HelpMeClean.ro — MVP Technical Specification

1. Product Overview

HelpMeClean.ro is Romania's first "Uber for cleaning" — a two-sided marketplace connecting clients who need home cleaning with verified cleaning companies and their cleaners. This MVP is a **functional prototype** for investor demonstration purposes.

Core value proposition: Formalize Romania's informal cleaning sector by making compliance the path of least resistance — digital payments, automatic invoicing, company verification, and transparent operations.

1.1 User Roles

Role	Description	Platforms
Client	Books and pays for cleaning services	iOS (SwiftUI native), Web (React)
Company Admin	Cleaning company owner/manager who applies to join, manages team	Web (React + Shadcn), Mobile (React Native) — *web is primary*
Cleaner	Performs cleaning jobs, receives assignments	Mobile (React Native — iOS + Android)
Global Admin	Platform operator — approves companies, monitors platform	Web (React + Shadcn), Mobile (React Native) — *web is primary*

> **Important:** A Company Admin can also be a Cleaner (PFA / one-man operation). The system must support this dual-role scenario where the admin invites themselves as a cleaner.

---

2. Architecture & Tech Stack

2.1 Backend — Go Monolith

```
┌──────────────────────────────────────────────────────────┐
│              Go Monolith (Single Binary)                 │
│                                                          │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────────┐  │
│  │ GraphQL      │  │ WebSocket    │  │ Background       │  │
│  │  API         │  │  Server      │  │  Workers         │  │
│  │ (gqlgen)     │  │  (Chat)      │  │  (Jobs/Notif)    │  │
│  └──────┬───────┘  └──────┬───────┘  └────────┬─────────┘  │
│         │                 │                   │            │
│  ┌──────┴─────────────────┴───────────────────┴───────┐    │
│  │              Service Layer                         │    │
│  │  Auth | Booking | Chat | Matching | Billing        │    │
│  └──────────────────────┬──────────────────────┘  │
│           │              │
│  ┌──────────────────────┴──────────────────────┐  │
│  │      Data Layer (sqlc / pgx)      │  │
│  └──────────────────────┬──────────────────────┘  │
│           │              │
└────────────────────────────┬──────────────────────────┘
          │
     ┌──────┴──────┐
     │ PostgreSQL │
     │  (Neon)  │
     └─────────────┘
```

Technology choices:

Component	Technology	Rationale
Language	**Go 1.22+**	Performance, single binary deployment, strong concurrency
GraphQL	**gqlgen**	Type-safe, code-first GraphQL for Go
Database	**PostgreSQL (Neon)**	Serverless Postgres, connection pooling included
DB Access	**sqlc**	Type-safe SQL → Go code generation
Migrations	**golang-migrate**	Version-controlled schema migrations
Auth	**Google OAuth 2.0 + JWT**	Simple auth, no password management
Real-time	**gorilla/websocket**	WebSocket for chat and live job updates
File Storage	**Local filesystem / GCS bucket**	Document uploads (company docs, profile photos)
Payments	**Stripe (mocked for MVP)**	Payment intents, saved cards — mock mode
Container	**Docker**	Single image, deploy to GCP Cloud Run
Push Notifications	**Firebase Cloud Messaging (FCM)**	Cross-platform push (iOS + Android)

2.2 Frontend Applications

App	Technology	Key Libraries
Client iOS	SwiftUI (Native)	Apollo GraphQL iOS, Liquid Glass design, MapKit
Client Web	React + TypeScript	Apollo Client, Shadcn/ui, TailwindCSS
Company Dashboard (Web)	React + TypeScript	Apollo Client, Shadcn/ui, TailwindCSS, Recharts

App	Technology	Key Libraries
Company Mobile	React Native (Expo)	Apollo Client, NativeWind
Cleaner Mobile	React Native (Expo)	Apollo Client, NativeWind, react-native-maps
Global Admin (Web)	React + TypeScript	Apollo Client, Shadcn/ui, TailwindCSS, Recharts

> **Design philosophy:** Everything must look polished and modern. The investor partner is non-technical — visual quality is paramount. Use Shadcn/ui components for all web interfaces. Use SwiftUI Liquid Glass effects for iOS. React Native apps should follow platform-native conventions with clean, modern styling.

2.3 Deployment

```
┌──────────────────────────────────────────────┐
│              GCP                             │
│                                              │
│  ┌──────────────┐  ┌──────────────────────┐  │
│  │ Cloud Run    │  │ Cloud Storage (GCS)  │  │
│  │ (Go Docker)  │  │ (uploads, documents) │  │
│  └──────┬───────┘  └──────────────────────┘  │
│         │                                    │
│  ┌──────┴───────┐                            │
│  │ Neon DB      │                            │
│  │ (Postgres)   │                            │
│  └──────────────┘                            │
│                                              │
│  ┌──────────────┐  ┌──────────────────────┐  │
│  │ Firebase     │  │ Vercel / Cloudflare  │  │
│  │ (FCM Push)   │  │ (Web frontends)      │  │
│  └──────────────┘  └──────────────────────┘  │
└──────────────────────────────────────────────┘
```

- **Backend:** Single Docker image → GCP Cloud Run (auto-scaling, HTTPS)
- **Web apps:** Deploy to Vercel or Cloudflare Pages
- **iOS app:** TestFlight for demo
- **Android:** Internal testing track on Play Store or direct APK
- **Database:** Neon serverless PostgreSQL (connection string provided)

2.4 Monorepo Structure

```
helpmeclean/
├── backend/              # Go monolith
│   ├── cmd/
│   │   └── server/
│   │       └── main.go        # Entry point
│   ├── internal/
│   │   ├── auth/              # Google OAuth, JWT, middleware
│   │   ├── graph/             # gqlgen resolvers + schema
│   │   │   ├── schema/        # .graphql schema files
│   │   │   ├── model/         # Generated + custom models
│   │   │   └── resolver/      # Resolver implementations
│   │   ├── service/           # Business logic layer
│   │   │   ├── booking/
│   │   │   ├── company/
│   │   │   ├── cleaner/
│   │   │   ├── chat/
│   │   │   ├── matching/
│   │   │   ├── payment/
│   │   │   ├── notification/
│   │   │   ├── review/
│   │   │   └── admin/
│   │   ├── db/                # sqlc queries + migrations
│   │   │   ├── migrations/
│   │   │   ├── queries/       # .sql files for sqlc
│   │   │   └── sqlc.yaml
│   │   ├── middleware/        # CORS, auth, logging
│   │   ├── ws/                # WebSocket hub for chat + live updates
│   │   └── storage/           # File upload handling (GCS or local)
│   ├── Dockerfile
│   ├── go.mod
│   └── go.sum
│
├── web/                       # All web frontends (React monorepo)
│   ├── packages/
│   │   ├── shared/            # Shared components, hooks, GraphQL codegen
│   │   │   ├── graphql/       # Shared .graphql operations + codegen
│   │   │   ├── components/    # Shared UI components
│   │   │   └── hooks/         # Shared React hooks
│   │   ├── client-web/        # Client-facing web app
│   │   ├── company-dashboard/ # Company admin dashboard
│   │   └── admin-dashboard/   # Global admin dashboard
│   ├── package.json
│   └── turbo.json             # Turborepo config
│
├── mobile/                    # React Native (Expo) apps
│   ├── packages/
│   │   ├── shared/            # Shared RN components, hooks, GraphQL
│   │   ├── cleaner-app/       # Cleaner mobile app
│   │   └── company-app/       # Company admin mobile app (if built)
│   ├── package.json
│   └── turbo.json
```

```
│
├── ios/                    # Native SwiftUI client app
│   ├── HelpMeClean/
│   │   ├── App/
│   │   ├── Features/
│   │   │   ├── Booking/
│   │   │   ├── Auth/
│   │   │   ├── Profile/
│   │   │   ├── Jobs/
│   │   │   ├── Chat/
│   │   │   └── Reviews/
│   │   ├── Shared/
│   │   │   ├── Components/
│   │   │   ├── GraphQL/       # Apollo iOS generated code
│   │   │   ├── Theme/         # Liquid Glass styling
│   │   │   └── Extensions/
│   │   └── Resources/
│   └── HelpMeClean.xcodeproj
│
├── docs/                    # Documentation
│   └── MVP-SPEC.md           # This file
│
└── docker-compose.yml         # Local development setup
```

