MoneySplit

Complete Source Code Appendix

Generated: October 05, 2025 at 11:25

Table of Contents

File 1: DB/__init__.py

```
1 |
```

File 2: DB/reset.py

```
"""
Maintenance tool for MoneySplit DB.
- Backup DB tables to CSV
- Reset DB with optional backup
- Restore DB from CSV backup
- Reset only tax brackets
- Export tax bracket CSV template
"""

from DB import setup
import csv
import os
from datetime import datetime

BACKUP_DIR = "backups"
os.makedirs(BACKUP_DIR, exist_ok=True)

def backup():
    """Backup tax_records and people to timestamped CSVs."""
    conn = setup.get_conn()
    cursor = conn.cursor()

    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")

    # Backup tax_records
    cursor.execute("SELECT * FROM tax_records")
    rows = cursor.fetchall()
    headers = [d[0] for d in cursor.description]
    tax_file = os.path.join(BACKUP_DIR, f"tax_records_{timestamp}.csv")
    with open(tax_file, "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(headers)
        writer.writerows(rows)

    # Backup people
    cursor.execute("SELECT * FROM people")
    rows = cursor.fetchall()
    headers = [d[0] for d in cursor.description]
    people_file = os.path.join(BACKUP_DIR, f"people_{timestamp}.csv")
    with open(people_file, "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(headers)
        writer.writerows(rows)

    conn.close()
    print(f"■ Backup complete → {tax_file}, {people_file}")


def reset():
    """Reset DB after optional backup."""
    confirm = input("■■ This will DELETE ALL DATA. Type 'RESET' to confirm: ").strip()
    if confirm != "RESET":
        print("■ Reset canceled.")
        return

    choice = input("Do you want to create a backup before reset? (y/n): ").strip().lower()
    if choice == "y":
        backup()

    setup.reset_db()


def reset_tax_brackets():
    """Reset only the tax_brackets table, with auto-backup."""
    confirm = input("■■ This will DELETE ALL TAX BRACKETS. Type 'RESET' to confirm: ").strip()
    if confirm != "RESET":
        print("■ Reset canceled.")
        return

    conn = setup.get_conn()
    cursor = conn.cursor()

    # Backup current brackets before deleting
    cursor.execute("SELECT * FROM tax_brackets")
```

```
    rows = cursor.fetchall()
    headers = [d[0] for d in cursor.description]

    if rows:  # Only back up if something exists
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        backup_file = os.path.join(BACKUP_DIR, f"tax_brackets_{timestamp}.csv")
        with open(backup_file, "w", newline="") as f:
            writer = csv.writer(f)
            writer.writerow(headers)
            writer.writerows(rows)
        print(f"■ Tax brackets backed up → {backup_file}")
    else:
        print("■■ No existing tax brackets found to back up.")

    # Delete and reseed defaults
    cursor.execute("DELETE FROM tax_brackets")
    conn.commit()
    conn.close()

    setup.seed_default_brackets()
    print("■ Tax brackets reset and reseeded with defaults.")

def restore_tax_brackets():
    """Restore tax brackets from a CSV backup."""
    filepath = input("Enter path to tax_brackets CSV backup: ").strip()

    if not os.path.exists(filepath):
        print("■ File not found.")
        return

    conn = setup.get_conn()
    cursor = conn.cursor()

    # Reset table before restoring
    cursor.execute("DELETE FROM tax_brackets")

    with open(filepath, "r") as f:
        reader = csv.reader(f)
        headers = next(reader)  # skip header
        for row in reader:
            cursor.execute(f"""
                INSERT INTO tax_brackets ({','.join(headers)})
                VALUES ({','.join(['?']*len(headers))})
            """, row)

    conn.commit()
    conn.close()
    print(f"■ Tax brackets restored from {filepath}")


def restore():
    """Restore DB from CSV backups in /backups/"""
    tax_file = input("Enter path to tax_records CSV: ").strip()
    people_file = input("Enter path to people CSV: ").strip()

    if not os.path.exists(tax_file) or not os.path.exists(people_file):
        print("■ One or both CSV files not found.")
        return

    setup.reset_db()  # start fresh

    conn = setup.get_conn()
    cursor = conn.cursor()

    # Restore tax_records
    with open(tax_file, "r") as f:
        reader = csv.reader(f)
        headers = next(reader)  # skip header
        for row in reader:
            cursor.execute(f"""
                INSERT INTO tax_records ({','.join(headers)})
                VALUES ({','.join(['?']*len(headers))})
            """, row)

    # Restore people
    with open(people_file, "r") as f:
        reader = csv.reader(f)
        headers = next(reader)
```

```
        for row in reader:
            cursor.execute(f"""
                INSERT INTO people ({','.join(headers)})
                VALUES ({','.join(['?']*len(headers))})
            """, row)

    conn.commit()
    conn.close()
    print("■ Database restored from CSV.")


def export_tax_template():
    """Export a blank tax bracket CSV template into backups/ folder."""
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    template_file = os.path.join(BACKUP_DIR, f"tax_bracket_template_{timestamp}.csv")

    headers = ["income_limit", "rate"]

    with open(template_file, "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(headers)
        # add one example row so user sees format
        writer.writerow([10000, 0.10])
        writer.writerow([40000, 0.20])
        writer.writerow([float("inf"), 0.30])

    print(f"■ Tax bracket template exported → {template_file}")
    print("■ Edit this CSV and then upload it back through the Tax Menu (Upload from CSV).")

def main():
    while True:
        print("\n=== DB Maintenance Tool ===")
        print("1. Backup database to CSV")
        print("2. Reset database ■■")
        print("3. Restore database from CSV")
        print("4. Reset tax brackets ■■")
        print("5. Restore tax brackets from backup")
        print("6. Export tax bracket CSV template")
        print("7. Back to main menu")

        choice = input("Choose an option (1-7): ").strip()

        if choice == "1":
            backup()
        elif choice == "2":
            reset()
        elif choice == "3":
            restore()
        elif choice == "4":
            reset_tax_brackets()
        elif choice == "5":
            restore_tax_brackets()
        elif choice == "6":
            export_tax_template()
        elif choice == "7":
            print("■ Returning to main menu.")
            break
        else:
            print("■ Invalid choice. Please enter 1-7.")

```

File 3: DB/setup.py

Note: This file has 954 lines. First 500 lines shown.

```
"""
Database setup and operations module.

AI ASSISTANCE DISCLOSURE:
The database schema design received AI consultation (ChatGPT/Claude).
- Prompts used: "Design a normalized database schema for tax records with people"
- Prompts used: "Should I use foreign keys? What's the best relationship structure?"
- AI helped recommend the 3-table design (tax_records, people, tax_brackets)
- AI suggested adding foreign key constraints and proper indexing

The final schema decisions were mine, but AI provided guidance on normalization,
relationships, and SQL best practices.
"""
import json
import sqlite3
import sys
import csv
import os
from datetime import datetime

# Editable fields at project level
ALLOWED_FIELDS = {
    "num_people", "revenue", "total_costs",
    "tax_origin", "tax_option"
    # other fields are derived → recalculated automatically
}

# ----------------------------
# Init
# ----------------------------

def _pb():
    """Return already-loaded ProgramBackend module without re-importing it."""
    m = sys.modules.get("MoneySplit.Logic.ProgramBackend")
    if m is None:
        raise RuntimeError("ProgramBackend not loaded; run a calculation first (option 1).")
    return m


def get_conn():
    """Get a SQLite connection with foreign keys enabled."""
    # Use test database when running tests
    db_name = os.environ.get('TEST_DB', 'example.db') if os.environ.get('TESTING') else 'example.db'
    conn = sqlite3.connect(db_name)
    conn.execute("PRAGMA foreign_keys = ON;")
    return conn


def init_db():
    """Initialize tax_records, people, and tax_brackets tables."""
    conn = get_conn()
    cursor = conn.cursor()

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS tax_records (
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          num_people INTEGER,
          revenue REAL,
          total_costs REAL,
          group_income REAL,
          individual_income REAL,
          tax_origin TEXT,
          tax_option TEXT,
          tax_amount REAL,
          net_income_per_person REAL,
          net_income_group REAL,
          created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS people (
```

```
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          record_id INTEGER,
          name TEXT NOT NULL,
          work_share REAL,
          gross_income REAL,
          tax_paid REAL,
          net_income REAL,
          FOREIGN KEY (record_id) REFERENCES tax_records(id) ON DELETE CASCADE
        )
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS tax_brackets (
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          country TEXT NOT NULL,
          tax_type TEXT NOT NULL,
          income_limit REAL NOT NULL,
          rate REAL NOT NULL
        )
    """)

    conn.commit()
    conn.close()


def seed_default_brackets():
    """Insert default US &amp;amp; Spain brackets if table is empty."""
    conn = get_conn()
    cursor = conn.cursor()
    cursor.execute("SELECT COUNT(*) FROM tax_brackets")
    count = cursor.fetchone()[0]

    if count &gt; 0:
        conn.close()
        return

    defaults = [
        # US Individual
        ("US", "Individual", 10275, 0.10),
        ("US", "Individual", 41775, 0.12),
        ("US", "Individual", 89075, 0.22),
        ("US", "Individual", 170050, 0.24),
        ("US", "Individual", 215950, 0.32),
        ("US", "Individual", 539900, 0.35),
        ("US", "Individual", float("inf"), 0.37),
        # US Business
        ("US", "Business", float("inf"), 0.21),
        # Spain Individual
        ("Spain", "Individual", 12450, 0.19),
        ("Spain", "Individual", 20200, 0.24),
        ("Spain", "Individual", 35200, 0.30),
        ("Spain", "Individual", 60000, 0.37),
        ("Spain", "Individual", 300000, 0.45),
        ("Spain", "Individual", float("inf"), 0.47),
        # Spain Business
        ("Spain", "Business", float("inf"), 0.25),
    ]

    cursor.executemany("""
        INSERT INTO tax_brackets (country, tax_type, income_limit, rate)
        VALUES (?, ?, ?, ?)
    """, defaults)

    conn.commit()
    conn.close()


init_db()
seed_default_brackets()

# ------------------------------
# CRUD for tax_records
# ------------------------------

def save_to_db():
    """Save the current ProgramBackend calculation to tax_records."""
    pb = _pb()

```

```
    tax_origin = "US" if pb.tax_origin == 1 else "Spain"
    tax_option = "Individual" if pb.tax_option == 1 else "Business"

    if pb.tax_option == 1:  # individual
        net_income_per_person = pb.individual_income - pb.tax
        net_income_group = net_income_per_person * pb.num_people
    else:  # business
        net_income_group = pb.group_income - pb.tax
        net_income_per_person = net_income_group / pb.num_people

    conn = get_conn()
    cursor = conn.cursor()

    cursor.execute("""
        INSERT INTO tax_records (
            num_people, revenue, total_costs, group_income, individual_income,
            tax_origin, tax_option, tax_amount,
            net_income_per_person, net_income_group
        ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    """, (
        pb.num_people,
        pb.revenue,
        pb.total_costs,
        pb.group_income,
        pb.individual_income,
        tax_origin,
        tax_option,
        pb.tax,
        net_income_per_person,
        net_income_group
    ))

    record_id = cursor.lastrowid
    conn.commit()
    conn.close()
    return record_id


def fetch_last_records(n=5):
    conn = get_conn()
    cursor = conn.cursor()
    cursor.execute("""
        SELECT id, tax_origin, tax_option,
               revenue, total_costs,
               tax_amount, net_income_group, net_income_per_person, created_at,
               num_people, group_income, individual_income
        FROM tax_records
        ORDER BY created_at DESC
        LIMIT ?
    """, (n,))
    rows = cursor.fetchall()
    conn.close()
    return rows


def get_record_by_id(record_id: int):
    conn = get_conn()
    cursor = conn.cursor()
    cursor.execute("""
        SELECT id, tax_origin, tax_option,
               revenue, total_costs,
               tax_amount, net_income_group, net_income_per_person, created_at,
               num_people, group_income, individual_income
        FROM tax_records
        WHERE id = ?
    """, (record_id,))
    row = cursor.fetchone()
    conn.close()
    return row


def delete_record(record_id: int):
    conn = get_conn()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM tax_records WHERE id = ?", (record_id,))
    conn.commit()
    conn.close()
    print(f"■■ Record {record_id} and linked people deleted.")
```

```
|
|
| def update_record(record_id: int, field: str, new_value):
|     """Update a record by ID. Only base fields can be edited; derived fields recalculated."""
|     if field not in ALLOWED_FIELDS:
|         raise ValueError(f"Invalid field: {field}. Allowed: {', '.join(sorted(ALLOWED_FIELDS))}")
|
|     conn = get_conn()
|     cursor = conn.cursor()
|     cursor.execute(f"UPDATE tax_records SET {field} = ? WHERE id = ?", (new_value, record_id))
|
|     # fetch values for recalculation
|     cursor.execute("""
|         SELECT num_people, revenue, total_costs, tax_origin, tax_option
|         FROM tax_records WHERE id = ?
|     """, (record_id,))
|     row = cursor.fetchone()
|
|     if row:
|         num_people, revenue, total_costs, origin, option = row
|         revenue, total_costs, num_people = float(revenue), float(total_costs), int(num_people)
|         income = revenue - total_costs
|         group_income = income
|         individual_income = income / num_people if num_people &gt; 0 else 0
|
|         # Calculate tax using DB-driven tax brackets
|         if option == "Individual":
|             tax = calculate_tax_from_db(individual_income, origin, option)
|         else:
|             tax = calculate_tax_from_db(group_income, origin, option)
|
|         if option == "Individual":
|             net_income_per_person = individual_income - tax
|             net_income_group = net_income_per_person * num_people
|         else:
|             net_income_group = group_income - tax
|             net_income_per_person = net_income_group / num_people if num_people &gt; 0 else 0
|
|         cursor.execute("""
|             UPDATE tax_records
|             SET group_income=?, individual_income=?, tax_amount=?,
|                 net_income_per_person=?, net_income_group=?
|             WHERE id=?
|         """, (group_income, individual_income, tax, net_income_per_person, net_income_group, record_id))
|
|     conn.commit()
|     conn.close()
|     print(f"═■ Record {record_id} updated and recalculated ({field} → {new_value})")
|
| # ----------------------------
| # CRUD for people
| # ----------------------------
|
| def add_person(record_id: int, name: str, work_share: float,
|                gross_income: float, tax_paid: float, net_income: float):
|     conn = get_conn()
|     cursor = conn.cursor()
|     cursor.execute("""
|         INSERT INTO people (record_id, name, work_share, gross_income, tax_paid, net_income)
|         VALUES (?, ?, ?, ?, ?, ?)
|     """, (record_id, name, work_share, gross_income, tax_paid, net_income))
|     person_id = cursor.lastrowid
|     conn.commit()
|     conn.close()
|     print(f"■ Added person {name} (ID: {person_id}) to record {record_id}.")
|     return person_id
|
|
| def fetch_people_by_record(record_id: int):
|     conn = get_conn()
|     cursor = conn.cursor()
|     cursor.execute("""
|         SELECT id, name, work_share, gross_income, tax_paid, net_income
|         FROM people
|         WHERE record_id = ?
|     """, (record_id,))
|     rows = cursor.fetchall()
|     conn.close()
```

```
    return rows


def delete_person(person_id: int):
    conn = get_conn()
    cursor = conn.cursor()
    cursor.execute("SELECT name FROM people WHERE id=?", (person_id,))
    row = cursor.fetchone()
    if not row:
        print(f"■ No person found with ID {person_id}.")
        conn.close()
        return
    name = row[0]
    cursor.execute("DELETE FROM people WHERE id=?", (person_id,))
    conn.commit()
    conn.close()
    print(f"■■ Deleted person {person_id} ({name}).")


def fetch_records_by_person(name: str):
    conn = get_conn()
    cursor = conn.cursor()
    cursor.execute("""
        SELECT p.id, p.record_id, p.name, p.work_share,
               p.gross_income, p.tax_paid, p.net_income, t.created_at
        FROM people p
        JOIN tax_records t ON p.record_id = t.id
        WHERE LOWER(p.name) = LOWER(?)
        ORDER BY t.created_at DESC
    """, (name,))
    rows = cursor.fetchall()
    conn.close()
    return rows

# ----------------------------
# DB Maintenance
# ----------------------------

def reset_db():
    """■■ Drop and recreate all tables. Use for testing only."""
    choice = input("Do you want to create a backup before reset? (y/n): ").strip().lower()
    if choice == "y":
        backup_db_to_csv()

    conn = get_conn()
    cursor = conn.cursor()

    cursor.execute("DROP TABLE IF EXISTS people")
    cursor.execute("DROP TABLE IF EXISTS tax_records")
    cursor.execute("DROP TABLE IF EXISTS tax_brackets")

    conn.commit()
    conn.close()

    print("■■ All tables dropped. Reinitializing...")
    init_db()
    seed_default_brackets()
    print("■ Database reset complete.")


def backup_db_to_csv():
    """Backup tax_records and people tables into CSV files."""
    conn = get_conn()
    cursor = conn.cursor()

    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")

    # Backup tax_records
    cursor.execute("SELECT * FROM tax_records")
    rows = cursor.fetchall()
    headers = [d[0] for d in cursor.description]
    with open(f"backup_tax_records_{timestamp}.csv", "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(headers)
        writer.writerows(rows)

    # Backup people
    cursor.execute("SELECT * FROM people")
```

```
    rows = cursor.fetchall()
    headers = [d[0] for d in cursor.description]
    with open(f"backup_people_{timestamp}.csv", "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(headers)
        writer.writerows(rows)

    conn.close()
    print(f"■ Backup complete → backup_tax_records_{timestamp}.csv, backup_people_{timestamp}.csv")


# ----------------------------
# Tax brackets management
# ----------------------------

def calculate_tax_from_db(income: float, country: str, tax_type: str) -&gt; float:
    """Generic tax calculator that fetches brackets from DB."""
    brackets = get_tax_brackets(country, tax_type)
    if not brackets:
        raise ValueError(f"No tax brackets found for {country} {tax_type}")

    tax = 0
    prev = 0
    for limit, rate in brackets:
        if income &gt; limit:
            tax += (limit - prev) * rate
            prev = limit
        else:
            tax += (income - prev) * rate
            break
    return tax


def get_tax_brackets(country: str, tax_type: str, include_id: bool = False):
    conn = get_conn()
    cursor = conn.cursor()
    if include_id:
        cursor.execute("""
            SELECT id, income_limit, rate
            FROM tax_brackets
            WHERE country=? AND tax_type=?
            ORDER BY income_limit ASC
        """, (country, tax_type))
    else:
        cursor.execute("""
            SELECT income_limit, rate
            FROM tax_brackets
            WHERE country=? AND tax_type=?
            ORDER BY income_limit ASC
        """, (country, tax_type))
    rows = cursor.fetchall()
    conn.close()
    return rows


def add_tax_brackets_from_csv(country: str, tax_type: str, filepath: str):
    """Upload multiple tax brackets from a CSV file with columns: income_limit, rate"""
    if not os.path.exists(filepath):
        print("■ File not found.")
        return

    conn = get_conn()
    cursor = conn.cursor()
    added, skipped = 0, 0

    with open(filepath, newline="") as csvfile:
        reader = csv.DictReader(csvfile)
        if "income_limit" not in reader.fieldnames or "rate" not in reader.fieldnames:
            print("■ CSV must contain 'income_limit' and 'rate'.")
            conn.close()
            return

        for row in reader:
            try:
                limit_raw = row.get("income_limit", "").strip()
                rate_raw = row.get("rate", "").strip()

                if not limit_raw or not rate_raw:
```

```
                    skipped += 1
                    continue

                income_limit = float("inf") if limit_raw.lower() == "inf" else float(limit_raw)
                rate = float(rate_raw)

                cursor.execute("""
                    INSERT INTO tax_brackets (country, tax_type, income_limit, rate)
                    VALUES (?, ?, ?, ?)
                """, (country, tax_type, income_limit, rate))
                added += 1
            except Exception as e:
                print(f"■■ Skipping row {row}: {e}")
                skipped += 1

    conn.commit()
    conn.close()
    print(f"■ Imported {added} brackets for {country} {tax_type} ({skipped} skipped).")


def update_tax_bracket(bracket_id: int, field: str, new_value):
    allowed = {"country", "tax_type", "income_limit", "rate"}
    if field not in allowed:
        raise ValueError(f"Invalid field. Allowed: {', '.join(allowed)}")
    conn = get_conn()
    cursor = conn.cursor()
    cursor.execute(f"UPDATE tax_brackets SET {field}=? WHERE id=?", (new_value, bracket_id))
    conn.commit()
    conn.close()
    print(f"═■ Bracket {bracket_id} updated: {field} → {new_value}")


def delete_tax_bracket(bracket_id: int):
    conn = get_conn()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM tax_brackets WHERE id=?", (bracket_id,))
    conn.commit()
    conn.close()
```

