סיכום מעודכן של קוד הפרויקט "SimFin Analyzer" והתפתחותו:

המטרה המרכזית של התוכנית נשארה זהה: יצירת אפליקציית Web (מבוססת Flask) לניתוח נתונים פיננסיים של חברות. האפליקציה שואבת נתונים מ-SimFin (דוחות כספיים) ומ-Yahoo Finance (נתוני מחירים), שומרת אותם (קבצי CSV וב-session), ומציגה אותם ויזואלית באמצעות גרפים אינטראקטיביים של Plotly. מתוכננת גם פונקציונליות עתידית להערכות שווי.

**השינוי המרכזי מאז הסיכום הקודם הוא המעבר לקוד מודולרי.** הקוד אורגן מחדש לקבצים נפרדים לפי תחומי אחריות, מה שמקל על התחזוקה, ההרחבה והבנת הקוד.

**מבנה הפרויקט הנוכחי (כפי שעולה מהקבצים וה-README):**

* **app.py**: נקודת הכניסה הראשית של האפליקציה. אחראי על יצירת מופע Flask, רישום Blueprints (ניתובים), הגדרות תצורה ראשוניות (כמו מפתח סודי וטעינת הגדרות SimFin), והפעלת שרת הפיתוח.
* **modules/**: מכיל את הלוגיקה המרכזית של האפליקציה, מחולקת למודולים:
  + **data\_loader.py**: אחראי על טעינת מפתח ה-API של SimFin מקובץ, הגדרת SimFin (קביעת מפתח API ותיקיית נתונים), ופונקציית עזר להצגת סטטוס מפתח ה-API בממשק המשתמש.
  + **financial\_statements.py**: מכיל פונקציות להורדת דוחות כספיים (הכנסות, מאזן, תזרים מזומנים – שנתי ורבעוני) מ-SimFin עבור טיקר ספציפי, שמירתם בקבצי CSV, וטעינתם מה-session או מקבצי CSV.
  + **price\_history.py**: מכיל פונקציות להורדת נתוני מחירים היסטוריים (OHLCV) מ-Yahoo Finance, חישוב ממוצעים נעים, ופוטנציאל לחישוב אינדיקטורים טכניים נוספים וסטטיסטיקות מחיר.
  + **chart\_creator.py**: מרכז את הפונקציונליות של יצירת הגרפים באמצעות Plotly. כולל פונקציה גנרית ליצירת גרפי Time Series (ברים או קווים) ופונקציה ייעודית ליצירת גרף נרות עם ממוצעים נעים. הפונקציות מחזירות את נתוני הגרף בפורמט JSON (data + layout) המתאים לרינדור בצד הלקוח.
  + **routes/**: מכיל את ה-Blueprints של Flask המגדירים את הניתובים (URLs) והלוגיקה של כל עמוד/קבוצת עמודים:
    - **home.py**: מטפל בניתובים הקשורים לדף הבית, בחירת טיקר, ועדכון מפתח API.
    - **graphs.py**: מטפל בניתובים להצגת גרפים שנתיים ורבעוניים של דוחות כספיים.
    - **valuations.py**: מטפל בניתוב לדף הערכות שווי (כרגע דף ממלא מקום).
* **utils/**: מכיל כלי עזר כלליים:
  + **helpers.py**: (קובץ זה לא סופק, אך מוזכר ב-financial\_statements.py, כנראה מכיל פונקציות כמו ensure\_directory\_exists).
  + **config\_loader.py**: (קובץ זה לא סופק, אך מוזכר ב-data\_loader.py ו-app.py, כנראה אחראי לטעינת הגדרות מקובץ config.ini).
* **templates/**: מכיל את תבניות ה-HTML של Jinja2:
  + **base\_layout.html**: תבנית הבסיס המשותפת לכל הדפים. כוללת את מבנה ה-HTML הכללי, תפריט צד (sidebar), סרגל עליון לבחירת טיקר, וחלון קופץ (modal) לעדכון מפתח API. היא גם טוענת את Bootstrap CSS ו-Plotly.js.
  + **content\_home.html**: תוכן ספציפי לדף הבית, כולל הצגת גרף הנרות וסטטוס הורדת נתונים.
  + **content\_graphs.html**: תוכן ספציפי לדפי הגרפים השנתיים/רבעוניים, כולל הצגת גרפי הכנסות ורווח נקי.
  + **content\_valuations.html**: תוכן ספציפי לדף הערכות שווי.
* **static/**: (התיקייה עצמה לא סופקה, אך מוזכרת ב-README) מיועדת לקבצים סטטיים כמו CSS, JavaScript ותמונות.
* **config/config.ini**: (הקובץ עצמו לא סופק, אך מוזכר) קובץ תצורה להגדרות האפליקציה (כמו נתיבים, מפתח API דיפולטיבי).
* **data/**: (התיקייה עצמה לא סופקה, אך מוזכרת) מכילה תיקיית משנה simfin\_data לאחסון נתוני SimFin גולמיים, ותיקיות משנה לכל טיקר עם קבצי ה-CSV של הדוחות הכספיים.
* **README.md**: קובץ הסבר על הפרויקט, מבנה, התקנה, שימוש וכו'.

