Ein Bild, das Logo, Uhr, Grafiken, Clipart enthält.

KI-generierte Inhalte können fehlerhaft sein.

**Entwicklungsschritte**

**1. Grundfunktionen**

**a. Für Kunden (Gemeinden / Unternehmen):**

* **Login- / Registrierungsseite**  
  mit E-Mail und Passwort, ggf. Zwei-Faktor-Authentifizierung (2FA)
* **Anfrageformular** mit folgenden Feldern:
  + Name der Firma / Institution
  + Adresse + Ort / Postleitzahl (für regionale Filterung)
  + Ansprechpartner, Telefonnummer, E-Mail
  + **Auswahl der gewünschten Dienstleistung** *(Dropdown-Menü)*, z. B.:
    - Umweltverträglichkeitsprüfung
    - Architekturplanung
    - Energieberatung
    - Bodengutachten
    - Bauphysikalische Beratung
* **Anfragenhistorie + Statusanzeige**  
  (z. B. *gesendet / in Bearbeitung / verbunden mit Anbieter*)

**b. Für Dienstleister (z. B. Umweltbüros, Architekturbüros):**

• Login- / Registrierungsseite

• Detailliertes Profil mit:

• Abgedeckte Regionen

• Fachgebiete / Dienstleistungen

• Projektbeispiele

• Benachrichtigung bei passenden Anfragen

**2. Matching-Funktionalität**

• Basierend auf der Region (z. B. über Postleitzahl oder Geodaten)

• Basierend auf der gewünschten Dienstleistung

• Zunächst einfache Filterung, später ggf. automatisierter Matching-Algorithmus

**3. Verwaltungsbereich (Admin-Panel)**

• Für dich als Administrator:

• Einsicht und Verwaltung aller Nutzer, Anfragen und Dienstleister

• Verifizierung von Dienstleistern

• Eingreifen bei Problemen

**4. Technologiewahl – Drei mögliche Umsetzungsvarianten**

Im Folgenden werden drei konkrete Technologievarianten vorgestellt, mit denen die Plattform ecoMatch.eu umgesetzt werden kann:

**🅰️ Variante A: Bubble.io (No-Code/Low-Code)**

* **Frontend & Backend**: Bubble.io All-in-One-Plattform (Drag & Drop, visuelle Workflows)
* **Authentifizierung**: Bubble-intern oder Plugin für 2FA / OAuth
* **Zahlungen**: Integration über Stripe/Mollie-Plugins
* **Datenbank**: Bubble.io interne Datenstruktur (tabellenbasiert, relational)
* **Karten / Standortdienste**: Google Maps Plugin oder Leaflet.js Plugin
* **Hosting**: Bubble.io Cloud (Serverstandort ggf. EU wählbar)
* **Besonderheiten**: Sehr schnelle Entwicklung, eingeschränkte Skalierbarkeit, wiederkehrende Kosten

**🅱️ Variante B: HTML, CSS, JavaScript (Vanilla) + PHP/Python Backend**

* **Frontend**: Manuell mit HTML5, CSS3, Vanilla JS
* **Backend**:
  + PHP (z. B. mit Laravel Slim) oder
  + Python (z. B. Flask als leichtgewichtige API)
* **Authentifizierung**: Eigene Lösung (Sessions, Passwort-Hashing, ggf. 2FA)
* **Zahlungen**: Stripe/Mollie API-Integration per JS + Webhooks im Backend
* **Datenbank**: MySQL oder PostgreSQL, über ORM oder direkte Queries
* **Karten / Standortdienste**: Google Maps API oder OpenStreetMap + Nominatim
* **Hosting**: Eigener vServer mit Nginx (z. B. Hetzner Cloud, DigitalOcean)
* **Besonderheiten**: Hohe Kontrolle, günstiges Hosting, aber längere Entwicklungszeit

**🅲️ Variante C: React + Python (Django/Flask)**

* **Frontend**: React (modular, performant, erweiterbar)
* **Backend**:
  + Django (mit Django REST Framework) oder
  + Flask (für schlanke APIs)
* **Authentifizierung**: Django-Auth-System (inkl. 2FA, Rollen) oder JWT-Login
* **Zahlungen**: Stripe/Mollie mit Python SDK + Webhooks
* **Datenbank**: PostgreSQL bevorzugt (wegen Geo-Funktionen & Performance)
* **Karten / Standortdienste**: Google Maps API oder OpenStreetMap + Nominatim
* **Hosting**: Docker-Deployment auf VPS (Hetzner, DigitalOcean, AWS)
* **Besonderheiten**: Beste Skalierbarkeit & Erweiterbarkeit, langfristig wartbar

