



ETF Portfolio Manager - Documentazione Tecnica Completa

Indice

1. [Panoramica del Progetto](#)
 2. [Architettura del Sistema](#)
 3. [Struttura del Database](#)
 4. [Sistema di Autenticazione Multi-Utente](#)
 5. [Gestione Portfolio e Sessioni Guest](#)
 6. [Backend - Server e API](#)
 7. [Frontend - Interfaccia Utente](#)
 8. [Dashboard Amministrativo](#)
 9. [Analytics e Reporting](#)
 10. [Sistema di Alert](#)
 11. [Docker e Deployment](#)
 12. [Configurazione e Environment](#)
 13. [API Endpoints](#)
 14. [Utility e Helper](#)
 15. [Sicurezza](#)
-

1. Panoramica del Progetto

1.1 Descrizione

ETF Portfolio Manager è una piattaforma web professionale per la gestione di portafogli di investimenti in ETF (Exchange-Traded Funds). L'applicazione offre funzionalità complete per il tracking, l'analisi e l'ottimizzazione degli investimenti con supporto multi-utente completo e gestione sessioni guest avanzata.






1.2 Caratteristiche Principali

- 👤 **Sistema Multi-Utente Completo:** Autenticazione, registrazione e gestione utenti
- 👥 **Separazione Dati per Utente:** Ogni utente vede solo i propri portfolio
- 👤 **Gestione Sessioni Guest:** Portfolio creati da guest vengono associati dopo login
- 📊 **Multi-Portfolio Support:** Gestione di portafogli multipli per utente
- ⚡ **Real-Time Data:** Aggiornamenti automatici dei prezzi ETF
- 📈 **Advanced Analytics:** Metriche di performance avanzate (Sharpe ratio, volatilità)
- 🛡️ **Admin Dashboard:** Interfaccia amministrativa completa
- 🐳 **Docker Ready:** Containerizzazione completa per deployment facile
- 🗄️ **Database Flessibile:** Supporto PostgreSQL e SQLite

1.3 Tecnologie Utilizzate

- Backend:** Node.js, Express.js
- Database:** SQLite (attuale), PostgreSQL (pianificato)
- Frontend:** EJS Templates, Bootstrap 5, Chart.js
- Authentication:** Session-based con express-session
- Security:** bcryptjs per password hashing
- Containerization:** Docker, Docker Compose

1.4 Novità Versione 3.0

-  **Sistema di Autenticazione Completo**
-  **Registrazione e Login Utenti**
-  **Associazione Automatica Portfolio Guest**
-  **Multi-Tenancy con Separazione Dati**
-  **Utenti Demo Predefiniti**

2. Architettura del Sistema

2.1 Architettura MVC

L'applicazione segue il pattern Model-View-Controller:

```
|— models/          # Modelli di dati (M)
|— views/           # Templates EJS (V)
|— controllers/     # Logica business (C)
|— routes/          # Routing delle richieste
|— middleware/      # Middleware Express
|— config/          # Configurazioni
```

2.2 Flusso delle Richieste

1. **Client Request** → **Express Router**
2. **Router** → **Middleware** (Auth, Validation)
3. **Middleware** → **Controller**
4. **Controller** → **Model** (Database)
5. **Model** → **Controller** (Dati)
6. **Controller** → **View** (Template)
7. **View** → **Client** (HTML Response)

2.3 Componenti Principali

- **Server.js**: Entry point dell'applicazione
- **Routes**: Gestione delle rotte HTTP
- **Controllers**: Logica di business
- **Models**: Interfaccia con il database
- **Middleware**: Autenticazione e validazione
- **Views**: Template engine per il frontend

3. Struttura del Database

3.1 Schema Principale (SQLite - Attuale)

Tabella Users

```
CREATE TABLE users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  username TEXT UNIQUE NOT NULL,
  email TEXT UNIQUE NOT NULL,
  password_hash TEXT NOT NULL,
  first_name TEXT,
```

```
last_name TEXT,  
role TEXT DEFAULT 'user',  
subscription_tier TEXT DEFAULT 'basic',  
is_active BOOLEAN DEFAULT 1,  
email_verified BOOLEAN DEFAULT 1,  
created_at DATETIME DEFAULT CURRENT_TIMESTAMP,  
updated_at DATETIME DEFAULT CURRENT_TIMESTAMP  
);
```