**מה התוכנית עושה כעת (לאחר המעבר למודולריות):**

הפונקציונליות הבסיסית נשארה דומה לסיכום הקודם, אך מאורגנת בצורה טובה יותר:

1. **אתחול אפליקציה (app.py):**
   * טוען הגדרות, מגדיר מפתח סודי.
   * מאתחל את SimFin עם מפתח API ותיקיית נתונים (data\_loader.py).
   * רושם את ה-Blueprints מהמודולים ב-modules/routes/.
2. **ממשק משתמש (base\_layout.html וקבצי תוכן):**
   * **סרגל עליון קבוע:** מאפשר למשתמש להזין טיקר מניה וללחוץ על כפתור "בחר מניה והורד נתונים". פעולה זו שולחת POST request לשרת.
   * **תפריט צד (Sidebar):** ניווט בין "דף הבית", "גרפים שנתיים", "גרפים רבעוניים", "הערכות שווי", וקישור לפתיחת חלון Modal לעדכון מפתח API של SimFin.
   * **הצגת תוכן דינמי:** התבנית הראשית (base\_layout.html) כוללת בלוק תוכן שמוחלף על ידי תבניות התוכן הספציפיות (content\_home.html, content\_graphs.html וכו') בהתאם לניתוב.
   * **הודעות Flash:** הצגת הודעות למשתמש (הצלחה, שגיאה, אזהרה, מידע).
3. **בחירת טיקר והורדת נתונים (בעיקר home.py ב-routes, ופונקציות מ-financial\_statements.py ו-price\_history.py):**
   * עם בחירת טיקר, השרת:
     + מוריד את 6 הדוחות הכספיים מ-SimFin (הכנסות, מאזן, תזרים – שנתי ורבעוני) באמצעות financial\_statements.download\_financial\_statements.
     + הפונקציה ב-financial\_statements.py טוענת את כל הדאטהסט הרלוונטי מ-SimFin (למשל, כל דוחות ההכנסה השנתיים) ואז מסננת לפי הטיקר המבוקש.
     + שומרת את הדוחות כקבצי CSV בתיקייה ייעודית לטיקר תחת data/[TICKER]/ באמצעות financial\_statements.save\_financial\_statements. סטטוס השמירה נשמר ב-session ומוצג בדף הבית.
     + ה-DataFrames של דוחות ההכנסה (כנראה שנתי ורבעוני) נשמרים ב-session כ-JSON (כדי להימנע מקריאה חוזרת מה-CSV בכל טעינת דף גרפים, אך כפי שצוין בפונקציה get\_dataframe\_from\_session\_or\_csv, יש Fallback לקריאה מ-CSV אם הנתונים לא ב-session או לא תקינים).
     + מורידה נתוני מחירים היסטוריים מ-Yahoo Finance (שנה אחרונה כברירת מחדל) ומחשבת ממוצעים נעים (20, 50, 100, 150, 200 יום כברירת מחדל) באמצעות price\_history.download\_price\_history\_with\_mavg.
4. **הצגת גרפים:**
   * **דף הבית (content\_home.html, home.py, chart\_creator.py):**
     + מציג גרף נרות יומי של המניה הנבחרת עם ממוצעים נעים. הגרף נוצר בשרת באמצעות chart\_creator.create\_candlestick\_chart\_with\_mavg ומועבר כ-JSON לתבנית ה-HTML, שם JavaScript (Plotly.js) מרנדר אותו.
   * **דפי גרפים שנתיים/רבעוניים (content\_graphs.html, graphs.py, chart\_creator.py):**
     + מציגים כרגע שני גרפים: גרף מכירות (Revenue) וגרף רווח נקי (Net Income).
     + הנתונים נטענים מה-session או מקבצי ה-CSV באמצעות financial\_statements.get\_dataframe\_from\_session\_or\_csv.
     + הגרפים נוצרים בשרת באמצעות chart\_creator.create\_timeseries\_chart ומועברים כ-JSON לתבנית ה-HTML לרינדור בצד הלקוח.
     + הגרפים מוצגים בשני טורים.
5. **עדכון מפתח API (base\_layout.html, home.py, data\_loader.py):**
   * חלון קופץ (Modal) מאפשר למשתמש להזין מפתח API חדש של SimFin.
   * שליחת הטופס מעדכנת את קובץ מפתח ה-API (api\_key.txt כברירת מחדל) ומאתחלת מחדש את SimFin עם המפתח החדש. המשתמש מקבל הודעת Flash על הצלחה/כישלון.
   * השארת השדה ריק מוחקת את המפתח מהקובץ ומחזירה לשימוש במפתח 'free'.
6. **דף הערכות שווי (content\_valuations.html, valuations.py):**
   * כרגע דף "ממלא מקום" עם הודעה שהפונקציונליות בפיתוח.