File 4: Logic/ProgramBackend.py

```
  1 | # ProgramBackend.py
  2 | from DB import setup
  3 | from Logic import validators
  4 |
  5 | # - Inputs -
  6 | num_people = validators.safe_int_input("Enter the number of people: ", "Number of people", min_value=1)
  7 | revenue = validators.safe_float_input("Enter the revenue: ", "Revenue")
  8 |
  9 | num_costs = validators.safe_int_input("Enter the number of different costs: ", "Number of costs", min_value=0)
 10 | total_costs = 0
 11 | for i in range(num_costs):
 12 |     cost = validators.safe_float_input(f"Enter cost {i + 1}: ", f"Cost {i + 1}")
 13 |     total_costs += cost
 14 |
 15 | print("Total costs:", total_costs)
 16 |
 17 | income = revenue - total_costs
 18 | group_income = income
 19 | individual_income = income / num_people if num_people &gt; 0 else 0
 20 |
 21 | tax_origin = validators.safe_int_input("Enter the country (1 for US, 2 for Spain): ", "Country", min_value=1,
max_value=...
 22 | tax_option = validators.safe_int_input("Enter tax option (1 for individual, 2 for business): ", "Tax option",
min_value=...
 23 |
 24 | # Convert numeric input to strings for DB-driven brackets
 25 | country = "US" if tax_origin == 1 else "Spain"
 26 | tax_type = "Individual" if tax_option == 1 else "Business"
 27 |
 28 |
 29 | # - Tax Calculation Functions -
 30 | def us_individual_tax(income: float) -&gt; float:
 31 |     return calculate_tax_from_db(income, "US", "Individual")
 32 |
 33 | def us_business_tax(income: float) -&gt; float:
 34 |     return calculate_tax_from_db(income, "US", "Business")
 35 |
 36 | def spain_individual_tax(income: float) -&gt; float:
 37 |     return calculate_tax_from_db(income, "Spain", "Individual")
 38 |
 39 | def spain_business_tax(income: float) -&gt; float:
 40 |     return calculate_tax_from_db(income, "Spain", "Business")
 41 |
 42 | # - Generic Tax Function (DB-driven) -
 43 | def calculate_tax_from_db(income: float, country: str, tax_type: str) -&gt; float:
 44 |     """
 45 |     Generic tax calculator that fetches brackets from DB.
 46 |     """
 47 |     brackets = setup.get_tax_brackets(country, tax_type)
 48 |     if not brackets:
 49 |         raise ValueError(f"No tax brackets found for {country} {tax_type}")
 50 |
 51 |     tax = 0
 52 |     prev = 0
 53 |     for limit, rate in brackets:
 54 |         if income &gt; limit:
 55 |             tax += (limit - prev) * rate
 56 |             prev = limit
 57 |         else:
 58 |             tax += (income - prev) * rate
 59 |             break
 60 |     return tax
 61 |
 62 |
 63 | # - Calculate and Display Results -
 64 | if tax_option == 1:  # Individual tax
 65 |     tax = calculate_tax_from_db(individual_income, country, tax_type)
 66 | else:  # Business tax
 67 |     tax = calculate_tax_from_db(group_income, country, tax_type)
 68 |
 69 | if tax_option == 1:
 70 |     print(f"\nEffective tax rate: {(tax / individual_income) * 100:,.2f}%")
 71 |     print(f"\nIndividual income: ${individual_income:,.2f}")
 72 |     print(f"Tax per person: ${tax:,.2f}")
```

```
    print(f"Net income per person: ${individual_income - tax:,.2f}")
    print(f"Total tax for all people: ${tax * num_people:,.2f}")
else:
    print(f"Effective tax rate: {(tax / group_income) * 100:,.2f}%")
    print(f"\nBusiness income: ${group_income:,.2f}")
    print(f"Business tax: ${tax:,.2f}")
    print(f"Net Business income: ${group_income - tax:,.2f}")
    print(f"\nNet income per person: {(group_income - tax) / num_people:,.2f}")


# ==============================
# Collect People Information
# ==============================
print("\nNow enter details for each person:")

record_id = setup.save_to_db()  # Save project-level data first
total_work_share = 0.0

people_shares = []
people_data = []  # Store person ID and details

for i in range(num_people):
    name = validators.safe_string_input(f"Name of person {i + 1}: ", f"Person {i + 1} name")

    # If only 1 person, auto-assign work_share = 1.0
    if num_people == 1:
        work_share = 1.0
        print(f"■ {name} is the only person - automatically assigned 100% work share")
    else:
        work_share = validators.safe_float_input(f"Work share for {name} (0.0–1.0): ", "Work share")
        try:
            work_share = validators.validate_work_share(work_share)
        except validators.ValidationError as e:
            print(f"■ {e}")
            work_share = validators.safe_float_input(f"Work share for {name} (0.0–1.0): ", "Work share")
            work_share = validators.validate_work_share(work_share)

    total_work_share += work_share
    people_shares.append(work_share)

    if tax_option == 1:  # Individual
        gross_income = individual_income * work_share * num_people
        tax_paid = tax * work_share
        net_income = gross_income - tax_paid
    else:  # Business → distributed after company tax
        gross_income = group_income * work_share
        tax_paid = tax * work_share
        net_income = gross_income - tax_paid

    person_id = setup.add_person(record_id, name, work_share, gross_income, tax_paid, net_income)
    people_data.append({
        'person_id': person_id,
        'name': name,
        'work_share': work_share,
        'gross_income': gross_income,
        'tax_paid': tax_paid,
        'net_income': net_income
    })

# After all people are added - validate total work shares
if num_people &gt; 1:  # Only validate if more than 1 person
    try:
        validators.validate_work_shares(people_shares)
        print("■ Work shares add up to 1.0")
    except validators.ValidationError as e:
        print(f"■■ Warning: {e}")

# Display summary of created people
print(f"\n■ Project Summary (Record ID: {record_id}):")
print(f"{'Person ID':&lt;10} | {'Name':&lt;15} | {'Work Share':&gt;12} | {'Gross Income':&gt;15} | {'Tax Paid':
&gt;12} |...
print("-" * 90)
for person in people_data:
    print(f"{person['person_id']:&lt;10} | {person['name']:&lt;15} | {person['work_share']:&gt;12.2%} | ${person[
'gross_...

# Export record_id for use by project_menu
LAST_RECORD_ID = record_id
```

149 |

File 5: Logic/__init__.py

```
1 |
```

File 6: Logic/forecasting.py

```
"""
Forecasting and prediction module for MoneySplit.
Provides revenue forecasting, tax optimization, and trend analysis.

AI ASSISTANCE DISCLOSURE:
This forecasting module was developed with AI assistance (ChatGPT/Claude).
- Prompts used: "Create revenue forecasting using scikit-learn LinearRegression"
- Prompts used: "Add tax optimization analysis comparing Individual vs Business"
- Prompts used: "Implement trend analysis with seasonality detection"

ACADEMIC BACKGROUND:
The machine learning concepts (linear regression, time-series forecasting) were
learned in Machine Learning course (2024). AI assistance was primarily used for:
- Writing the Python implementation code
- Structuring the data preprocessing pipeline
- Integrating sklearn with the database layer
The underlying ML theory and approach were based on coursework knowledge.
"""
import numpy as np
from datetime import datetime, timedelta
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import sys
import os
sys.path.insert(0, os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
from DB import setup


def get_historical_data():
    """Fetch historical revenue data grouped by month."""
    conn = setup.get_conn()
    cursor = conn.cursor()

    cursor.execute("""
        SELECT strftime('%Y-%m', created_at) as month,
               SUM(revenue) as total_revenue,
               SUM(total_costs) as total_costs,
               SUM(net_income_group) as total_profit,
               COUNT(*) as num_projects,
               AVG(tax_amount * 100.0 / NULLIF(group_income, 0)) as avg_tax_rate
        FROM tax_records
        GROUP BY month
        ORDER BY month
    """)
    rows = cursor.fetchall()
    conn.close()

    return rows


def forecast_revenue(months_ahead=3):
    """
    Forecast revenue using an enhanced algorithm.

    Uses polynomial regression when enough data is available (better for non-linear trends).
    Falls back to linear regression for smaller datasets.
    Includes moving average smoothing to reduce noise.
    """
    historical = get_historical_data()

    if len(historical) &lt; 2:
        return {
            'success': False,
            'message': 'Not enough historical data (need at least 2 months)',
            'predictions': [],
            'explanation': 'Create more projects across different months to enable AI forecasting.'
        }

    # Prepare data
    revenues = np.array([row[1] for row in historical])
    num_projects = np.array([row[4] for row in historical])

    # Apply moving average smoothing for datasets with enough data
    # IMPORTANT: Use 'valid' mode to avoid data leakage (no look-ahead bias)
```

```
    if len(revenues) &gt;= 4:
        # 3-point moving average - 'valid' mode loses 2 data points but prevents leakage
        smoothed_revenues = np.convolve(revenues, np.ones(3)/3, mode='valid')
        y = smoothed_revenues
        # Adjust indices to match smoothed data length (starts from index 1)
        months_indices = np.array([i+1 for i in range(len(smoothed_revenues))]).reshape(-1, 1)
    else:
        y = revenues
        months_indices = np.array([i for i in range(len(revenues))]).reshape(-1, 1)

    # Choose model based on data size
    if len(historical) &gt;= 6:
        # Polynomial regression for better curve fitting
        poly_features = PolynomialFeatures(degree=2)
        X_poly = poly_features.fit_transform(months_indices)
        model = LinearRegression()
        model.fit(X_poly, y)
        r2_score = model.score(X_poly, y)
        model_type = "Polynomial (curved trend)"
    else:
        # Linear regression for smaller datasets
        model = LinearRegression()
        model.fit(months_indices, y)
        r2_score = model.score(months_indices, y)
        model_type = "Linear (straight trend)"

    # Enhanced confidence scoring
    if r2_score &gt; 0.8:
        confidence = "High"
        confidence_desc = "Very reliable - strong pattern detected"
    elif r2_score &gt; 0.6:
        confidence = "Medium-High"
        confidence_desc = "Reliable - clear pattern with minor variation"
    elif r2_score &gt; 0.4:
        confidence = "Medium"
        confidence_desc = "Moderate - pattern exists but with some variability"
    elif r2_score &gt; 0.2:
        confidence = "Low-Medium"
        confidence_desc = "Less reliable - high variability in data"
    else:
        confidence = "Low"
        confidence_desc = "Unreliable - very inconsistent data"

    # Predict future months
    predictions = []
    last_month = historical[-1][0]  # Format: YYYY-MM
    last_date = datetime.strptime(last_month, '%Y-%m')

    for i in range(1, months_ahead + 1):
        future_index = len(historical) + i - 1

        if len(historical) &gt;= 6:
            future_x_poly = poly_features.transform([[future_index]])
            predicted_revenue = model.predict(future_x_poly)[0]
        else:
            predicted_revenue = model.predict([[future_index]])[0]

        # Prevent negative predictions
        predicted_revenue = max(0, predicted_revenue)

        # Calculate next month
        next_month = last_date + timedelta(days=30*i)
        month_str = next_month.strftime('%B %Y')  # e.g., "November 2025"

        # Calculate confidence interval (95%)
        std_error = np.std(y - model.predict(X_poly if len(historical) &gt;= 6 else months_indices))
        lower_bound = max(0, predicted_revenue - 1.96 * std_error)
        upper_bound = predicted_revenue + 1.96 * std_error

        predictions.append({
            'month': month_str,
            'revenue': predicted_revenue,
            'confidence': confidence,
            'lower_bound': lower_bound,
            'upper_bound': upper_bound,
            'range': f"${lower_bound:,.0f} - ${upper_bound:,.0f}"
        })

```

```
    # Calculate trend with clearer description
    if len(historical) &gt;= 6:
        slope = revenues[-1] - revenues[0]
    else:
        slope = model.coef_[0] if len(model.coef_) == 1 else model.coef_[1]

    if slope &gt; 100:
        trend = "Strongly Increasing"
    elif slope &gt; 0:
        trend = "Growing"
    elif slope &lt; -100:
        trend = "Declining"
    else:
        trend = "Stable"

    trend_strength = abs(slope / len(historical)) if len(historical) &gt; 0 else 0

    # Generate plain English explanation
    avg_revenue = np.mean(revenues)
    last_revenue = revenues[-1]
    growth_rate = ((last_revenue - revenues[0]) / revenues[0] * 100) if revenues[0] &gt; 0 else 0

    explanation = f"""■ What this means:

Your business has {len(historical)} months of data. The AI analyzed this and found a {trend.lower()} pattern.

• Average monthly revenue: ${avg_revenue:,.0f}
• Last month: ${last_revenue:,.0f}
• Overall growth: {growth_rate:+.1f}%
• Confidence level: {confidence} ({confidence_desc})
• Prediction model: {model_type}

■ Next month prediction: ${predictions[0]['revenue']:,.0f}
(Range: {predictions[0]['range']})

■ The AI is {r2_score*100:.0f}% confident in this pattern. {
        "This is very reliable!" if r2_score &gt; 0.7 else
        "Add more data over time for better accuracy." if r2_score &lt; 0.5 else
        "Predictions are reasonably accurate."
    }"""

    return {
        'success': True,
        'predictions': predictions,
        'trend': trend,
        'trend_strength': trend_strength,
        'r2_score': r2_score,
        'confidence': confidence,
        'confidence_description': confidence_desc,
        'historical_avg': avg_revenue,
        'model_slope': slope,
        'growth_rate': growth_rate,
        'model_type': model_type,
        'explanation': explanation.strip(),
        'data_quality': "Excellent" if len(historical) &gt;= 10 else "Good" if len(historical) &gt;= 6 else
"Fair"
    }


def tax_optimization_analysis():
    """Analyze tax strategies and provide optimization recommendations."""
    conn = setup.get_conn()
    cursor = conn.cursor()

    # Compare Individual vs Business tax for recent projects
    cursor.execute("""
        SELECT tax_option,
               AVG(tax_amount) as avg_tax,
               AVG(tax_amount * 100.0 / NULLIF(group_income, 0)) as avg_rate,
               COUNT(*) as count,
               AVG(revenue) as avg_revenue
        FROM tax_records
        GROUP BY tax_option
    """)
    tax_comparison = cursor.fetchall()

    # Get most efficient strategy per country
    cursor.execute("""
```

```
        SELECT tax_origin, tax_option,
               AVG(tax_amount * 100.0 / NULLIF(group_income, 0)) as avg_rate
        FROM tax_records
        GROUP BY tax_origin, tax_option
        ORDER BY tax_origin, avg_rate
    """)
    country_analysis = cursor.fetchall()

    # Get overall rate before closing connection
    cursor.execute("""
        SELECT AVG(tax_amount * 100.0 / NULLIF(group_income, 0)) as overall_rate
        FROM tax_records
    """)
    overall_rate = cursor.fetchone()[0] or 0

    conn.close()

    recommendations = []

    # Analyze tax comparison
    if len(tax_comparison) &gt;= 2:
        individual = next((t for t in tax_comparison if t[0] == 'Individual'), None)
        business = next((t for t in tax_comparison if t[0] == 'Business'), None)

        if individual and business:
            if individual[2] &lt; business[2]:
                savings = business[1] - individual[1]
                recommendations.append(
                    f"Individual tax is {business[2] - individual[2]:.1f}% lower on average. "
                    f"Consider Individual tax for future projects (potential savings: ${savings:,.2f} per
project)"
                )
            else:
                savings = individual[1] - business[1]
                recommendations.append(
                    f"Business tax is {individual[2] - business[2]:.1f}% lower on average. "
                    f"Consider Business tax for future projects (potential savings: ${savings:,.2f} per project)"
                )

    # Country-specific recommendations
    country_dict = {}
    for row in country_analysis:
        country = row[0]
        if country not in country_dict:
            country_dict[country] = []
        country_dict[country].append((row[1], row[2]))

    for country, strategies in country_dict.items():
        if len(strategies) &gt;= 2:
            best_strategy = min(strategies, key=lambda x: x[1])
            recommendations.append(
                f"For {country}: {best_strategy[0]} tax offers best rate ({best_strategy[1]:.1f}%)"
            )

    # General recommendations
    if overall_rate &gt; 25:
        recommendations.append(
            f"Your average tax rate is {overall_rate:.1f}%. Consider reviewing tax brackets and deductions."
        )

    return {
        'tax_comparison': [
            {
                'type': t[0],
                'avg_tax': t[1],
                'avg_rate': t[2],
                'count': t[3],
                'avg_revenue': t[4]
            } for t in tax_comparison
        ],
        'recommendations': recommendations if recommendations else ['Your tax strategy appears optimal based on
current ...
    }


def break_even_analysis(revenue, costs):
    """Calculate break-even point and margin of safety."""
    profit = revenue - costs
```

```
    margin = (profit / revenue * 100) if revenue &gt; 0 else 0

    return {
        'revenue': revenue,
        'costs': costs,
        'profit': profit,
        'profit_margin': margin,
        'break_even_revenue': costs,
        'margin_of_safety': revenue - costs,
        'margin_of_safety_pct': margin
    }


def trend_analysis():
    """Analyze trends in revenue, costs, and profitability."""
    historical = get_historical_data()

    if len(historical) &lt; 3:
        return {
            'success': False,
            'message': 'Need at least 3 months of data for trend analysis'
        }

    months = [row[0] for row in historical]
    revenues = [row[1] for row in historical]
    costs = [row[2] for row in historical]
    profits = [row[3] for row in historical]
    num_projects = [row[4] for row in historical]

    # Calculate trends
    revenue_trend = "increasing" if revenues[-1] &gt; revenues[0] else "decreasing"
    cost_trend = "increasing" if costs[-1] &gt; costs[0] else "decreasing"
    profit_trend = "increasing" if profits[-1] &gt; profits[0] else "decreasing"

    # Calculate growth rates
    revenue_growth = ((revenues[-1] - revenues[0]) / revenues[0] * 100) if revenues[0] &gt; 0 else 0
    cost_growth = ((costs[-1] - costs[0]) / costs[0] * 100) if costs[0] &gt; 0 else 0
    profit_growth = ((profits[-1] - profits[0]) / profits[0] * 100) if profits[0] &gt; 0 else 0

    # Seasonality detection (simple moving average)
    if len(revenues) &gt;= 6:
        avg_revenue = np.mean(revenues)
        volatility = np.std(revenues) / avg_revenue if avg_revenue &gt; 0 else 0
        seasonality = "High seasonality detected" if volatility &gt; 0.3 else "Low seasonality" if volatility
&gt; 0.1 e...
    else:
        seasonality = "Insufficient data"

    return {
        'success': True,
        'months_analyzed': len(historical),
        'revenue_trend': revenue_trend,
        'cost_trend': cost_trend,
        'profit_trend': profit_trend,
        'revenue_growth': revenue_growth,
        'cost_growth': cost_growth,
        'profit_growth': profit_growth,
        'seasonality': seasonality,
        'avg_projects_per_month': np.mean(num_projects),
        'current_month_projects': num_projects[-1] if num_projects else 0,
        'insights': generate_insights(revenue_trend, cost_trend, profit_trend, revenue_growth, cost_growth)
    }


def generate_insights(rev_trend, cost_trend, profit_trend, rev_growth, cost_growth):
    """Generate actionable insights from trend analysis."""
    insights = []

    if rev_trend == "increasing" and cost_trend == "increasing":
        if cost_growth &gt; rev_growth:
            insights.append("■■ Warning: Costs are growing faster than revenue. Review cost management.")
        else:
            insights.append("■ Good: Revenue growth is outpacing cost growth.")

    if rev_trend == "decreasing":
        insights.append("■■ Revenue is declining. Consider marketing efforts or new revenue streams.")

    if profit_trend == "decreasing":
```

```
        insights.append("■■ Profitability is declining. Focus on cost reduction or pricing optimization.")

    if profit_trend == "increasing":
        insights.append("■ Profitability is improving. Maintain current strategy.")

    if abs(rev_growth) &lt; 5:
        insights.append("■ Revenue is relatively stable. Consider growth strategies.")

    return insights if insights else ["■ Business metrics are stable."]


def comprehensive_forecast():
    """Generate comprehensive forecast with all insights."""
    revenue_forecast = forecast_revenue(3)
    tax_optimization = tax_optimization_analysis()
    trends = trend_analysis()

    recommendations = []

    # Add revenue forecast recommendations
    if revenue_forecast['success']:
        if revenue_forecast['trend'] == 'increasing':
            recommendations.append(
                f"■ Revenue is trending upward (+${revenue_forecast['trend_strength']:,.0f}/month). "
                f"Expected next month: ${revenue_forecast['predictions'][0]['revenue']:,.2f}"
            )
        else:
            recommendations.append(
                f"■ Revenue is trending downward (-${revenue_forecast['trend_strength']:,.0f}/month). "
                "Consider strategies to reverse this trend."
            )

        if revenue_forecast['confidence'] == 'Low':
            recommendations.append(
                "■■ Low forecast confidence. Predictions may be unreliable due to data volatility."
            )

    # Add tax optimization recommendations
    recommendations.extend(tax_optimization['recommendations'])

    # Add trend insights
    if trends['success']:
        recommendations.extend(trends['insights'])

    return {
        'revenue_forecast': revenue_forecast,
        'tax_optimization': tax_optimization,
        'trend_analysis': trends,
        'recommendations': recommendations
    }

