripe Payment Form
│   ├── CreatorDashboard
│   │   ├── Stats Cards
│   │   ├── Submit Form
│   │   └── My Shortcuts List
│   ├── AdminDashboard
│   │   └── Pending Submissions
│   └── Library
│       └── Purchased Shortcuts Grid
└── Providers
    ├── ThemeProvider
```

```
    ├── tRPC Provider
    └── TooltipProvider
```

State Management Pattern

tRPC React Query Integration

TypeScript

```typescript
// Client setup
import { createTRPCReact } from "@trpc/react-query";
import type { AppRouter } from "../../../server/routers";

export const trpc = createTRPCReact<AppRouter>();

// Usage in components
function ShortcutList() {
  const { data, isLoading } = trpc.shortcuts.list.useQuery({
    featured: true,
    limit: 6,
  });

  const downloadMutation = trpc.shortcuts.download.useMutation();

  const handleDownload = async (id: number) => {
    await downloadMutation.mutateAsync({ id });
  };

  return (
    <div>
      {data?.map(shortcut => (
        <ShortcutCard key={shortcut.id} {...shortcut} />
      ))}
    </div>
  );
}
```

Routing Configuration

File: `client/src/App.tsx`

TypeScript

```typescript
import { Route, Switch } from "wouter";

function Router() {
```

```
  return (
    <Switch>
      <Route path="/" component={Home} />
      <Route path="/shortcut/:slug" component={ShortcutDetail} />
      <Route path="/checkout/:slug" component={Checkout} />
      <Route path="/creator" component={CreatorDashboard} />
      <Route path="/admin" component={AdminDashboard} />
      <Route path="/library" component={Library} />
      <Route component={NotFound} />
    </Switch>
  );
}
```

Design System

Tailwind Configuration

TypeScript

```
// tailwind.config.ts
export default {
  theme: {
    extend: {
      colors: {
        background: "oklch(var(--background))",
        foreground: "oklch(var(--foreground))",
        primary: "oklch(var(--primary))",
        // ... more colors
      },
      fontFamily: {
        sans: ["Inter", "system-ui", "sans-serif"],
      },
    },
  },
};
```

CSS Variables (client/src/index.css)

CSS

```
:root {
  --background: 100% 0 0;
  --foreground: 10% 0 0;
  --primary: 60% 0.2 270;  /* Purple */
  --radius: 0.5rem;
}
```

Payment Integration

Stripe Checkout Flow

Plain Text

```
User clicks "Buy"
       ↓
Create Checkout Session (Backend)
       ↓
Redirect to Stripe Checkout
       ↓
User completes payment (Apple Pay / Card)
       ↓
Stripe redirects to success URL
       ↓
Verify payment (Backend)
       ↓
Record purchase in database
       ↓
Add to user library
       ↓
Provide download link
```

Implementation

Stripe Helper (`server/stripe.ts`)

TypeScript

```typescript
import Stripe from "stripe";

const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, {
  apiVersion: "2024-12-18.acacia",
});

export async function createCheckoutSession(params: {
  priceAmount: number;
  currency: string;
  shortcutId: number;
  shortcutTitle: string;
  successUrl: string;
  cancelUrl: string;
}) {
```

```
  return await stripe.checkout.sessions.create({
    payment_method_types: ["card"],
    line_items: [{
      price_data: {
        currency: params.currency,
        product_data: {
          name: params.shortcutTitle,
          description: "iOS Shortcut - Instant Download",
        },
        unit_amount: params.priceAmount,
      },
      quantity: 1,
    }],
    mode: "payment",
    success_url: params.successUrl,
    cancel_url: params.cancelUrl,
    metadata: {
      shortcutId: params.shortcutId.toString(),
    },
  });
}
```

Frontend Integration

TypeScript