**בעיות שזוהו (או שנותרו רלוונטיות מהסיכום הקודם) והפתרונות/שינויים שבוצעו:**

1. **שגיאת SimFin: load() got an unexpected keyword argument 'ticker' (טופל במעקף):**
   * **הבעיה:** הפונקציות sf.load\_\* בגרסה מסוימת של SimFin לא קיבלו פרמטר ticker ישירות כפי שציפינו.
   * **הטיפול (שנשמר גם בקוד המודולרי):** הפונקציה download\_financial\_statements ב-financial\_statements.py טוענת את כל הדאטהסט הרלוונטי עבור סוג הדוח והתדירות (למשל, כל דוחות ההכנסה השנתיים), ולאחר מכן ה-DataFrame הגדול מסונן לפי הטיקר המבוקש. זה אולי פחות יעיל מהורדה ישירה פר-טיקר אם ה-API היה תומך בכך בצורה מיטבית, אך זהו מעקף שעובד.
2. **גרפים לא מוצגים בדפדפן (טופל):**
   * **הבעיה:** בעבר היו מגוון בעיות שגרמו לגרפים לא להופיע.
   * **הטיפול (שמיושם בקוד הנוכחי):**
     + הפונקציות ב-chart\_creator.py מחזירות את נתוני הגרף כ-JSON (אובייקט המכיל data ו-layout).
     + תבניות ה-HTML (content\_home.html, content\_graphs.html) מקבלות את ה-JSON הזה ומשתמשות בפונקציה Plotly.newPlot() בצד הלקוח (JavaScript) כדי לרנדר את הגרפים. זהו הפתרון היציב.
     + הקישור ל-Plotly.js עדכני.
     + בדיקות תקינות נתונים והודעות שגיאה משופרות ב-JavaScript המרנדר את הגרפים (למשל, ב-content\_graphs.html).
3. **ניהול נתונים ב-Session לעומת CSV:**
   * **האתגר:** איזון בין מהירות (טעינה מה-session) לבין עמידות וחיסכון בזיכרון (טעינה מ-CSV).
   * **הגישה הנוכחית (financial\_statements.get\_dataframe\_from\_session\_or\_csv):** מנסה תחילה לטעון מה-session. אם הנתונים לא שם, לא תקינים, או ריקים, היא פונה לקובץ ה-CSV. עבור דוחות הכנסה שנטענים מ-CSV, הם נשמרים בחזרה ב-session (כנראה כדי לייעל טעינות עתידיות באותו session). זה נראה כמו פשרה סבירה. יש לוודא ניקוי ה-session בעת בחירת טיקר חדש כדי למנוע הצגת נתונים ישנים.
4. **טיפול בשגיאות (משופר, אך תמיד יש מקום לשיפור):**
   * הקוד המודולרי כולל יותר בדיקות try-except והחזרת הודעות שגיאה מפורטות יותר מהפונקציות (למשל, ב-chart\_creator.py וב-financial\_statements.py).
   * הצגת הודעות שגיאה/מידע למשתמש בתבניות ה-HTML (דרך משתני context והודעות Flash).