```

File 7: Logic/pdf_generator.py

```
"""
PDF Report Generator for MoneySplit.
Creates professional PDF reports for projects, summaries, and forecasts.
"""
from reportlab.lib import colors
from reportlab.lib.pagesizes import letter, A4
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.units import inch
from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, Paragraph, Spacer, PageBreak
from reportlab.platypus import Image as RLImage
from reportlab.lib.enums import TA_CENTER, TA_RIGHT, TA_LEFT
from datetime import datetime
import os


def generate_project_pdf(record_data, people_data, filepath="reports/project_report.pdf"):
    """Generate PDF report for a single project/record."""
    os.makedirs("reports", exist_ok=True)

    doc = SimpleDocTemplate(filepath, pagesize=letter)
    story = []
    styles = getSampleStyleSheet()

    # Custom styles
    title_style = ParagraphStyle(
        'CustomTitle',
        parent=styles['Heading1'],
        fontSize=24,
        textColor=colors.HexColor('#2c3e50'),
        spaceAfter=30,
        alignment=TA_CENTER
    )

    heading_style = ParagraphStyle(
        'CustomHeading',
        parent=styles['Heading2'],
        fontSize=16,
        textColor=colors.HexColor('#34495e'),
        spaceAfter=12,
    )

    # Title
    story.append(Paragraph("■ MoneySplit Project Report", title_style))
    story.append(Spacer(1, 0.2*inch))

    # Project Info
    story.append(Paragraph("Project Information", heading_style))

    project_info = [
        ['Record ID:', str(record_data[0])],
        ['Date:', record_data[8]],
        ['Country:', record_data[1]],
        ['Tax Type:', record_data[2]],
        ['Number of People:', str(record_data[9])],
    ]

    project_table = Table(project_info, colWidths=[2*inch, 4*inch])
    project_table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (0, -1), colors.HexColor('#ecf0f1')),
        ('TEXTCOLOR', (0, 0), (-1, -1), colors.black),
        ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
        ('FONTNAME', (0, 0), (0, -1), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, -1), 10),
        ('BOTTOMPADDING', (0, 0), (-1, -1), 8),
        ('GRID', (0, 0), (-1, -1), 1, colors.grey),
    ]))
    story.append(project_table)
    story.append(Spacer(1, 0.3*inch))

    # Financial Summary
    story.append(Paragraph("Financial Summary", heading_style))

    financial_data = [
        ['Item', 'Amount'],
```

```
        ['Revenue', f"${record_data[3]:,.2f}"],
        ['Total Costs', f"${record_data[4]:,.2f}"],
        ['Group Income', f"${record_data[10]:,.2f}"],
        ['Tax Amount', f"${record_data[5]:,.2f}"],
        ['Net Income (Group)', f"${record_data[6]:,.2f}"],
        ['Net Income (Per Person)', f"${record_data[7]:,.2f}"],
    ]

    financial_table = Table(financial_data, colWidths=[3*inch, 3*inch])
    financial_table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#3498db')),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
        ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
        ('ALIGN', (1, 1), (1, -1), 'RIGHT'),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, -1), 10),
        ('BOTTOMPADDING', (0, 0), (-1, -1), 8),
        ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
        ('GRID', (0, 0), (-1, -1), 1, colors.grey),
        ('BACKGROUND', (0, -1), (-1, -1), colors.HexColor('#2ecc71')),
        ('TEXTCOLOR', (0, -1), (-1, -1), colors.whitesmoke),
        ('FONTNAME', (0, -1), (-1, -1), 'Helvetica-Bold'),
    ]))
    story.append(financial_table)
    story.append(Spacer(1, 0.3*inch))

    # Team Breakdown
    story.append(Paragraph("Team Breakdown", heading_style))

    team_data = [['Name', 'Work Share', 'Gross Income', 'Tax Paid', 'Net Income']]
    for person in people_data:
        team_data.append([
            person[1],  # name
            f"{person[2]*100:.1f}%",  # work_share
            f"${person[3]:,.2f}",  # gross_income
            f"${person[4]:,.2f}",  # tax_paid
            f"${person[5]:,.2f}",  # net_income
        ])

    team_table = Table(team_data, colWidths=[1.5*inch, 1.2*inch, 1.3*inch, 1.3*inch, 1.3*inch])
    team_table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#3498db')),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
        ('ALIGN', (0, 0), (0, -1), 'LEFT'),
        ('ALIGN', (1, 0), (-1, -1), 'CENTER'),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, -1), 9),
        ('BOTTOMPADDING', (0, 0), (-1, -1), 6),
        ('GRID', (0, 0), (-1, -1), 1, colors.grey),
        ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white, colors.lightgrey]),
    ]))
    story.append(team_table)

    # Footer
    story.append(Spacer(1, 0.5*inch))
    footer_text = f"Generated by MoneySplit on {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
    footer_style = ParagraphStyle('Footer', parent=styles['Normal'], fontSize=8, textColor=colors.grey,
alignment=TA_CEN...
    story.append(Paragraph(footer_text, footer_style))

    # Build PDF
    doc.build(story)
    return filepath


def generate_summary_pdf(records, stats, filepath="reports/summary_report.pdf"):
    """Generate PDF summary report for all records."""
    os.makedirs("reports", exist_ok=True)

    doc = SimpleDocTemplate(filepath, pagesize=letter)
    story = []
    styles = getSampleStyleSheet()

    # Custom styles
    title_style = ParagraphStyle(
        'CustomTitle',
        parent=styles['Heading1'],
        fontSize=24,
```

```
        textColor=colors.HexColor('#2c3e50'),
        spaceAfter=30,
        alignment=TA_CENTER
    )

    heading_style = ParagraphStyle(
        'CustomHeading',
        parent=styles['Heading2'],
        fontSize=16,
        textColor=colors.HexColor('#34495e'),
        spaceAfter=12,
    )

    # Title
    story.append(Paragraph("■ MoneySplit Summary Report", title_style))
    story.append(Spacer(1, 0.2*inch))

    # Overall Statistics
    story.append(Paragraph("Overall Statistics", heading_style))

    stats_data = [
        ['Metric', 'Value'],
        ['Total Records', str(stats['total_records'])],
        ['Total Revenue', f"${stats['total_revenue']:,.2f}"],
        ['Total Costs', f"${stats['total_costs']:,.2f}"],
        ['Total Tax Paid', f"${stats['total_tax']:,.2f}"],
        ['Total Net Income', f"${stats['total_net_income']:,.2f}"],
        ['Average Tax Rate', f"{stats['average_tax_rate']:.2f}%"],
        ['Unique People', str(stats['unique_people'])],
    ]

    stats_table = Table(stats_data, colWidths=[3*inch, 3*inch])
    stats_table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#3498db')),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
        ('ALIGN', (0, 0), (0, -1), 'LEFT'),
        ('ALIGN', (1, 0), (1, -1), 'RIGHT'),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, -1), 10),
        ('BOTTOMPADDING', (0, 0), (-1, -1), 8),
        ('GRID', (0, 0), (-1, -1), 1, colors.grey),
        ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.beige, colors.white]),
    ]))
    story.append(stats_table)
    story.append(Spacer(1, 0.4*inch))

    # Recent Records
    story.append(Paragraph(f"Recent Records (Last {len(records)})", heading_style))

    records_data = [['ID', 'Date', 'Country', 'Tax Type', 'Revenue', 'Tax', 'Net Income']]
    for r in records:
        records_data.append([
            str(r[0]),
            r[8][:10],  # date only
            r[1],
            r[2],
            f"${r[3]:,.0f}",
            f"${r[5]:,.0f}",
            f"${r[6]:,.0f}",
        ])

    records_table = Table(records_data, colWidths=[0.5*inch, 1*inch, 1*inch, 1.2*inch, 1.2*inch, 1*inch, 1.
2*inch])
    records_table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#3498db')),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
        ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, -1), 8),
        ('BOTTOMPADDING', (0, 0), (-1, -1), 6),
        ('GRID', (0, 0), (-1, -1), 1, colors.grey),
        ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white, colors.lightgrey]),
    ]))
    story.append(records_table)

    # Footer
    story.append(Spacer(1, 0.5*inch))
    footer_text = f"Generated by MoneySplit on {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
```

```
    footer_style = ParagraphStyle('Footer', parent=styles['Normal'], fontSize=8, textColor=colors.grey,
alignment=TA_CEN...
    story.append(Paragraph(footer_text, footer_style))

    # Build PDF
    doc.build(story)
    return filepath


def generate_forecast_pdf(forecast_data, filepath="reports/forecast_report.pdf"):
    """Generate PDF forecast report with predictions."""
    os.makedirs("reports", exist_ok=True)

    doc = SimpleDocTemplate(filepath, pagesize=letter)
    story = []
    styles = getSampleStyleSheet()

    # Custom styles
    title_style = ParagraphStyle(
        'CustomTitle',
        parent=styles['Heading1'],
        fontSize=24,
        textColor=colors.HexColor('#2c3e50'),
        spaceAfter=30,
        alignment=TA_CENTER
    )

    heading_style = ParagraphStyle(
        'CustomHeading',
        parent=styles['Heading2'],
        fontSize=16,
        textColor=colors.HexColor('#34495e'),
        spaceAfter=12,
    )

    # Title
    story.append(Paragraph("■ MoneySplit Forecast Report", title_style))
    story.append(Spacer(1, 0.2*inch))

    # Predictions
    story.append(Paragraph("Revenue Predictions (Next 3 Months)", heading_style))

    predictions_data = [['Month', 'Predicted Revenue', 'Confidence']]
    for pred in forecast_data['predictions']:
        predictions_data.append([
            pred['month'],
            f"${pred['revenue']:,.2f}",
            pred['confidence']
        ])

    pred_table = Table(predictions_data, colWidths=[2*inch, 2.5*inch, 2*inch])
    pred_table.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#3498db')),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
        ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('FONTSIZE', (0, 0), (-1, -1), 10),
        ('BOTTOMPADDING', (0, 0), (-1, -1), 8),
        ('GRID', (0, 0), (-1, -1), 1, colors.grey),
        ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.lightblue, colors.white]),
    ]))
    story.append(pred_table)
    story.append(Spacer(1, 0.3*inch))

    # Recommendations
    if 'recommendations' in forecast_data:
        story.append(Paragraph("■ Recommendations", heading_style))
        for rec in forecast_data['recommendations']:
            story.append(Paragraph(f"• {rec}", styles['BodyText']))
            story.append(Spacer(1, 0.1*inch))

    # Footer
    story.append(Spacer(1, 0.5*inch))
    footer_text = f"Generated by MoneySplit on {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
    footer_style = ParagraphStyle('Footer', parent=styles['Normal'], fontSize=8, textColor=colors.grey,
alignment=TA_CEN...
    story.append(Paragraph(footer_text, footer_style))

```

```
305 |     # Build PDF
306 |     doc.build(story)
307 |     return filepath
308 |
```

File 8: Logic/tax_calculator.py

```
"""
Tax calculation module - testable functions without side effects.
"""

from DB import setup


def calculate_tax(income: float, tax_brackets: list[tuple[float, float]]) -&gt; float:
    """
    Calculate tax using progressive tax brackets.

    Args:
        income: The income amount to calculate tax for
        tax_brackets: List of (limit, rate) tuples

    Returns:
        Total tax amount
    """
    tax = 0
    prev = 0
    for limit, rate in tax_brackets:
        if income &gt; limit:
            tax += (limit - prev) * rate
            prev = limit
        else:
            tax += (income - prev) * rate
            break
    return tax


def calculate_tax_from_db(income: float, country: str, tax_type: str) -&gt; float:
    """
    Calculate tax by fetching brackets from database.

    Args:
        income: The income amount
        country: Country name (e.g., "US", "Spain")
        tax_type: Tax type ("Individual" or "Business")

    Returns:
        Total tax amount

    Raises:
        ValueError: If no tax brackets found for country/type
    """
    brackets = setup.get_tax_brackets(country, tax_type)
    if not brackets:
        raise ValueError(f"No tax brackets found for {country} {tax_type}")

    return calculate_tax(income, brackets)


def split_work_shares(total_amount: float, work_shares: list[float]) -&gt; list[float]:
    """
    Distribute total amount based on work shares.

    Args:
        total_amount: Total amount to distribute
        work_shares: List of work share percentages (should sum to 1.0)

    Returns:
        List of distributed amounts
    """
    return [total_amount * share for share in work_shares]


def calculate_profit(revenue: float, costs: list[float]) -&gt; float:
    """
    Calculate profit from revenue and costs.

    Args:
        revenue: Total revenue
        costs: List of cost amounts

```

```
    Returns:
        Profit amount (revenue - total costs)
    """
    return revenue - sum(costs)
```

File 9: Logic/validators.py

```
"""
Input validation utilities for MoneySplit.
"""

class ValidationError(Exception):
    """Custom exception for validation errors."""
    pass


def validate_positive_number(value: float, field_name: str) -&gt; float:
    """Validate that a number is positive."""
    if value &lt; 0:
        raise ValidationError(f"{field_name} must be positive, got {value}")
    return value


def validate_work_shares(shares: list[float]) -&gt; None:
    """Validate that work shares sum to 1.0 (with small tolerance)."""
    total = sum(shares)
    if abs(total - 1.0) &gt; 0.01:
        raise ValidationError(f"Work shares must sum to 1.0, got {total:.2f}")


def validate_work_share(share: float) -&gt; float:
    """Validate a single work share is between 0 and 1."""
    if not 0 &lt;= share &lt;= 1:
        raise ValidationError(f"Work share must be between 0 and 1, got {share}")
    return share


def validate_non_empty_string(value: str, field_name: str) -&gt; str:
    """Validate that a string is not empty."""
    value = value.strip()
    if not value:
        raise ValidationError(f"{field_name} cannot be empty")
    return value


def validate_country(country: str) -&gt; str:
    """Validate country name (allows any non-empty string)."""
    country = country.strip()
    if not country:
        raise ValidationError("Country cannot be empty")
    return country


def validate_tax_type(tax_type: str) -&gt; str:
    """Validate tax type."""
    valid_types = ["Individual", "Business"]
    tax_type = tax_type.strip().title()
    if tax_type not in valid_types:
        raise ValidationError(f"Tax type must be one of {valid_types}, got '{tax_type}'")
    return tax_type


def validate_tax_rate(rate: float) -&gt; float:
    """Validate tax rate is between 0 and 1."""
    if not 0 &lt;= rate &lt;= 1:
        raise ValidationError(f"Tax rate must be between 0 and 1, got {rate}")
    return rate


def safe_float_input(prompt: str, field_name: str = "Value", allow_negative: bool = False) -&gt; float:
    """Safely get float input from user with validation."""
    while True:
        try:
            value = float(input(prompt))
            if not allow_negative:
                validate_positive_number(value, field_name)
            return value
        except ValueError:
            print(f"■ Invalid number. Please enter a valid number for {field_name}.")
        except ValidationError as e:
            print(f"■ {e}")
```

```
  75 |
  76 |
  77 | def safe_int_input(prompt: str, field_name: str = "Value", min_value: int = None, max_value: int = None) -&gt;
int:
  78 |     """Safely get integer input from user with validation."""
  79 |     while True:
  80 |         try:
  81 |             value = int(input(prompt))
  82 |             if min_value is not None and value &lt; min_value:
  83 |                 print(f"■ {field_name} must be at least {min_value}.")
  84 |                 continue
  85 |             if max_value is not None and value &gt; max_value:
  86 |                 print(f"■ {field_name} must be at most {max_value}.")
  87 |                 continue
  88 |             return value
  89 |         except ValueError:
  90 |             print(f"■ Invalid integer. Please enter a valid integer for {field_name}.")
  91 |
  92 |
  93 | def safe_string_input(prompt: str, field_name: str = "Value") -&gt; str:
  94 |     """Safely get non-empty string input from user."""
  95 |     while True:
  96 |         try:
  97 |             value = input(prompt).strip()
  98 |             return validate_non_empty_string(value, field_name)
  99 |         except ValidationError as e:
 100 |             print(f"■ {e}")
 101 |
```

File 10: Menus/__init__.py

```
1 |
```

File 11: Menus/data_menu.py

```
  1 | from DB import setup
  2 |
  3 | def data_menu():
  4 |     while True:
  5 |         print("\n=== Data Menu ■ ===")
  6 |         print("1. Export data to CSV")
  7 |         print("2. Export data to JSON")
  8 |         print("3. Import data from CSV")
  9 |         print("4. Import data from JSON")
 10 |         print("5. Back")
 11 |
 12 |         choice = input("Choose an option (1-5): ").strip()
 13 |         if choice == "1":
 14 |             filepath = input("Enter base filename (default: export): ").strip() or "export"
 15 |             setup.export_to_csv(filepath)
 16 |         elif choice == "2":
 17 |             filepath = input("Enter filename (default: export.json): ").strip() or "export.json"
 18 |             setup.export_to_json(filepath)
 19 |         elif choice == "3":
 20 |             rec_file = input("Enter records CSV filename (default: export_records.csv): ").strip() or
"export_records.cs...
 21 |             ppl_file = input("Enter people CSV filename (default: export_people.csv): ").strip() or
"export_people.csv"
 22 |             setup.import_from_csv(rec_file, ppl_file)
 23 |         elif choice == "4":
 24 |             filepath = input("Enter JSON filename (default: export.json): ").strip() or "export.json"
 25 |             setup.import_from_json(filepath)
 26 |         elif choice == "5":
 27 |             break
 28 |         else:
 29 |             print("■ Invalid choice. Please enter 1-5.")
 30 |
```

File 12: Menus/db_menu.py

```
from DB import setup
from Logic import validators

def show_last_records(n=5):
    print(f"\n=== Last {n} Saved Records ===")
    records = setup.fetch_last_records(n)

    if not records:
        print("No records found.")
        return

    header = (
        f"{'ID':&lt;3} | {'Origin':&lt;6} | {'Option':&lt;10} | "
        f"{'Revenue':&gt;12} | {'Costs':&gt;10} | {'Tax':&gt;10} | "
        f"{'Net Group':&gt;12} | {'Net Person':&gt;12} | {'Created At'}"
    )
    print(header)
    print("-" * len(header))

    for r in records:
        id, origin, option, revenue, costs, tax, net_group, net_person, created = r
        print(
            f"{id:&lt;3} | {origin:&lt;6} | {option:&lt;10} | "
            f"{float(revenue):&gt;12,.2f} | {float(costs):&gt;10,.2f} | {float(tax):&gt;10,.2f} | "
            f"{float(net_group):&gt;12,.2f} | {float(net_person):&gt;12,.2f} | {created}"
        )


def show_people_for_record():
    try:
        record_id = int(input("Enter the ID of the record to view people: "))
        people = setup.fetch_people_by_record(record_id)

        if not people:
            print(f"■ No people found for record {record_id}.")
            return

        print(f"\n=== People for Record {record_id} ===")
        header = f"{'ID':&lt;3} | {'Name':&lt;10} | {'Work Share':&gt;10} | {'Gross':&gt;12} | {'Tax Paid':&gt;
10} | {'N...
        print(header)
        print("-" * len(header))

        for p in people:
            pid, name, work_share, gross, tax_paid, net_income = p
            print(
                f"{pid:&lt;3} | {name:&lt;10} | {work_share:&gt;10.2f} | "
                f"{gross:&gt;12,.2f} | {tax_paid:&gt;10,.2f} | {net_income:&gt;12,.2f}"
            )

    except ValueError:
        print("■ Invalid input. Please enter a number.")


def delete_record_menu():
    try:
        record_id = int(input("Enter the ID of the record to delete: "))
        record = setup.get_record_by_id(record_id)

        if not record:
            print(f"■ No record found with ID {record_id}.")
            return

        confirm = input(f"Are you sure you want to delete record {record_id} (and all linked people)? (y/n): ").
strip()....
        if confirm == "y":
            setup.delete_record(record_id)
        else:
            print("■ Deletion canceled.")

    except ValueError:
        print("■ Invalid ID. Please enter a number.")


```

```
def update_record_menu():
    try:
        record_id = validators.safe_int_input("Enter the ID of the record to update: ", "Record ID", min_value=1)
        record = setup.get_record_by_id(record_id)

        if not record:
            print(f"■ No record found with ID {record_id}.")
            return

        print("\nYou can update the following fields:")
        for f in sorted(setup.ALLOWED_FIELDS):
            print(f" - {f}")

        field = validators.safe_string_input("\nEnter the field to update: ", "Field name")

        if field not in setup.ALLOWED_FIELDS:
            print(f"■ '{field}' is not editable. Allowed: {', '.join(sorted(setup.ALLOWED_FIELDS))}")
            return

        if field == "num_people":
            new_value = validators.safe_int_input(f"Enter new value for {field}: ", field, min_value=1)
        elif field in ["revenue", "total_costs"]:
            new_value = validators.safe_float_input(f"Enter new value for {field}: ", field)
        else:
            new_value = validators.safe_string_input(f"Enter new value for {field}: ", field)

        setup.update_record(record_id, field, new_value)

    except validators.ValidationError as e:
        print(f"■ {e}")


def show_person_history():
    name = input("Enter the person's name: ").strip()
    records = setup.fetch_records_by_person(name)

    if not records:
        print(f"■ No records found for {name}.")
        return

    print(f"\n=== Records for {name} ===")
    header = f"{'PersonID':&lt;8} | {'RecordID':&lt;8} | {'Work Share':&gt;10} | {'Gross':&gt;12} | {'Tax Paid':
&gt;10} ...
    print(header)
    print("-" * len(header))

    total_gross = total_tax = total_net = 0

    for r in records:
        pid, record_id, pname, work_share, gross, tax_paid, net_income, created = r
        print(
            f"{pid:&lt;8} | {record_id:&lt;8} | {work_share:&gt;10.2f} | "
            f"{gross:&gt;12,.2f} | {tax_paid:&gt;10,.2f} | {net_income:&gt;12,.2f} | {created}"
        )
        total_gross += gross
        total_tax += tax_paid
        total_net += net_income

    print("\n--- Totals ---")
    print(f"Total Gross: {total_gross:,.2f}")
    print(f"Total Tax:   {total_tax:,.2f}")
    print(f"Total Net:   {total_net:,.2f}")


def update_person_menu():
    try:
        person_id = validators.safe_int_input("Enter the ID of the person to update: ", "Person ID", min_value=1)
        print("\nYou can update the following fields:")
        print(" - name")
        print(" - work_share")

        field = validators.safe_string_input("Enter the field to update: ", "Field name")

        if field == "work_share":
            new_value = validators.safe_float_input(f"Enter new value for {field} (0.0-1.0): ", field)
            new_value = validators.validate_work_share(new_value)
        elif field == "name":
            new_value = validators.safe_string_input(f"Enter new value for {field}: ", field)
```

```
        else:
            print(f"■ Invalid field. Only 'name' and 'work_share' can be updated.")
            return

        setup.update_person(person_id, field, new_value)

    except validators.ValidationError as e:
        print(f"■ {e}")


def delete_person_menu():
    try:
        person_id = int(input("Enter the ID of the person to delete: "))
        confirm = input(f"Are you sure you want to delete person {person_id}? (y/n): ").strip().lower()
        if confirm == "y":
            setup.delete_person(person_id)
        else:
            print("■ Deletion canceled.")
    except ValueError:
        print("■ Invalid input. Please enter a number.")


def deduplicate_people_menu():
    try:
        record_id = int(input("Enter the record ID to deduplicate people: "))
        setup.deduplicate_people(record_id)
    except ValueError:
        print("■ Invalid input. Please enter a number.")


# --- Maintenance ---
def reset_db_menu():
    confirm = input("■■ This will DELETE ALL tax records and people. Type 'RESET' to confirm: ").strip()
    if confirm == "RESET":
        setup.reset_db()
    else:
        print("■ Reset canceled.")


def reset_tax_brackets_menu():
    confirm = input("■■ This will DELETE ALL tax brackets and restore defaults. Type 'RESET' to confirm: ").
strip()
    if confirm == "RESET":
        setup.reset_tax_brackets()
    else:
        print("■ Reset canceled.")


def export_template_menu():
    filepath = input("Enter filename for template (default: tax_template.csv): ").strip()
    if not filepath:
        filepath = "tax_template.csv"
    setup.export_tax_template(filepath)


def view_tax_brackets_menu():
    country = input("Enter country (e.g. US, Spain): ").strip()
    tax_type = input("Enter type (Individual/Business): ").strip().title()
    rows = setup.get_tax_brackets(country, tax_type, include_id=True)
    if not rows:
        print("■ No brackets found.")
    else:
        print(f"\nBrackets for {country} {tax_type}:")
        print(f"{'ID':&lt;4} | {'Income Limit':&gt;15} | {'Rate':&gt;8}")
        print("-" * 35)
        for bid, limit, rate in rows:
            limit_txt = "∞" if limit == float("inf") else f"{limit:,.0f}"
            print(f"{bid:&lt;4} | {limit_txt:&gt;15} | {rate*100:&gt;7.2f}%")


# --- Advanced ---
def clone_record_menu():
    try:
        record_id = int(input("Enter the ID of the record to clone: "))
        setup.clone_record(record_id)
    except ValueError:
        print("■ Invalid input. Please enter a number.")