Tabella Portfolios

```
CREATE TABLE portfolios (  
  id INTEGER PRIMARY KEY AUTOINCREMENT,  
  user_id INTEGER,  
  name TEXT NOT NULL,  
  description TEXT,  
  is_default BOOLEAN DEFAULT 0,  
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP,  
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE SET NULL  
);
```

Tabella Investments

```
CREATE TABLE investments (  
  id INTEGER PRIMARY KEY AUTOINCREMENT,  
  user_id INTEGER,  
  portfolio_id INTEGER NOT NULL,  
  ticker TEXT NOT NULL,  
  name TEXT,  
  shares REAL NOT NULL,  
  buy_price REAL NOT NULL,  
  buy_date DATE NOT NULL,  
  current_price REAL,  
  last_updated DATETIME DEFAULT CURRENT_TIMESTAMP,  
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE SET NULL,  
  FOREIGN KEY (portfolio_id) REFERENCES portfolios(id) ON DELETE CASCADE  
);
```

Tabella Alerts

```
CREATE TABLE alerts (  
  id INTEGER PRIMARY KEY AUTOINCREMENT,  
  user_id INTEGER,  
  portfolio_id INTEGER,  
  ticker TEXT,  
  alert_type TEXT NOT NULL,  
  target_value REAL,  
  condition_type TEXT,  
  is_active BOOLEAN DEFAULT 1,
```

```

    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    triggered_at DATETIME,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (portfolio_id) REFERENCES portfolios(id) ON DELETE CASCADE
);

```

3.2 Gestione Multi-Utente

- **user_id NULL:** Portfolio/investimenti guest (prima del login)
- **user_id SET:** Portfolio/investimenti associati a utente specifico
- **Isolamento Dati:** Ogni utente vede solo i propri dati
- **Associazione Automatica:** Portfolio guest vengono associati dopo login

4. Sistema di Autenticazione Multi-Utente

4.1 Architettura Autenticazione

Il sistema utilizza **express-session** per la gestione delle sessioni web:

```

// Configurazione sessioni
app.use(session({
  secret: process.env.SESSION_SECRET || 'etf-portfolio-secret-key',
  resave: false,
  saveUninitialized: false,
  cookie: {
    secure: false,
    httpOnly: true,
    maxAge: 24 * 60 * 60 * 1000 // 24 ore
  }
}));

```

4.2 Middleware di Autenticazione

requireAuth - Protegge Route Autenticate

```

export const requireAuth = (req, res, next) => {
  if (!req.session || !req.session.user) {
    return res.redirect('/auth/login');
  }
  next();
};

```

requireGuest - Solo Utenti Non Autenticati

```

export const requireGuest = (req, res, next) => {
  if (req.session && req.session.user) {
    return res.redirect('/');
  }
}

```

```
    next();  
  }  
};
```

4.3 Controller di Autenticazione

Registrazione Utente

```
export const registerUser = async (req, res) => {  
  const { username, email, password, first_name, last_name } = req.body;  
  
  // Hash password con bcrypt  
  const hashedPassword = await bcrypt.hash(password, 12);  
  
  // Crea utente  
  const result = await runQuery(`  
    INSERT INTO users (username, email, password_hash, first_name, last_name)  
    VALUES (?, ?, ?, ?, ?)  
  `, [username, email, hashedPassword, first_name, last_name]);  
  
  // Associa portfolio guest all'utente  
  await associateGuestDataToUser(req.sessionID, result.lastID);  
  
  // Crea sessione  
  req.session.user = { id: result.lastID, username, email, role: 'user' };  
  res.redirect('/');  
};
```

Login Utente

```
export const loginUser = async (req, res) => {  
  const { username, password } = req.body;  
  
  // Trova utente  
  const user = await get(`  
    SELECT id, username, email, password_hash, role  
    FROM users WHERE (username = ? OR email = ?) AND is_active = 1  
  `, [username, username]);  
  
  // Valida password  
  const isValidPassword = await bcrypt.compare(password, user.password_hash);  
  
  if (isValidPassword) {  
    // Associa portfolio guest  
    await associateGuestDataToUser(req.sessionID, user.id);  
  
    // Crea sessione  
    req.session.user = { id: user.id, username: user.username, email: user.email, role:  
user.role };  
    res.redirect('/');
```

```
}  
};
```

4.4 Utenti Demo Predefiniti

Il sistema include utenti demo per test immediato:

Username	Password	Ruolo	Descrizione
admin	Admin123!	admin	Amministratore sistema
demo_user	DemoUser123!	user	Utente standard demo
test_premium	Premium123!	user	Utente premium test

5. Gestione Portfolio e Sessioni Guest

5.1 Sistema Guest-to-User

Funzionalità chiave che permette agli utenti non autenticati di creare portfolio che vengono poi associati automaticamente dopo login/registrazione.