---

3. Database Schema

3.1 Core Tables

```
sql
```

```sql
-- ============================================
-- USERS & AUTH
-- ============================================

CREATE TYPE user_role AS ENUM ('client', 'company_admin', 'cleaner', 'global_admin');
CREATE TYPE user_status AS ENUM ('active', 'inactive', 'suspended', 'pending');

CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    full_name VARCHAR(255) NOT NULL,
    phone VARCHAR(50),
    avatar_url TEXT,
    role user_role NOT NULL,
    status user_status NOT NULL DEFAULT 'active',
    google_id VARCHAR(255) UNIQUE,
    fcm_token TEXT,                    -- Firebase push token
    preferred_language VARCHAR(5) DEFAULT 'ro',
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- ============================================
-- CLIENT-SPECIFIC
-- ============================================

CREATE TABLE client_addresses (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(id),
    label VARCHAR(100),                -- "Home", "Office", etc.
    street_address TEXT NOT NULL,
    city VARCHAR(100) NOT NULL,
    county VARCHAR(100) NOT NULL,      -- județ
    postal_code VARCHAR(20),
    floor VARCHAR(20),
    apartment VARCHAR(20),
    entry_code VARCHAR(50),            -- cod interfon
    latitude DOUBLE PRECISION,
    longitude DOUBLE PRECISION,
    notes TEXT,                        -- special instructions
    is_default BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE TABLE client_payment_methods (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(id),
    stripe_payment_method_id VARCHAR(255),  -- mocked for MVP
    card_last_four VARCHAR(4),
    card_brand VARCHAR(50),           -- visa, mastercard
    is_default BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- ============================================
-- COMPANIES
-- ============================================

CREATE TYPE company_status AS ENUM ('pending_review', 'approved', 'rejected', 'suspended');
CREATE TYPE company_type AS ENUM ('srl', 'pfa', 'ii');  -- Romanian business types

CREATE TABLE companies (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    admin_user_id UUID REFERENCES users(id),  -- NULL until they sign up
    company_name VARCHAR(255) NOT NULL,
    cui VARCHAR(20) UNIQUE NOT NULL,          -- Cod Unic de Înregistrare
    company_type company_type NOT NULL,
    legal_representative VARCHAR(255) NOT NULL,
    contact_email VARCHAR(255) NOT NULL,
    contact_phone VARCHAR(50) NOT NULL,
    address TEXT NOT NULL,
    city VARCHAR(100) NOT NULL,
    county VARCHAR(100) NOT NULL,
    description TEXT,
    logo_url TEXT,
    status company_status NOT NULL DEFAULT 'pending_review',
    rejection_reason TEXT,
    max_service_radius_km INTEGER DEFAULT 20,  -- how far they'll travel
    rating_avg DECIMAL(3,2) DEFAULT 0.00,
    total_jobs_completed INTEGER DEFAULT 0,
    approved_at TIMESTAMPTZ,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE TABLE company_documents (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    company_id UUID NOT NULL REFERENCES companies(id),
    document_type VARCHAR(100) NOT NULL,    -- 'certificat_constatator', 'asigurare', 'cui', etc.
    file_url TEXT NOT NULL,
    file_name VARCHAR(255) NOT NULL,
    uploaded_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

```sql
-- ============================================
-- CLEANERS
-- ============================================

CREATE TYPE cleaner_status AS ENUM ('invited', 'active', 'inactive', 'suspended');

CREATE TABLE cleaners (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id),        -- NULL until they accept invite
    company_id UUID NOT NULL REFERENCES companies(id),
    full_name VARCHAR(255) NOT NULL,
    phone VARCHAR(50),
    email VARCHAR(255),
    avatar_url TEXT,
    status cleaner_status NOT NULL DEFAULT 'invited',
    is_company_admin BOOLEAN DEFAULT FALSE,   -- true if admin added themselves
    invite_token VARCHAR(255) UNIQUE,
    invite_expires_at TIMESTAMPTZ,
    rating_avg DECIMAL(3,2) DEFAULT 0.00,
    total_jobs_completed INTEGER DEFAULT 0,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Cleaner availability schedule
CREATE TABLE cleaner_availability (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    cleaner_id UUID NOT NULL REFERENCES cleaners(id),
    day_of_week INTEGER NOT NULL CHECK (day_of_week BETWEEN 0 AND 6), -- 0=Monday
    start_time TIME NOT NULL,
    end_time TIME NOT NULL,
    is_available BOOLEAN DEFAULT TRUE
);

-- ============================================
-- SERVICES & PRICING
-- ============================================

CREATE TYPE service_type AS ENUM (
    'standard_cleaning',
    'deep_cleaning',
    'move_in_out_cleaning',
    'post_construction',
    'office_cleaning',
    'window_cleaning'
);
```

```sql
CREATE TABLE service_definitions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    service_type service_type NOT NULL,
    name_ro VARCHAR(255) NOT NULL,          -- Romanian name
    name_en VARCHAR(255) NOT NULL,          -- English name
    description_ro TEXT,
    description_en TEXT,
    base_price_per_hour DECIMAL(10,2) NOT NULL,  -- in RON
    min_hours DECIMAL(3,1) NOT NULL DEFAULT 2.0,
    icon VARCHAR(50),                       -- icon identifier
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE TABLE service_extras (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name_ro VARCHAR(255) NOT NULL,
    name_en VARCHAR(255) NOT NULL,
    price DECIMAL(10,2) NOT NULL,           -- flat fee in RON
    icon VARCHAR(50),
    is_active BOOLEAN DEFAULT TRUE
);

-- ============================================
-- BOOKINGS / JOBS
-- ============================================

CREATE TYPE booking_status AS ENUM (
    'pending',            -- client submitted, awaiting assignment
    'assigned',           -- cleaner assigned
    'confirmed',          -- cleaner confirmed
    'in_progress',        -- cleaner started
    'completed',          -- cleaner finished
    'cancelled_by_client',
    'cancelled_by_company',
    'cancelled_by_admin'
);