```

```
|
| def copy_people_menu():
|     try:
|         source_id = int(input("Enter source record ID: "))
|         target_id = int(input("Enter target record ID: "))
|         setup.copy_people(source_id, target_id)
|
|         # ■ Auto-update num_people in target
|         people = setup.fetch_people_by_record(target_id)
|         new_count = len(people)
|         setup.update_record(target_id, "num_people", new_count)
|
|         # ■ Run deduplication
|         removed = setup.deduplicate_people(target_id)
|         print(f"■ Target record {target_id} updated. num_people = {new_count}. "
|               f"Deduplicated {removed} duplicate(s).")
|
|     except ValueError:
|         print("■ Invalid input. Please enter numbers.")
|
|
| def advanced_options_menu():
|     while True:
|         print("\n=== Advanced DB Options ===")
|         print("1. Clone a record")
|         print("2. Copy people between records")
|         print("3. Back")
|
|         choice = input("Choose an option (1-3): ").strip()
|
|         if choice == "1":
|             clone_record_menu()
|         elif choice == "2":
|             copy_people_menu()
|         elif choice == "3":
|             break
|         else:
|             print("■ Invalid choice. Please enter 1-3.")
|
|
| # --- Submenus ---
| def search_records_menu():
|     print("\n=== Search Filters ===")
|     country = input("Country (leave blank to skip): ").strip() or None
|     tax_option = input("Tax option (Individual/Business, leave blank to skip): ").strip().title() or None
|     start_date = input("Start date (YYYY-MM-DD, leave blank to skip): ").strip() or None
|     end_date = input("End date (YYYY-MM-DD, leave blank to skip): ").strip() or None
|
|     rows = setup.search_records(country, tax_option, start_date, end_date)
|     if not rows:
|         print("■ No matching records found.")
|         return
|
|     print(f"\nFound {len(rows)} matching records:")
|     header = (
|         f"{'ID':&lt;3} | {'Origin':&lt;6} | {'Option':&lt;10} | "
|         f"{'Revenue':&gt;12} | {'Costs':&gt;10} | {'Tax':&gt;10} | "
|         f"{'Net Group':&gt;12} | {'Net Person':&gt;12} | {'Created At'}"
|     )
|     print(header)
|     print("-" * len(header))
|
|     for r in rows:
|         id, origin, option, revenue, costs, tax, net_group, net_person, created = r
|         print(
|             f"{id:&lt;3} | {origin:&lt;6} | {option:&lt;10} | "
|             f"{float(revenue):&gt;12,.2f} | {float(costs):&gt;10,.2f} | {float(tax):&gt;10,.2f} | "
|             f"{float(net_group):&gt;12,.2f} | {float(net_person):&gt;12,.2f} | {created}"
|         )
|
|
| def merge_records_menu():
|     try:
|         r1 = int(input("Enter the first record ID: "))
|         r2 = int(input("Enter the second record ID: "))
|         new_id = setup.merge_records(r1, r2)
|
|         # ■ Run deduplication
```

```
        removed = setup.deduplicate_people(new_id)
        print(f"■ Merged into record {new_id}. Deduplicated {removed} duplicate(s).")

    except ValueError:
        print("■ Invalid input. Please enter numbers.")

def records_menu():
    while True:
        print("\n=== Records Menu ■ ===")
        print("1. View last 5 records")
        print("2. Search records")
        print("3. Clone record")
        print("4. Merge records")
        print("5. Update record by ID")
        print("6. Delete record by ID")
        print("7. Advanced options ■■")
        print("8. Back")

        choice = input("Choose an option (1-8): ").strip()
        if choice == "1":
            show_last_records(5)
        elif choice == "2":
            search_records_menu()
        elif choice == "3":
            clone_record_menu()
        elif choice == "4":
            merge_records_menu()
        elif choice == "5":
            update_record_menu()
        elif choice == "6":
            delete_record_menu()
        elif choice == "7":
            advanced_options_menu()
        elif choice == "8":
            break
        else:
            print("■ Invalid choice. Please enter 1-8.")


def people_menu():
    while True:
        print("\n=== People Menu ■ ===")
        print("1. View people for a record")
        print("2. View person history")
        print("3. Update person by ID")
        print("4. Delete person by ID")
        print("5. Deduplicate people in record ■")
        print("6. Back")

        choice = input("Choose an option (1-6): ").strip()
        if choice == "1":
            show_people_for_record()
        elif choice == "2":
            show_person_history()
        elif choice == "3":
            update_person_menu()
        elif choice == "4":
            delete_person_menu()
        elif choice == "5":
            deduplicate_people_menu()
        elif choice == "6":
            break
        else:
            print("■ Invalid choice. Please enter 1-6.")


def maintenance_menu():
    while True:
        print("\n=== Maintenance Menu ■■ ===")
        print("1. Reset database ■■")
        print("2. Reset tax brackets ■■")
        print("3. Export CSV template")
        print("4. View tax brackets")
        print("5. Global deduplication ■")
        print("6. Back")

        choice = input("Choose an option (1-6): ").strip()
        if choice == "1":
```

```
            reset_db_menu()
        elif choice == "2":
            reset_tax_brackets_menu()
        elif choice == "3":
            export_template_menu()
        elif choice == "4":
            view_tax_brackets_menu()
        elif choice == "5":
            setup.deduplicate_all_records()
        elif choice == "6":
            break
        else:
            print("■ Invalid choice. Please enter 1-6.")


# --- Main ---
def show_db_menu():
    while True:
        print("\n=== DB Menu ===")
        print("1. Records ■")
        print("2. People ■")
        print("3. Maintenance ■■")
        print("4. Back to main menu")

        choice = input("Choose an option (1-4): ").strip()
        if choice == "1":
            records_menu()
        elif choice == "2":
            people_menu()
        elif choice == "3":
            maintenance_menu()
        elif choice == "4":
            break
        else:
            print("■ Invalid choice. Please enter 1-4.")

```

File 13: Menus/project_menu.py

```
import sys, importlib
from DB import setup
from Menus import report_menu


def run_new_project():
    """Run a fresh MoneySplit calculation and save it to DB."""
    importlib.invalidate_caches()
    if "Logic.ProgramBackend" in sys.modules:
        del sys.modules["Logic.ProgramBackend"]

    # This import executes ProgramBackend.py (asks for inputs, runs calculation, and saves to DB)
    pb = importlib.import_module("Logic.ProgramBackend")

    # Get the record_id that was already saved in ProgramBackend
    record_id = pb.LAST_RECORD_ID
    print(f"\n■ Project results saved (record {record_id}).")
    print("■ Calculation finished and stored in database.")

    # Auto-report (summary + top contributors)
    print("\n■ Auto-generated report:")
    report_menu.show_report_menu(auto=True, record_id=record_id)

```

File 14: Menus/report_menu.py

Note: This file has 1238 lines. First 500 lines shown.

```
  1 | from DB import setup
  2 | from Logic import pdf_generator, forecasting
  3 | import csv
  4 | import plotly.graph_objects as go
  5 | import plotly.express as px
  6 | from plotly.subplots import make_subplots
  7 | import webbrowser
  8 | import os
  9 |
 10 | def summary_report():
 11 |     conn = setup.get_conn()
 12 |     cursor = conn.cursor()
 13 |
 14 |     cursor.execute("""
 15 |         SELECT tax_origin, tax_option,
 16 |                COUNT(*), SUM(revenue), SUM(total_costs), SUM(tax_amount),
 17 |                SUM(net_income_group)
 18 |         FROM tax_records
 19 |         GROUP BY tax_origin, tax_option
 20 |     """)
 21 |     rows = cursor.fetchall()
 22 |     conn.close()
 23 |
 24 |     print("\n=== Summary Report ===")
 25 |     print(f"{'Origin':&lt;8} | {'Option':&lt;10} | {'Records':&lt;7} | {'Revenue':&gt;12} | {'Costs':&gt;12} | {
    'Tax':&g...
 26 |     print("-" * 75)
 27 |     for origin, option, cnt, rev, cost, tax, net in rows:
 28 |         print(f"{origin:&lt;8} | {option:&lt;10} | {cnt:&lt;7} | {rev:&gt;12,.2f} | {cost:&gt;12,.2f} | {tax:&gt;
    12,.2f}...
 29 |
 30 |
 31 | def person_report():
 32 |     conn = setup.get_conn()
 33 |     cursor = conn.cursor()
 34 |
 35 |     cursor.execute("""
 36 |         SELECT name, SUM(gross_income), SUM(tax_paid), SUM(net_income)
 37 |         FROM people
 38 |         GROUP BY name
 39 |         ORDER BY SUM(gross_income) DESC
 40 |     """)
 41 |     rows = cursor.fetchall()
 42 |     conn.close()
 43 |
 44 |     print("\n=== Per-Person Report ===")
 45 |     print(f"{'Name':&lt;12} | {'Gross':&gt;12} | {'Tax Paid':&gt;12} | {'Net':&gt;12}")
 46 |     print("-" * 55)
 47 |     for name, gross, tax, net in rows:
 48 |         print(f"{name:&lt;12} | {gross:&gt;12,.2f} | {tax:&gt;12,.2f} | {net:&gt;12,.2f}")
 49 |
 50 |
 51 | def record_stats():
 52 |     conn = setup.get_conn()
 53 |     cursor = conn.cursor()
 54 |
 55 |     cursor.execute("""
 56 |         SELECT COUNT(*), AVG(revenue), AVG(tax_amount), MIN(net_income_group), MAX(net_income_group)
 57 |         FROM tax_records
 58 |     """)
 59 |     total, avg_rev, avg_tax, min_net, max_net = cursor.fetchone()
 60 |     conn.close()
 61 |
 62 |     print("\n=== Record Statistics ===")
 63 |     print(f"Total Records: {total}")
 64 |     print(f"Average Revenue: {avg_rev:,.2f}")
 65 |     print(f"Average Tax: {avg_tax:,.2f}")
 66 |     print(f"Min Net Income (Group): {min_net:,.2f}")
 67 |     print(f"Max Net Income (Group): {max_net:,.2f}")
 68 |
 69 |
 70 | def show_report_menu():
```

```
    while True:
        print("\n=== Reports ■ ===")
        print("1. Summary Report")
        print("2. Per-Person Report")
        print("3. Record Statistics")
        print("4. Back")

        choice = input("Choose an option (1-4): ").strip()
        if choice == "1":
            summary_report()
        elif choice == "2":
            person_report()
        elif choice == "3":
            record_stats()
        elif choice == "4":
            break
        else:
            print("■ Invalid choice.")

def export_to_csv(filename, headers, rows):
    with open(filename, "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(headers)
        writer.writerows(rows)
    print(f"■ Exported report → {filename}")

def revenue_summary_report():
    rows = setup.get_revenue_summary()
    if not rows:
        print("■ No data found.")
        return

    # Extract data
    years = [row[0] for row in rows]
    revenues = [row[1] for row in rows]
    costs = [row[2] for row in rows]
    net_incomes = [row[3] for row in rows]

    # Create interactive chart
    fig = make_subplots(
        rows=2, cols=1,
        subplot_titles=("Revenue &amp;amp; Costs by Year", "Net Income by Year"),
        vertical_spacing=0.15
    )

    # Revenue and Costs
    fig.add_trace(
        go.Bar(name="Revenue", x=years, y=revenues, marker_color='rgb(55, 83, 109)'),
        row=1, col=1
    )
    fig.add_trace(
        go.Bar(name="Costs", x=years, y=costs, marker_color='rgb(219, 64, 82)'),
        row=1, col=1
    )

    # Net Income
    fig.add_trace(
        go.Scatter(name="Net Income", x=years, y=net_incomes,
                   mode='lines+markers', marker_color='rgb(50, 171, 96)',
                   line=dict(width=3)),
        row=2, col=1
    )

    fig.update_layout(
        title_text="Revenue Summary by Year",
        showlegend=True,
        height=700
    )
    fig.update_xaxes(title_text="Year", row=2, col=1)
    fig.update_yaxes(title_text="Amount ($)", row=1, col=1)
    fig.update_yaxes(title_text="Net Income ($)", row=2, col=1)

    # Save and open
    filepath = "reports/revenue_summary.html"
    os.makedirs("reports", exist_ok=True)
    fig.write_html(filepath)
    webbrowser.open('file://' + os.path.abspath(filepath))
    print(f"■ Visualization opened in browser: {filepath}")
```

```
|
|     # Also print text summary
|     print("\n=== Revenue Summary (Text) ===")
|     print(f"{'Year':&lt;6} | {'Total Revenue':&gt;15} | {'Total Costs':&gt;15} | {'Net Income':&gt;15}")
|     print("-" * 60)
|     for year, rev, cost, net in rows:
|         print(f"{year:&lt;6} | {rev:&gt;15,.2f} | {cost:&gt;15,.2f} | {net:&gt;15,.2f}")
|
|     choice = input("\nExport to CSV? (y/n): ").strip().lower()
|     if choice == "y":
|         filename = input("Enter filename (default: report_revenue_summary.csv): ").strip()
|         if not filename:
|             filename = "report_revenue_summary.csv"
|         elif not filename.endswith(".csv"):
|             filename += ".csv"
|         headers = ["Year", "Total Revenue", "Total Costs", "Net Income"]
|         export_to_csv(filename, headers, rows)
|
| def top_people_report():
|     rows = setup.get_top_people()
|     if not rows:
|         print("■ No data found.")
|         return
|
|     # Extract data
|     names = [row[0] for row in rows]
|     gross_incomes = [row[1] for row in rows]
|     taxes_paid = [row[2] for row in rows]
|     net_incomes = [row[3] for row in rows]
|
|     # Create horizontal bar chart
|     fig = make_subplots(
|         rows=1, cols=2,
|         subplot_titles=("Net Income", "Gross Income vs Tax Paid"),
|         specs=[[{"type": "bar"}, {"type": "bar"}]]
|     )
|
|     # Net income chart
|     fig.add_trace(
|         go.Bar(name="Net Income", y=names, x=net_incomes, orientation='h',
|                marker_color='rgb(50, 171, 96)', text=net_incomes,
|                texttemplate='$%{text:,.0f}', textposition='outside'),
|         row=1, col=1
|     )
|
|     # Gross vs Tax chart
|     fig.add_trace(
|         go.Bar(name="Gross Income", y=names, x=gross_incomes, orientation='h',
|                marker_color='rgb(55, 83, 109)'),
|         row=1, col=2
|     )
|     fig.add_trace(
|         go.Bar(name="Tax Paid", y=names, x=taxes_paid, orientation='h',
|                marker_color='rgb(219, 64, 82)'),
|         row=1, col=2
|     )
|
|     fig.update_layout(
|         title_text="Top People by Net Income",
|         showlegend=True,
|         height=max(400, len(names) * 40),
|         barmode='group'
|     )
|     fig.update_xaxes(title_text="Net Income ($)", row=1, col=1)
|     fig.update_xaxes(title_text="Amount ($)", row=1, col=2)
|
|     # Save and open
|     filepath = "reports/top_people.html"
|     os.makedirs("reports", exist_ok=True)
|     fig.write_html(filepath)
|     webbrowser.open('file://' + os.path.abspath(filepath))
|     print(f"■ Visualization opened in browser: {filepath}")
|
|     # Also print text summary
|     headers = ["Name", "Total Gross", "Total Tax Paid", "Total Net Income"]
|     print("\n=== Top People by Net Income (Text) ===")
|     print(f"{headers[0]:&lt;15} | {headers[1]:&gt;15} | {headers[2]:&gt;15} | {headers[3]:&gt;15}")
|     print("-" * 70)
```

```
    for name, gross, tax, net in rows:
        print(f"{name:&lt;15} | {gross:&gt;15,.2f} | {tax:&gt;15,.2f} | {net:&gt;15,.2f}")

    choice = input("\nExport to CSV? (y/n): ").strip().lower()
    if choice == "y":
        filename = input("Enter filename (default: report_top_people.csv): ").strip()
        if not filename:
            filename = "report_top_people.csv"
        elif not filename.endswith(".csv"):
            filename += ".csv"
        export_to_csv(filename, headers, rows)

def show_report_menu():
    while True:
        print("\n=== Reports Menu ■ ===")
        print("1. Revenue summary by year")
        print("2. Top people by net income")
        print("3. Back")

        choice = input("Choose an option (1-3): ").strip()
        if choice == "1":
            revenue_summary_report()
        elif choice == "2":
            top_people_report()
        elif choice == "3":
            break
        else:
            print("■ Invalid choice. Please enter 1-3.")


def show_summary(record_id: int):
    """Show summary for a given record."""
    rec = setup.get_record_by_id(record_id)
    if not rec:
        print(f"■ Record {record_id} not found.")
        return

    rid, origin, option, revenue, costs, tax, net_group, net_person, created = rec
    print(f"\n=== Summary for Record {rid} ({origin} {option}) ===")
    print(f"Revenue:        {float(revenue):,.2f}")
    print(f"Total Costs:    {float(costs):,.2f}")
    print(f"Tax Paid:       {float(tax):,.2f}")
    print(f"Net Group:      {float(net_group):,.2f}")
    print(f"Net Per Person: {float(net_person):,.2f}")


def show_top_contributors(record_id: int, top_n: int = 5):
    """Show top contributors (by net income) for a record."""
    people = setup.fetch_people_by_record(record_id)
    if not people:
        print(f"■ No people found for record {record_id}.")
        return

    ranked = sorted(people, key=lambda x: x[5], reverse=True)  # net_income = idx 5
    print(f"\n=== Top {min(top_n, len(ranked))} Contributors (by Net Income) ===")
    for i, (pid, name, ws, gross, tax_paid, net) in enumerate(ranked[:top_n], start=1):
        print(f"{i}. {name:&lt;12} → Net {net:,.2f} (Gross {gross:,.2f}, Tax {tax_paid:,.2f})")


def single_record_menu():
    """Menu for single record reports."""
    while True:
        print("\n=== Single Record Reports ■ ===")
        print("1. View summary for a record")
        print("2. View top contributors for a record")
        print("3. Export record to PDF ■")
        print("4. Back")

        choice = input("Choose an option (1-4): ").strip()

        if choice == "1":
            try:
                record_id = int(input("Enter record ID: "))
                show_summary(record_id)
            except ValueError:
                print("■ Invalid input.")
        elif choice == "2":
            try:
```

```
                record_id = int(input("Enter record ID: "))
                show_top_contributors(record_id)
            except ValueError:
                print("■ Invalid input.")
        elif choice == "3":
            export_record_to_pdf()
        elif choice == "4":
            break
        else:
            print("■ Invalid choice. Please enter 1-4.")


def aggregate_reports_menu():
    """Menu for aggregate reports across all data."""
    while True:
        print("\n=== Aggregate Reports ■ ===")
        print("1. Revenue summary by year")
        print("2. Top people (across all records)")
        print("3. Tax strategy comparison")
        print("4. Overall statistics")
        print("5. Monthly trends ■")
        print("6. Work distribution analysis ■")
        print("7. Person performance timeline ■■")
        print("8. Tax efficiency report ■")
        print("9. Project profitability analysis ■")
        print("10. Revenue forecasting &amp;amp; predictions ■")
        print("11. Export summary to PDF ■")
        print("12. Back")

        choice = input("Choose an option (1-12): ").strip()

        if choice == "1":
            revenue_summary_report()
        elif choice == "2":
            top_people_report()
        elif choice == "3":
            tax_type_comparison_report()
        elif choice == "4":
            overall_statistics()
        elif choice == "5":
            monthly_trends_report()
        elif choice == "6":
            work_distribution_report()
        elif choice == "7":
            person_performance_timeline()
        elif choice == "8":
            tax_efficiency_report()
        elif choice == "9":
            project_profitability_report()
        elif choice == "10":
            show_forecast_report()
        elif choice == "11":
            export_summary_to_pdf()
        elif choice == "12":
            break
        else:
            print("■ Invalid choice. Please enter 1-12.")


def tax_type_comparison_report():
    """Compare Individual vs Business tax strategies."""
    conn = setup.get_conn()
    cursor = conn.cursor()

    cursor.execute("""
        SELECT tax_origin, tax_option,
               COUNT(*) as records,
               AVG(revenue) as avg_revenue,
               AVG(tax_amount) as avg_tax,
               AVG(tax_amount * 100.0 / NULLIF(group_income, 0)) as avg_tax_rate,
               SUM(net_income_group) as total_net
        FROM tax_records
        GROUP BY tax_origin, tax_option
        ORDER BY tax_origin, tax_option
    """)
    rows = cursor.fetchall()
    conn.close()