Flow Operativo:

1. **Guest** accede alla homepage senza autenticazione
2. **Guest** crea portfolio e investimenti (salvati con `user_id = NULL`)
3. **Guest** decide di registrarsi o fare login
4. **Sistema** associa automaticamente tutti i portfolio guest all'utente

5.2 Funzione di Associazione

```
async function associateGuestDataToUser(sessionId, userId) {  
  // Trova portfolio guest (user_id IS NULL)  
  const guestPortfolios = await getAll(`  
    SELECT id, name, description FROM portfolios  
    WHERE user_id IS NULL AND id > 1  
  `);  
  
  // Associa ogni portfolio all'utente  
  for (const portfolio of guestPortfolios) {  
    // Aggiorna portfolio  
    await runQuery(`  
      UPDATE portfolios  
      SET user_id = ?, updated_at = CURRENT_TIMESTAMP  
      WHERE id = ?  
    `, [userId, portfolio.id]);  
  
    // Aggiorna investimenti del portfolio  
    await runQuery(`  
      UPDATE investments  
      SET user_id = ?, last_updated = CURRENT_TIMESTAMP  
      WHERE portfolio_id = ?  
    `);  
  }  
}
```

```

        `, [userId, portfolio.id]);
    }
}

```

5.3 Controller Portfolio Multi-Utente

```

// Ottieni portfolio per utente specifico
export function getPortfolios(userId = null) {
  if (userId) {
    // Portfolio utente autenticato
    return getAll(
      'SELECT id, name, description, created_at FROM portfolios WHERE user_id = ?
ORDER BY id',
      [userId]
    );
  } else {
    // Portfolio guest
    return getAll(
      'SELECT id, name, description, created_at FROM portfolios WHERE user_id IS NULL
ORDER BY id'
    );
  }
}

// Crea portfolio associato a utente
export function createPortfolio(portfolio, userId = null) {
  const { name, description } = portfolio;
  return runQuery(
    'INSERT INTO portfolios (name, description, user_id) VALUES (?, ?, ?)',
    [name.trim(), description || '', userId]
  );
}

```

5.4 Separazione Dati Multi-Tenant

- **Homepage:** Mostra solo portfolio dell'utente autenticato
- **Guest Access:** Portfolio temporanei per utenti non autenticati
- **Data Isolation:** Ogni utente vede esclusivamente i propri dati
- **Automatic Migration:** Portfolio guest → utente dopo autenticazione

6. Backend - Server e API

6.1 Configurazione Server Express

```

// server.js - Configurazione principale
import express from 'express';
import session from 'express-session';
import { setUserLocals } from './middleware/auth.js';

```

```

const app = express();

// Middleware sessioni
app.use(session({
  secret: process.env.SESSION_SECRET || 'etf-portfolio-secret-key',
  resave: false,
  saveUninitialized: false,
  cookie: { secure: false, httpOnly: true, maxAge: 24 * 60 * 60 * 1000 }
}));

// Middleware autenticazione
app.use(setUserLocals);

// Route principali
app.use('/', indexRoutes);
app.use('/auth', authRoutes);
app.use('/admin', adminRoutes);

```

6.2 Route Principali

Route Autenticazione (/auth)

- GET /auth/login - Pagina login
- POST /auth/login - Processo login
- GET /auth/register - Pagina registrazione
- POST /auth/register - Processo registrazione
- POST /auth/logout - Logout utente
- GET /auth/profile - Profilo utente

Route Portfolio (/)

- GET / - Homepage con portfolio (guest o autenticato)
- POST /add - Aggiungi investimento
- GET /portfolios - API lista portfolio utente
- POST /portfolios/create - API crea portfolio
- PUT /portfolios/:id - API aggiorna portfolio

