

Prototyping Projektdokumentation

Name: Darren Glatzl

E-Mail: glatzdar@students.zhaw.ch

Projekt: investify-ch.netlify.app

1. Einleitung

Investify – Ihre persönliche Investment-Plattform

Investify ist eine moderne Webanwendung, die privaten Nutzer*innen (Customers) einen sicheren und intuitiven Zugang zu Investitionen in verschiedene Finanzwerte (Assets) bietet. Der Name „Investify“ steht für einfache, transparente und technologiegestützte Portfolio-verwaltung auf Basis aktueller Marktdaten.

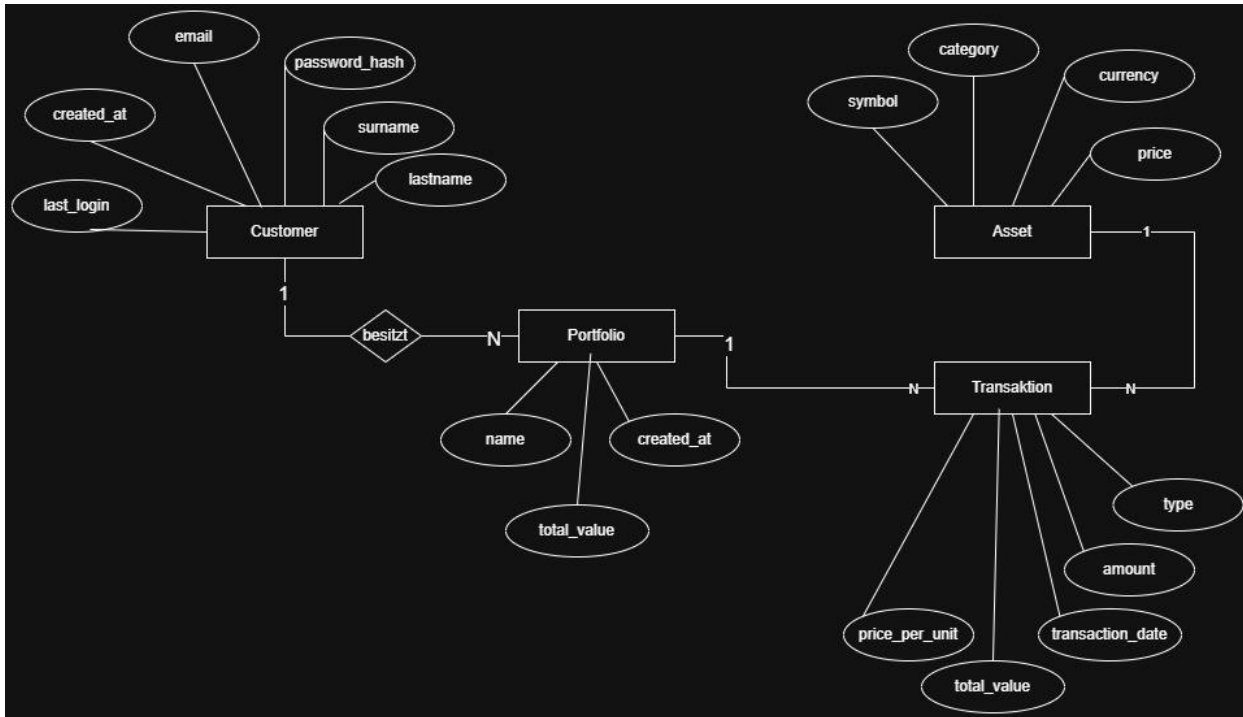
Idee

Viele Privatanleger*innen wünschen sich eine zentrale Anlaufstelle, um verfügbare Aktien, ETFs oder Kryptowährungen zu recherchieren und direkt zu kaufen, ohne zwischen mehreren Börsen, Brokern oder Datenportalen hin- und herzuschalten. Investify löst dieses Problem, indem es eine **interne API** für Registrierung, Authentifizierung und Portfoliomanagement bereitstellt und gleichzeitig via **externer API** stets aktuelle Asset-Preise sowie Kursverläufe lädt. So entsteht ein End-to-End-Workflow von der Anmeldung bis zu detaillierten Performance-Analysen.