CREATE TABLE bookings (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    reference_code VARCHAR(20) UNIQUE NOT NULL,   -- human-readable: HMC-XXXX
    client_user_id UUID NOT NULL REFERENCES users(id),
    company_id UUID REFERENCES companies(id),     -- assigned company
    cleaner_id UUID REFERENCES cleaners(id),      -- assigned cleaner
    address_id UUID NOT NULL REFERENCES client_addresses(id),
    -- Service details
    service_type service_type NOT NULL,
    scheduled_date DATE NOT NULL,
    scheduled_start_time TIME NOT NULL,
    estimated_duration_hours DECIMAL(3,1) NOT NULL,

    -- Property details
    property_type VARCHAR(50),                    -- apartment, house, office
    num_rooms INTEGER,
    num_bathrooms INTEGER,
    area_sqm INTEGER,
    has_pets BOOLEAN DEFAULT FALSE,
    special_instructions TEXT,

    -- Pricing
    hourly_rate DECIMAL(10,2) NOT NULL,
    estimated_total DECIMAL(10,2) NOT NULL,
    final_total DECIMAL(10,2),                    -- after job completion
    platform_commission_pct DECIMAL(5,2) DEFAULT 25.00,
    platform_commission_amount DECIMAL(10,2),

    -- Status & timing
    status booking_status NOT NULL DEFAULT 'pending',
    started_at TIMESTAMPTZ,
    completed_at TIMESTAMPTZ,
    cancelled_at TIMESTAMPTZ,
    cancellation_reason TEXT,

    -- Payment
    stripe_payment_intent_id VARCHAR(255),        -- mocked
    payment_status VARCHAR(50) DEFAULT 'pending', -- pending, paid, refunded
    paid_at TIMESTAMPTZ,

    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Extras selected for a booking
CREATE TABLE booking_extras (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    booking_id UUID NOT NULL REFERENCES bookings(id),
    extra_id UUID NOT NULL REFERENCES service_extras(id),
    price DECIMAL(10,2) NOT NULL,
    quantity INTEGER DEFAULT 1
);

-- ============================================
-- REVIEWS
-- ============================================

CREATE TABLE reviews (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    booking_id UUID NOT NULL REFERENCES bookings(id) UNIQUE,
    reviewer_user_id UUID NOT NULL REFERENCES users(id),
    reviewed_user_id UUID REFERENCES users(id),       -- client reviewing cleaner or vice versa
    reviewed_cleaner_id UUID REFERENCES cleaners(id),  -- when client reviews cleaner
    rating INTEGER NOT NULL CHECK (rating BETWEEN 1 AND 5),
    comment TEXT,
    review_type VARCHAR(20) NOT NULL,  -- 'client_to_cleaner' or 'cleaner_to_client'
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- ============================================
-- CHAT / MESSAGING
-- ============================================

CREATE TABLE chat_rooms (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    booking_id UUID REFERENCES bookings(id),      -- job-specific chats
    room_type VARCHAR(50) NOT NULL,               -- 'booking', 'company_internal', 'admin_support'
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE TABLE chat_participants (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    room_id UUID NOT NULL REFERENCES chat_rooms(id),
    user_id UUID NOT NULL REFERENCES users(id),
    joined_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    UNIQUE(room_id, user_id)
);

CREATE TABLE chat_messages (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    room_id UUID NOT NULL REFERENCES chat_rooms(id),
    sender_id UUID NOT NULL REFERENCES users(id),
    content TEXT NOT NULL,
    message_type VARCHAR(20) DEFAULT 'text',      -- 'text', 'image', 'system'
    is_read BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- ============================================
-- NOTIFICATIONS
-- ============================================
```

```sql
CREATE TYPE notification_type AS ENUM (
    'booking_created',
    'booking_assigned',
    'booking_confirmed',
    'booking_started',
    'booking_completed',
    'booking_cancelled',
    'cleaner_invited',
    'company_approved',
    'company_rejected',
    'new_message',
    'review_received',
    'payment_processed'
);

CREATE TABLE notifications (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(id),
    type notification_type NOT NULL,
    title VARCHAR(255) NOT NULL,
    body TEXT NOT NULL,
    data JSONB,                          -- extra payload (booking_id, etc.)
    is_read BOOLEAN DEFAULT FALSE,
    is_pushed BOOLEAN DEFAULT FALSE,   -- whether FCM push was sent
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- ============================================
-- PLATFORM STATS (for admin dashboard)
-- ============================================

CREATE TABLE platform_events (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    event_type VARCHAR(100) NOT NULL,
    entity_type VARCHAR(50),           -- 'booking', 'company', 'user'
    entity_id UUID,
    metadata JSONB,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- ============================================
-- INDEXES
-- ============================================

CREATE INDEX idx_bookings_client ON bookings(client_user_id);
CREATE INDEX idx_bookings_company ON bookings(company_id);
CREATE INDEX idx_bookings_cleaner ON bookings(cleaner_id);
CREATE INDEX idx_bookings_status ON bookings(status);
CREATE INDEX idx_bookings_date ON bookings(scheduled_date);
CREATE INDEX idx_cleaners_company ON cleaners(company_id);
CREATE INDEX idx_chat_messages_room ON chat_messages(room_id);
CREATE INDEX idx_notifications_user ON notifications(user_id, is_read);
CREATE INDEX idx_platform_events_type ON platform_events(event_type, created_at);
```

---

4. GraphQL API Schema

4.1 Schema Overview

The GraphQL schema is split into domain modules. All mutations require authentication except `createBookingRequest` (guest flow) and `applyAsCompany`.

```
graphql
```

```graphql
# ================================================
# SCALARS & COMMON TYPES
# ================================================

scalar DateTime
scalar Upload
scalar JSON

type PageInfo {
  hasNextPage: Boolean!
  endCursor: String
}

type Coordinates {
  latitude: Float!
  longitude: Float!
}

# ================================================
# AUTH
# ================================================

type AuthPayload {
  token: String!
  user: User!
  isNewUser: Boolean!
}

extend type Mutation {
  # Google Sign-In — works for all roles
  signInWithGoogle(idToken: String!, role: UserRole!): AuthPayload!