**5. Entwicklungsphasen**

1. Einfaches MVP mit Login, Formular und manueller Weiterleitung an passende Dienstleister

2. Automatisches Matching + Benachrichtigungen

3. Erweiterte Profile für Dienstleister und Kunden

4. Verlauf, Statistiken, Feedbacksystem

🔒 Zugang nur mit aktivem Abonnement:

• Kunden (Kommunen / Unternehmen) können nur dann Anfragen stellen, wenn ihr Abonnement aktiv ist.

• Dienstleister (z. B. Umweltbüros) erhalten nur dann passende Anfragen, wenn sie ein aktives Abo haben.

• Es kann eine kostenlose Testphase (z. B. 14 Tage) angeboten werden.

⸻

**💳 Wie funktionieren Abonnements?**

1. Integration eines Zahlungsanbieters

**💶 Geeignete Zahlungsanbieter für den deutschen / EU-Markt**

| **Anbieter** | **Vorteile** |
| --- | --- |
| **Stripe** | Sehr beliebt, unterstützt SEPA & Kreditkarte, leicht integrierbar |
| **PayPal** | Einfach zu verwenden, aber eingeschränkt bei wiederkehrenden Zahlungen |
| **Mollie** | Ideal für den EU-Markt, unterstützt SEPA-Lastschrift & Rechnungserstellung |
| **Digistore24** | Perfekt für Deutschland, automatische Abwicklung der Umsatzsteuer (USt) |

2. Benötigte Funktionen auf der Plattform

🔸 Für Kunden und Dienstleister:

• Seite „Mein Abonnement“ mit:

• Auswahl des Abonnements

• Eingabe der Zahlungsdaten

• Bestellübersicht & Zahlungsbestätigung

• Nach erfolgreicher Zahlung:

• Eintrag in die Datenbank: Abo = aktiv

• Automatische Prüfung des Abostatus bei jeder Anmeldung

• Bei abgelaufenem Abo: Kein Zugriff + Hinweis „Bitte Abo verlängern“

🔸 Für Administrator (dich):

• Admin-Dashboard mit:

• Liste aller Nutzer mit Abo-Status

• Aktive, abgelaufene & Test-Abos

• Export von Rechnungen (optional)

💰 Beispielhafte Abo-Modelle

| **Abo-Typ** | **Preis / Monat** | **Vorteile** |
| --- | --- | --- |
| **Kunde – Basic** | 19 € | Bis zu 5 Anfragen pro Monat |
| **Kunde – Pro** | 49 € | Unbegrenzte Anfragen |
| **Dienstleister – Standard** | 29 € | Erhält Anfragen aus seinem PLZ-Gebiet |
| **Dienstleister – Premium** | 79 € | Anfragen + Zugriff auf Statistikfunktionen |

Optional: Rabatt bei Jahreszahlung (z. B. 10 % Ermäßigung)

🧩 Umsetzungsreihenfolge mit Bezahlfunktion

1. ✅ Registrierung & Login

2. ✅ Anfrage-Formular & Anbieterprofil

3. ✅ Automatische Matching-Logik

4. ➕ Integration von Stripe / Mollie

5. ✅ Automatische Zugangsprüfung (Abo aktiv?)

6. ✅ Rechnungsversand automatisch / per PDF

⚠ **Rechtliches in Deutschland**

Für rechtssicheren Betrieb mit Bezahlung musst du:

• Bruttopreise inkl. USt (falls umsatzsteuerpflichtig) angeben

• Rechnungen ausstellen (automatisch oder manuell)

• Die Plattform braucht folgende Pflichtseiten:

• Impressum

• Datenschutzerklärung (DSGVO)

• AGB

• Widerrufsrecht (14 Tage Rücktritt)