```

```
    if not rows:
        print("■ No data found.")
        return

    # Extract data
    labels = [f"{row[0]} - {row[1]}" for row in rows]
    avg_revenues = [row[3] for row in rows]
    avg_taxes = [row[4] for row in rows]
    avg_rates = [row[5] for row in rows]
    total_nets = [row[6] for row in rows]

    # Create comparison charts
    fig = make_subplots(
        rows=2, cols=2,
        subplot_titles=("Average Tax Rate (%)", "Total Net Income",
                        "Average Revenue vs Tax", "Record Distribution"),
        specs=[[{"type": "bar"}, {"type": "bar"}],
               [{"type": "bar"}, {"type": "pie"}]]
    )

    # Avg tax rate
    fig.add_trace(
        go.Bar(x=labels, y=avg_rates, name="Avg Tax Rate",
               marker_color='rgb(219, 64, 82)', text=avg_rates,
               texttemplate='%{text:.2f}%', textposition='outside'),
        row=1, col=1
    )

    # Total net income
    fig.add_trace(
        go.Bar(x=labels, y=total_nets, name="Total Net",
               marker_color='rgb(50, 171, 96)', text=total_nets,
               texttemplate='$%{text:,.0f}', textposition='outside'),
        row=1, col=2
    )

    # Avg revenue vs tax
    fig.add_trace(
        go.Bar(x=labels, y=avg_revenues, name="Avg Revenue",
               marker_color='rgb(55, 83, 109)'),
        row=2, col=1
    )
    fig.add_trace(
        go.Bar(x=labels, y=avg_taxes, name="Avg Tax",
               marker_color='rgb(219, 64, 82)'),
        row=2, col=1
    )

    # Record distribution pie
    record_counts = [row[2] for row in rows]
    fig.add_trace(
        go.Pie(labels=labels, values=record_counts, textinfo='label+percent+value'),
        row=2, col=2
    )

    fig.update_layout(
        title_text="Tax Strategy Comparison",
        showlegend=True,
        height=800
    )
    fig.update_yaxes(title_text="Rate (%)", row=1, col=1)
    fig.update_yaxes(title_text="Net Income ($)", row=1, col=2)
    fig.update_yaxes(title_text="Amount ($)", row=2, col=1)

    # Save and open
    filepath = "reports/tax_strategy_comparison.html"
    os.makedirs("reports", exist_ok=True)
    fig.write_html(filepath)
    webbrowser.open('file://' + os.path.abspath(filepath))
    print(f"■ Visualization opened in browser: {filepath}")

    # Also print text summary
    print("\n=== Tax Strategy Comparison (Text) ===")
    print(f"{'Origin':&lt;8} | {'Strategy':&lt;10} | {'Records':&lt;7} | {'Avg Revenue':&gt;12} | {'Avg Tax':&gt;
12} | {...
    print("-" * 90)
    for origin, option, cnt, avg_rev, avg_tax, avg_rate, total_net in rows:
        print(f"{origin:&lt;8} | {option:&lt;10} | {cnt:&lt;7} | {avg_rev:&gt;12,.2f} | {avg_tax:&gt;12,.2f} | {
```

```
avg_rate...
```

```
 460 |
 461 |
 462 | def overall_statistics():
 463 |     """Show overall database statistics."""
 464 |     conn = setup.get_conn()
 465 |     cursor = conn.cursor()
 466 |
 467 |     # Records stats
 468 |     cursor.execute("""
 469 |         SELECT COUNT(*),
 470 |                SUM(revenue),
 471 |                SUM(total_costs),
 472 |                SUM(tax_amount),
 473 |                SUM(net_income_group),
 474 |                AVG(tax_amount * 100.0 / NULLIF(group_income, 0))
 475 |         FROM tax_records
 476 |     """)
 477 |     total_records, total_rev, total_costs, total_tax, total_net, avg_rate = cursor.fetchone()
 478 |
 479 |     # People stats
 480 |     cursor.execute("SELECT COUNT(*), COUNT(DISTINCT name) FROM people")
 481 |     total_people_entries, unique_people = cursor.fetchone()
 482 |
 483 |     conn.close()
 484 |
 485 |     if total_records == 0:
 486 |         print("■ No data found.")
 487 |         return
 488 |
 489 |     # Create dashboard with multiple visualizations
 490 |     fig = make_subplots(
 491 |         rows=2, cols=2,
 492 |         subplot_titles=("Financial Breakdown", "Revenue Flow (Sankey)",
 493 |                         "Database Overview", "Tax Efficiency"),
 494 |         specs=[[{"type": "pie"}, {"type": "sankey"}],
 495 |                [{"type": "indicator"}, {"type": "indicator"}]]
 496 |     )
 497 |
 498 |     # Pie chart - Financial breakdown
 499 |     fig.add_trace(
 500 |         go.Pie(labels=["Net Income", "Tax Paid", "Costs"],
```

File 15: Menus/tax_menu.py

```
  1 | from DB import setup
  2 | from Logic import validators
  3 |
  4 | def manage_brackets_menu():
  5 |     while True:
  6 |         print("\n=== Manage Tax Brackets ■ ===")
  7 |         print("1. Upload tax brackets")
  8 |         print("2. Update tax bracket")
  9 |         print("3. Delete tax bracket")
 10 |         print("4. View tax brackets")
 11 |         print("5. Back")
 12 |
 13 |         choice = input("Choose an option (1-5): ").strip()
 14 |
 15 |         if choice == "1":  # Upload
 16 |             print("\n=== Upload Options ===")
 17 |             print("1. Enter manually")
 18 |             print("2. Upload from CSV")
 19 |             print("3. Back")
 20 |             sub = input("Choose an option (1-3): ").strip()
 21 |
 22 |             if sub == "1":  # Manual entry
 23 |                 try:
 24 |                     country = validators.validate_country(input("Enter country (US/Spain): ").strip())
 25 |                     tax_type = validators.validate_tax_type(input("Enter type (Individual/Business): ").strip())
 26 |                     n = validators.safe_int_input("How many brackets to add? ", "Number of brackets",
min_value=1)
 27 |                     for i in range(n):
 28 |                         limit = input(f"Bracket {i+1} income limit (number or 'inf'): ").strip()
 29 |                         income_limit = float("inf") if limit.lower() == "inf" else validators.safe_float_input(
f"Re-ente...
 30 |                         rate = validators.safe_float_input(f"Bracket {i+1} rate (0.0-1.0, e.g. 0.21): ", "Tax
rate")
 31 |                         rate = validators.validate_tax_rate(rate)
 32 |                         setup.add_tax_bracket(country, tax_type, income_limit, rate)
 33 |                     print(f"■ Added {n} brackets for {country} {tax_type}")
 34 |                 except validators.ValidationError as e:
 35 |                     print(f"■ {e}")
 36 |
 37 |             elif sub == "2":  # CSV
 38 |                 try:
 39 |                     country = validators.validate_country(input("Enter country (US/Spain): ").strip())
 40 |                     tax_type = validators.validate_tax_type(input("Enter type (Individual/Business): ").strip())
 41 |                     filepath = validators.safe_string_input("Enter path to CSV file: ", "File path")
 42 |                     setup.add_tax_brackets_from_csv(country, tax_type, filepath)
 43 |                 except validators.ValidationError as e:
 44 |                     print(f"■ {e}")
 45 |
 46 |             elif sub == "3":
 47 |                 continue
 48 |
 49 |         elif choice == "2":  # Update
 50 |             bracket_id = int(input("Enter bracket ID to update: "))
 51 |             field = input("Which field (country, tax_type, income_limit, rate)? ").strip()
 52 |             new_value = input("Enter new value: ").strip()
 53 |             if field in ("income_limit", "rate") and new_value.lower() != "inf":
 54 |                 new_value = float(new_value)
 55 |             elif new_value.lower() == "inf":
 56 |                 new_value = float("inf")
 57 |             setup.update_tax_bracket(bracket_id, field, new_value)
 58 |
 59 |         elif choice == "3":  # Delete
 60 |             bracket_id = int(input("Enter bracket ID to delete: "))
 61 |             setup.delete_tax_bracket(bracket_id)
 62 |
 63 |         elif choice == "4":  # View
 64 |             country = input("Enter country (e.g. US, Spain): ").strip()
 65 |             tax_type = input("Enter type (Individual/Business): ").strip().title()
 66 |             rows = setup.get_tax_brackets(country, tax_type, include_id=True)
 67 |             if not rows:
 68 |                 print("■ No brackets found.")
 69 |             else:
 70 |                 print(f"\nBrackets for {country} {tax_type}:")
 71 |                 print(f"{'ID':&lt;4} | {'Income Limit':&gt;15} | {'Rate':&gt;8}")
```

```
                print("-" * 35)
                for bid, limit, rate in rows:
                    limit_txt = "∞" if limit == float("inf") else f"{limit:,.0f}"
                    print(f"{bid:&lt;4} | {limit_txt:&gt;15} | {rate*100:&gt;7.2f}%")

        elif choice == "5":  # Back
            break
        else:
            print("■ Invalid choice. Please enter 1-5.")


def maintenance_menu():
    while True:
        print("\n=== Tax Maintenance ■■ ===")
        print("1. Reset tax brackets ■■")
        print("2. Export CSV template")
        print("3. Back")

        choice = input("Choose an option (1-3): ").strip()
        if choice == "1":
            setup.reset_tax_brackets()
        elif choice == "2":
            setup.export_tax_template()
        elif choice == "3":
            break
        else:
            print("■ Invalid choice. Please enter 1-3.")


def show_tax_menu():
    while True:
        print("\n=== Tax Menu ===")
        print("1. Manage Brackets ■")
        print("2. Maintenance ■■")
        print("3. Back to main menu")

        choice = input("Choose an option (1-3): ").strip()
        if choice == "1":
            manage_brackets_menu()
        elif choice == "2":
            maintenance_menu()
        elif choice == "3":
            break
        else:
            print("■ Invalid choice. Please enter 1-3.")

```

File 16: __init__.py

```
1 |
```

File 17: __main__.py

```
from Menus import project_menu, db_menu, tax_menu, report_menu
from DB import reset as db_reset
from DB import setup

# Init DB + defaults
setup.init_db()
setup.seed_default_brackets()

def main():
    while True:
        print("\n=== MoneySplit Main Menu ===")
        print("1. New Project ■")
        print("2. Play with DB ■")
        print("3. Tax ■")
        print("4. Reports ■")
        print("5. DB Maintenance ■■")
        print("6. Exit ■")

        choice = input("Choose an option (1-6): ").strip()

        if choice == "1":
            project_menu.run_new_project()
        elif choice == "2":
            db_menu.show_db_menu()
        elif choice == "3":
            tax_menu.show_tax_menu()
        elif choice == "4":
            report_menu.show_report_menu()
        elif choice == "5":
            db_reset.main()  # run the maintenance tool
        elif choice == "6":
            print("■ Exiting MoneySplit. Goodbye!")
            break
        else:
            print("■ Invalid choice. Please enter 1-6.")

if __name__ == "__main__":
    main()

```

File 18: api/__init__.py

```
1 | # API package
2 |
```

File 19: api/main.py

Note: This file has 1605 lines. First 500 lines shown.

```
"""
MoneySplit FastAPI Application
RESTful API for commission-based income splitting with tax calculations.

AI ASSISTANCE DISCLOSURE:
This API was developed with AI assistance (ChatGPT/Claude).
- Prompts used: "Create FastAPI endpoints for CRUD operations on tax records"
- Prompts used: "Add forecasting endpoints with scikit-learn integration"
- Prompts used: "Generate PDF export endpoints using ReportLab"
- AI helped structure the API following REST principles and FastAPI best practices
"""
from fastapi import FastAPI, HTTPException, Query
from fastapi.responses import HTMLResponse, FileResponse
from fastapi.middleware.cors import CORSMiddleware
from typing import List, Optional
import sys
import os

# Add parent directory to path for imports
sys.path.insert(0, os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from DB import setup
from Logic import pdf_generator, forecasting
from api.models import (
    ProjectCreate, ProjectCreateResponse, RecordResponse,
    RecordWithPeople, PersonResponse, TaxBracketCreate,
    TaxBracketResponse, RecordUpdate, MessageResponse
)

app = FastAPI(
    title="MoneySplit API",
    description="RESTful API for commission-based income splitting with tax calculations",
    version="1.0.0"
)

# Enable CORS for frontend integration
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)


# ===== Project/Record Endpoints =====

@app.post("/api/projects", response_model=ProjectCreateResponse, status_code=201)
async def create_project(project: ProjectCreate):
    """Create a new project with people and calculate taxes."""
    try:
        # Calculate totals
        total_costs = sum(project.costs)
        income = project.revenue - total_costs
        group_income = income
        individual_income = income / project.num_people if project.num_people &gt; 0 else 0

        # Calculate tax
        if project.tax_type == "Individual":
            tax = setup.calculate_tax_from_db(individual_income, project.country, project.tax_type)
        else:
            tax = setup.calculate_tax_from_db(group_income, project.country, project.tax_type)

        # Calculate net incomes
        if project.tax_type == "Individual":
            net_income_per_person = individual_income - tax
            net_income_group = net_income_per_person * project.num_people
        else:
            net_income_group = group_income - tax
            net_income_per_person = net_income_group / project.num_people if project.num_people &gt; 0 else 0

        # Save to database
```

```
        conn = setup.get_conn()
        cursor = conn.cursor()
        cursor.execute("""
            INSERT INTO tax_records (
                num_people, revenue, total_costs, group_income, individual_income,
                tax_origin, tax_option, tax_amount,
                net_income_per_person, net_income_group
            ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        """, (
            project.num_people,
            project.revenue,
            total_costs,
            group_income,
            individual_income,
            project.country,
            project.tax_type,
            tax,
            net_income_per_person,
            net_income_group
        ))
        record_id = cursor.lastrowid

        # Save people
        for person in project.people:
            if project.tax_type == "Individual":
                gross_income = individual_income * person.work_share * project.num_people
                tax_paid = tax * person.work_share
                net_income = gross_income - tax_paid
            else:
                gross_income = group_income * person.work_share
                tax_paid = tax * person.work_share
                net_income = gross_income - tax_paid

            cursor.execute("""
                INSERT INTO people (record_id, name, work_share, gross_income, tax_paid, net_income)
                VALUES (?, ?, ?, ?, ?, ?)
            """, (record_id, person.name, person.work_share, gross_income, tax_paid, net_income))

        conn.commit()
        conn.close()

        return ProjectCreateResponse(
            record_id=record_id,
            message="Project created successfully",
            summary={
                "revenue": project.revenue,
                "total_costs": total_costs,
                "tax_amount": tax,
                "net_income_group": net_income_group,
                "net_income_per_person": net_income_per_person
            }
        )
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))


@app.get("/api/records", response_model=List[RecordResponse])
async def get_records(limit: int = Query(10, ge=1, le=100)):
    """Get recent records (default: last 10)."""
    records = setup.fetch_last_records(limit)
    return [
        RecordResponse(
            id=r[0], tax_origin=r[1], tax_option=r[2],
            revenue=r[3], total_costs=r[4], tax_amount=r[5],
            net_income_group=r[6], net_income_per_person=r[7],
            created_at=r[8],
            num_people=r[9], group_income=r[10], individual_income=r[11]
        ) for r in records
    ]


@app.get("/api/records/{record_id}", response_model=RecordWithPeople)
async def get_record(record_id: int):
    """Get a specific record with its people."""
    record = setup.get_record_by_id(record_id)
    if not record:
        raise HTTPException(status_code=404, detail=f"Record {record_id} not found")

```

```
    people = setup.fetch_people_by_record(record_id)

    return RecordWithPeople(
        id=record[0], tax_origin=record[1], tax_option=record[2],
        revenue=record[3], total_costs=record[4], tax_amount=record[5],
        net_income_group=record[6], net_income_per_person=record[7],
        created_at=record[8],
        num_people=record[9], group_income=record[10], individual_income=record[11],
        people=[
            PersonResponse(
                id=p[0], name=p[1], work_share=p[2],
                gross_income=p[3], tax_paid=p[4], net_income=p[5]
            ) for p in people
        ]
    )


@app.put("/api/records/{record_id}", response_model=MessageResponse)
async def update_record(record_id: int, update: RecordUpdate):
    """Update a record field."""
    try:
        setup.update_record(record_id, update.field, update.value)
        return MessageResponse(message=f"Record {record_id} updated successfully")
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))


@app.delete("/api/records/{record_id}", response_model=MessageResponse)
async def delete_record(record_id: int):
    """Delete a record and its people."""
    record = setup.get_record_by_id(record_id)
    if not record:
        raise HTTPException(status_code=404, detail=f"Record {record_id} not found")

    setup.delete_record(record_id)
    return MessageResponse(message=f"Record {record_id} deleted successfully")


# ===== Tax Bracket Endpoints =====

@app.get("/api/tax-brackets", response_model=List[TaxBracketResponse])
async def get_tax_brackets(country: str, tax_type: str):
    """Get tax brackets for a country and type."""
    import math
    brackets = setup.get_tax_brackets(country, tax_type, include_id=True)
    return [
        TaxBracketResponse(
            id=b[0],
            income_limit=999999999 if math.isinf(b[1]) else b[1],  # Convert inf to large number for JSON
            rate=b[2],
            country=country,
            tax_type=tax_type
        ) for b in brackets
    ]


@app.post("/api/tax-brackets", response_model=MessageResponse, status_code=201)
async def create_tax_bracket(bracket: TaxBracketCreate):
    """Add a new tax bracket."""
    try:
        setup.add_tax_bracket(bracket.country, bracket.tax_type, bracket.income_limit, bracket.rate)
        return MessageResponse(message="Tax bracket created successfully")
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))


@app.delete("/api/tax-brackets/{bracket_id}", response_model=MessageResponse)
async def delete_tax_bracket(bracket_id: int):
    """Delete a tax bracket."""
    try:
        setup.delete_tax_bracket(bracket_id)
        return MessageResponse(message=f"Tax bracket {bracket_id} deleted successfully")
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))


# ===== People Endpoints =====