  # Refresh JWT
  refreshToken: AuthPayload!

  # Register FCM token for push notifications
  registerDeviceToken(token: String!): Boolean!
}

# ================================================
# USERS
# ================================================

enum UserRole {
  CLIENT
  COMPANY_ADMIN
  CLEANER
  GLOBAL_ADMIN
}

enum UserStatus {
  ACTIVE
  INACTIVE
  SUSPENDED
  PENDING
}

type User {
  id: ID!
  email: String!
  fullName: String!
  phone: String
  avatarUrl: String
  role: UserRole!
  status: UserStatus!
  preferredLanguage: String!
  createdAt: DateTime!
}

extend type Query {
  me: User!
}

extend type Mutation {
  updateProfile(input: UpdateProfileInput!): User!
}

input UpdateProfileInput {
  fullName: String
  phone: String
  avatarUrl: String
  preferredLanguage: String
}

# ============================================
# CLIENT
# ============================================

type Address {
  id: ID!
  label: String
  streetAddress: String!
  city: String!
```

```graphql
  county: String!
  postalCode: String
  floor: String
  apartment: String
  entryCode: String
  coordinates: Coordinates
  notes: String
  isDefault: Boolean!
}

type PaymentMethod {
  id: ID!
  cardLastFour: String!
  cardBrand: String!
  isDefault: Boolean!
}

extend type Query {
  myAddresses: [Address!]!
  myPaymentMethods: [PaymentMethod!]!
}

extend type Mutation {
  addAddress(input: AddAddressInput!): Address!
  updateAddress(id: ID!, input: UpdateAddressInput!): Address!
  deleteAddress(id: ID!): Boolean!
  setDefaultAddress(id: ID!): Address!

  addPaymentMethod(stripeToken: String!): PaymentMethod!  # mocked
  deletePaymentMethod(id: ID!): Boolean!
  setDefaultPaymentMethod(id: ID!): PaymentMethod!
}

input AddAddressInput {
  label: String
  streetAddress: String!
  city: String!
  county: String!
  postalCode: String
  floor: String
  apartment: String
  entryCode: String
  latitude: Float
  longitude: Float
  notes: String
}
```

```graphql
input UpdateAddressInput {
  label: String
  streetAddress: String
  city: String
  county: String
  postalCode: String
  floor: String
  apartment: String
  entryCode: String
  latitude: Float
  longitude: Float
  notes: String
}

# ============================================
# SERVICES & PRICING
# ============================================

enum ServiceType {
  STANDARD_CLEANING
  DEEP_CLEANING
  MOVE_IN_OUT_CLEANING
  POST_CONSTRUCTION
  OFFICE_CLEANING
  WINDOW_CLEANING
}

type ServiceDefinition {
  id: ID!
  serviceType: ServiceType!
  nameRo: String!
  nameEn: String!
  descriptionRo: String
  descriptionEn: String
  basePricePerHour: Float!
  minHours: Float!
  icon: String
}

type ServiceExtra {
  id: ID!
  nameRo: String!
  nameEn: String!
  price: Float!
  icon: String
}
```

```graphql
type PriceEstimate {
  hourlyRate: Float!
  estimatedHours: Float!
  subtotal: Float!
  extras: [ExtraLineItem!]!
  total: Float!
}

type ExtraLineItem {
  extra: ServiceExtra!
  quantity: Int!
  lineTotal: Float!
}

extend type Query {
  availableServices: [ServiceDefinition!]!
  availableExtras: [ServiceExtra!]!
  estimatePrice(input: PriceEstimateInput!): PriceEstimate!
}

input PriceEstimateInput {
  serviceType: ServiceType!
  numRooms: Int!
  numBathrooms: Int!
  areaSqm: Int
  extras: [ExtraInput!]
}

input ExtraInput {
  extraId: ID!
  quantity: Int!
}

# ============================================
# BOOKINGS
# ============================================

enum BookingStatus {
  PENDING
  ASSIGNED
  CONFIRMED
  IN_PROGRESS
  COMPLETED
  CANCELLED_BY_CLIENT
  CANCELLED_BY_COMPANY
  CANCELLED_BY_ADMIN
}
```

```graphql
type Booking {
  id: ID!
  referenceCode: String!
  client: User!
  company: Company
  cleaner: CleanerProfile
  address: Address!

  # Service info
  serviceType: ServiceType!
  serviceName: String!
  scheduledDate: String!
  scheduledStartTime: String!
  estimatedDurationHours: Float!

  # Property
  propertyType: String
  numRooms: Int
  numBathrooms: Int
  areaSqm: Int
  hasPets: Boolean
  specialInstructions: String

  # Pricing
  hourlyRate: Float!
  estimatedTotal: Float!
  finalTotal: Float
  platformCommissionPct: Float!
  extras: [BookingExtra!]!

  # Status
  status: BookingStatus!
  startedAt: DateTime
  completedAt: DateTime
  cancelledAt: DateTime
  cancellationReason: String

  # Payment
  paymentStatus: String!
  paidAt: DateTime

  # Relations
  review: Review
  chatRoom: ChatRoom

  createdAt: DateTime!
```

```graphql
}

type BookingExtra {
  extra: ServiceExtra!
  price: Float!
  quantity: Int!
}

type BookingConnection {
  edges: [Booking!]!
  pageInfo: PageInfo!
  totalCount: Int!
}

extend type Query {
  # Client queries
  myBookings(status: BookingStatus, first: Int, after: String): BookingConnection!
  booking(id: ID!): Booking!

  # Company queries
  companyBookings(status: BookingStatus, first: Int, after: String): BookingConnection!

  # Cleaner queries
  myAssignedJobs(status: BookingStatus): [Booking!]!
  todaysJobs: [Booking!]!

  # Admin queries
  allBookings(
    status: BookingStatus
    companyId: ID
    dateFrom: String
    dateTo: String
    first: Int
    after: String
  ): BookingConnection!
}

extend type Mutation {
  # Guest booking flow — no auth required!
  createBookingRequest(input: CreateBookingInput!): Booking!

  # Client actions
  cancelBooking(id: ID!, reason: String): Booking!
  payForBooking(id: ID!, paymentMethodId: ID): Booking!  # mocked

  # Company/Admin actions
  assignCleanerToBooking(bookingId: ID!, cleanerId: ID!): Booking!
  # Cleaner actions
  confirmBooking(id: ID!): Booking!
  startJob(id: ID!): Booking!
  completeJob(id: ID!): Booking!
}

input CreateBookingInput {
  # Address (inline for guest flow)
  address: AddAddressInput!

  # Service
  serviceType: ServiceType!
  scheduledDate: String!
  scheduledStartTime: String!

  # Property
  propertyType: String
  numRooms: Int!
  numBathrooms: Int!
  areaSqm: Int
  hasPets: Boolean
  specialInstructions: String

  # Extras
  extras: [ExtraInput!]

  # Guest info (if not logged in)
  guestEmail: String
  guestName: String
  guestPhone: String
}

# ============================================
# COMPANIES
# ============================================

enum CompanyStatus {
  PENDING_REVIEW
  APPROVED
  REJECTED
  SUSPENDED
}

enum CompanyType {
  SRL
  PFA
  II
}

type Company {
  id: ID!
  companyName: String!
  cui: String!
  companyType: CompanyType!
  legalRepresentative: String!
  contactEmail: String!
  contactPhone: String!
  address: String!
  city: String!
  county: String!
  description: String
  logoUrl: String
  status: CompanyStatus!
  rejectionReason: String
  maxServiceRadiusKm: Int!
  ratingAvg: Float!
  totalJobsCompleted: Int!
  documents: [CompanyDocument!]!
  cleaners: [CleanerProfile!]!
  admin: User
  createdAt: DateTime!
}

type CompanyDocument {
  id: ID!
  documentType: String!
  fileUrl: String!
  fileName: String!
  uploadedAt: DateTime!
}

extend type Query {
  # Company admin
  myCompany: Company!

