

InvestmentSim

HADIYA ALEXANDRA RAILEANU

Fecha de finalización: 23 de octubre de 2025

TFM - Introducción a Python

ÍNDICE DE CONTENIDOS

1. PRESENTACIÓN DEL PROYECTO
 2. STACK TECNOLÓGICO Y ALTERNATIVAS EVALUADAS
 3. ESTRUCTURA DEL PROYECTO
 4. BASE DE DATOS (models.py)
 5. FUNCIONES PRINCIPALES (main.py)
 6. MANUAL DE INSTALACIÓN
 7. PREGUNTAS Y RESPUESTAS
 8. NÚMEROS DEL PROYECTO
 9. POSIBLES MEJORAS A FUTURO
 10. PEQUEÑO RESUMEN
 11. EJEMPLOS DEL FUNCIONAMIENTO
 12. CONCLUSIONES
-

1. PRESENTACIÓN DEL PROYECTO

¿Qué es InvestmentSim?

InvestmentSim es una aplicación web interactiva desarrollada con Python que simula inversiones en el mercado financiero real. Esto permite a los usuarios experimentar con la compra y venta de acciones sin riesgo económico, utilizando datos reales del mercado

obtenidos en tiempo real. Esta está programada en inglés, para mayor inclusión. En esta web será donde:

- Los usuarios se registran, con 10,000€ iniciales de saldo (**¡NO REALES!**)
- Compran y venden acciones reales con su correspondiente valor de mercado y el valor según la cantidad que queramos comprar o vender (Apple, Microsoft, Google, etc.)
- Se muestran gráficos históricos de precios, facilitando el análisis de las inversiones
- Se puede gestionar un portafolio personal donde aparecerán todas las operaciones realizadas

2. STACK TECNOLÓGICO Y ALTERNATIVAS EVALUADAS

Backend:

- **Python 3.13** - Lenguaje de programación principal
- **Flask 3.0.0** - Framework web ligero
- **Flask-Login 0.6.3** - Gestión de sesiones de usuario
- **Flask-SQLAlchemy 3.1.1** - ORM para base de datos
- **Werkzeug 3.0.1**

Base de datos:

- **SQLite 3** - Base de datos

Obtención de datos:

- **yfinance 0.2.32** - API para datos de Yahoo Finance
- **pandas 2.1.3** - Manipulación de datos financieros

Visualización:

- **Matplotlib 3.8.2** - Generación de gráficos históricos

Frontend:

- **Jinja2** - Motor de plantillas HTML
- **Tailwind CSS (CDN)** - Framework CSS para el diseño
- **HTML5** - Estructura de páginas

A futuro, las alternativas a tener en cuenta podrían ser: Django (framework y base de datos), Alpha Vantage (finanzas), Bootstrap (diseño) o Plotly / Chart.js (gráficos).

3. ESTRUCTURA DEL PROYECTO

```
TFM/
├── main.py          # Archivo principal (rutas y lógica)
├── config.py        # Configuración
├── models.py        # Modelos de base de datos
├── finance_fetch.py # Obtención de precios
├── requirements.txt # Librerías necesarias
├── app.db            # Base de datos
└── templates/
    ├── base.html
    ├── index.html
    ├── login.html
    ├── register.html
    ├── dashboard.html
    ├── asset.html
    └── portfolio.html
```

4. BASE DE DATOS (models.py)

Los 4 modelos principales:

1. User (Usuario)

```
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(120))
    email = db.Column(db.String(120), unique=True)
    password_hash = db.Column(db.String(128))
    cash = db.Column(db.Float, default=10000.0)
```

Guarda: Los datos del usuario y su dinero disponible.

2. Asset (Acción)

```
class Asset(db.Model):
    ticker = db.Column(db.String(20), unique=True)
    name = db.Column(db.String(200))
    last_price = db.Column(db.Float)
    holdings = db.relationship('Holding', backref='asset')
```

Aquí se guarda la información de cada acción y su precio actual.

3. Holding (Posesión)

```
class Holding(db.Model):
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    asset_id = db.Column(db.Integer, db.ForeignKey('asset.id'))
    quantity = db.Column(db.Float)
    avg_price = db.Column(db.Float)
```

Aquí se guardan cuáles acciones posee cada usuario y cuántas.

4. Operation (Operación)

```
class Operation(db.Model):
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    asset_id = db.Column(db.Integer, db.ForeignKey('asset.id'))
    type = db.Column(db.String(10))
    quantity = db.Column(db.Float)
    price = db.Column(db.Float)
    timestamp = db.Column(db.DateTime)
```

Aquí se guarda el historial de todas las compras y ventas.