**✅ 2. Vergleichstabelle: Bubble.io vs HTML/CSS/JS vs React + Python**

| **Kriterium** | **Bubble.io** | **HTML/CSS/JS Vanilla** | **React + Python (Django/Flask)** |
| --- | --- | --- | --- |
| **Entwicklungsdauer MVP** (1 Entwickler) | **2–4 Wochen** | 6–8 Wochen | 8–12 Wochen |
| **Mit Zahlungsfunktion & Admin** | **4–6 Wochen** | 10–12 Wochen | 12–16 Wochen |
| **Lernkurve** | 🔓 Sehr niedrig | 🟡 Mittel | 🔐 Hoch |
| **Schnelles UI-Design** | ✅ Drag & Drop | ❌ Manuell | ⚠ Nur mit Frameworks (z. B. Tailwind) |
| **Monatliche Kosten** | 💲 29–129 €/Monat | ✅ Nur Hosting-Kosten | ✅ Nur Hosting-Kosten |
| **Individuelle Anpassung** | ⚠ Eingeschränkt | ✅ Vollständig | ✅ Vollständig |
| **Skalierbarkeit** | ⚠ Eingeschränkt | 🟡 Gut | ✅ Sehr gut |
| **API- & Logik-Kontrolle** | ⚠ Mittel | ✅ Voll | ✅ Voll |
| **Zahlungen (Stripe/Mollie)** | ✅ Einfach integrierbar | 🟡 Mit JS-Code | ✅ Offizielle SDKs |
| **Authentifizierung (2FA, Rollen)** | ✅ Mit Plugin | 🟡 Selbst entwickeln | ✅ Voll integriert |
| **Matching-Logik** | ✅ Über Workflows | 🔧 Eigenentwicklung | ✅ Backend-Logik möglich |
| **Rechtliches (DSGVO, Hosting DE)** | ✅ Möglich, Server prüfen | ✅ Voll kontrollierbar | ✅ Voll kontrollierbar |
| **SEO** | ⚠ Schwach (SPA) | ✅ Gut | ✅ Gut (z. B. mit SSR) |

**📌 Empfehlung je nach Ziel:**

| **Ziel** | **Empfohlene Lösung** |
| --- | --- |
| Schnelles MVP zum Testen am Markt | ✅ **Bubble.io** |
| Günstiges, einfaches MVP mit mehr Kontrolle | 🟡 **HTML/CSS/JS Vanilla** |
| Langfristige, skalierbare Plattform | ✅ **React + Python (Django oder Flask)** |

**🕒 Geschätzte Entwicklungsdauer für einen einzelnen Entwickler**

| **Funktionalität** | **Bubble.io** | **HTML/CSS/JS** | **React + Django** |
| --- | --- | --- | --- |
| Registrierung/Login + 2FA | 2 Tage | 4 Tage | 5 Tage |
| Kundenformular mit Dropdown + Validierung | 2 Tage | 3 Tage | 3 Tage |
| Dienstleisterprofil | 2 Tage | 3 Tage | 4 Tage |
| Matching-Logik (einfach) | 3 Tage | 5 Tage | 5 Tage |
| Zahlungsintegration (Stripe/Mollie) | 2 Tage | 4 Tage | 4 Tage |
| Admin-Bereich | 3 Tage | 4–5 Tage | 6 Tage |
| Abo-Prüfung + Testphase | 2 Tage | 3 Tage | 4 Tage |
| Automatische Benachrichtigungen (E-Mail) | 1 Tag | 2 Tage | 2 Tage |
| **Gesamt (geschätzt)** | **4–6 Wochen** | **10 Wochen** | **12–14 Wochen** |

Ein Bild, das Text, Zahl, Reihe, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Diagramm, parallel, Dokument enthält.

KI-generierte Inhalte können fehlerhaft sein.

**Domain**

Ich habe die Domain „ecoMatch“ geprüft und festgestellt, dass sowohl die .de- als auch die .eu-Domain bereits vergeben sind, aber zum Verkauf stehen.

|  |
| --- |
| Ein Bild, das Text, Screenshot, Multimedia, Software enthält.  KI-generierte Inhalte können fehlerhaft sein.Ein Bild, das Text, Screenshot, Software, Multimedia enthält.  KI-generierte Inhalte können fehlerhaft sein. |

|  |
| --- |
| Ein Bild, das Text, Screenshot, Zahl, parallel enthält.  KI-generierte Inhalte können fehlerhaft sein.Ein Bild, das Text, Screenshot, Software, Computersymbol enthält.  KI-generierte Inhalte können fehlerhaft sein. |