Grundfunktionen

- Authentifizierung & Nutzerverwaltung:**
 - Sicheres Anlegen und Einloggen von Accounts (Passwort-Hashing, JWT/Cookie-basierte Sessions)
 - Geschützter Zugriff auf Portfolio-Funktionen
- Asset-Datenintegration:**
 - Abruf aktueller Echtzeit-Preise (z. B. Aktienkurse, Crypto-Quotes) über eine externe Datenquelle (REST-API)
 - Caching-Mechanismus, um Rate-Limits zu schonen und Ausfallzeiten abzufedern
 - Historische Kursdaten (z. B. 30-Tage-Verlauf) für Chart-Darstellungen
- Portfolio-Management:**
 - Jeder Kauf eines Assets wird in einer eigenen „portfolio“-Collection gespeichert (mit Feldern wie assetSymbol, amount, purchasePrice, purchaseDate)
 - Dashboard mit KPI-Panel (Gesamtinvestition, Gesamtwert, Gewinn/Verlust in € und %)
 - Dynamisches Pie-Chart zur Visualisierung der prozentualen Aufteilung nach Asset-Werten
 - Tabelle mit detaillierten Einträgen: Symbol, Menge, Kaufpreis, aktueller Wert und Klick-Navigation zu Asset-Detailseiten
- Asset-Detailseite:**
 - Interaktives Line-Chart für historische Preisverläufe (z. B. letzte 30 Tage)
 - Aktueller Kurs, 24-Stunden-Änderung und weitere Kennzahlen
 - Möglichkeit, direkt aus der Detailansicht einen neuen Kauf zu initiieren

2. Datenmodell



2.1. Kardinalitäten und Primär-/Fremdschlüssel im Überblick

- Kunde (customer_id) 1 — N Portfolio (portfolio_id)
Ein Kunde kann null oder mehrere Portfolios führen, jedes Portfolio gehört zu genau einem Kunden.
→ Fremdschlüssel: Portfolio.customer_id referenziert Kunde.customer_id.
- Portfolio (portfolio_id) 1 — N Transaktion (transaction_id)
Ein Portfolio kann beliebig viele Transaktionen enthalten; jede Transaktion ist genau einem Portfolio zugeordnet.
→ Fremdschlüssel: Transaktion.portfolio_id referenziert Portfolio.portfolio_id.
- Asset (symbol) 1 — N Transaktion (transaction_id)
Ein Asset kann in vielen Transaktionen gekauft/verkauft werden; jede Transaktion referenziert genau ein vorhandenes Asset.
→ Fremdschlüssel: Transaktion.asset_symbol referenziert Asset.symbol.

2.2. Funktionaler Ablauf und eingesetzte APIs

Benutzer-Authentifizierung (better-auth API)

- Bei Registrierung und Login nutzt das System die better-auth-API, um Passwörter sicher zu hashen, Sitzungscookies zu verwalten und Session-Daten (wie session.data.user) zu erzeugen.
- Nach erfolgreicher Authentifizierung steht der Customer's customer_id / Benutzername in jeder Server-Funktion (z. B. locals.user oder session.data.user) zur Verfügung, sodass nur berechtigte Nutzer auf ihre Portfolios und Transaktionen zugreifen können.

Verwaltung von Portfolios

- Ein eingeloggter User (Kunde) kann in der UI mehrere Portfolios (Portfolio) anlegen.
- Beim Anlegen eines neuen Portfolios wird in der Datenbank ein neuer Datensatz mit portfolio_id und dem FK customer_id des eingeloggten Users erstellt.

Durchführung von Käufen und Verkäufen (Transaktionen)

- Sobald ein User ein Asset kauft oder verkauft, entsteht ein Eintrag in der Entität „Transaktion“.
- In Transaktion werden u. a. gespeichert:
 - transaction_id (eindeutige ID der Buchung)
 - portfolio_id (zeigt, in welchem Portfolio gebucht wird)
 - asset_symbol (zeigt, um welches Asset es sich handelt)
 - transaction_date, amount (Menge), price_per_unit, total_value, type („BUY“/„SELL“).
- Dadurch bleibt jederzeit nachvollziehbar, wer welches Asset wann und zu welchem Preis gehandelt hat.

Asset-Datenpflege (yahoo-finance-api)

- In der Entität „Asset“ werden alle relevanten Wertpapier-Symbole (symbol) gepflegt.
- Um historische und aktuelle Marktdaten zu erhalten, nutzt das Projekt das NPM-Paket yahoo-finance-api.
- Beispiel-Ablauf: Bei jeder neuen Transaktion liest das Backend den aktuellen Kurs zu asset_symbol die API aus , um price_per_unit automatisch vorzubelegen bzw. den korrekten Tages-Schlusskurs zu verwenden.