```

```
@app.get("/api/people/{person_id}", response_model=PersonResponse)
async def get_person(person_id: int):
    """Get a specific person by ID."""
    conn = setup.get_conn()
    cursor = conn.cursor()
    cursor.execute("SELECT id, name, work_share, gross_income, tax_paid, net_income FROM people WHERE id=?", (
person_id,...
    person = cursor.fetchone()
    conn.close()

    if not person:
        raise HTTPException(status_code=404, detail=f"Person {person_id} not found")

    return PersonResponse(
        id=person[0], name=person[1], work_share=person[2],
        gross_income=person[3], tax_paid=person[4], net_income=person[5]
    )


@app.get("/api/people/history/{name}", response_model=List[dict])
async def get_person_history(name: str):
    """Get all records for a person by name."""
    records = setup.fetch_records_by_person(name)
    return [
        {
            "person_id": r[0],
            "record_id": r[1],
            "name": r[2],
            "work_share": r[3],
            "gross_income": r[4],
            "tax_paid": r[5],
            "net_income": r[6],
            "created_at": r[7]
        } for r in records
    ]


# ===== Report Endpoints =====

@app.get("/api/reports/revenue-summary")
async def revenue_summary():
    """Get revenue summary by year."""
    rows = setup.get_revenue_summary()
    return [
        {
            "year": r[0],
            "total_revenue": r[1],
            "total_costs": r[2],
            "net_income": r[3]
        } for r in rows
    ]


@app.get("/api/reports/top-people")
async def top_people(limit: int = Query(10, ge=1, le=50)):
    """Get top people by net income."""
    rows = setup.get_top_people(limit)
    return [
        {
            "name": r[0],
            "total_gross": r[1],
            "total_tax_paid": r[2],
            "total_net": r[3]
        } for r in rows
    ]


@app.get("/api/reports/statistics")
async def overall_statistics():
    """Get overall database statistics."""
    conn = setup.get_conn()
    cursor = conn.cursor()

    cursor.execute("""
        SELECT COUNT(*),
               COALESCE(SUM(revenue), 0),
               COALESCE(SUM(total_costs), 0),
               COALESCE(SUM(tax_amount), 0),
```

```
                COALESCE(SUM(net_income_group), 0),
                COALESCE(AVG(CASE
                    WHEN group_income &gt; 0 THEN tax_amount * 100.0 / group_income
                    ELSE 0
                END), 0)
            FROM tax_records
        """)
        result = cursor.fetchone()
        total_records = result[0] or 0
        total_rev = result[1] or 0
        total_costs = result[2] or 0
        total_tax = result[3] or 0
        total_net = result[4] or 0
        avg_rate = result[5] or 0

        cursor.execute("SELECT COUNT(*), COUNT(DISTINCT name) FROM people")
        people_result = cursor.fetchone()
        total_people_entries = people_result[0] or 0
        unique_people = people_result[1] or 0

        conn.close()

        return {
            "total_records": total_records,
            "total_revenue": float(total_rev),
            "total_costs": float(total_costs),
            "total_tax": float(total_tax),
            "total_net_income": float(total_net),
            "average_tax_rate": float(avg_rate),
            "total_people_entries": total_people_entries,
            "unique_people": unique_people
        }


# ===== Visualization Endpoints =====

def create_stunning_html(plotly_fig, title, emoji, description):
    """Wrap Plotly figure in world-class premium HTML template."""
    plot_config = {'displayModeBar': True, 'displaylogo': False}
    plotly_html = plotly_fig.to_html(include_plotlyjs='cdn', full_html=False, config=plot_config)

    return f"""
&lt;!DOCTYPE html&gt;
&lt;html lang="en"&gt;
&lt;head&gt;
    &lt;meta charset="UTF-8"&gt;
    &lt;meta name="viewport" content="width=device-width, initial-scale=1.0"&gt;
    &lt;title&gt;{title} - MoneySplit Analytics&lt;/title&gt;
    &lt;link rel="preconnect" href="https://fonts.googleapis.com"&gt;
    &lt;link rel="preconnect" href="https://fonts.gstatic.com" crossorigin&gt;
    &lt;link href="https://fonts.googleapis.com/css2?family=Inter:wght@300;400;500;600;700;800;900&amp;
family=Poppins:wg...
    &lt;style&gt;
        :root {{
            --bg-primary: #0a0a0f;
            --bg-secondary: #13131a;
            --bg-tertiary: #1a1a24;
            --accent-primary: #6366f1;
            --accent-secondary: #8b5cf6;
            --accent-tertiary: #ec4899;
            --text-primary: #ffffff;
            --text-secondary: #a1a1aa;
            --border-subtle: rgba(255, 255, 255, 0.08);
            --glow-purple: rgba(139, 92, 246, 0.5);
            --glow-pink: rgba(236, 72, 153, 0.5);
        }}

        * {{
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }}

        html {{
            scroll-behavior: smooth;
        }}

        body {{
```

```
            font-family: 'Inter', -apple-system, BlinkMacSystemFont, sans-serif;
            background: var(--bg-primary);
            color: var(--text-primary);
            min-height: 100vh;
            padding: 0;
            position: relative;
            overflow-x: hidden;
            line-height: 1.6;
            -webkit-font-smoothing: antialiased;
            -moz-osx-font-smoothing: grayscale;
        }}

        /* ULTRA-PREMIUM MESH GRADIENT BACKGROUND */
        body::before {{
            content: '';
            position: fixed;
            top: 0;
            left: 0;
            width: 100%;
            height: 100%;
            z-index: 0;
            background:
                radial-gradient(ellipse 80% 50% at 50% -20%, rgba(139, 92, 246, 0.15), transparent),
                radial-gradient(ellipse 60% 40% at 10% 40%, rgba(99, 102, 241, 0.1), transparent),
                radial-gradient(ellipse 60% 40% at 90% 60%, rgba(236, 72, 153, 0.1), transparent);
            pointer-events: none;
        }}

        /* ANIMATED GRID PATTERN */
        body::after {{
            content: '';
            position: fixed;
            top: 0;
            left: 0;
            width: 100%;
            height: 100%;
            z-index: 0;
            background-image:
                linear-gradient(rgba(255, 255, 255, 0.02) 1px, transparent 1px),
                linear-gradient(90deg, rgba(255, 255, 255, 0.02) 1px, transparent 1px);
            background-size: 50px 50px;
            mask-image: radial-gradient(ellipse 100% 100% at 50% 50%, black 40%, transparent 100%);
            pointer-events: none;
        }}

        .container {{
            max-width: 1600px;
            margin: 0 auto;
            padding: 60px 40px;
            position: relative;
            z-index: 1;
        }}

        @media (max-width: 768px) {{
            .container {{
                padding: 40px 20px;
            }}
        }}

        /* NAVIGATION BAR */
        .nav {{
            position: fixed;
            top: 0;
            left: 0;
            right: 0;
            z-index: 1000;
            backdrop-filter: blur(20px) saturate(180%);
            background: rgba(10, 10, 15, 0.7);
            border-bottom: 1px solid var(--border-subtle);
        }}

        .nav-content {{
            max-width: 1600px;
            margin: 0 auto;
            padding: 16px 40px;
            display: flex;
            align-items: center;
            justify-content: space-between;
```

```
        }}

        .logo {{
            display: flex;
            align-items: center;
            gap: 12px;
            font-family: 'Poppins', sans-serif;
            font-size: 20px;
            font-weight: 700;
            background: linear-gradient(135deg, var(--accent-primary), var(--accent-secondary));
            -webkit-background-clip: text;
            -webkit-text-fill-color: transparent;
        }}

        .badge {{
            padding: 4px 12px;
            border-radius: 12px;
            font-size: 11px;
            font-weight: 600;
            text-transform: uppercase;
            letter-spacing: 0.5px;
            background: linear-gradient(135deg, var(--accent-primary), var(--accent-secondary));
            color: white;
        }}

        /* HERO SECTION */
        .header {{
            margin-top: 80px;
            margin-bottom: 48px;
            text-align: center;
            animation: fadeInDown 0.6s cubic-bezier(0.16, 1, 0.3, 1);
        }}

        @keyframes fadeInDown {{
            from {{
                opacity: 0;
                transform: translateY(-20px);
            }}
            to {{
                opacity: 1;
```

File 20: api/models.py

```
"""
Pydantic models for API request/response validation.
"""
from pydantic import BaseModel, Field, field_validator
from typing import List, Optional
from datetime import datetime


# ===== Request Models =====

class PersonInput(BaseModel):
    name: str = Field(..., min_length=1, description="Person's name")
    work_share: float = Field(..., ge=0, le=1, description="Work share (0.0 to 1.0)")


class ProjectCreate(BaseModel):
    num_people: int = Field(..., gt=0, description="Number of people")
    revenue: float = Field(..., ge=0, description="Total revenue")
    costs: List[float] = Field(..., description="List of costs")
    country: str = Field(..., min_length=1, description="Country (e.g., US, Spain, UK, etc.)")
    tax_type: str = Field(..., pattern="^(Individual|Business)$", description="Tax type: Individual or Business")
    people: List[PersonInput] = Field(..., description="List of people with work shares")

    @field_validator('people')
    @classmethod
    def validate_people_count(cls, v, info):
        if 'num_people' in info.data and len(v) != info.data['num_people']:
            raise ValueError(f"Expected {info.data['num_people']} people, got {len(v)}")
        return v

    @field_validator('people')
    @classmethod
    def validate_work_shares(cls, v):
        total_share = sum(person.work_share for person in v)
        if abs(total_share - 1.0) &gt; 0.01:
            raise ValueError(f"Work shares must sum to 1.0, got {total_share:.2f}")
        return v


class TaxBracketCreate(BaseModel):
    country: str = Field(..., min_length=1, description="Country name")
    tax_type: str = Field(..., pattern="^(Individual|Business)$")
    income_limit: float = Field(..., ge=0)
    rate: float = Field(..., ge=0, le=1)


class RecordUpdate(BaseModel):
    field: str = Field(..., pattern="^(num_people|revenue|total_costs|tax_origin|tax_option)$")
    value: str | int | float


# ===== Response Models =====

class PersonResponse(BaseModel):
    id: int
    name: str
    work_share: float
    gross_income: float
    tax_paid: float
    net_income: float

    class Config:
        from_attributes = True


class RecordResponse(BaseModel):
    id: int
    num_people: int
    revenue: float
    total_costs: float
    group_income: float
    individual_income: float
    tax_origin: str
    tax_option: str
```

```
    tax_amount: float
    net_income_per_person: float
    net_income_group: float
    created_at: str

    class Config:
        from_attributes = True


class RecordWithPeople(RecordResponse):
    people: List[PersonResponse] = []


class TaxBracketResponse(BaseModel):
    id: int
    country: str
    tax_type: str
    income_limit: float
    rate: float

    class Config:
        from_attributes = True


class ProjectCreateResponse(BaseModel):
    record_id: int
    message: str
    summary: dict


class MessageResponse(BaseModel):
    message: str
    details: Optional[dict] = None

```

File 21: frontend/package.json

```
{
  "name": "frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@tanstack/react-query": "^5.90.2",
    "@testing-library/dom": "^10.4.1",
    "@testing-library/jest-dom": "^6.9.1",
    "@testing-library/react": "^16.3.0",
    "@testing-library/user-event": "^13.5.0",
    "@types/jest": "^27.5.2",
    "@types/node": "^16.18.126",
    "@types/react": "^19.2.0",
    "@types/react-dom": "^19.2.0",
    "axios": "^1.12.2",
    "lucide-react": "^0.544.0",
    "react": "^19.2.0",
    "react-dom": "^19.2.0",
    "react-router-dom": "^7.9.3",
    "react-scripts": "5.0.1",
    "recharts": "^3.2.1",
    "typescript": "^4.9.5",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      "&gt;0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}

```

File 22: frontend/src/App.css

Note: This file has 1078 lines. First 500 lines shown.

```
@import url('https://fonts.googleapis.com/css2?family=Inter:wght@300;400;500;600;700;800;900&amp;display=swap');
@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@700;800;900&amp;display=swap');

* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: 'Inter', -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', sans-serif;
  -webkit-font-smoothing: subpixel-antialiased;
  -moz-osx-font-smoothing: auto;
  background: #1a1625;
  background: linear-gradient(135deg, #1a1625 0%, #2d2640 50%, #1f1b2e 100%);
  min-height: 100vh;
  overflow-x: hidden;
  position: relative;
  transform: translateZ(0);
  backface-visibility: hidden;
  perspective: 1000px;
}

/* STATIC BACKGROUND - No Animation */
body::before {
  content: '';
  position: fixed;
  width: 100%;
  height: 100%;
  top: 0;
  left: 0;
  z-index: 0;
  background:
    radial-gradient(circle at 20% 80%, rgba(102, 126, 234, 0.15) 0%, transparent 50%),
    radial-gradient(circle at 80% 20%, rgba(118, 75, 162, 0.15) 0%, transparent 50%),
    radial-gradient(circle at 40% 40%, rgba(99, 102, 241, 0.15) 0%, transparent 50%),
    radial-gradient(circle at 60% 80%, rgba(138, 116, 249, 0.15) 0%, transparent 50%);
  pointer-events: none;
}

@keyframes rotate {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}

/* FLOATING GRADIENT ORBS */
body::after {
  content: '';
  position: fixed;
  width: 100%;
  height: 100%;
  top: 0;
  left: 0;
  z-index: 0;
  background:
    radial-gradient(circle at 15% 90%, rgba(102, 126, 234, 0.35) 0%, transparent 25%),
    radial-gradient(circle at 85% 10%, rgba(118, 75, 162, 0.35) 0%, transparent 25%),
    radial-gradient(circle at 50% 50%, rgba(138, 116, 249, 0.25) 0%, transparent 30%);
  animation: float 20s ease-in-out infinite;
  pointer-events: none;
}

@keyframes float {
  0%, 100% {
    transform: translateY(0) ;
    opacity: 0.8;
  }
  33% {
    transform: translateY(-30px) ;
    opacity: 1;
  }
  66% {
```

```
    transform: translateY(30px) ;
    opacity: 0.9;
  }
}

.app {
  display: flex;
  min-height: 100vh;
  position: relative;
  z-index: 1;
}

/* STARS/PARTICLES EFFECT */
.app::before {
  content: '';
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-image:
    radial-gradient(2px 2px at 20px 30px, rgba(255, 255, 255, 0.3), transparent),
    radial-gradient(2px 2px at 60px 70px, rgba(255, 255, 255, 0.2), transparent),
    radial-gradient(1px 1px at 50px 50px, rgba(255, 255, 255, 0.4), transparent),
    radial-gradient(1px 1px at 130px 80px, rgba(255, 255, 255, 0.3), transparent),
    radial-gradient(2px 2px at 90px 10px, rgba(255, 255, 255, 0.5), transparent);
  background-size: 200px 200px;
  animation: sparkle 4s ease-in-out infinite;
  pointer-events: none;
  z-index: 0;
}

@keyframes sparkle {
  0%, 100% { opacity: 1; }
  50% { opacity: 0.5; }
}

/* PREMIUM 3D SIDEBAR */
.sidebar {
  width: 280px;
  background: rgba(20, 18, 35, 0.98);
  color: #ffffff;
  padding: 32px 24px;
  display: flex;
  flex-direction: column;
  box-shadow:
    4px 0 30px rgba(0, 0, 0, 0.4),
    inset -1px 0 0 rgba(255, 255, 255, 0.08);
  position: relative;
  z-index: 10;
  border-right: 1px solid rgba(138, 116, 249, 0.2);
  animation: slideInLeft 0.8s cubic-bezier(0.16, 1, 0.3, 1);
  transition: transform 0.4s cubic-bezier(0.4, 0, 0.2, 1);
}

.sidebar.collapsed {
  transform: translateX(-100%);
}

@keyframes slideInLeft {
  from {
    transform: translateX(-100%);
    opacity: 0;
  }
  to {
    transform: translateX(0);
    opacity: 1;
  }
}

.sidebar::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  height: 6px;
  background: linear-gradient(90deg,
```

```
    #667eea 0%,
    #764ba2 25%,
    #f093fb 50%,
    #764ba2 75%,
    #667eea 100%);
  background-size: 200% 100%;
  animation: shimmer 3s linear infinite;
}

@keyframes shimmer {
  0% { background-position: 0% 0%; }
  100% { background-position: 200% 0%; }
}

.sidebar::after {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background: linear-gradient(180deg, rgba(102, 126, 234, 0.05) 0%, transparent 100%);
  pointer-events: none;
}

.sidebar h1 {
  font-family: 'Poppins', sans-serif;
  font-size: 28px;
  font-weight: 900;
  margin-bottom: 56px;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 50%, #f093fb 100%);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
  background-clip: text;
  display: flex;
  align-items: center;
  gap: 8px;
  letter-spacing: -0.5px;
  position: relative;
  z-index: 1;
  animation: titleGlow 3s ease-in-out infinite;
}

@keyframes titleGlow {
  0%, 100% {
    filter: drop-shadow(0 0 20px rgba(102, 126, 234, 0.6));
  }
  50% {
    filter: drop-shadow(0 0 40px rgba(118, 75, 162, 0.8));
  }
}

.logo-emoji {
  font-size: 32px;
  filter: drop-shadow(0 4px 16px rgba(102, 126, 234, 0.8));
  animation: bounce 2s ease-in-out infinite;
  display: inline-block;
  line-height: 1;
}

@keyframes bounce {
  0%, 100% {
    transform: translateY(0) ;
  }
  50% {
    transform: translateY(-8px) ;
  }
}

.nav-menu {
  display: flex;
  flex-direction: column;
  gap: 8px;
}

.nav-item {
  padding: 16px 20px;
  border-radius: 12px;
```

```
  cursor: pointer;
  transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
  display: flex;
  align-items: center;
  gap: 14px;
  text-decoration: none;
  color: rgba(255, 255, 255, 0.65);
  font-weight: 500;
  font-size: 15px;
  position: relative;
  overflow: hidden;
  background: rgba(255, 255, 255, 0.03);
  border: 1px solid rgba(255, 255, 255, 0.05);
}

.nav-item::before {
  content: '';
  position: absolute;
  left: 0;
  top: 0;
  bottom: 0;
  width: 5px;
  background: linear-gradient(180deg, #667eea 0%, #764ba2 100%);
  transform: scaleY(0);
  transition: transform 0.4s cubic-bezier(0.175, 0.885, 0.32, 1.275);
  box-shadow: 0 0 20px rgba(102, 126, 234, 0.8);
}

.nav-item::after {
  content: '';
  position: absolute;
  inset: 0;
  background: linear-gradient(135deg, rgba(102, 126, 234, 0.2), rgba(118, 75, 162, 0.2));
  opacity: 0;
  transition: opacity 0.4s ease;
  border-radius: 18px;
}

.nav-item:hover {
  background: linear-gradient(135deg, rgba(102, 126, 234, 0.18) 0%, rgba(118, 75, 162, 0.18) 100%);
  color: rgba(255, 255, 255, 0.95);
  transform: translateX(8px) ;
  box-shadow:
    0 8px 24px rgba(102, 126, 234, 0.25),
    inset 0 1px 0 rgba(255, 255, 255, 0.08);
}

.nav-item:hover::before {
  transform: scaleY(1);
}

.nav-item:hover::after {
  opacity: 1;
}

.nav-item.active {
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  color: white;
  font-weight: 700;
  box-shadow:
    0 12px 40px rgba(102, 126, 234, 0.6),
    0 0 60px rgba(102, 126, 234, 0.4),
    inset 0 1px 0 rgba(255, 255, 255, 0.2);
  transform: translateX(8px) ;
  animation: activeGlow 2s ease-in-out infinite;
}

@keyframes activeGlow {
  0%, 100% {
    box-shadow:
      0 12px 40px rgba(102, 126, 234, 0.6),
      0 0 60px rgba(102, 126, 234, 0.4),
      inset 0 1px 0 rgba(255, 255, 255, 0.2);
  }
  50% {
    box-shadow:
      0 12px 40px rgba(118, 75, 162, 0.8),
      0 0 80px rgba(118, 75, 162, 0.6),
```

```
      inset 0 1px 0 rgba(255, 255, 255, 0.3);
  }
}

.nav-item.active::before {
  transform: scaleY(1);
  background: white;
}

.nav-item span:first-child {
  font-size: 28px;
  filter: drop-shadow(0 4px 8px rgba(0, 0, 0, 0.3));
  transition: transform 0.3s ease;
}

.nav-item:hover span:first-child,
.nav-item.active span:first-child {
  transform:  rotate(5deg);
}

/* Menu Toggle Button */
.menu-toggle {
  position: fixed;
  top: 20px;
  left: 20px;
  z-index: 100;
  width: 50px;
  height: 50px;
  border-radius: 16px;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  border: none;
  cursor: pointer;
  display: flex;
  align-items: center;
  justify-content: center;
  box-shadow:
    0 10px 30px rgba(102, 126, 234, 0.5),
    0 0 40px rgba(102, 126, 234, 0.3);
  transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
}

.menu-toggle:hover {
  transform:  translateY(-2px);
  box-shadow:
    0 15px 40px rgba(102, 126, 234, 0.7),
    0 0 60px rgba(102, 126, 234, 0.5);
}

.menu-toggle svg {
  width: 24px;
  height: 24px;
  stroke: white;
  stroke-width: 2.5;
  fill: none;
}

/* Top Right Logo */
.top-logo {
  position: fixed;
  top: 24px;
  right: 32px;
  z-index: 100;
  display: flex;
  align-items: center;
  gap: 12px;
  padding: 14px 24px;
  background: rgba(20, 18, 35, 0.95);
  border-radius: 20px;
  border: 2px solid rgba(138, 116, 249, 0.3);
  box-shadow:
    0 10px 30px rgba(0, 0, 0, 0.3),
    0 0 40px rgba(102, 126, 234, 0.2);
}

.top-logo-emoji {
  font-size: 28px;
  filter: drop-shadow(0 4px 12px rgba(102, 126, 234, 0.6));
}
```

```
|
| .top-logo-text {
|   font-family: 'Poppins', sans-serif;
|   font-size: 20px;
|   font-weight: 800;
|   background: linear-gradient(135deg, #667eea 0%, #764ba2 50%, #f093fb 100%);
|   -webkit-background-clip: text;
|   -webkit-text-fill-color: transparent;
|   background-clip: text;
|   letter-spacing: -0.5px;
| }
|
| /* Main Content */
| .main-content {
|   flex: 1;
|   padding: 40px;
|   overflow-y: auto;
|   position: relative;
|   z-index: 1;
| }
|
| .page-header {
|   margin-bottom: 56px;
|   animation: fadeInDown 0.8s cubic-bezier(0.16, 1, 0.3, 1);
|   position: relative;
| }
|
| .page-header::after {
|   content: '';
|   position: absolute;
|   bottom: -20px;
|   left: 0;
|   width: 120px;
|   height: 6px;
|   background: linear-gradient(90deg, #667eea 0%, #764ba2 50%, #f093fb 100%);
|   border-radius: 3px;
|   box-shadow: 0 0 30px rgba(102, 126, 234, 0.8);
|   animation: widthGrow 0.8s ease 0.4s both;
| }
|
| @keyframes widthGrow {
|   from { width: 0; }
|   to { width: 120px; }
| }
|
| .page-header h2 {
|   font-family: 'Poppins', sans-serif;
|   font-size: 56px;
|   font-weight: 900;
|   color: white;
|   margin-bottom: 16px;
|   letter-spacing: -2px;
|   text-shadow:
|     0 4px 20px rgba(0, 0, 0, 0.4),
|     0 0 60px rgba(102, 126, 234, 0.5);
|   animation: titleBounce 1s ease;
| }
|
| @keyframes titleBounce {
|   0%, 100% { transform: translateY(0); }
|   50% { transform: translateY(-10px); }
| }
|
| .page-header p {
|   color: rgba(255, 255, 255, 0.9);
|   font-size: 20px;
|   font-weight: 600;
|   text-shadow: 0 2px 8px rgba(0, 0, 0, 0.3);
|   letter-spacing: 0.3px;
| }
|
| /* PREMIUM 3D CARDS */
| .card {
|   background: rgba(15, 12, 41, 0.85);
|   border-radius: 32px;
|   padding: 40px;
|   box-shadow:
|     0 30px 80px rgba(0, 0, 0, 0.5),
```

```
    inset 0 1px 0 rgba(255, 255, 255, 0.1),
    0 0 60px rgba(102, 126, 234, 0.2);
  margin-bottom: 32px;
  border: 2px solid rgba(102, 126, 234, 0.3);
  animation: fadeInUp 0.8s cubic-bezier(0.16, 1, 0.3, 1);
  transition: all 0.5s cubic-bezier(0.175, 0.885, 0.32, 1.275);
  position: relative;
  overflow: visible;
  transform-style: preserve-3d;
  word-wrap: break-word;
  overflow-wrap: break-word;
}

.card::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background: radial-gradient(circle at 20% 50%, rgba(102, 126, 234, 0.15), transparent 60%);
  opacity: 0;
  transition: opacity 0.5s ease;
  pointer-events: none;
}

.card:hover {
  transform: translateY(-8px)  ;
  box-shadow:
    0 40px 100px rgba(0, 0, 0, 0.6),
    0 0 100px rgba(102, 126, 234, 0.4),
    inset 0 1px 0 rgba(255, 255, 255, 0.15);
  border-color: rgba(102, 126, 234, 0.5);
}

.card:hover::before {
  opacity: 1;
}
```