Route Admin (/admin)

- GET /admin - Dashboard amministrativo (solo admin)
- GET /admin/users - Gestione utenti (solo admin)

6.3 Middleware Personalizzati

setUserLocals - Variabili Globali Template

```

export const setUserLocals = (req, res, next) => {
  res.locals.user = req.session?.user || null;
  res.locals.isAuthenticated = !!req.session?.user;
  res.locals.isAdmin = req.session?.user?.role === 'admin';
  next();
};

```


6.4 Database SQLite Integration

```
// config/database.js
import sqlite3 from 'sqlite3';

// Connection pool management
let _db = null;

export async function get(query, params = []) {
  const db = await getDb();
  return new Promise((resolve, reject) => {
    db.get(query, params, (err, row) => {
      if (err) reject(err);
      else resolve(row);
    });
  });
}

export async function runQuery(query, params = []) {
  const db = await getDb();
  return new Promise((resolve, reject) => {
    db.run(query, params, function(err) {
      if (err) reject(err);
      else resolve({ lastID: this.lastID, changes: this.changes });
    });
  });
}
```

7. Frontend - Interfaccia Utente

7.1 Template Engine - EJS

L'applicazione utilizza EJS per il rendering server-side:

```
views/
├─ layouts/
│   └─ main.ejs      # Layout principale
├─ partials/
│   └─ header.ejs    # Componenti riutilizzabili
├─ admin/
│   └─ dashboard.ejs # Dashboard admin
│   └─ users.ejs     # Gestione utenti
├─ index.ejs         # Homepage
└─ error.ejs         # Pagina errori
```

7.2 Assets Statici

CSS (public/css/style.css)

- **Bootstrap 5:** Framework CSS responsive
- **Custom Styles:** Personalizzazioni per il tema

- **Dark/Light Mode:** Supporto tema scuro/chiaro

JavaScript (public/js/)

- **main.js:** Funzionalità generali
- **charts.js:** Gestione grafici Chart.js

7.3 Responsive Design

- **Mobile First:** Design ottimizzato per mobile
 - **Bootstrap Grid:** Sistema di griglia responsive
 - **Touch Friendly:** Interfaccia touch-friendly
-

8. Dashboard Amministrativo

8.1 Funzionalità Admin

- **User Management:** CRUD completo utenti
- **System Stats:** Statistiche sistema real-time
- **Monitoring:** Monitoraggio performance
- **Logs:** Visualizzazione logs sistema

8.2 Routes Admin

File: `routes/admin/adminRoutes.js`

- `GET /admin/dashboard` - Dashboard principale
- `GET /admin/users` - Lista utenti
- `POST /admin/users` - Creazione utenti
- `PUT /admin/users/:id` - Aggiornamento utenti
- `DELETE /admin/users/:id` - Eliminazione utenti

8.3 Security Admin

- **Role-Based Access:** Solo utenti admin
 - **Session Management:** Gestione sessioni admin
 - **Audit Logging:** Log delle operazioni admin
-

9. Analytics e Reporting

9.1 Metriche Calcolate

- **Total Value:** Valore totale portfolio
- **Profit/Loss:** Profitti/perdite realizzate
- **Performance %:** Percentuale di performance
- **Sharpe Ratio:** Rapporto rischio/rendimento
- **Volatility:** Volatilità del portfolio
- **Diversification Score:** Punteggio diversificazione

9.2 Grafici e Visualizzazioni

Chart.js Implementation:

- **Pie Chart:** Allocazione assets
- **Line Chart:** Performance nel tempo
- **Bar Chart:** Comparazione investimenti

- **Doughnut Chart:** Distribuzione settori

9.3 Reports

- **Performance Report:** Report performance completo
 - **Holdings Report:** Report partecipazioni
 - **Transaction History:** Storico transazioni
-

10. Sistema di Alert

10.1 Tipi di Alert

- **Price Alerts:** Alert sui prezzi
- **Performance Alerts:** Alert su performance
- **Portfolio Alerts:** Alert sul portfolio
- **System Alerts:** Alert di sistema

10.2 Configurazione Alert

```
// Struttura alert
{
  ticker: 'VTI',
  alert_type: 'price',
  threshold_value: 250.00,
  condition_type: 'above',
  is_active: true
}
```