3. Beschreibung der Anwendung

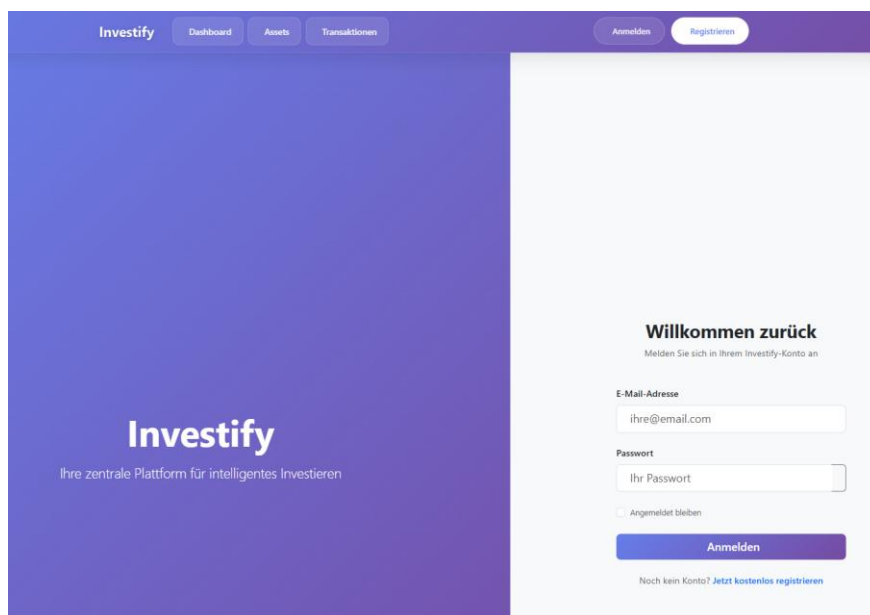
3.1. Authentifizierung

Route: /auth

Die Authentifizierung ist die zentrale Komponente für den Zugriff auf alle geschützten Bereiche der Anwendung. Das System nutzt Better Auth für die Session-Verwaltung.

3.1.1. Login (/auth/login)

Auf dieser Seite können sich bestehende Benutzer mit E-Mail und Passwort anmelden. Nach erfolgreicher Authentifizierung werden Benutzer automatisch zum Dashboard weitergeleitet. Die Session wird im authClient gespeichert und ist anwendungsweit verfügbar.



Dateien:

- routes/auth/login/+page.svelte
- routes/auth/login/+page.server.js
- lib/auth-client.js

3.1.2. Registrierung (/auth/register)

Neue Benutzer können hier ein Konto erstellen. Erforderliche Angaben sind Vorname, Nachname, E-Mail und Passwort. Nach erfolgreicher Registrierung erfolgt eine automatische Anmeldung und Weiterleitung zum Dashboard. Die Benutzerdaten werden in der MongoDB-Collection user gespeichert.

Dateien:

- routes/auth/register/+page.svelte
- routes/auth/register/+page.server.js
- lib/server/db.js (createUser-Funktion)

3.1.3. Logout (/auth/logout)

Beendet die aktuelle Benutzersession und leitet zur Login-Seite weiter.

Dateien:

- routes/auth/logout/+page.server.js

3.2. Dashboard

Route: /dashboard

Willkommen zurück, darren.glatz!

Hier ist Ihre Investitionsübersicht

Das Dashboard ist die zentrale Übersichtsseite für angemeldete Benutzer. Hier werden alle Portfolios des Benutzers mit ihren aktuellen Werten, Gewinnen/Verlusten und Performance-Metriken angezeigt.

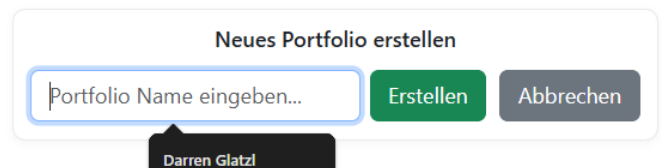
Hauptfunktionen:

- **KPI-Karten:** Zeigt Gesamtwert, Gesamtrendite, Tagesveränderung und investiertes Kapital
- **Portfolio-Verteilung:** Pie-Chart-Visualisierung aller Portfolios
- **Letzte Transaktionen:** Übersicht der 10 neuesten Käufe/Verkäufe
- **Portfolio-Liste:** Alle Portfolios mit individueller Performance

3.2.1. Portfolio erstellen:

Über den Button "Neues Portfolio erstellen" öffnet sich ein Modal-Dialog. Nach Eingabe eines Portfolio-Namens wird dieses über die Server-Action create angelegt.