File 23: frontend/src/App.tsx

```
/**
 * MoneySplit React Frontend Application
 *
 * AI ASSISTANCE DISCLOSURE:
 * This entire frontend was developed with AI assistance (ChatGPT/Claude).
 * - Prompts used: "Create a React TypeScript app with sidebar navigation for MoneySplit"
 * - Prompts used: "Build Dashboard page with statistics cards and recent projects table"
 * - Prompts used: "Create Projects page with form for creating tax calculation projects"
 * - Prompts used: "Add responsive sidebar with gradient purple theme"
 * - AI helped with React hooks, state management, TypeScript types, and CSS styling
 */
import React, { useState } from 'react';
import './App.css';
import Dashboard from './pages/Dashboard';
import Projects from './pages/Projects';
import Reports from './pages/Reports';
import RecordsManagement from './pages/RecordsManagement';
import TaxBracketsManagement from './pages/TaxBracketsManagement';

type Page = 'dashboard' | 'projects' | 'reports' | 'records' | 'tax-brackets';

function App() {
  const [currentPage, setCurrentPage] = useState&lt;Page&gt;('dashboard');
  const [sidebarCollapsed, setSidebarCollapsed] = useState(false);

  const renderPage = () =&gt; {
    switch (currentPage) {
      case 'dashboard':
        return &lt;Dashboard /&gt;;
      case 'projects':
        return &lt;Projects /&gt;;
      case 'reports':
        return &lt;Reports /&gt;;
      case 'records':
        return &lt;RecordsManagement /&gt;;
      case 'tax-brackets':
        return &lt;TaxBracketsManagement /&gt;;
      default:
        return &lt;Dashboard /&gt;;
    }
  };

  return (
    &lt;div className="app"&gt;
      &lt;button
        className="menu-toggle"
        onClick={() =&gt; setSidebarCollapsed(!sidebarCollapsed)}
        aria-label="Toggle menu"
      &gt;
        &lt;svg viewBox="0 0 24 24"&gt;
          {sidebarCollapsed ? (
            &lt;path d="M3 12h18M3 6h18M3 18h18" strokeLinecap="round" strokeLinejoin="round"/&gt;
          ) : (
            &lt;path d="M6 18L18 6M6 6l12 12" strokeLinecap="round" strokeLinejoin="round"/&gt;
          )}
        &lt;/svg&gt;
      &lt;/button&gt;
      &lt;div className="top-logo"&gt;
        &lt;span className="top-logo-emoji"&gt;■&lt;/span&gt;
        &lt;span className="top-logo-text"&gt;MoneySplit&lt;/span&gt;
      &lt;/div&gt;
      &lt;aside className={`sidebar ${sidebarCollapsed ? 'collapsed' : ''}`}&gt;
        &lt;h1&gt;&lt;span className="logo-emoji"&gt;■&lt;/span&gt;MoneySplit&lt;/h1&gt;
        &lt;nav className="nav-menu"&gt;
          &lt;div
            className={`nav-item ${currentPage === 'dashboard' ? 'active' : ''}`}
            onClick={() =&gt; setCurrentPage('dashboard')}
          &gt;
            &lt;span&gt;■&lt;/span&gt;
            &lt;span&gt;Dashboard&lt;/span&gt;
          &lt;/div&gt;
          &lt;div
            className={`nav-item ${currentPage === 'projects' ? 'active' : ''}`}
            onClick={() =&gt; setCurrentPage('projects')}
```

```
          &gt;
            &lt;span&gt;■&lt;/span&gt;
            &lt;span&gt;New Project&lt;/span&gt;
          &lt;/div&gt;
          &lt;div
            className={`nav-item ${currentPage === 'reports' ? 'active' : ''}`}
            onClick={() =&gt; setCurrentPage('reports')}
          &gt;
            &lt;span&gt;■&lt;/span&gt;
            &lt;span&gt;Reports&lt;/span&gt;
          &lt;/div&gt;
          &lt;div
            className={`nav-item ${currentPage === 'records' ? 'active' : ''}`}
            onClick={() =&gt; setCurrentPage('records')}
          &gt;
            &lt;span&gt;■&lt;/span&gt;
            &lt;span&gt;Manage Records&lt;/span&gt;
          &lt;/div&gt;
          &lt;div
            className={`nav-item ${currentPage === 'tax-brackets' ? 'active' : ''}`}
            onClick={() =&gt; setCurrentPage('tax-brackets')}
          &gt;
            &lt;span&gt;■&lt;/span&gt;
            &lt;span&gt;Tax Brackets&lt;/span&gt;
          &lt;/div&gt;
        &lt;/nav&gt;
      &lt;/aside&gt;
      &lt;main className="main-content"&gt;
        {renderPage()}
      &lt;/main&gt;
    &lt;/div&gt;
  );
}

export default App;

```

File 24: frontend/src/api/client.ts

```
import axios from 'axios';

const API_BASE_URL = process.env.REACT_APP_API_URL || 'http://localhost:8000/api';

export const apiClient = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json',
  },
});

// Types
export interface Person {
  name: string;
  work_share: number;
}

export interface ProjectCreate {
  num_people: number;
  revenue: number;
  costs: number[];
  country: string;
  tax_type: 'Individual' | 'Business';
  people: Person[];
}

export interface Record {
  id: number;
  num_people: number;
  revenue: number;
  total_costs: number;
  group_income: number;
  individual_income: number;
  tax_origin: string;
  tax_option: string;
  tax_amount: number;
  net_income_per_person: number;
  net_income_group: number;
  created_at: string;
}

export interface PersonResponse {
  id: number;
  name: string;
  work_share: number;
  gross_income: number;
  tax_paid: number;
  net_income: number;
}

export interface RecordWithPeople extends Record {
  people: PersonResponse[];
}

export interface Statistics {
  total_records: number;
  total_revenue: number;
  total_costs: number;
  total_tax: number;
  total_net_income: number;
  average_tax_rate: number;
  total_people_entries: number;
  unique_people: number;
}

export interface Forecast {
  success: boolean;
  predictions: {
    month: string;
    revenue: number;
    confidence: string;
    lower_bound: number;
    upper_bound: number;
    range: string;
```

```
  }[];
  trend: string;
  trend_strength: number;
  r2_score: number;
  confidence: string;
  confidence_description: string;
  historical_avg: number;
  model_slope: number;
  growth_rate: number;
  model_type: string;
  explanation: string;
  data_quality: string;
  recommendations?: string[];
}

// API Functions
export interface RecordUpdate {
  field: string;
  value: string | number;
}

export interface TaxBracket {
  id: number;
  country: string;
  tax_type: string;
  income_limit: number;
  rate: number;
}

export interface TaxBracketCreate {
  country: string;
  tax_type: string;
  income_limit: number;
  rate: number;
}

export const projectsApi = {
  create: (data: ProjectCreate) =&gt; apiClient.post('/projects', data),
  getRecords: (limit = 10) =&gt; apiClient.get&lt;Record[]&gt;(`/records?limit=${limit}`),
  getRecord: (id: number) =&gt; apiClient.get&lt;RecordWithPeople&gt;(`/records/${id}`),
  updateRecord: (id: number, update: RecordUpdate) =&gt; apiClient.put(`/records/${id}`, update),
  deleteRecord: (id: number) =&gt; apiClient.delete(`/records/${id}`),
};

export const taxBracketsApi = {
  getTaxBrackets: (country: string, taxType: string) =&gt; apiClient.get&lt;TaxBracket[]&gt;(`/tax-
brackets?country=${co...
  createTaxBracket: (data: TaxBracketCreate) =&gt; apiClient.post('/tax-brackets', data),
  deleteTaxBracket: (id: number) =&gt; apiClient.delete(`/tax-brackets/${id}`),
};

export const reportsApi = {
  getStatistics: () =&gt; apiClient.get&lt;Statistics&gt;('/reports/statistics'),
  getRevenueSummary: () =&gt; apiClient.get('/reports/revenue-summary'),
  getTopPeople: (limit = 10) =&gt; apiClient.get(`/reports/top-people?limit=${limit}`),
};

export const forecastApi = {
  getRevenueForecast: (months = 3) =&gt; apiClient.get&lt;Forecast&gt;(`/forecast/revenue?months=${months}`),
  getComprehensive: () =&gt; apiClient.get('/forecast/comprehensive'),
  getTaxOptimization: () =&gt; apiClient.get('/forecast/tax-optimization'),
  getTrends: () =&gt; apiClient.get('/forecast/trends'),
};

export const exportApi = {
  exportRecordPDF: (id: number) =&gt; `${API_BASE_URL}/export/record/${id}/pdf`,
  exportSummaryPDF: () =&gt; `${API_BASE_URL}/export/summary/pdf`,
  exportForecastPDF: () =&gt; `${API_BASE_URL}/export/forecast/pdf`,
};

export const visualizationApi = {
  getRevenueSummary: () =&gt; `${API_BASE_URL}/visualizations/revenue-summary`,
  getMonthlyTrends: () =&gt; `${API_BASE_URL}/visualizations/monthly-trends`,
  getWorkDistribution: () =&gt; `${API_BASE_URL}/visualizations/work-distribution`,
  getTaxComparison: () =&gt; `${API_BASE_URL}/visualizations/tax-comparison`,
  getPersonPerformance: (name: string) =&gt; `${API_BASE_URL}/visualizations/person-performance/${name}`,
  getProjectProfitability: () =&gt; `${API_BASE_URL}/visualizations/project-profitability`,
};
```

File 25: frontend/src/pages/Dashboard.tsx

```
import React, { useEffect, useState } from 'react';
import { reportsApi, projectsApi, Statistics, Record } from '../api/client';

// Format dollar amounts - round down to nearest dollar
const formatCurrency = (amount: number): string =&gt; {
  return Math.floor(amount).toLocaleString();
};

// Determine font size based on string length
const getValueLength = (value: string | number): string =&gt; {
  const strValue = String(value);
  if (strValue.length &gt; 10) return 'very-long';
  if (strValue.length &gt; 7) return 'long';
  return 'normal';
};

const Dashboard: React.FC = () =&gt; {
  const [stats, setStats] = useState&lt;Statistics | null&gt;(null);
  const [recentRecords, setRecentRecords] = useState&lt;Record[]&gt;([]);
  const [loading, setLoading] = useState(true);

  useEffect(() =&gt; {
    const fetchData = async () =&gt; {
      try {
        const [statsRes, recordsRes] = await Promise.all([
          reportsApi.getStatistics(),
          projectsApi.getRecords(5),
        ]);
        setStats(statsRes.data);
        setRecentRecords(recordsRes.data);
      } catch (error) {
        console.error('Error fetching dashboard data:', error);
      } finally {
        setLoading(false);
      }
    };

    fetchData();
  }, []);

  if (loading) {
    return (
      &lt;div className="loading"&gt;
        &lt;div className="spinner"&gt;&lt;/div&gt;
      &lt;/div&gt;
    );
  }

  return (
    &lt;div&gt;
      &lt;div className="page-header"&gt;
        &lt;h2&gt;■ Dashboard&lt;/h2&gt;
        &lt;p&gt;Overview of your commission splitting business&lt;/p&gt;
      &lt;/div&gt;

      &lt;div className="stats-grid"&gt;
        &lt;div className="stat-card"&gt;
          &lt;div style={{ fontSize: '36px', marginBottom: '8px' }}&gt;■&lt;/div&gt;
          &lt;h4&gt;Total Revenue&lt;/h4&gt;
          &lt;div className="stat-value" data-length={stats ? getValueLength(`$${formatCurrency(stats.
total_revenue)}`) ...
            ${stats ? formatCurrency(stats.total_revenue) : 0}
          &lt;/div&gt;
          &lt;div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}&gt;
            All time earnings
          &lt;/div&gt;
        &lt;/div&gt;
        &lt;div className="stat-card"&gt;
          &lt;div style={{ fontSize: '36px', marginBottom: '8px' }}&gt;■&lt;/div&gt;
          &lt;h4&gt;Total Projects&lt;/h4&gt;
          &lt;div className="stat-value" data-length={getValueLength(stats?.total_records || 0)}&gt;
            {stats?.total_records || 0}
          &lt;/div&gt;
          &lt;div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}&gt;
```

```
  74 |             Completed projects
  75 |           &lt;/div&gt;
  76 |         &lt;/div&gt;
  77 |         &lt;div className="stat-card"&gt;
  78 |           &lt;div style={{ fontSize: '36px', marginBottom: '8px' }}&gt;■■&lt;/div&gt;
  79 |           &lt;h4&gt;Tax Paid&lt;/h4&gt;
  80 |           &lt;div className="stat-value" data-length={stats ? getValueLength(`$${formatCurrency(stats.
total_tax)}`) : 'n...
  81 |             ${stats ? formatCurrency(stats.total_tax) : 0}
  82 |           &lt;/div&gt;
  83 |           &lt;div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}&gt;
  84 |             Total contributions
  85 |           &lt;/div&gt;
  86 |         &lt;/div&gt;
  87 |         &lt;div className="stat-card"&gt;
  88 |           &lt;div style={{ fontSize: '36px', marginBottom: '8px' }}&gt;■&lt;/div&gt;
  89 |           &lt;h4&gt;Net Income&lt;/h4&gt;
  90 |           &lt;div className="stat-value" data-length={stats ? getValueLength(`$${formatCurrency(stats.
total_net_income)}...
  91 |             ${stats ? formatCurrency(stats.total_net_income) : 0}
  92 |           &lt;/div&gt;
  93 |           &lt;div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}&gt;
  94 |             After tax profits
  95 |           &lt;/div&gt;
  96 |         &lt;/div&gt;
  97 |       &lt;/div&gt;
  98 |
  99 |       &lt;div className="card"&gt;
 100 |         &lt;h3 style={{ marginBottom: '24px' }}&gt;■ Recent Projects&lt;/h3&gt;
 101 |         {recentRecords.length &gt; 0 ? (
 102 |           &lt;div style={{ overflowX: 'auto' }}&gt;
 103 |             &lt;table className="table"&gt;
 104 |               &lt;thead&gt;
 105 |                 &lt;tr&gt;
 106 |                   &lt;th&gt;ID&lt;/th&gt;
 107 |                   &lt;th&gt;Date&lt;/th&gt;
 108 |                   &lt;th&gt;Country&lt;/th&gt;
 109 |                   &lt;th&gt;Tax Type&lt;/th&gt;
 110 |                   &lt;th&gt;Revenue&lt;/th&gt;
 111 |                   &lt;th&gt;Net Income&lt;/th&gt;
 112 |                 &lt;/tr&gt;
 113 |               &lt;/thead&gt;
 114 |               &lt;tbody&gt;
 115 |                 {recentRecords.map((record) =&gt; (
 116 |                   &lt;tr key={record.id}&gt;
 117 |                     &lt;td&gt;
 118 |                       &lt;span style={{
 119 |                         background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
 120 |                         color: 'white',
 121 |                         padding: '4px 12px',
 122 |                         borderRadius: '12px',
 123 |                         fontSize: '13px',
 124 |                         fontWeight: 600
 125 |                       }}&gt;
 126 |                         #{record.id}
 127 |                       &lt;/span&gt;
 128 |                     &lt;/td&gt;
 129 |                     &lt;td&gt;{new Date(record.created_at).toLocaleDateString()}&lt;/td&gt;
 130 |                     &lt;td&gt;
 131 |                       &lt;span style={{ fontWeight: 600 }}&gt;{record.tax_origin}&lt;/span&gt;
 132 |                     &lt;/td&gt;
 133 |                     &lt;td&gt;
 134 |                       &lt;span style={{
 135 |                         padding: '4px 12px',
 136 |                         borderRadius: '8px',
 137 |                         fontSize: '13px',
 138 |                         fontWeight: 500,
 139 |                         background: record.tax_option === 'Individual'
 140 |                           ? 'rgba(102, 126, 234, 0.1)'
 141 |                           : 'rgba(118, 75, 162, 0.1)',
 142 |                         color: record.tax_option === 'Individual'
 143 |                           ? '#667eea'
 144 |                           : '#764ba2'
 145 |                       }}&gt;
 146 |                         {record.tax_option}
 147 |                       &lt;/span&gt;
 148 |                     &lt;/td&gt;
 149 |                     &lt;td style={{ fontWeight: 600, color: '#2d3748' }}&gt;
```

```
                      ${formatCurrency(record.revenue)}
                    &lt;/td&gt;
                    &lt;td style={{
                      fontWeight: 700,
                      background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
                      WebkitBackgroundClip: 'text',
                      WebkitTextFillColor: 'transparent'
                    }}&gt;
                      ${formatCurrency(record.net_income_group)}
                    &lt;/td&gt;
                  &lt;/tr&gt;
                ))}
              &lt;/tbody&gt;
            &lt;/table&gt;
          &lt;/div&gt;
        ) : (
          &lt;div style={{
            textAlign: 'center',
            padding: '48px 24px',
            color: '#718096'
          }}&gt;
            &lt;div style={{ fontSize: '64px', marginBottom: '16px' }}&gt;■&lt;/div&gt;
            &lt;p style={{ fontSize: '18px', fontWeight: 500 }}&gt;No projects yet&lt;/p&gt;
            &lt;p style={{ fontSize: '14px', marginTop: '8px' }}&gt;Create your first project to get started!&lt;
/p&gt;
          &lt;/div&gt;
        )}
      &lt;/div&gt;

      &lt;div className="stats-grid"&gt;
        &lt;div className="stat-card"&gt;
          &lt;div style={{ fontSize: '36px', marginBottom: '8px' }}&gt;■&lt;/div&gt;
          &lt;h4&gt;Average Tax Rate&lt;/h4&gt;
          &lt;div className="stat-value"&gt;{stats?.average_tax_rate.toFixed(1)}%&lt;/div&gt;
          &lt;div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}&gt;
            Effective rate
          &lt;/div&gt;
        &lt;/div&gt;
        &lt;div className="stat-card"&gt;
          &lt;div style={{ fontSize: '36px', marginBottom: '8px' }}&gt;■&lt;/div&gt;
          &lt;h4&gt;Unique People&lt;/h4&gt;
          &lt;div className="stat-value"&gt;{stats?.unique_people || 0}&lt;/div&gt;
          &lt;div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}&gt;
            Team members
          &lt;/div&gt;
        &lt;/div&gt;
        &lt;div className="stat-card"&gt;
          &lt;div style={{ fontSize: '36px', marginBottom: '8px' }}&gt;■&lt;/div&gt;
          &lt;h4&gt;Total Costs&lt;/h4&gt;
          &lt;div className="stat-value" data-length={stats ? getValueLength(`$${formatCurrency(stats.
total_costs)}`) : ...
            ${stats ? formatCurrency(stats.total_costs) : 0}
          &lt;/div&gt;
          &lt;div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}&gt;
            Business expenses
          &lt;/div&gt;
        &lt;/div&gt;
        &lt;div className="stat-card"&gt;
          &lt;div style={{ fontSize: '36px', marginBottom: '8px' }}&gt;■&lt;/div&gt;
          &lt;h4&gt;Profit Margin&lt;/h4&gt;
          &lt;div className="stat-value" data-length="normal"&gt;
            {stats
              ? Math.floor((stats.total_net_income / stats.total_revenue) * 100)
              : 0}%
          &lt;/div&gt;
          &lt;div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}&gt;
            Net profit ratio
          &lt;/div&gt;
        &lt;/div&gt;
      &lt;/div&gt;
    &lt;/div&gt;
  );
};

export default Dashboard;
```