**בעיות פוטנציאליות או נקודות למחשבה עתידית (חלקן מהסיכום הקודם, רלוונטיות גם כעת):**

1. **יעילות הורדת נתונים מ-SimFin:** כפי שצוין, הורדת כל הדאטהסט וסינון מקומי עשויה להיות לא יעילה אם עובדים עם מספר רב של טיקרים או אם רוצים רק נתונים עדכניים. ייתכן שכדאי לבדוק אם גרסאות חדשות יותר של SimFin מאפשרות שאילתות יעילות יותר פר-טיקר.
2. **ניהול Session:**
   * **גודל ה-Session:** שמירת DataFrames (אפילו כ-JSON) ב-session עלולה להפוך אותו לגדול מאוד אם משתמש בודק טיקרים רבים. יש לשקול אסטרטגיות ניקוי או הגבלת הנתונים הנשמרים ב-session.
   * **ניקוי Session בבחירת טיקר חדש:** חיוני לוודא שכאשר משתמש בוחר טיקר חדש, נתונים ספציפיים לטיקר הקודם (כמו data\_download\_status או DataFrames שמורים) מתנקים מה-session כדי למנוע בלבול או הצגת מידע שגוי. (בדיקה מהירה של home.py (לא סופק במלואו, רק ה-Blueprint) תהיה נחוצה כדי לראות איך זה מטופל).
3. **טיפול בשדות חסרים/לא סטנדרטיים בדוחות הכספיים:** דוחות כספיים יכולים להשתנות בין חברות ותעשיות. הפונקציה create\_timeseries\_chart צריכה להיות עמידה מספיק לטפל במקרים שבהם עמודה מסוימת (למשל, 'Net Income Common Stockholders') לא קיימת, ואולי להציע חלופות או הודעות ברורות. כרגע היא בודקת קיום עמודה ומחזירה שגיאה אם אינה קיימת, שזה טוב.
4. **התאמה אישית של גרפים:**
   * **גרף הנרות:** כרגע הממוצעים הנעים קבועים בקוד (price\_history.py ו-home.py). היכולת שהמשתמש יבחר אילו ממוצעים להציג וטווח התאריכים נשארה כדרישה עתידית.
   * **גרפים פיננסיים:** הרחבת מגוון הגרפים (רווח גולמי, תפעולי וכו') כפי שתוכנן. המבנה המודולרי עם chart\_creator.py ו-graphs.py אמור להקל על כך.
5. **מודול "הערכות שווי":** עדיין בשלב תכנון/ממלא מקום.
6. **עיצוב ו-UX:** base\_layout.html כולל עיצוב בסיסי עם Bootstrap. יש מקום לשיפורים אסתטיים והתאמה אישית של מראה הגרפים של Plotly.

**סיכום היתרונות של המבנה המודולרי הנוכחי:**

* **קריאות ותחזוקתיות:** קל יותר להבין ולתחזק כל חלק של הקוד בנפרד.
* **שימוש חוזר:** פונקציות כמו יצירת גרפים (chart\_creator.py) או הורדת נתונים (financial\_statements.py, price\_history.py) ניתנות לשימוש חוזר על ידי מספר Routes.
* **בדיקות (Testing):** קל יותר לכתוב בדיקות יחידה (Unit Tests) עבור מודולים ספציפיים (כפי שמוזכר ב-README).
* **הרחבה:** קל יותר להוסיף פונקציונליות חדשה על ידי הוספת מודולים או הרחבת מודולים קיימים מבלי להשפיע באופן דרמטי על חלקים אחרים של האפליקציה.

לסיכום, המעבר לקוד מודולרי הוא צעד משמעותי קדימה שהופך את הפרויקט למאורגן, גמיש וקל יותר לפיתוח עתידי. הדרישות המקוריות עדיין בתהליך מימוש, אך התשתית הקיימת חזקה יותר.