Meine Portfolios



Dateien:

- routes/dashboard/+page.svelte
- routes/dashboard/+page.server.js
- lib/components/PortfolioCard.svelte
- lib/components/PieChart.svelte

3.3. Portfolio

Route: /portfolio

3.3.1. Portfolio-Details (/portfolio/[id])

Detailansicht eines spezifischen Portfolios mit drei Tabs:

- **Übersicht:** Performance-Chart, Asset-Verteilung, Top/Worst Performer
- **Positionen:** Tabellarische Ansicht aller Assets mit Quick-Actions
- **Transaktionen:** Chronologische Liste aller Käufe/Verkäufe

3.3.2. Quick Actions:

- **Buy:** Schnellkauf direkt aus der Positionsübersicht
- **Sell:** Schnellverkauf mit Bestandsprüfung

Dateien:

- routes/portfolio/[id]/+page.svelte
- routes/portfolio/[id]/+page.server.js
- lib/components/LineChart.svelte

3.3.3. Portfolio-Verwaltung:

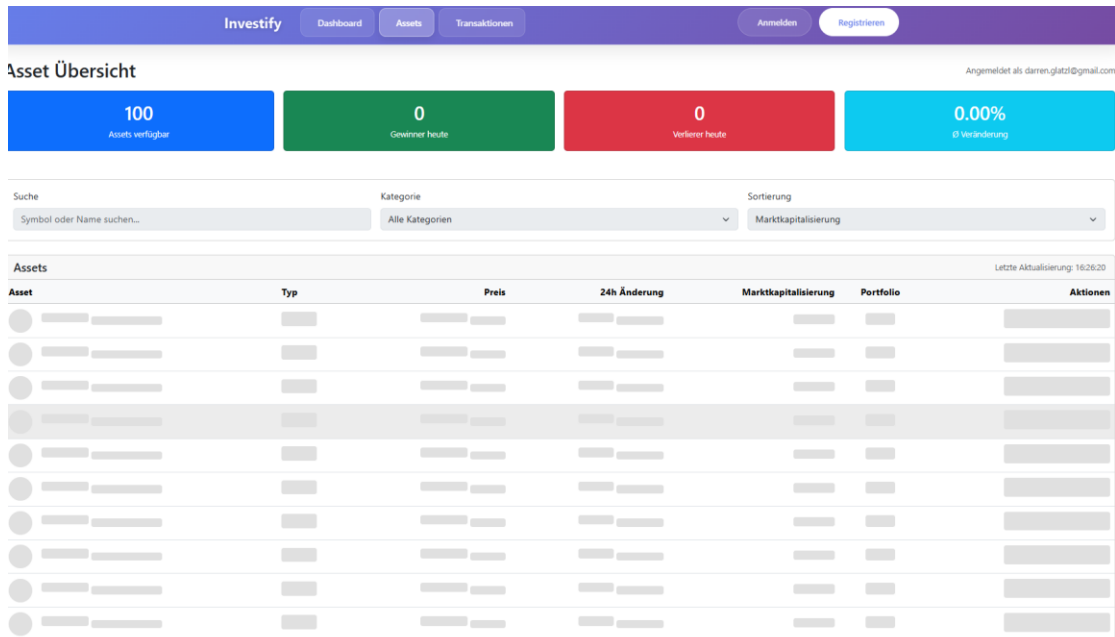
- **Create:** Erfolgt über das Dashboard mit der Action createPortfolio
- **Delete:** Funktion in lib/server/db.js vorhanden (deletePortfolio), aber keine dedizierte UI-Implementation ersichtlich

3.4. Assets

Route: /assets

3.4.1. Asset-Übersicht (/assets)

Listet alle verfügbaren Assets (Aktien, Krypto, ETFs) mit Filtermöglichkeiten nach Kategorie, Suchfunktion und verschiedenen Sortieroptionen. Zeigt für jedes Asset aktuelle Kurse, 24h-Änderung und Portfolio-Positionen des Benutzers.



Dateien:

- routes/assets/+page.svelte
- routes/assets/+page.server.js
- lib/server/assets.js

3.4.2. Asset-Details (/assets/[symbol])

Detailseite für einzelne Assets mit:

- Kursverlauf (30-Tage-Chart)
- Marktdaten (Kapitalisierung, Volumen, Sektor)
- Benutzer-Position (falls vorhanden)
- Ähnliche Assets

3.4.3. Trading-Funktionen:

- **Buy-Modal:**
 - Portfolio-Auswahl
 - Mengenangabe
 - Echtzeit-Preisberechnung
 - Transaktion wird über addTransaction in der DB gespeichert
- **Sell-Modal:**
 - Nur verfügbar bei vorhandener Position
 - Bestandsprüfung
 - Gewinn/Verlust-Anzeige
 - Validierung der Verkaufsmenge