```
import { loadStripe } from "@stripe/stripe-js";

const stripePromise =
loadStripe(import.meta.env.VITE_STRIPE_PUBLISHABLE_KEY);

function CheckoutButton({ shortcutId }: { shortcutId: number }) {
  const checkoutMutation = trpc.payment.createCheckoutSession.useMutation();

  const handleCheckout = async () => {
    const { url } = await checkoutMutation.mutateAsync({ shortcutId });
    if (url) window.location.href = url;
  };

  return (
    <button onClick={handleCheckout}>
      Buy with Apple Pay
    </button>
  );
}
```

Apple Pay Configuration

1. **Stripe Dashboard**: Enable Apple Pay in payment methods
2. **Domain Verification**: Add domain to Apple Pay settings
3. **Automatic**: Stripe handles Apple Pay display based on device

---

Authentication & Authorization

JWT-Based Sessions

TypeScript

```typescript
// Session creation (after OAuth)
const token = jwt.sign(
  { userId: user.id, openId: user.openId },
  process.env.JWT_SECRET!,
  { expiresIn: "7d" }
);

res.cookie("session", token, {
  httpOnly: true,
  secure: process.env.NODE_ENV === "production",
  sameSite: "lax",
  maxAge: 7 * 24 * 60 * 60 * 1000, // 7 days
});
```

Authorization Middleware

TypeScript

```typescript
// Protected procedure
export const protectedProcedure = publicProcedure.use(async ({ ctx, next }) => {
  if (!ctx.user) {
    throw new TRPCError({ code: "UNAUTHORIZED" });
  }
  return next({ ctx: { ...ctx, user: ctx.user } });
});

// Admin procedure
export const adminProcedure = protectedProcedure.use(async ({ ctx, next }) => {
  if (ctx.user.role !== "admin") {
```

```
    throw new TRPCError({ code: "FORBIDDEN" });
  }
  return next({ ctx });
});
```

Role-Based Access Control

TypeScript

```
enum UserRole {
  USER = "user",      // Can browse and purchase
  CREATOR = "creator", // Can submit shortcuts
  ADMIN = "admin",    // Full access
}

// Frontend check
function AdminPanel() {
  const { user } = useAuth();

  if (user?.role !== "admin") {
    return <AccessDenied />;
  }

  return <AdminDashboard />;
}
```

File Structure

Plain Text

```
taptask/
├── client/                    # Frontend application
│   ├── public/                # Static assets
│   ├── src/
│   │   ├── pages/             # Route components
│   │   │   ├── Home.tsx
│   │   │   ├── ShortcutDetail.tsx
│   │   │   ├── Checkout.tsx
│   │   │   ├── CreatorDashboard.tsx
│   │   │   ├── AdminDashboard.tsx
│   │   │   └── Library.tsx
│   │   ├── components/        # Reusable components
│   │   │   ├── ui/            # shadcn/ui components
│   │   │   └── ...
```

```
│   │   ├── hooks/              # Custom React hooks
│   │   ├── lib/                # Utilities
│   │   │   ├── trpc.ts         # tRPC client
│   │   │   └── utils.ts
│   │   ├── contexts/           # React contexts
│   │   ├── App.tsx             # Root component
│   │   ├── main.tsx            # Entry point
│   │   └── index.css           # Global styles
│   └── index.html
├── server/                       # Backend application
│   ├── _core/                   # Core functionality
│   │   ├── trpc.ts             # tRPC setup
│   │   ├── context.ts          # Request context
│   │   ├── oauth.ts            # Authentication
│   │   └── index.ts            # Express server
│   ├── routers.ts              # API routes
│   ├── db.ts                   # Database queries
│   ├── stripe.ts               # Payment integration
│   └── storage.ts              # File storage (optional)
├── drizzle/                      # Database
│   ├── schema.ts               # Table definitions
│   ├── migrations/             # SQL migrations
│   └── relations.ts            # Foreign keys
├── shared/                       # Shared code
│   ├── types.ts                # TypeScript types
│   └── const.ts                # Constants
├── scripts/                      # Utility scripts
│   └── seed.ts                 # Database seeding
├── package.json
├── tsconfig.json
├── vite.config.ts
├── drizzle.config.ts
└── .env                         # Environment variables
```

Key Features Implementation

1. Marketplace Browse & Search

Backend Query

TypeScript