5. FUNCIONES PRINCIPALES (main.py)

Ruta: Página Principal

```
@app.route('/')
def index():
    assets = Asset.query.all()
    return render_template('index.html', assets=assets)
```

¿Qué hace?: Muestra todas las acciones disponibles con sus precios.

Ruta: Registro

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        name = request.form['name']
        email = request.form['email']
        pwd = request.form['password']
```

```

if User.query.filter_by(email=email).first():
    flash('Email already registered', 'danger')
    return redirect(url_for('register'))

user = User(name=name, email=email,
            password_hash=generate_password_hash(pwd))
db.session.add(user)
db.session.commit()
return redirect(url_for('login'))

```

Puntos clave:

- generate_password_hash(): Encripta la contraseña
- db.session.commit(): Guarda el registro en la base de datos
- Cada usuario empieza con 10,000€ de saldo como hemos mencionado con anterioridad

Ruta: Login

```

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        pwd = request.form['password']
        user = User.query.filter_by(email=email).first()

        if user and check_password_hash(user.password_hash, pwd):
            login_user(user)
            return redirect(url_for('dashboard'))

    flash('Incorrect email or password', 'danger')

```

Puntos clave:

- check_password_hash(): Compara la contraseña encriptada
- login_user(): Crea una sesión
- current_user: Variable global del usuario registrado

Ruta: Comprar acción (LA MÁS IMPORTANTE)

```

if op_type == 'buy':
    cost = qty * price

```

```

if current_user.cash < cost:
    flash('Insufficient funds', 'danger')
else:
    current_user.cash -= cost

h = Holding.query.filter_by(
    user_id=current_user.id,
    asset_id=asset.id
).first()

if not h:
    h = Holding(user_id=current_user.id,
                asset_id=asset.id,
                quantity=qty, avg_price=price)
    db.session.add(h)
else:
    prev_val = h.quantity * h.avg_price
    h.quantity += qty
    h.avg_price = (prev_val + cost) / h.quantity

db.session.commit()

```

CÁLCULO DEL PRECIO PROMEDIO:

Ejemplo:

- Tengo: 10 acciones a 100€ cada una = $10 \times 100 = 1,000\text{€}$
 - Compro: 5 acciones a 120€ cada una = $5 \times 120 = 600\text{€}$
 - Total: tenemos 15 acciones (se guardarán en el Portfolio)
 - Precio promedio = $(1,000 + 600) / 15 = 106.67\text{€}$
-

6. MANUAL DE INSTALACIÓN

Requisitos previos:

- **Python 3.8 o superior** instalado
- **pip** (gestor de paquetes de Python)
- **Navegador web**
- **Conexión a internet** (para obtener precios de acciones)

Paso 1: Descargar el proyecto

```
git clone [URL_DEL_REPOITORIO]
```

```
cd TFM
```

Paso 2: Crear entorno virtual

```
# En Windows:
```

```
python -m venv venv
```

```
venv\Scripts\activate
```

```
# En macOS/Linux:
```

```
python3 -m venv venv
```

```
source venv/bin/activate
```

Paso 3: Instalar dependencias

```
pip install -r requirements.txt
```

Contenido de requirements.txt:

```
Flask==3.0.0
Flask-Login==0.6.3
Flask-SQLAlchemy==3.1.1
Flask-WTF==1.2.1
yfinance==0.2.32
pandas==2.1.3
matplotlib==3.8.2
Werkzeug==3.0.1
```

Paso 4: Configurar la aplicación

El archivo config.py ya está configurado por defecto. Si se quisiera personalizar:

```
# config.py
SECRET_KEY = 'tu-clave-secreta-aqui'
SQLALCHEMY_DATABASE_URI = 'sqlite:///app.db'
```

Paso 5: Ejecutar la aplicación

```
python main.py
```

Salida esperada:

```
* Serving Flask app 'main'
* Debug mode: on
* Running on http://127.0.0.1:5000
```

Paso 6: Acceder a la aplicación

1. Abre el navegador
2. Ve a: `http://127.0.0.1:5000` o `http://localhost:5000`
3. Deberías ver la página principal con las acciones disponibles

Paso 7: Crear tu primera cuenta

1. Haz clic en "**Register**"
2. Completa el formulario:
 - o Name: tu nombre
 - o Email: tu mail
 - o Password: tu contraseña
3. Haz clic en "**Create Account**"
4. Inicia sesión con tus credenciales

Tras esto, ya podrías empezar a comprar y vender acciones con el saldo inicial.

Solución de problemas comunes:

Los precios no cargan → Verifica conexión a internet. yfinance necesita acceso a Yahoo Finance.

Error de base de datos → Elimina el archivo `app.db` y vuelve a ejecutar. La base de datos se creará automáticamente.

7. PREGUNTAS Y RESPUESTAS

1. ¿Por qué Flask? → he usado Flask ya que es el recomendado en proyectos financieros y educativos en Python, por ser más “fácil” que otras librerías.

2. ¿Cómo garantizo la seguridad de las contraseñas? → usando `generate_password_hash()` garantizamos la seguridad de este dato, nunca guardadas en texto plano.

3. ¿Qué es SQLAlchemy? → SQLAlchemy es un ORM (Object-Relational Mapping) que nos permite trabajar con bases de datos usando objetos de Python en lugar de escribir SQL directo (facilita el código).

4. ¿Qué hace el decorador `@login_required`? → este protege rutas para que solo usuarios identificados puedan acceder. Si alguien no registrado intenta entrar, lo redirige al login repetidamente.

5. ¿De dónde obtienes los precios de las acciones? → para la búsqueda de precios, uso la librería yfinance que consulta Yahoo Finance. Los precios se actualizan al iniciar la aplicación, así no afectará al usuario al momento de acceder, y podrá ser un proyecto al que se pueda acceder a futuro.

6. ¿Por qué SQLite y no MySQL? → SQLite es ideal para proyectos educativos: no necesita instalación ni servidor, y los datos se guardan en un solo archivo.

7. ¿Cómo funciona la relación entre User y Holding? → esta relación es una relación uno-a-muchos, esto quiere decir que, un usuario puede tener múltiples holdings. Se conectan mediante user_id como Foreign Key. Por tanto, con un solo usuario, puedo realizar varias operaciones.

8. ¿Qué es current_user? → esta es una variable global de Flask-Login que representa al usuario actual registrado. Contiene todos sus datos.

9. ¿Por qué calcular el precio promedio? → porque si compras acciones a diferentes precios, necesitas saber tu coste promedio para calcular ganancias o pérdidas correctamente. He intentado simular operaciones que normalmente se harían en finanzas en Excel.

8. NÚMEROS DEL PROYECTO

- 7 rutas principales
 - 4 modelos de base de datos
 - 7 plantillas HTML
 - 15 acciones disponibles
 - 10,000€ de presupuesto inicial por usuario
-

9. POSIBLES MEJORAS A FUTURO

1. Más gráficos de comparativas o evolución
 2. Historial completo de operaciones
 3. Análisis técnico (RSI, ROA,ROS, etc)
 4. Exportar reportes en PDF o de cualquier otra forma