  # Global admin
  companies(status: CompanyStatus, first: Int, after: String): CompanyConnection!
  company(id: ID!): Company!
}

type CompanyConnection {
  edges: [Company!]!
  pageInfo: PageInfo!
```

```graphql
  totalCount: Int!
}

extend type Mutation {
  # Public — application from landing page
  applyAsCompany(input: CompanyApplicationInput!): Company!

  # Company admin
  updateCompanyProfile(input: UpdateCompanyInput!): Company!
  uploadCompanyDocument(companyId: ID!, documentType: String!, file: Upload!): CompanyDocument!

  # Global admin
  approveCompany(id: ID!): Company!
  rejectCompany(id: ID!, reason: String!): Company!
  suspendCompany(id: ID!, reason: String!): Company!
}

input CompanyApplicationInput {
  companyName: String!
  cui: String!
  companyType: CompanyType!
  legalRepresentative: String!
  contactEmail: String!
  contactPhone: String!
  address: String!
  city: String!
  county: String!
  description: String
  # Documents uploaded separately after initial application
}

input UpdateCompanyInput {
  description: String
  contactPhone: String
  maxServiceRadiusKm: Int
}

# ============================================
# CLEANERS
# ============================================

enum CleanerStatus {
  INVITED
  ACTIVE
  INACTIVE
  SUSPENDED
}
```

```graphql
type CleanerProfile {
  id: ID!
  user: User
  company: Company!
  fullName: String!
  phone: String
  email: String
  avatarUrl: String
  status: CleanerStatus!
  isCompanyAdmin: Boolean!
  ratingAvg: Float!
  totalJobsCompleted: Int!
  availability: [AvailabilitySlot!]!
  createdAt: DateTime!
}

type AvailabilitySlot {
  id: ID!
  dayOfWeek: Int!        # 0=Monday
  startTime: String!
  endTime: String!
  isAvailable: Boolean!
}

type CleanerStats {
  totalJobsCompleted: Int!
  averageRating: Float!
  totalReviews: Int!
  thisMonthJobs: Int!
  thisMonthEarnings: Float!
}

extend type Query {
  # Company admin
  myCleaners: [CleanerProfile!]!

  # Cleaner
  myCleanerProfile: CleanerProfile!
  myCleanerStats: CleanerStats!
}

extend type Mutation {
  # Company admin
  inviteCleaner(input: InviteCleanerInput!): CleanerProfile!
  inviteSelfAsCleaner: CleanerProfile!   # Admin adds themselves
  updateCleanerStatus(id: ID!, status: CleanerStatus!): CleanerProfile!
  # Cleaner
  acceptInvitation(token: String!): CleanerProfile!
  updateAvailability(slots: [AvailabilitySlotInput!]!): [AvailabilitySlot!]!
}

input InviteCleanerInput {
  fullName: String!
  email: String!
  phone: String
}

input AvailabilitySlotInput {
  dayOfWeek: Int!
  startTime: String!
  endTime: String!
  isAvailable: Boolean!
}

# ============================================
# REVIEWS
# ============================================

type Review {
  id: ID!
  booking: Booking!
  reviewer: User!
  rating: Int!
  comment: String
  reviewType: String!
  createdAt: DateTime!
}

extend type Mutation {
  submitReview(input: SubmitReviewInput!): Review!
}

input SubmitReviewInput {
  bookingId: ID!
  rating: Int!
  comment: String
}

# ============================================
# CHAT
# ============================================
```

```graphql
type ChatRoom {
  id: ID!
  booking: Booking
  roomType: String!
  participants: [ChatParticipant!]!
  messages(first: Int, after: String): ChatMessageConnection!
  lastMessage: ChatMessage
  createdAt: DateTime!
}

type ChatParticipant {
  user: User!
  joinedAt: DateTime!
}

type ChatMessage {
  id: ID!
  sender: User!
  content: String!
  messageType: String!
  isRead: Boolean!
  createdAt: DateTime!
}

type ChatMessageConnection {
  edges: [ChatMessage!]!
  pageInfo: PageInfo!
}

extend type Query {
  myChatRooms: [ChatRoom!]!
  chatRoom(id: ID!): ChatRoom!
}

extend type Mutation {
  sendMessage(roomId: ID!, content: String!, messageType: String): ChatMessage!
  markMessagesAsRead(roomId: ID!): Boolean!

  # Admin: initiate chat with any user
  createAdminChatRoom(userId: ID!): ChatRoom!
}

# WebSocket subscription for real-time messages
extend type Subscription {
  messageSent(roomId: ID!): ChatMessage!
  bookingUpdated(bookingId: ID!): Booking!
  notificationReceived: Notification!
```

```graphql
}

# ============================================
# NOTIFICATIONS
# ============================================

type Notification {
  id: ID!
  type: String!
  title: String!
  body: String!
  data: JSON
  isRead: Boolean!
  createdAt: DateTime!
}

extend type Query {
  myNotifications(first: Int, after: String, unreadOnly: Boolean): NotificationConnection!
  unreadNotificationCount: Int!
}

type NotificationConnection {
  edges: [Notification!]!
  pageInfo: PageInfo!
  totalCount: Int!
}

extend type Mutation {
  markNotificationRead(id: ID!): Notification!
  markAllNotificationsRead: Boolean!
}

# ============================================
# ADMIN DASHBOARD
# ============================================

type PlatformStats {
  totalClients: Int!
  totalCompanies: Int!
  totalCleaners: Int!
  totalBookings: Int!
  totalRevenue: Float!
  platformCommissionTotal: Float!
  averageRating: Float!

  # Period-specific
  bookingsThisMonth: Int!
  revenueThisMonth: Float!
  newClientsThisMonth: Int!
  newCompaniesThisMonth: Int!
}

type BookingsByStatus {
  status: BookingStatus!
  count: Int!
}

type RevenueByMonth {
  month: String!
  revenue: Float!
  commission: Float!
  bookingCount: Int!
}

type CompanyPerformance {
  company: Company!
  totalBookings: Int!
  totalRevenue: Float!
  averageRating: Float!
  completionRate: Float!
}

extend type Query {
  platformStats(dateFrom: String, dateTo: String): PlatformStats!
  bookingsByStatus: [BookingsByStatus!]!
  revenueByMonth(months: Int): [RevenueByMonth!]!
  companyPerformance(first: Int): [CompanyPerformance!]!
  pendingCompanyApplications: [Company!]!
}

extend type Mutation {
  # Admin can cancel any booking
  adminCancelBooking(id: ID!, reason: String!): Booking!