```typescript
export async function getShortcuts(filters: {
  category?: string;
  search?: string;
  featured?: boolean;
```

```
  trending?: boolean;
  status?: string;
  limit?: number;
}) {
  let query = db.select().from(shortcuts);

  const conditions = [];

  if (filters.category) {
    conditions.push(eq(shortcuts.category, filters.category));
  }

  if (filters.search) {
    conditions.push(
      or(
        like(shortcuts.title, `%${filters.search}%`),
        like(shortcuts.description, `%${filters.search}%`)
      )
    );
  }

  if (filters.featured) {
    conditions.push(eq(shortcuts.featured, 1));
  }

  if (filters.status) {
    conditions.push(eq(shortcuts.status, filters.status));
  }

  if (conditions.length > 0) {
    query = query.where(and(...conditions));
  }

  if (filters.limit) {
    query = query.limit(filters.limit);
  }

  return await query;
}
```

Frontend Component

TypeScript

```
function MarketplaceBrowse() {
  const [category, setCategory] = useState<string>();

  const { data: shortcuts, isLoading } = trpc.shortcuts.list.useQuery({
```

```
    category,
    status: "approved",
  });

  return (
    <div>
      <CategoryFilter onChange={setCategory} />
      <ShortcutGrid shortcuts={shortcuts} loading={isLoading} />
    </div>
  );
}
```

2. Purchase Flow

Step 1: Create Checkout Session

TypeScript

```typescript
const checkoutMutation = trpc.payment.createCheckoutSession.useMutation();

const handlePurchase = async (shortcutId: number) => {
  const { url } = await checkoutMutation.mutateAsync({ shortcutId });
  window.location.href = url; // Redirect to Stripe
};
```

Step 2: Handle Success Callback

TypeScript

```typescript
// After Stripe redirects back
function ShortcutDetail() {
  const [searchParams] = useSearchParams();
  const paymentStatus = searchParams.get("payment");

  useEffect(() => {
    if (paymentStatus === "success") {
      toast.success("Purchase successful! Download link available.");
      // Shortcut is now in user's library
    }
  }, [paymentStatus]);
}
```

Step 3: Record Purchase

TypeScript

```
// Stripe webhook handler (optional for instant verification)
app.post("/api/webhooks/stripe", async (req, res) => {
  const sig = req.headers["stripe-signature"];
  const event = stripe.webhooks.constructEvent(
    req.body,
    sig,
    process.env.STRIPE_WEBHOOK_SECRET
  );

  if (event.type === "checkout.session.completed") {
    const session = event.data.object;
    const shortcutId = session.metadata.shortcutId;

    await db.createPurchase({
      userId: session.customer,
      shortcutId: parseInt(shortcutId),
      price: session.amount_total,
      stripePaymentIntentId: session.payment_intent,
      status: "completed",
    });
  }

  res.json({ received: true });
});
```

3. Creator Dashboard

Submit Shortcut Form

TypeScript

```
function SubmitShortcutForm() {
  const submitMutation = trpc.creator.submitShortcut.useMutation();

  const onSubmit = async (data: FormData) => {
    await submitMutation.mutateAsync({
      title: data.title,
      description: data.description,
      category: data.category,
      tags: data.tags.split(","),
      price: parseFloat(data.price) * 100, // Convert to cents
      iCloudLink: data.iCloudLink,
      previewImage: data.previewImage,
      requiredIOSVersion: data.requiredIOSVersion,
    });

    toast.success("Shortcut submitted for review!");
```

```
  };

  return <form onSubmit={handleSubmit(onSubmit)}>...</form>;
}
```

Creator Stats

TypeScript

```
function CreatorStats() {
  const { data: profile } = trpc.creator.getProfile.useQuery();

  return (
    <div className="grid grid-cols-4 gap-4">
      <StatCard
        label="Total Earnings"
        value={`$${(profile?.totalEarnings || 0) / 100}`}
      />
      <StatCard
        label="Shortcuts Submitted"
        value={profile?.shortcutsSubmitted || 0}
      />
      <StatCard
        label="Shortcuts Approved"
        value={profile?.shortcutsApproved || 0}
      />
      <StatCard
        label="Total Sales"
        value={profile?.shortcutsSold || 0}
      />
    </div>
  );
}
```

4. Admin Moderation

Pending Submissions

TypeScript