10.3 Processing Engine

- **Price Monitoring:** Monitoraggio prezzi real-time
 - **Condition Evaluation:** Valutazione condizioni
 - **Notification Dispatch:** Invio notifiche
-

11. Docker e Deployment

11.1 Dockerfile

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

11.2 Docker Compose

File: docker-compose.yml

Servizi configurati:

- **app**: Applicazione Node.js
- **postgres**: Database PostgreSQL
- **redis**: Cache Redis
- **nginx**: Reverse proxy
- **adminer**: Interface database

11.3 Deployment Stack

- **Production**: PostgreSQL + Redis + Nginx
 - **Development**: SQLite + Local server
 - **Testing**: In-memory database
-

12. Configurazione e Environment

12.1 File di Configurazione

- **.env**: Variabili ambiente produzione
- **.env.docker**: Template Docker
- **config/database.js**: Configurazione database
- **config/postgresql.js**: Setup PostgreSQL

12.2 Variabili Ambiente

```
NODE_ENV=production
PORT=3000
JWT_SECRET=your-secret-key
POSTGRES_HOST=localhost
POSTGRES_PORT=5432
POSTGRES_DB=etf_portfolio
POSTGRES_USER=username
POSTGRES_PASSWORD=password
```

12.3 Migration Scripts

File: scripts/migrate.js

- Migrazione SQLite → PostgreSQL
 - Backup automatico
 - Rollback capability
 - Data validation
-

13. API Endpoints

13.1 Authentication Routes

```
POST /auth/register  # Registrazione utente
POST /auth/login     # Login utente
POST /auth/logout    # Logout utente
GET  /auth/profile   # Profilo utente
```

13.2 Portfolio Routes

GET	/portfolios	# Lista portfolios
POST	/portfolios	# Crea portfolio
GET	/portfolios/:id	# Dettagli portfolio
PUT	/portfolios/:id	# Aggiorna portfolio
DELETE	/portfolios/:id	# Elimina portfolio

13.3 Investment Routes

GET	/investments	# Lista investimenti
POST	/investments	# Crea investimento
GET	/investments/:id	# Dettagli investimento
PUT	/investments/:id	# Aggiorna investimento
DELETE	/investments/:id	# Elimina investimento

13.4 Analytics Routes

GET	/analytics/dashboard	# Dashboard analytics
GET	/analytics/performance	# Performance metrics
GET	/analytics/allocation	# Asset allocation
GET	/analytics/reports	# Generazione reports

13.5 Admin Routes

GET	/admin/dashboard	# Dashboard admin
GET	/admin/users	# Lista utenti
POST	/admin/users	# Crea utente
PUT	/admin/users/:id	# Aggiorna utente
DELETE	/admin/users/:id	# Elimina utente
GET	/admin/stats	# Statistiche sistema

14. Utility e Helper

14.1 Price Utilities

File: `utils/priceUtils.js`

- **fetchETFPrice()**: Recupero prezzi ETF
- **updatePortfolioPrices()**: Aggiornamento prezzi portfolio
- **calculatePerformance()**: Calcolo performance
- **getMarketData()**: Dati mercato

14.2 Helper Functions

```
// Esempi di utility functions
export const formatCurrency = (amount) => {
  return new Intl.NumberFormat('it-IT', {
    style: 'currency',
```

```
        currency: 'EUR'
    }).format(amount);
};

export const calculateChange = (current, previous) => {
    return ((current - previous) / previous) * 100;
};
```

15. Sicurezza

15.1 Misure di Sicurezza Implementate

- **Password Hashing:** bcrypt per hash password
- **JWT Tokens:** Autenticazione basata su token
- **Input Validation:** Validazione input con express-validator
- **Rate Limiting:** Protezione contro attacchi DDoS
- **CORS Protection:** Configurazione CORS
- **Helmet.js:** Security headers HTTP

15.2 Data Protection

- **User Isolation:** Isolamento dati per utente
- **SQL Injection:** Protezione via ORM Sequelize
- **XSS Protection:** Escape automatico template EJS
- **CSRF Protection:** Token CSRF per form

15.3 Admin Security

- **Role-Based Access:** Controllo accessi basato su ruoli
 - **Session Management:** Gestione sicura sessioni
 - **Audit Logging:** Registrazione operazioni sensibili
-