**Kurze Übersicht der bisher erledigten Schritte**

1. **Python und virtuelle Umgebung (venv) eingerichtet**
   * Python 3.11 installiert
   * Virtuelle Umgebung mit python -m venv venv erstellt und aktiviert
2. **Django und Django REST Framework installiert**
   * Mit pip install django djangorestframework
3. **Django-Projekt und App erstellt**
   * django-admin startproject backend
   * Innerhalb des Projekts eine App namens core erstellt
4. **Modelle definiert (models.py)**
   * Benutzerdefiniertes User-Modell (User) mit Feldern is\_client und is\_provider
   * Modelle für ServiceType, ProviderProfile und Request definiert
5. **AUTH\_USER\_MODEL in den Einstellungen gesetzt**
   * AUTH\_USER\_MODEL = 'core.User' in settings.py
6. **Migrationen erstellt und angewendet**
   * python manage.py makemigrations core
   * python manage.py migrate (Probleme mit Migrationen wurden behoben)
7. **Superuser erstellt**
   * python manage.py createsuperuser
   * Benutzername, Email und Passwort eingegeben
8. **Im Django-Shell gearbeitet**
   * python manage.py shell geöffnet
   * Superuser abgefragt und sichergestellt, dass is\_staff und is\_superuser auf True gesetzt sind
9. **Django-Entwicklungsserver gestartet**
   * python manage.py runserver
   * Server läuft lokal unter <http://127.0.0.1:8000>

**✅ Ce ai făcut deja (aprox. 65% complet):**

**🔧 1. Setup proiect Django:**

* Instalare Django + DRF ✔
* Creare proiect și aplicație core ✔
* Configurare AUTH\_USER\_MODEL personalizat ✔
* Configurare settings.py corect ✔
* Pornit server local ✔

**🗃️ 2. Modele (models.py):**

* User personalizat ✔
* ServiceType ✔
* ProviderProfile ✔
* Request ✔

**🛠️ 3. Migrații:**

* Creat și aplicat migrațiile ✔

**🔐 4. Autentificare:**

* Creat superuser, verificat shell ✔

**🧪 5. Testat local:**

* Pornit server, accesat admin ✔

**🌐 6. API:**

* Creat RequestViewSet ✔
* Configurat router în core/urls.py ✔
* Inclus în backend/urls.py ✔

**🔜 Ce mai urmează (restul de 35%):**

**🔹 7. API complet:**

* ViewSets pentru User, ServiceType, ProviderProfile
* Serializers pentru toate modelele
* Autorizare (JWT sau TokenAuth)
* Validări personalizate (region, status etc.)

**🔹 8. Funcționalități avansate:**

* Upload fișiere (opțional)
* Business Logic (ex: matching automat)
* Filtrări (ex: după regiune sau status)
* Permisiuni: cine poate vedea/posta ce

**🔹 9. Finalizare:**

* Documentație API (Swagger/OpenAPI)
* (Opțional) Teste automate

✅ **Concluzie**:  
Ai terminat baza solidă și ți-ai configurat corect toate elementele fundamentale.  
Urmează partea de expunere a tuturor datelor prin API + autentificare → apoi frontend-ul (React).

Spune-mi dacă vrei să începem acum pasul următor: **serializers + API complet**.

✅ **Was ich bereits gemacht habe:**

🔧 **1. Django-Projekt Setup:**

* Django + Django REST Framework installiert ✔
* Projekt und App core erstellt ✔
* AUTH\_USER\_MODEL angepasst und konfiguriert ✔
* settings.py korrekt eingerichtet ✔
* Lokaler Server gestartet ✔

🗃️ **2. Datenmodelle (models.py):**

* Benutzerdefiniertes User-Modell ✔
* ServiceType-Modell ✔
* ProviderProfile-Modell ✔
* Request-Modell ✔

🛠️ **3. Migrationen:**

* Migrationen erstellt und angewendet ✔

🔐 **4. Authentifizierung:**

* Superuser erstellt und in der Shell getestet ✔

🧪 **5. Lokale Tests:**

* Server ausgeführt, Admin-Bereich aufgerufen ✔

🌐 **6. API:**

* RequestViewSet erstellt ✔
* Router in core/urls.py konfiguriert ✔
* In backend/urls.py eingebunden ✔

🔜 **Was noch fehlt:**

🔹 **7. API komplettieren:**

* ViewSets für User, ServiceType, ProviderProfile
* Serializers für alle Modelle
* Authentifizierung mit Token oder JWT
* Individuelle Validierungen (Region, Status etc.)

🔹 **8. Erweiterte Funktionen:**

* Datei-Uploads (optional)
* Business-Logik (z. B. automatisches Matching)
* Filter (z. B. nach Region oder Status)
* Berechtigungen: Wer darf was sehen oder senden

🔹 **9. Abschluss:**

* API-Dokumentation mit Swagger/OpenAPI
* (Optional) Automatisierte Tests