  # Admin can manage users
  suspendUser(id: ID!, reason: String!): User!
  reactivateUser(id: ID!): User!
}

# ============================================
# FILE UPLOAD
# ============================================

type UploadResult {
  url: String!
  fileName: String!
}

extend type Mutation {
  uploadFile(file: Upload!, purpose: String!): UploadResult!
}
```

---

5. Feature Specifications by Role

5.1 Client (iOS SwiftUI + Web)

5.1.1 Booking Flow (Guest-Friendly)

The booking flow is the **hero feature** — it must be smooth, beautiful, and work without authentication.

Step-by-step flow:

```
1. SELECT SERVICE TYPE
   → Cards showing service types with icons, descriptions, starting prices
   → e.g., "Curățenie Standard", "Curățenie Generală", "După Constructor"

2. PROPERTY DETAILS
   → Property type (apartment/house/office)
   → Number of rooms (visual selector, not text input)
   → Number of bathrooms
   → Approximate area (sqm) — optional
   → Has pets? (toggle)

3. SELECT EXTRAS
   → Grid of extra services with prices
   → e.g., "Interior frigider (+30 RON)", "Interior cuptor (+25 RON)",
     "Călcat rufe (+40 RON)", "Curățat geamuri interioare (+35 RON)"
   → Each extra has quantity selector if applicable

4. PRICE ESTIMATE (shown live as they configure)
   → Hourly rate × estimated hours
   → Extras total
   → Grand total
   → "Preț estimat: 210 RON" — prominent display

5. SCHEDULE
   → Calendar date picker (min 24h in advance)
   → Time slot selection (8:00, 9:00, 10:00, etc.)
   → Duration shown based on property size

6. ADDRESS
   → Google Places autocomplete
   → Map pin confirmation
   → Floor, apartment, entry code fields
   → Special instructions textarea

7. REVIEW & CONFIRM
   → Full summary: service, property, extras, price, date, time, address
   → "Confirmă Rezervarea" button

8. PAYMENT (mocked for MVP)
   → Add card (Stripe Elements — mocked)
   → Or "Plătește la final" option
   → Confirmation animation

9. SIGN UP PROMPT
   → "Creează un cont pentru a urmări comanda ta"
```

→ Google Sign-In button
→ "Continuă fără cont" option (but they can't track order)

Design notes for iOS (SwiftUI):

- Use Liquid Glass effects for cards and modals
- Smooth page transitions with matched geometry
- Haptic feedback on selections
- Large, tappable elements
- Bottom sheet for extras selection
- Live price update animation

5.1.2 Client Dashboard (Post-Auth)

Screen	Content
Home	Active booking card, quick rebook, upcoming bookings
My Bookings	List with status badges, pull-to-refresh, filter by status
Booking Detail	Full info, assigned cleaner with photo/rating, job timeline, chat button, cancel option
Chat	Real-time messaging with assigned cleaner, message bubbles
Review	Star rating (1-5), optional comment, submit after completion
Profile	Name, phone, avatar
Addresses	List, add/edit/delete, set default
Payment Methods	Saved cards, add new (mocked), set default
Notifications	In-app notification center

5.1.3 Client Web Version

Same functionality as iOS but implemented in React + Shadcn/ui. Focus on:

- Responsive design (desktop + mobile browser)
- Booking flow as a multi-step form with progress indicator
- Dashboard with sidebar navigation

5.2 Company Admin (Web Dashboard Primary + React Native)

5.2.1 Application Flow (Public — Landing Page)

Step-by-step from landing page:

```
1. LANDING PAGE
  → Hero section: "Devino partener HelpMeClean"
  → Benefits of joining (more clients, digital payments, simplified admin)
  → "Aplică acum" CTA button

2. COMPANY INFORMATION FORM
  → Company name
  → CUI (Cod Unic de Înregistrare)
  → Company type: SRL / PFA / II
  → Legal representative name
  → Contact email & phone
  → Address, city, county
  → Description (what makes them special)

3. DOCUMENT UPLOAD
  → Certificat Constatator (required)
  → Asigurare de Răspundere Civilă (required)
  → CUI document (required)
  → Other supporting documents (optional)
  → Drag-and-drop or file picker
  → Progress indicator for uploads

4. GOOGLE SIGN-UP
  → "Creează contul de administrator"
  → Google Sign-In
  → Associates Google account with company application

5. PENDING CONFIRMATION
  → "Aplicația ta a fost trimisă cu succes!"
  → "Vei primi un email când contul tău este aprobat."
  → Show pending dashboard preview (greyed out)
```

5.2.2 Company Dashboard (Post-Approval — Web)

Navigation sidebar:

- 📊 Dashboard (overview)
- 📋 Comenzi (bookings)
- 👥 Echipa mea (team management)
- 💬 Mesaje (chat)

- ⚙️ Setări (settings)

Section	Features
Dashboard	Today's jobs, weekly revenue chart, team availability overview, recent reviews
Comenzi	All bookings (pending, assigned, in progress, completed), assign cleaner to booking, filter/search
Echipa Mea	List of cleaners with status badges, invite new cleaner (email invite), "Add myself as cleaner" button, edit cleaner details, toggle active/inactive, availability calendar view
Mesaje	Internal team chat, booking-specific chats (cleaner ↔ client), admin support chat
Setări	Company profile, logo upload, service radius, documents management

5.2.3 Invite Flow for Cleaners

```
Company Admin clicks "Invită un curățător"
  → Enter: name, email, phone
  → System generates invite link with unique token
  → Email sent with invite link (or share link manually for MVP)
  → Cleaner opens link → download app prompt
  → Cleaner signs in with Google → account linked to company
  → Status: invited → active
```

Self-invite for PFA/solo operators:

```
Company Admin clicks "Adaugă-te ca și curățător"
  → Confirmation dialog
  → Creates cleaner profile linked to their user account
  → They now see both Company Dashboard (web) and Cleaner features (mobile)
```

5.3 Cleaner (React Native — iOS + Android)

5.3.1 Screens

Screen	Content
Home / Today	Today's assigned jobs as cards with time, address, service type, client name. Next job highlighted.
Job Detail	Full booking details, client info, address with map + directions button (opens native maps), checklist of tasks based on service type, "Începe curățenia" / "Finalizează" buttons
Job Timer	Active timer when job is in progress, elapsed time display
My Schedule	Calendar view of upcoming jobs, weekly overview
Chat	Chat with current client (booking-specific), chat with company admin
Stats	Jobs completed (total, this month), average rating, total reviews, earnings summary
Review Client	After job completion: rate client (1-5), optional comment
Profile	Personal info, avatar, availability schedule editor

5.3.2 Job Lifecycle (Cleaner Perspective)

```
1. NOTIFICATION: "Ai o nouă comandă!"
   → Push notification + in-app notification

2. VIEW ASSIGNMENT
   → See job details, address, service type, client info
   → "Confirm" button → status: CONFIRMED

3. DAY OF JOB
   → Job appears in "Today" tab
   → "Navigate" button → opens Apple Maps / Google Maps
   → "Am ajuns" optional check-in

4. START JOB
   → "Începe curățenia" button → status: IN_PROGRESS
   → Timer starts
   → Client gets notification

5. DURING JOB
   → Timer running
   → Can chat with client if needed
   → Checklist of tasks to complete

6. COMPLETE JOB
   → "Finalizează curățenia" button → status: COMPLETED
   → Timer stops, duration recorded
   → Prompted to review client