 5. Añadir criptomonedas tal y como accedemos a las acciones, incluso compras de oro u otros materiales.
 6. Ranking de usuarios, pudiendo ver quienes están registrados y que tal van sus operaciones.
-

10. PEQUEÑO RESUMEN

1. Cómo funciona “login” ?:

- El usuario ingresa su email y contraseña
- check_password_hash verifica la contraseña
- login_user crea la sesión

2. Cómo funciona la función “comprar una acción” ?:

- Verifica fondos
- Resta el dinero al usuario según el precio de la acción y la cantidad
- Crea o actualiza la función “Holding”
- Calcula el precio promedio
- Guarda en la función “Operation”

3. De dónde vienen los precios?

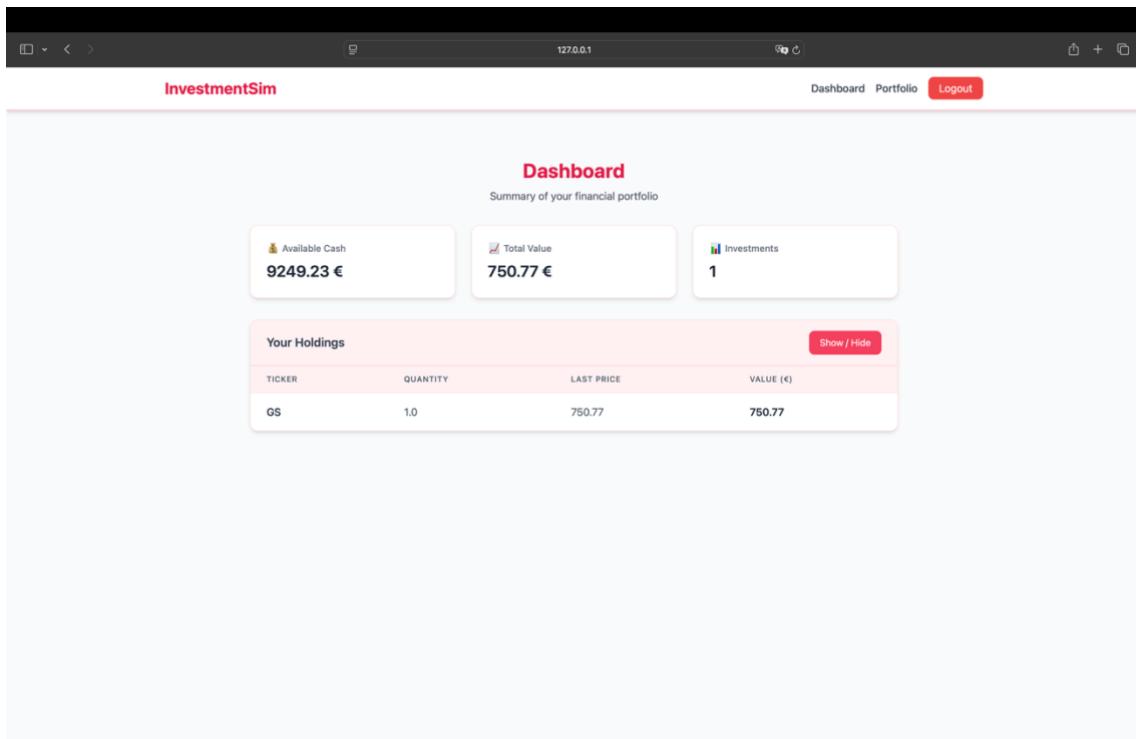
- Usando yfinance que consulta Yahoo Finance
- Estos se actualizan al iniciar la app
- Se guardan en Asset.last_price

11. EJEMPLOS DEL FUNCIONAMIENTO

The screenshot shows a web application interface titled "Available Assets". The title bar includes the URL "127.0.0.1" and standard browser controls. Below the title, a sub-header reads "Explore and buy stocks from the market". The main content area displays a grid of 12 stock entries, each in its own card:

Symbol	Type	Last Price
AAPL	Stock	252.29 €
MSFT	Stock	513.58 €
GOOGL	Stock	253.30 €
AMZN	Stock	213.04 €
TSLA	Stock	439.31 €
JPM	Stock	297.56 €
BAC	Stock	51.28 €
GS	Stock	750.77 €
MS	Stock	158.67 €
V	Stock	341.89 €
MA	Stock	560.97 €
NVDA	Stock	183.22 €
META	Stock	716.92 €
IBM	Stock	281.28 €
ORCL	Stock	291.31 €

Each card contains the stock symbol, type, last price, and a "View Details →" button.



12. CONCLUSIONES

Tras la realización del proyecto, iniciado en el mes de septiembre, he intentado plasmar en el proyecto mi idea inicial, cuyo objetivo era la unión entre lo aprendido durante el curso de Introducción a Python, con el Grado Universitario en Administración y Empresas que estoy cursando actualmente. Para poder lograr esto, he intentado repasar todo lo visto, además de buscar nueva información y librerías especializadas, para integrar ambas disciplinas de la mejor forma posible. No ha sido tarea fácil incluir código desconocido en el TFM, pero he intentado dar lo mejor posible, informándome sobre este, escribiéndolo detalladamente y traduciendo las funciones al inglés. Además, he intentado pedir ayuda siempre que no he comprendido algún punto o me he quedado estancada.

Tras un arduo trabajo, este proyecto representa la unión perfecta entre tecnología y finanzas, dos campos que cada vez están más interrelacionados en nuestro día a día. Por tanto, podría ser de gran utilidad por ejemplo para aquellas personas que pretendan estudiar disciplinas como “*trading*” y quieran practicar antes de lanzarse al mercado con un saldo real.

InvestmentSim es solo el punto de partida. Con las mejoras propuestas anteriormente, este proyecto podría convertirse en una herramienta educativa útil para otros estudiantes de finanzas o incluso en una plataforma comercial. Estoy orgullosa del resultado final y de todo el esfuerzo empleado, y del apoyo brindado.