Dateien:

- routes/assets/[symbol]/+page.svelte
- routes/assets/[symbol]/+page.server.js
- lib/server/db.js (addTransaction-Funktion)

4. Technische Abhängigkeiten

Authentifizierung:

- Alle geschützten Routes prüfen die Session über locals.user oder die auth()-Funktion
- Nicht authentifizierte Benutzer werden automatisch zu /auth/login weitergeleitet
- Die Session wird im +layout.svelte global verwaltet

Datenfluss:

1. **Server-Side:** Daten werden in +page.server.js aus MongoDB geladen
2. **Client-Side:** Better Auth Session wird für UI-States verwendet
3. **Form Actions:** Portfolio-Erstellung und Transaktionen nutzen SvelteKit Form Actions

Sicherheit:

- Portfolio-Zugriff wird serverseitig validiert (Benutzer kann nur eigene Portfolios sehen)
- Alle Transaktionen werden mit Benutzer-ID verknüpft
- Verkäufe werden gegen verfügbare Bestände geprüft

5. Erweiterungen

5.1. Komplexes Authentifizierungssystem mit Better Auth

- Implementierung:
 - /routes/auth/login/+page.svelte
 - /routes/auth/register/+page.svelte
 - /lib/auth-client.js
 - /lib/server/auth.js
 - src/hooks.server.js
- Features: Session-Management, Auto-Login nach Registrierung, Protected Routes
- Besonderheit: Integration von Better Auth Library statt einfacher Cookie-Lösung

5.2. Real-Time Asset-Preisabfrage über externe APIs (yahoo finace-api)

- Implementierung:
 - /lib/server/assets.js → fetchAssetPrice(), fetchStockPrice(), fetchCryptoPrice()
 - /routes/api/assets/+server.js
 - /lib/server/db.js → getCachedPrice(), setCachedPrice()
- Features: Integration von Alpha Vantage und CoinDesk APIs
- Caching: Implementierter Price-Cache zur API-Optimierung

5.3. Interaktive Datenvisualisierung

- PieChart: /lib/components/PieChart.svelte - Portfolio-Verteilung
- LineChart: /lib/components/LineChart.svelte - Performance-Charts
- Implementierung:
 - /lib/components/PieChart.svelte
 - /lib/components/LineChart.svelte
 - /routes/dashboard/+page.svelte
 - /routes/portfolio/[id]/+page.svelte
- Features: Chart.js Integration, responsive Design, dynamische Updates

5.4. Portfolio-Performance-Tracking

Implementierung:

- /routes/dashboard/+page.server.js → load() Funktion, Zeile 20-75
- /lib/server/db.js → calculatePortfolioValue(), getPortfolioPositions()
- /routes/portfolio/[id]/+page.server.js

Features:

- Echtzeit-Gewinn/Verlust-Berechnung
- Tagesveränderungen
- Portfolio-übergreifende Aggregation
- Top/Worst Performer Analyse

5.5. Mehrstufige Portfolio-Verwaltung

- Portfolio-Ebene: Mehrere Portfolios pro User (/routes/portfolio/[id])
- Positions-Management: Aggregierte Positionen aus Transaktionen
- Features: FIFO-Durchschnittspreisberechnung, Portfolio-Isolation

5.6. Quick-Trading Funktionalität

- Implementierung: /routes/portfolio/[id]/+page.server.js - Actions quickBuy, quickSell
- Modal-Dialoge: Direkte Käufe/Verkäufe aus Portfolio- und Asset-Übersicht
- Validierung: Bestandsprüfung, Echtzeit-Preisberechnung

5.7. Transaction History mit Audit Trail

- Implementierung: Vollständige Transaktionshistorie pro Portfolio
- Features: Chronologische Darstellung, Portfolio-übergreifende Ansicht
- Datenpersistenz: Unveränderliche Transaktions-Records

5.8. Error Handling und User Feedback

- Implementierung: Durchgängig in allen Server-Actions
- Features:
 - Toast-Notifications
 - Form-Validierung mit spezifischen Fehlermeldungen
 - Graceful Degradation bei API-Fehlern

5.9. Sicherheitsfeatures

- Portfolio-Isolation: Serverseitige Zugriffsprüfung (`portfolio.customer_id === user.id`)
- Input-Validierung: Umfangreiche Validierung aller User-Inputs
- Password-Hashing: bcrypt-Implementation in `/lib/server/db.js`