16. Performance e Ottimizzazione

16.1 Database Optimization

- **Indexes:** Indici ottimizzati per query frequenti
- **Connection Pooling:** Pool connessioni database
- **Query Optimization:** Query ottimizzate con Sequelize

16.2 Caching Strategy

- **Redis Cache:** Cache per dati frequenti
- **Static Assets:** Cache per file statici
- **Database Results:** Cache risultati query

16.3 Frontend Performance

- **Minification:** Minificazione CSS/JS
 - **Compression:** Compressione gzip
 - **Lazy Loading:** Caricamento differito contenuti
-

17. Monitoraggio e Logging

17.1 Logging System

- **Winston Logger:** Sistema logging avanzato
- **Log Levels:** Debug, Info, Warn, Error
- **File Rotation:** Rotazione automatica log
- **Structured Logging:** Log strutturati JSON

17.2 Health Monitoring

- **Health Check Endpoint:** /health
- **Database Connectivity:** Verifica connessione DB
- **System Metrics:** CPU, memoria, disk usage
- **Application Metrics:** Response time, error rate

17.3 Error Handling

- **Global Error Handler:** Gestione errori centralizzata
 - **Custom Error Classes:** Classi errore personalizzate
 - **Error Reporting:** Reporting automatico errori
 - **Graceful Degradation:** Gestione elegante fallimenti
-

18. Deployment e DevOps

18.1 Deployment Options

1. **Docker Compose:** Deployment locale/staging
2. **Kubernetes:** Deployment produzione scalabile
3. **Traditional Server:** Deployment server tradizionale

18.2 CI/CD Pipeline

```
# Esempio GitHub Actions
name: Deploy ETF Portfolio Manager
on:
  push:
    branches: [main]
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Build Docker Image
      - name: Deploy to Production
```

18.3 Environment Management

- **Development:** Ambiente sviluppo locale
 - **Staging:** Ambiente test pre-produzione
 - **Production:** Ambiente produzione
-

19. Troubleshooting e Manutenzione

19.1 Common Issues

- **Database Connection:** Problemi connessione DB
- **Authentication Errors:** Errori autenticazione
- **Performance Issues:** Problemi performance
- **Memory Leaks:** Perdite memoria

19.2 Maintenance Tasks

- **Database Backup:** Backup automatici database
- **Log Cleanup:** Pulizia log obsoleti
- **Performance Monitoring:** Monitoraggio continuo
- **Security Updates:** Aggiornamenti sicurezza

19.3 Support Tools

- **Adminer:** Interface database
 - **Health Dashboard:** Dashboard stato sistema
 - **Log Viewer:** Visualizzatore log
 - **Performance Monitor:** Monitor performance
-

20. Roadmap e Future Enhancements

20.1 Planned Features

- **Mobile App:** Applicazione mobile React Native
- **Advanced Analytics:** Analytics AI-powered
- **Social Features:** Sharing e social trading
- **API Public:** API pubblica per integrazioni

20.2 Technical Improvements

- **Microservices:** Architettura microservizi
- **GraphQL:** API GraphQL
- **Real-time Updates:** WebSocket per aggiornamenti real-time
- **Machine Learning:** ML per raccomandazioni investimenti




20.3 Scalability Enhancements




- **Load Balancing:** Bilanciamento carico
 - **Database Sharding:** Sharding database
 - **CDN Integration:** Rete distribuzione contenuti
 - **Auto-scaling:** Scalabilità automatica
-

Conclusioni

ETF Portfolio Manager rappresenta una soluzione completa e professionale per la gestione di portafogli ETF, implementata con tecnologie moderne e best practices di sviluppo. L'architettura modulare e scalabile permette facilità di manutenzione e futuro sviluppo.

Punti di Forza

-  **Architettura Solida:** MVC pattern ben implementato
-  **Sicurezza Robusta:** Implementazione sicurezza enterprise-grade
-  **Scalabilità:** Design scalabile con Docker e PostgreSQL

-  **User Experience:** Interface moderna e responsive
-  **Admin Tools:** Dashboard amministrativo completo
-  **Documentation:** Documentazione completa e dettagliata

Ready for Production

L'applicazione è pronta per il deployment in produzione con tutte le funzionalità core implementate e testate.

Versione: 2.0.0

Data: Dicembre 2024

Autore: ETF Manager Team

Licenza: MIT