7. REVIEW CLIENT
   → Rate 1-5 stars
   → Optional comment
   → Submit
```

5.4 Global Admin (Web Dashboard + Mobile)

5.4.1 Web Dashboard

Navigation sidebar:

- 📊 Dashboard (platform overview)
- 🏢 Companii (company management)
- 📋 Comenzi (all bookings)
- 👥 Utilizatori (user management)
- 💬 Mesaje (admin chat)

- 📈 Rapoarte (reports)
- ⚙️ Setări (settings)

Section	Features
Dashboard	Key metrics cards (total bookings, revenue, active users, pending applications), revenue chart (line, by month), bookings by status (donut chart), recent activity feed
Companii	All companies list with status filter, pending applications queue with approve/reject actions, company detail view (docs, cleaners, stats), suspend/reactivate
Comenzi	All platform bookings, advanced filters (status, company, date range, city), booking detail view, admin cancel capability
Utilizatori	All users list with role filter, user detail view, suspend/reactivate, search by name/email
Mesaje	Initiate chat with any user (support), list of active admin conversations
Rapoarte	Revenue by month (filterable by company), bookings by status over time, company performance ranking, top cleaners by rating, client retention metrics, exportable (CSV — nice to have)
Setări	Platform pricing configuration, commission rate, service types management

5.4.2 Admin Mobile (React Native — Lower Priority)

Simplified version:

- View and approve/reject company applications
- View platform stats
- Chat with users
- Push notifications for new applications and urgent issues

---

6. Additional Features (Recommended for MVP)

These features aren't in the original requirements but significantly improve the demo quality and completeness:

6.1 Push Notifications (Essential)

Without push notifications, the multi-role flow feels disconnected. Use Firebase Cloud Messaging (FCM):

Trigger	Recipients	Message
New booking created	Company admins in area	"Comandă nouă: Curățenie Standard, 3 camere"
Cleaner assigned	Client	"Un curățător a fost desemnat pentru comanda ta"
Cleaner confirms	Client	"Curățătorul tău a confirmat prezența"
Job started	Client	"Curățenia a început!"
Job completed	Client	"Curățenia s-a finalizat! Lasă o recenzie"
New message	Recipient	"Mesaj nou de la [name]"
Company approved	Company admin	"Felicitări! Contul tău a fost aprobat"
Company rejected	Company admin	"Aplicația ta necesită modificări"
Cleaner invited	Cleaner (email)	"Ai fost invitat să te alături echipei [company]"
Review received	Cleaner/Client	"Ai primit o recenzie nouă: ⭐⭐⭐⭐⭐ "

6.2 In-App Notification Center

Every user role sees a bell icon with unread count badge. Tapping opens a scrollable notification list grouped by date. Each notification is tappable and navigates to the relevant screen.

6.3 Booking Reference Codes

Human-readable codes like `HMC-7A3F` for easy phone/chat reference instead of UUIDs.

6.4 Job Matching Logic (Simplified for MVP)

When a booking is created:

1. Find companies within service radius of the booking address
2. Notify company admins in that area
3. Company admin manually assigns a cleaner from their team
4. Cleaner confirms the assignment

Future: auto-matching based on availability, ratings, and proximity.

6.5 Multi-Language Support (Romanian + English)

All user-facing strings should support `ro` and `en`. The backend returns both `nameRo` and `nameEn` for services; clients choose their preferred language.

6.6 Service Checklists

Each service type has a standard checklist of tasks. When a cleaner starts a job, they see what's expected:

Standard Cleaning:

- ✅ Aspirat toate camerele
- ✅ Șters praful de pe suprafețe
- ✅ Curățat baia (chiuvetă, WC, cadă/duș)
- ✅ Curățat bucătăria (chiuvetă, aragaz, blat)
- ✅ Spălat pe jos
- ✅ Golit coșurile de gunoi

6.7 Landing Page

A polished public landing page for the platform:

- Hero with value proposition
- How it works (3 steps for clients)
- Service types with pricing
- "Become a partner" CTA for companies
- FAQ section
- Footer with legal links

6.8 Email Notifications (Basic)

For MVP, use a simple email service (or even just log emails) for:

- Booking confirmations
- Company application status updates
- Cleaner invitations
- Password-less login links (if needed)

6.9 Booking Cancellation Policy

Simple rules:

- Free cancellation up to 24 hours before scheduled time
- 50% charge for cancellations within 24 hours
- Full charge for no-shows

6.10 Photo Upload for Profiles

Cleaners and company admins should be able to upload profile photos. Clients see cleaner photos when they're assigned — it builds trust.

---

7. Design Guidelines

7.1 Brand Identity

Element	Value
Primary Color	🟦 #2563EB (blue-600) — trust, cleanliness
Secondary Color	🟩 #10B981 (emerald-500) — freshness, success
Accent Color	🟧 #F59E0B (amber-500) — ratings, highlights
Danger Color	🟥 #EF4444 (red-500) — errors, cancellations
Background	⬜ #FAFBFC — clean, bright
Text Primary	⬛ #111827 (gray-900)
Text Secondary	⬛ #6B7280 (gray-500)
Font (Web)	Inter — clean, modern, great for UI
Font (iOS)	SF Pro — system default, Liquid Glass compatible
Border Radius	12px — modern, soft
Spacing Scale	4px base (4, 8, 12, 16, 20, 24, 32, 48, 64)

7.2 Web (Shadcn/ui)

- Use Shadcn/ui as the component foundation
- TailwindCSS for all styling — no custom CSS files
- Dark mode support (nice to have)
- Responsive: mobile-first approach
- Sidebar navigation with collapsible menu
- Toast notifications for actions
- Loading skeletons for data fetching states
- Empty states with illustrations
- Consistent card-based layouts

7.3 iOS (SwiftUI — Liquid Glass)

- Use iOS 26+ Liquid Glass material effects
- Navigation: TabView with custom styling
- Glass-effect cards for booking items and job cards
- Smooth transitions with matchedGeometryEffect
- Haptic feedback on key interactions
- Pull-to-refresh on all lists
- Large, accessible touch targets (min 44pt)
- Bottom sheet modals for selections
- Custom animations for status changes

7.4 React Native (Cleaner + Company Mobile)

- NativeWind (TailwindCSS for RN) for styling
- Platform-adaptive components (iOS vs Android)
- Native navigation (React Navigation)
- Gesture handling for swipe actions
- Clean, card-based layout matching the web dashboard aesthetic
- Loading states and skeletons
- Pull-to-refresh

---

8. Implementation Priority

Phase 1: Foundation (Week 1-2)

1. **Backend:** Project setup, database migrations, auth (Google OAuth + JWT), basic GraphQL schema
2. **Shared:** GraphQL codegen setup for all clients
3. **Landing page:** Public site with company application form

Phase 2: Core Booking Flow (Week 3-4)

1. **Backend:** Service definitions, pricing engine, booking CRUD, guest booking flow
2. **Client iOS:** Complete booking flow (the hero feature)
3. **Client Web:** Booking flow (can lag behind iOS)
4. **Company Web:** Application flow, pending dashboard

Phase 3: Company & Cleaner (Week 5-6)

1. **Backend:** Company management, cleaner invites, job assignment
2. **Company Dashboard:** Team management, booking management
3. **Cleaner App:** Accept invite, view jobs, start/complete flow
4. **Admin Dashboard:** Company approval queue

Phase 4: Communication & Polish (Week 7-8)

1. **Backend:** Chat (WebSocket), notifications, reviews
2. **All clients:** Chat integration, notification center
3. **Reviews:** Client → cleaner and cleaner → client
4. **Admin Dashboard:** Full stats, reports, user management
5. **Polish:** Animations, empty states, error handling, loading states

---

9. API Authentication & Authorization

9.1 Auth Flow