```
function AdminModeration() {
  const { data: pending, refetch } = trpc.admin.pendingShortcuts.useQuery();
  const approveMutation = trpc.admin.approveShortcut.useMutation();
  const rejectMutation = trpc.admin.rejectShortcut.useMutation();

  const handleApprove = async (id: number) => {
    await approveMutation.mutateAsync({ id });
```

```
    toast.success("Shortcut approved!");
    refetch();
  };

  const handleReject = async (id: number) => {
    await rejectMutation.mutateAsync({ id });
    toast.success("Shortcut rejected");
    refetch();
  };

  return (
    <div>
      {pending?.map(shortcut => (
        <ShortcutReviewCard
          key={shortcut.id}
          shortcut={shortcut}
          onApprove={() => handleApprove(shortcut.id)}
          onReject={() => handleReject(shortcut.id)}
        />
      ))}
    </div>
  );
}
```

5. User Library

Track Purchases

TypeScript

```
// Add to library after purchase
export async function addToUserLibrary(userId: number, shortcutId: number) {
  const user = await db.select().from(users).where(eq(users.id,
userId)).limit(1);

  if (user.length > 0) {
    const library = user[0].library ? JSON.parse(user[0].library) : [];

    if (!library.includes(shortcutId)) {
      library.push(shortcutId);
      await db.update(users)
        .set({ library: JSON.stringify(library) })
        .where(eq(users.id, userId));
    }
  }
}
```

Display Library

TypeScript

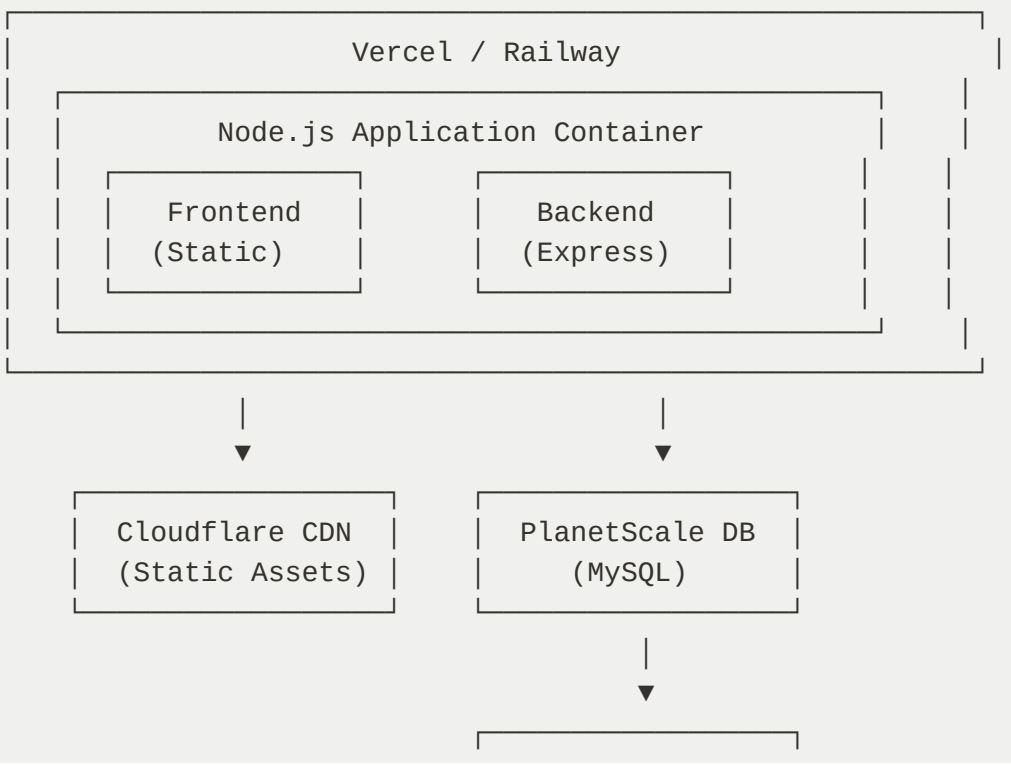
```
function UserLibrary() {
  const { data: library, isLoading } = trpc.shortcuts.myLibrary.useQuery();

  return (
    <div className="grid grid-cols-3 gap-6">
      {library?.map(shortcut => (
        <ShortcutCard
          key={shortcut.id}
          shortcut={shortcut}
          owned={true}
        />
      ))}
    </div>
  );
}
```

Deployment Architecture

Production Environment

Plain Text



```
                                    │   Stripe API     │
                                    │   (Payments)     │
                                    └──────────────────┘
```

Deployment Steps

1. Build Configuration

JSON

```json
// package.json
{
  "scripts": {
    "build": "vite build && tsc --project tsconfig.server.json",
    "start": "NODE_ENV=production node dist/server/_core/index.js"
  }
}
```

2. Environment Variables (Production)

Plain Text

```
NODE_ENV=production
DATABASE_URL=mysql://user:pass@db.planetscale.com/taptask
STRIPE_SECRET_KEY=sk_live_...
VITE_STRIPE_PUBLISHABLE_KEY=pk_live_...
JWT_SECRET=<strong-random-secret>
APP_URL=https://taptask.com
```

3. Database Migration

Bash

```bash
# Generate migration
pnpm drizzle-kit generate

# Apply to production
pnpm drizzle-kit migrate
```

4. Deploy to Vercel

Bash

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

```
# Deploy
vercel --prod
```

5. Configure Domain

- Point domain to Vercel
- Enable SSL (automatic)
- Configure Stripe webhook URL

Security Considerations

1. Authentication Security

- **JWT Tokens**: HttpOnly cookies prevent XSS attacks
- **CSRF Protection**: SameSite cookie attribute
- **Token Expiration**: 7-day expiry with refresh mechanism
- **Password Hashing**: Not applicable (OAuth-based)

2. Payment Security

- **PCI Compliance**: Stripe handles card data (no storage)
- **Webhook Verification**: Validate Stripe signatures
- **Amount Verification**: Server-side price validation
- **Idempotency**: Prevent duplicate charges

TypeScript

```
// Verify webhook signature
const sig = req.headers["stripe-signature"];
const event = stripe.webhooks.constructEvent(
  req.body,
  sig,
  process.env.STRIPE_WEBHOOK_SECRET
);
```

3. API Security

- **Rate Limiting**: Prevent abuse
- **Input Validation**: Zod schemas on all inputs

- **SQL Injection**: Drizzle ORM parameterized queries
- **Authorization**: Role-based access control

TypeScript

```typescript
// Input validation
const schema = z.object({
  title: z.string().min(1).max(255),
  price: z.number().min(0).max(9999),
  iCloudLink: z.string().url(),
});
```

4. Data Protection

- **Sensitive Data**: Never log credit cards or tokens
- **Database Encryption**: Enable at-rest encryption
- **HTTPS Only**: Enforce SSL in production
- **CORS**: Restrict origins

Performance Optimization

1. Database Optimization

SQL

```sql
-- Composite indexes for common queries
CREATE INDEX idx_shortcuts_status_featured ON shortcuts(status, featured);
CREATE INDEX idx_shortcuts_status_category ON shortcuts(status, category);

-- Analyze query performance
EXPLAIN SELECT * FROM shortcuts WHERE status = 'approved' AND featured = 1;
```

2. Frontend Optimization

Code Splitting

TypeScript

```
// Lazy load routes
const AdminDashboard = lazy(() => import("./pages/AdminDashboard"));
const CreatorDashboard = lazy(() => import("./pages/CreatorDashboard"));
```

Image Optimization

TypeScript

```
// Use next-gen formats
<img
  src={shortcut.previewImage}
  loading="lazy"
  decoding="async"
  alt={shortcut.title}
/>
```

React Query Caching

TypeScript