```
Client App          Backend           Google
   │                  │                 │
   │── Google Sign-In ──────────►│                 │
   │                  │── Verify ID Token ──────────►│
   │                  │◄── User Info ─────────────────│
   │                  │                 │
   │                  │── Find/Create User ──►DB  │
   │                  │                 │
   │◄── JWT Token + User ──────────│                 │
   │                  │                 │
   │── GraphQL Request ──────────►│                 │
   │  (Authorization: Bearer) │                 │
   │                  │── Verify JWT ──────────────►│
   │                  │── Check Role ──────────────►│
   │◄── Response ─────────────────────│                 │
```

9.2 Authorization Matrix

Operation	Guest	Client	Company Admin	Cleaner	Global Admin
Create booking	✅	✅	❌	❌	❌
View own bookings	❌	✅	❌	❌	✅

Operation	Guest	Client	Company Admin	Cleaner	Global Admin
Cancel own booking	❌	✅	❌	❌	✅
Apply as company	✅	❌	❌	❌	❌
Manage team	❌	❌	✅	❌	❌
Assign cleaner	❌	❌	✅	❌	✅
Start/complete job	❌	❌	❌	✅	❌
Approve companies	❌	❌	❌	❌	✅
View platform stats	❌	❌	❌	❌	✅
Admin chat	❌	❌	❌	❌	✅
Send message in booking chat	❌	✅ *	❌	✅ *	✅

*Only participants of that booking's chat room.

---

10. Environment Variables

```env
```

```
# Server
PORT=8080
ENVIRONMENT=development   # development | staging | production

# Database
DATABASE_URL=postgresql://user:pass@host/dbname?sslmode=require

# Auth
GOOGLE_CLIENT_ID=your-google-client-id
GOOGLE_CLIENT_ID_IOS=your-ios-client-id
GOOGLE_CLIENT_ID_ANDROID=your-android-client-id
JWT_SECRET=your-jwt-secret
JWT_EXPIRY=24h

# Stripe (mocked for MVP)
STRIPE_SECRET_KEY=sk_test_xxx
STRIPE_PUBLISHABLE_KEY=pk_test_xxx
STRIPE_WEBHOOK_SECRET=whsec_xxx

# Storage
GCS_BUCKET=helpmeclean-uploads
GCS_PROJECT_ID=your-gcp-project

# Firebase (Push Notifications)
FIREBASE_PROJECT_ID=your-firebase-project
FIREBASE_SERVICE_ACCOUNT_KEY=path/to/key.json

# CORS
ALLOWED_ORIGINS=http://localhost:3000,http://localhost:3001,http://localhost:3002

# Email (basic — can use logs for MVP)
SMTP_HOST=
SMTP_PORT=
SMTP_USER=
SMTP_PASS=
FROM_EMAIL=noreply@helpmeclean.ro
```

---

11. Development Setup

bash

```
```

```
# 1. Clone and setup
git clone <repo>
cd helpmeclean

# 2. Backend
cd backend
cp .env.example .env          # Fill in values
go mod download
make migrate-up               # Run database migrations
make generate                 # Generate sqlc + gqlgen code
make run                      # Start server on :8080

# 3. Web apps
cd web
npm install
npm run dev:client            # Client web on :3000
npm run dev:company           # Company dashboard on :3001
npm run dev:admin             # Admin dashboard on :3002

# 4. Mobile apps
cd mobile
npm install
npx expo start                # Expo dev server

# 5. iOS app
cd ios
open HelpMeClean.xcodeproj    # Open in Xcode
# Run on simulator or device

# Docker (full stack)
docker-compose up             # Backend + DB + all services
```

---

12. Key Decisions & Constraints

Decision	Choice	Rationale
Monolith vs Microservices	**Monolith**	Simplicity, single deployment, good enough for MVP scale
REST vs GraphQL	**GraphQL**	Multiple clients with different data needs, type safety
Native iOS vs React Native	**SwiftUI native** for client	Best UX for the hero app, Liquid Glass design

Decision	Choice	Rationale
React Native for others	**Expo**	Faster development for cleaner/company mobile apps
Auth provider	**Google OAuth only**	Simplicity, most Romanians have Google accounts
Payments	**Stripe (mocked)**	Industry standard, easy to unmock later
Real-time	**WebSockets**	Chat + live job updates, Go's goroutines handle well
File storage	**GCS**	Already on GCP, simple integration
State management (web)	**Apollo Client cache**	GraphQL-native, reduces boilerplate
Styling (web)	**Shadcn/ui + Tailwind**	Modern, customizable, polished out of the box
i18n	**Romanian primary, English secondary**	Target market is Romania, but investor may prefer English
No external integrations	**Confirmed**	No e-factura, no ANAF API, no Google Maps API — keep it simple

---

13. Mocked / Simplified for MVP

Feature	MVP Approach	Production Approach
Payments	Stripe test mode, mock payment flow	Full Stripe Connect with real charges
Company verification	Manual admin review	ANAF API auto-verification of CUI
E-factura	Not included	facturează.ro integration
Geocoding	Manual lat/lng or browser geolocation	Google Places API + Geocoding
Email	Console log or basic SMTP	SendGrid / Postmark
Push notifications	FCM basic implementation	FCM with rich notifications + deep links
Job matching	Manual assignment by company admin	Auto-matching algorithm
Maps in cleaner app	Link to external maps app	Embedded map with directions
Analytics	Basic SQL queries for admin dashboard	PostHog or Mixpanel
File storage	Local filesystem (dev), GCS (prod)	GCS with CDN

14. Success Criteria for Demo

The MVP is successful if an investor can:

1. ✅ **Visit the landing page** and understand the value proposition
2. ✅ **Book a cleaning as a client** — smooth, beautiful booking flow
3. ✅ **See the booking confirmed** with pricing and details
4. ✅ **Apply as a cleaning company** from the landing page
5. ✅ **Admin approves the company** from the admin dashboard
6. ✅ **Company admin invites a cleaner** (or adds themselves)
7. ✅ **Cleaner sees and accepts the job** on their mobile app
8. ✅ **Cleaner starts and completes the job** with timer
9. ✅ **Client reviews the service** after completion
10. ✅ **Chat works** between client and cleaner
11. ✅ **Admin dashboard shows stats** — the platform is alive
12. ✅ **Everything looks polished** — modern, clean, trustworthy

The full lifecycle from booking → assignment → completion → review should work end-to-end in a live demo.