```
// Configure cache times
const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      staleTime: 5 * 60 * 1000, // 5 minutes
      cacheTime: 10 * 60 * 1000, // 10 minutes
    },
  },
});
```

3. API Optimization

Batch Queries

TypeScript

```
// Fetch related data in single query
const shortcutsWithCreators = await db
  .select()
  .from(shortcuts)
  .leftJoin(users, eq(shortcuts.creatorId, users.id));
```

Pagination

TypeScript

```
export async function getShortcuts(page: number, limit: number = 20) {
  const offset = (page - 1) * limit;
  return await db
    .select()
    .from(shortcuts)
    .limit(limit)
    .offset(offset);
}
```

Future Enhancements

Phase 2 Features

1. **Search & Filters**
   - Full-text search with Algolia or Meilisearch
   - Advanced filtering (price range, iOS version, ratings)
   - Sort by popularity, date, price
2. **Reviews & Ratings**
   - 5-star rating system
   - Written reviews with moderation
   - Creator response to reviews
3. **Collections & Bundles**
   - Curated shortcut collections
   - Bundle pricing (buy multiple shortcuts)
   - Themed collections (Productivity Pack, etc.)
4. **Creator Analytics**
   - Sales charts and trends
   - Geographic data
   - Conversion rates
5. **Social Features**
   - Follow favorite creators
   - Share shortcuts on social media
   - Wishlist functionality

Phase 3: iOS Native App

Architecture for iOS

Plain Text

```
TapTask iOS App (React Native / Swift)
├── Shared Business Logic (60-70% reuse)
│   ├── API Client (tRPC)
│   ├── Data Models
│   └── Business Rules
├── Native UI (iOS-specific)
│   ├── SwiftUI Views
│   ├── UIKit Components
│   └── Native Navigation
└── Platform Features
    ├── Apple Pay Integration
    ├── Shortcuts App Integration
    ├── iCloud Sync
    └── Push Notifications
```

Implementation Options

1. **React Native** (Faster, code reuse)
   - Reuse tRPC client
   - Share business logic
   - Native modules for Shortcuts integration
2. **Swift Native** (Better performance)
   - Generate Swift types from tRPC
   - Implement native UI
   - Full iOS ecosystem integration

Appendix: Renaming from AutoFlow to TapTask

Global Find & Replace

Files to Update:

1. **Environment Variables**

Plain Text

```
VITE_APP_TITLE="TapTask"
VITE_APP_LOGO="https://taptask.com/logo.png"
```

1. **Package.json**

```json
{
  "name": "taptask",
  "description": "The marketplace for iPhone automations"
}
```

1. **Frontend Constants** (`shared/const.ts`)

```typescript
export const APP_TITLE = "TapTask";
export const APP_DESCRIPTION = "The App Store for iPhone Automations";
```

1. **Database Seed Data** (`scripts/seed.ts`)

```typescript
// Update creator names, descriptions, etc.
```

1. **README Files**

```markdown
# TapTask - iOS Shortcuts Marketplace
```

Brand Assets Needed

- Logo (SVG, PNG in multiple sizes)
- Favicon
- App icons (iOS sizes)
- Social media preview image
- Email templates header

Conclusion

This architecture provides a solid foundation for TapTask, a production-ready iOS Shortcuts marketplace. The tech stack is modern, type-safe, and scalable. The modular design allows for easy feature additions and platform expansion.

Key Strengths:

- **Type Safety**: End-to-end TypeScript with tRPC
- **Performance**: Optimized queries and caching
- **Security**: Industry-standard authentication and payment handling
- **Scalability**: Horizontal scaling with stateless architecture
- **Developer Experience**: Hot reload, type inference, excellent tooling

Next Steps for Your Developer:

1. Review this document thoroughly
2. Set up local development environment
3. Familiarize with tRPC and Drizzle ORM
4. Start with database schema and migrations
5. Build API layer with tRPC routers
6. Implement frontend pages
7. Integrate Stripe payments
8. Test thoroughly
9. Deploy to staging
10. Launch to production

Document Version: 1.0

Last Updated: October 25, 2025

Prepared for: TapTask Development Team