File 26: frontend/src/pages/Projects.tsx

```
import React, { useState } from 'react';
import { projectsApi, ProjectCreate, Person } from '../api/client';

const Projects: React.FC = () =&gt; {
  const [numPeople, setNumPeople] = useState&lt;number&gt;(2);
  const [revenue, setRevenue] = useState&lt;string&gt;('');
  const [costs, setCosts] = useState&lt;string&gt;('');
  const [country, setCountry] = useState&lt;string&gt;('US');
  const [taxType, setTaxType] = useState&lt;'Individual' | 'Business'&gt;('Individual');
  const [people, setPeople] = useState&lt;Person[]&gt;([
    { name: '', work_share: 0.5 },
    { name: '', work_share: 0.5 },
  ]);
  const [successMessage, setSuccessMessage] = useState&lt;string&gt;('');
  const [errorMessage, setErrorMessage] = useState&lt;string&gt;('');
  const [loading, setLoading] = useState(false);

  const handleNumPeopleChange = (value: number) =&gt; {
    setNumPeople(value);
    const newPeople: Person[] = [];
    const sharePerPerson = 1 / value;

    for (let i = 0; i &lt; value; i++) {
      newPeople.push({
        name: people[i]?.name || '',
        work_share: sharePerPerson,
      });
    }
    setPeople(newPeople);
  };

  const handlePersonChange = (index: number, field: keyof Person, value: string | number) =&gt; {
    const newPeople = [...people];
    newPeople[index] = { ...newPeople[index], [field]: value };
    setPeople(newPeople);
  };

  const handleSubmit = async (e: React.FormEvent) =&gt; {
    e.preventDefault();
    setSuccessMessage('');
    setErrorMessage('');
    setLoading(true);

    try {
      const costsArray = costs.split(',').map(c =&gt; parseFloat(c.trim()));

      const data: ProjectCreate = {
        num_people: numPeople,
        revenue: parseFloat(revenue),
        costs: costsArray,
        country,
        tax_type: taxType,
        people,
      };

      const response = await projectsApi.create(data);
      setSuccessMessage(`Project created successfully! Record ID: ${response.data.record_id}`);

      // Reset form
      setRevenue('');
      setCosts('');
      handleNumPeopleChange(2);
    } catch (error: any) {
      setErrorMessage(error.response?.data?.detail || 'Error creating project');
    } finally {
      setLoading(false);
    }
  };

  return (
    &lt;div&gt;
      &lt;div className="page-header"&gt;
        &lt;h2&gt;■ New Project&lt;/h2&gt;
        &lt;p&gt;Create a new commission splitting project&lt;/p&gt;
```

```
 75 |       &lt;/div&gt;
 76 |
 77 |       {successMessage &amp;&amp; (
 78 |         &lt;div className="alert alert-success" style={{
 79 |           display: 'flex',
 80 |           alignItems: 'center',
 81 |           gap: '12px'
 82 |         }}&gt;
 83 |           &lt;span style={{ fontSize: '24px' }}&gt;■&lt;/span&gt;
 84 |           &lt;span&gt;{successMessage}&lt;/span&gt;
 85 |         &lt;/div&gt;
 86 |       )}
 87 |
 88 |       {errorMessage &amp;&amp; (
 89 |         &lt;div className="alert alert-error" style={{
 90 |           display: 'flex',
 91 |           alignItems: 'center',
 92 |           gap: '12px'
 93 |         }}&gt;
 94 |           &lt;span style={{ fontSize: '24px' }}&gt;■&lt;/span&gt;
 95 |           &lt;span&gt;{errorMessage}&lt;/span&gt;
 96 |         &lt;/div&gt;
 97 |       )}
 98 |
 99 |       &lt;div className="card"&gt;
100 |         &lt;form onSubmit={handleSubmit}&gt;
101 |           &lt;div className="form-group"&gt;
102 |             &lt;label&gt;Number of People&lt;/label&gt;
103 |             &lt;input
104 |               type="number"
105 |               min="1"
106 |               value={numPeople}
107 |               onChange={(e) =&gt; handleNumPeopleChange(parseInt(e.target.value))}
108 |               required
109 |             /&gt;
110 |           &lt;/div&gt;
111 |
112 |           &lt;div className="form-group"&gt;
113 |             &lt;label&gt;Revenue ($)&lt;/label&gt;
114 |             &lt;input
115 |               type="number"
116 |               step="0.01"
117 |               value={revenue}
118 |               onChange={(e) =&gt; setRevenue(e.target.value)}
119 |               placeholder="10000"
120 |               required
121 |             /&gt;
122 |           &lt;/div&gt;
123 |
124 |           &lt;div className="form-group"&gt;
125 |             &lt;label&gt;Costs (comma-separated)&lt;/label&gt;
126 |             &lt;input
127 |               type="text"
128 |               value={costs}
129 |               onChange={(e) =&gt; setCosts(e.target.value)}
130 |               placeholder="1000, 500, 300"
131 |               required
132 |             /&gt;
133 |           &lt;/div&gt;
134 |
135 |           &lt;div className="form-group"&gt;
136 |             &lt;label&gt;Country&lt;/label&gt;
137 |             &lt;select value={country} onChange={(e) =&gt; setCountry(e.target.value)}&gt;
138 |               &lt;option value="US"&gt;United States&lt;/option&gt;
139 |               &lt;option value="Spain"&gt;Spain&lt;/option&gt;
140 |               &lt;option value="UK"&gt;United Kingdom&lt;/option&gt;
141 |               &lt;option value="Canada"&gt;Canada&lt;/option&gt;
142 |               &lt;option value="Other"&gt;Other&lt;/option&gt;
143 |             &lt;/select&gt;
144 |           &lt;/div&gt;
145 |
146 |           &lt;div className="form-group"&gt;
147 |             &lt;label&gt;Tax Type&lt;/label&gt;
148 |             &lt;select value={taxType} onChange={(e) =&gt; setTaxType(e.target.value as 'Individual' |
'Business')}&gt;
149 |               &lt;option value="Individual"&gt;Individual&lt;/option&gt;
150 |               &lt;option value="Business"&gt;Business&lt;/option&gt;
151 |             &lt;/select&gt;
```

```
152 |           &lt;/div&gt;
153 |
154 |           &lt;h3 style={{ marginTop: '32px', marginBottom: '20px' }}&gt;■ Team Members&lt;/h3&gt;
155 |           {people.map((person, index) =&gt; (
156 |             &lt;div key={index} className="person-card"&gt;
157 |               &lt;div className="form-group"&gt;
158 |                 &lt;label&gt;Person {index + 1} - Name&lt;/label&gt;
159 |                 &lt;input
160 |                   type="text"
161 |                   value={person.name}
162 |                   onChange={(e) =&gt; handlePersonChange(index, 'name', e.target.value)}
163 |                   placeholder="John Doe"
164 |                   required
165 |                 /&gt;
166 |               &lt;/div&gt;
167 |               &lt;div className="form-group"&gt;
168 |                 &lt;label&gt;Work Share (0 - 1)&lt;/label&gt;
169 |                 &lt;input
170 |                   type="number"
171 |                   step="0.01"
172 |                   min="0"
173 |                   max="1"
174 |                   value={person.work_share}
175 |                   onChange={(e) =&gt; handlePersonChange(index, 'work_share', parseFloat(e.target.value))}
176 |                   required
177 |                 /&gt;
178 |                 &lt;small style={{ display: 'block', marginTop: '8px', color: '#718096', fontWeight: 500 }}&gt;
179 |                   {(person.work_share * 100).toFixed(0)}% of total work
180 |                 &lt;/small&gt;
181 |               &lt;/div&gt;
182 |             &lt;/div&gt;
183 |           ))}
184 |
185 |           &lt;button type="submit" className="btn btn-primary" disabled={loading}&gt;
186 |             {loading ? 'Creating...' : 'Create Project'}
187 |           &lt;/button&gt;
188 |         &lt;/form&gt;
189 |       &lt;/div&gt;
190 |     &lt;/div&gt;
191 |   );
192 | };
193 |
194 | export default Projects;
195 |
```

File 27: requirements.txt

```
# Core dependencies
plotly==6.3.0

# API dependencies
fastapi==0.116.1
uvicorn==0.35.0
pydantic==2.11.7

# PDF generation
reportlab==4.4.4

# Machine Learning &amp;amp; Data Science
scikit-learn&amp;gt;=1.0.0
numpy&amp;gt;=1.20.0
pandas&amp;gt;=2.0.0

# Python version requirement
# Python &amp;gt;= 3.8

```

File 28: tests/__init__.py

```
1 | # Test suite for MoneySplit application
2 |
```

File 29: tests/conftest.py

```
"""Pytest configuration and fixtures."""

import pytest
import os
import sys
import sqlite3

# Add parent directory to path for imports
sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

# Use same database as production for school project
# os.environ['TESTING'] = '1'
# os.environ['TEST_DB'] = 'test_example.db'


@pytest.fixture(scope="session", autouse=True)
def test_database():
    """Use production database for school project."""
    # No test database setup - using production database
    yield "example.db"


@pytest.fixture
def sample_project_data():
    """Provide sample project data for tests."""
    return {
        "num_people": 2,
        "revenue": 10000,
        "costs": [1000, 500, 300],
        "country": "US",
        "tax_type": "Individual",
        "people": [
            {"name": "Alice", "work_share": 0.6},
            {"name": "Bob", "work_share": 0.4}
        ]
    }


@pytest.fixture
def sample_tax_brackets():
    """Provide sample tax brackets for tests."""
    return [
        {'min': 0, 'max': 11000, 'rate': 0.10},
        {'min': 11000, 'max': 44725, 'rate': 0.12},
        {'min': 44725, 'max': 95375, 'rate': 0.22},
        {'min': 95375, 'max': 182100, 'rate': 0.24}
    ]

```

File 30: tests/test_api.py

```
"""
Integration tests for FastAPI endpoints.

AI ASSISTANCE DISCLOSURE:
All tests in this file were generated with AI assistance (ChatGPT/Claude).
- Prompts used: "Write pytest tests for FastAPI CRUD endpoints"
- Prompts used: "Create integration tests for forecasting and export endpoints"
- Prompts used: "Add edge case tests for invalid inputs and error handling"
- AI helped structure test cases, assertions, and test data fixtures
"""

import pytest
from fastapi.testclient import TestClient
from api.main import app

client = TestClient(app)


class TestProjectEndpoints:
    """Test project CRUD endpoints."""

    def test_create_project_success(self):
        """Test successful project creation."""
        payload = {
            "num_people": 2,
            "revenue": 10000,
            "costs": [1000, 500],
            "country": "US",
            "tax_type": "Individual",
            "people": [
                {"name": "Alice", "work_share": 0.6},
                {"name": "Bob", "work_share": 0.4}
            ]
        }

        response = client.post("/api/projects", json=payload)

        assert response.status_code == 201  # FastAPI returns 201 for created
        data = response.json()
        assert "record_id" in data
        assert data["message"] == "Project created successfully"

    def test_create_project_invalid_work_shares(self):
        """Test project creation with invalid work shares."""
        payload = {
            "num_people": 2,
            "revenue": 10000,
            "costs": [1000],
            "country": "US",
            "tax_type": "Individual",
            "people": [
                {"name": "Alice", "work_share": 0.7},
                {"name": "Bob", "work_share": 0.7}  # Sums to 1.4
            ]
        }

        response = client.post("/api/projects", json=payload)
        assert response.status_code == 422  # FastAPI validation error

    def test_create_project_negative_revenue(self):
        """Test project creation with negative revenue."""
        payload = {
            "num_people": 1,
            "revenue": -5000,
            "costs": [1000],
            "country": "US",
            "tax_type": "Individual",
            "people": [
                {"name": "Alice", "work_share": 1.0}
            ]
        }

        response = client.post("/api/projects", json=payload)
        assert response.status_code == 422  # Validation error
```

```
|
|     def test_get_recent_records(self):
|         """Test fetching recent records."""
|         response = client.get("/api/records")
|
|         assert response.status_code == 200
|         data = response.json()
|         assert isinstance(data, list)
|
|     def test_get_record_by_id(self):
|         """Test fetching specific record by ID."""
|         # First create a record
|         payload = {
|             "num_people": 1,
|             "revenue": 5000,
|             "costs": [500],
|             "country": "US",
|             "tax_type": "Individual",
|             "people": [
|                 {"name": "Test User", "work_share": 1.0}
|             ]
|         }
|
|         create_response = client.post("/api/projects", json=payload)
|         record_id = create_response.json()["record_id"]
|
|         # Fetch the record
|         response = client.get(f"/api/records/{record_id}")
|
|         assert response.status_code == 200
|         data = response.json()
|         assert data["id"] == record_id  # API uses "id" not "record_id"
|
|     def test_get_nonexistent_record(self):
|         """Test fetching a non-existent record."""
|         response = client.get("/api/records/999999")
|
|         assert response.status_code == 404
|
|
| class TestReportEndpoints:
|     """Test report and statistics endpoints."""
|
|     def test_get_statistics(self):
|         """Test overall statistics endpoint."""
|         response = client.get("/api/reports/statistics")
|
|         assert response.status_code == 200
|         data = response.json()
|         assert "total_revenue" in data
|         assert "total_records" in data  # API uses "total_records" not "total_projects"
|         assert "total_tax" in data  # API uses "total_tax" not "total_tax_paid"
|
|     def test_get_revenue_summary(self):
|         """Test revenue summary by year."""
|         response = client.get("/api/reports/revenue-summary")
|
|         assert response.status_code == 200
|         data = response.json()
|         assert isinstance(data, list)
|
|     def test_get_top_people(self):
|         """Test top contributors endpoint."""
|         response = client.get("/api/reports/top-people")
|
|         assert response.status_code == 200
|         data = response.json()
|         assert isinstance(data, list)
|         if len(data) &gt; 0:
|             assert "name" in data[0]
|             assert "total_gross" in data[0]  # API uses "total_gross" not "total_earned"
|
|
| class TestForecastingEndpoints:
|     """Test forecasting and ML endpoints."""
|
|     def test_revenue_forecast_default(self):
|         """Test revenue forecasting with default months."""
```

```
        response = client.get("/api/forecast/revenue")

        assert response.status_code == 200
        data = response.json()
        assert "predictions" in data
        # May not have enough data, so just check structure exists

    def test_revenue_forecast_custom_months(self):
        """Test revenue forecasting with custom months."""
        response = client.get("/api/forecast/revenue?months=6")

        assert response.status_code == 200
        data = response.json()
        assert "predictions" in data
        # May be empty if insufficient data

    def test_comprehensive_forecast(self):
        """Test comprehensive forecast endpoint."""
        response = client.get("/api/forecast/comprehensive")

        assert response.status_code == 200
        data = response.json()
        assert "revenue_forecast" in data
        assert "recommendations" in data  # API uses "recommendations" directly

    def test_tax_optimization(self):
        """Test tax optimization recommendations."""
        response = client.get("/api/forecast/tax-optimization")

        assert response.status_code == 200
        data = response.json()
        assert "tax_comparison" in data  # API structure is different
        assert "recommendations" in data

    def test_trends_analysis(self):
        """Test trend analysis endpoint."""
        response = client.get("/api/forecast/trends")

        assert response.status_code == 200
        data = response.json()
        # May not have enough data, check for success or message
        assert "success" in data or "message" in data


class TestVisualizationEndpoints:
    """Test visualization endpoints."""

    def test_revenue_summary_visualization(self):
        """Test revenue summary visualization."""
        response = client.get("/api/visualizations/revenue-summary")

        assert response.status_code == 200
        assert "text/html" in response.headers["content-type"]
        assert b"Revenue Summary" in response.content

    def test_monthly_trends_visualization(self):
        """Test monthly trends visualization."""
        response = client.get("/api/visualizations/monthly-trends")

        assert response.status_code == 200
        assert "text/html" in response.headers["content-type"]

    def test_work_distribution_visualization(self):
        """Test work distribution visualization."""
        response = client.get("/api/visualizations/work-distribution")

        assert response.status_code == 200
        assert "text/html" in response.headers["content-type"]

    def test_tax_comparison_visualization(self):
        """Test tax comparison visualization."""
        response = client.get("/api/visualizations/tax-comparison")

        assert response.status_code == 200
        assert "text/html" in response.headers["content-type"]

    def test_profitability_visualization(self):
        """Test project profitability visualization."""
```

```
        response = client.get("/api/visualizations/project-profitability")

        assert response.status_code == 200
        assert "text/html" in response.headers["content-type"]


class TestPDFExportEndpoints:
    """Test PDF export endpoints."""

    def test_export_record_pdf(self):
        """Test exporting a record as PDF."""
        # First create a record
        payload = {
            "num_people": 1,
            "revenue": 5000,
            "costs": [500],
            "country": "US",
            "tax_type": "Individual",
            "people": [
                {"name": "PDF Test User", "work_share": 1.0}
            ]
        }

        create_response = client.post("/api/projects", json=payload)
        record_id = create_response.json()["record_id"]

        # Export as PDF
        response = client.get(f"/api/export/record/{record_id}/pdf")

        assert response.status_code == 200
        assert response.headers["content-type"] == "application/pdf"

    def test_export_summary_pdf(self):
        """Test exporting summary as PDF."""
        response = client.get("/api/export/summary/pdf")

        assert response.status_code == 200
        assert response.headers["content-type"] == "application/pdf"

    def test_export_forecast_pdf(self):
        """Test exporting forecast as PDF."""
        response = client.get("/api/export/forecast/pdf")

        assert response.status_code == 200
        assert response.headers["content-type"] == "application/pdf"


class TestHealthCheck:
    """Test health check endpoint."""

    def test_root_endpoint(self):
        """Test API root endpoint."""
        response = client.get("/")

        assert response.status_code == 200
        # Root may return HTML, not JSON
        assert response.text is not None

```

File 31: tests/test_backend_logic.py

```
"""
Unit tests for backend business logic.

AI ASSISTANCE DISCLOSURE:
All tests in this file were generated with AI assistance (ChatGPT/Claude).
- Prompts used: "Create unit tests for tax calculation functions"
- Prompts used: "Write tests for work share distribution logic"
- Prompts used: "Add parameterized tests for different income levels"
- AI helped create comprehensive test coverage for all business logic functions
"""

import pytest
from Logic.tax_calculator import calculate_tax, calculate_tax_from_db, split_work_shares, calculate_profit
from Logic.validators import (
    validate_positive_number,
    validate_work_shares,
    validate_work_share,
    validate_non_empty_string,
    validate_country,
    validate_tax_type,
    validate_tax_rate,
    ValidationError
)


class TestTaxCalculations:
    """Test tax calculation logic."""

    def test_calculate_progressive_tax(self):
        """Test progressive tax calculation with mock brackets."""
        # Mock tax brackets: 10% up to 10k, 20% above 10k
        from unittest.mock import patch

        mock_brackets = [(10000, 0.10), (50000, 0.20)]

        with patch('Logic.tax_calculator.setup.get_tax_brackets', return_value=mock_brackets):
            # Income of 15000: (10000 * 0.10) + (5000 * 0.20) = 1000 + 1000 = 2000
            tax = calculate_tax_from_db(15000, "US", "Individual")
            assert tax == 2000.0

    def test_calculate_tax_single_bracket(self):
        """Test tax calculation within single bracket."""
        from unittest.mock import patch

        mock_brackets = [(10000, 0.10), (50000, 0.20)]

        with patch('Logic.tax_calculator.setup.get_tax_brackets', return_value=mock_brackets):
            # Income of 5000: entirely in first bracket
            tax = calculate_tax_from_db(5000, "US", "Individual")
            assert tax == 500.0  # 5000 * 0.10

    def test_calculate_tax_zero_income(self):
        """Test tax calculation for zero income."""
        from unittest.mock import patch

        mock_brackets = [(10000, 0.10)]

        with patch('Logic.tax_calculator.setup.get_tax_brackets', return_value=mock_brackets):
            tax = calculate_tax_from_db(0, "US", "Individual")
            assert tax == 0.0

    def test_calculate_tax_direct(self):
        """Test direct tax calculation without DB."""
        brackets = [(10000, 0.10), (50000, 0.20)]

        # Income of 15000: (10000 * 0.10) + (5000 * 0.20) = 1000 + 1000 = 2000
        tax = calculate_tax(15000, brackets)
        assert tax == 2000.0

        # Income of 5000: entirely in first bracket
        tax = calculate_tax(5000, brackets)
        assert tax == 500.0


```

```
class TestWorkShareDistribution:
    """Test work share distribution logic."""

    def test_equal_split_two_people(self):
        """Test equal work share split between two people."""
        profit = 10000
        work_shares = [0.5, 0.5]

        distribution = split_work_shares(profit, work_shares)

        assert len(distribution) == 2
        assert distribution[0] == 5000.0
        assert distribution[1] == 5000.0

    def test_unequal_split_three_people(self):
        """Test unequal work share split."""
        profit = 10000
        work_shares = [0.5, 0.3, 0.2]

        distribution = split_work_shares(profit, work_shares)

        assert len(distribution) == 3
        assert distribution[0] == 5000.0
        assert distribution[1] == 3000.0
        assert distribution[2] == 2000.0


class TestValidators:
    """Test input validation functions."""

    def test_validate_positive_number_valid(self):
        """Test positive number validation with valid value."""
        result = validate_positive_number(10000, "Revenue")
        assert result == 10000

    def test_validate_positive_number_zero(self):
        """Test positive number validation with zero."""
        result = validate_positive_number(0, "Revenue")
        assert result == 0

    def test_validate_positive_number_negative(self):
        """Test positive number validation with negative value."""
        with pytest.raises(ValidationError):
            validate_positive_number(-5000, "Revenue")

    def test_validate_work_shares_valid(self):
        """Test work shares validation with valid shares."""
        # Should not raise exception
        validate_work_shares([0.5, 0.5])
        validate_work_shares([0.6, 0.4])
        validate_work_shares([0.33, 0.33, 0.34])

    def test_validate_work_shares_invalid_sum(self):
        """Test work shares validation with invalid sum."""
        with pytest.raises(ValidationError):
            validate_work_shares([0.6, 0.6])  # Sums to 1.2

        with pytest.raises(ValidationError):
            validate_work_shares([0.3, 0.3])  # Sums to 0.6

    def test_validate_work_share_valid(self):
        """Test single work share validation."""
        assert validate_work_share(0.5) == 0.5
        assert validate_work_share(0.0) == 0.0
        assert validate_work_share(1.0) == 1.0

    def test_validate_work_share_invalid(self):
        """Test single work share validation with invalid values."""
        with pytest.raises(ValidationError):
            validate_work_share(-0.1)

        with pytest.raises(ValidationError):
            validate_work_share(1.5)

    def test_validate_non_empty_string_valid(self):
        """Test non-empty string validation."""
        assert validate_non_empty_string("Alice", "Name") == "Alice"
        assert validate_non_empty_string("  Bob  ", "Name") == "Bob"
```

```
|
|     def test_validate_non_empty_string_invalid(self):
|         """Test non-empty string validation with empty string."""
|         with pytest.raises(ValidationError):
|             validate_non_empty_string("", "Name")
|
|         with pytest.raises(ValidationError):
|             validate_non_empty_string("   ", "Name")
|
|     def test_validate_country_valid(self):
|         """Test country validation."""
|         assert validate_country("US") == "US"
|         assert validate_country("Spain") == "Spain"
|
|     def test_validate_country_invalid(self):
|         """Test country validation with empty string."""
|         with pytest.raises(ValidationError):
|             validate_country("")
|
|     def test_validate_tax_type_valid(self):
|         """Test tax type validation."""
|         assert validate_tax_type("Individual") == "Individual"
|         assert validate_tax_type("Business") == "Business"
|         assert validate_tax_type("individual") == "Individual"  # Case insensitive
|
|     def test_validate_tax_type_invalid(self):
|         """Test tax type validation with invalid type."""
|         with pytest.raises(ValidationError):
|             validate_tax_type("Corporate")
|
|     def test_validate_tax_rate_valid(self):
|         """Test tax rate validation."""
|         assert validate_tax_rate(0.10) == 0.10
|         assert validate_tax_rate(0.0) == 0.0
|         assert validate_tax_rate(1.0) == 1.0
|
|     def test_validate_tax_rate_invalid(self):
|         """Test tax rate validation with invalid values."""
|         with pytest.raises(ValidationError):
|             validate_tax_rate(-0.1)
|
|         with pytest.raises(ValidationError):
|             validate_tax_rate(1.5)
|
|
| class TestProfitCalculations:
|     """Test profit calculation logic."""
|
|     def test_profit_calculation_basic(self):
|         """Test basic profit calculation."""
|         revenue = 10000
|         costs = [1000, 500, 300]
|
|         profit = calculate_profit(revenue, costs)
|         assert profit == 8200
|
|     def test_profit_calculation_no_profit(self):
|         """Test profit calculation with zero profit."""
|         revenue = 5000
|         costs = [3000, 2000]
|
|         profit = calculate_profit(revenue, costs)
|         assert profit == 0
|
|     def test_profit_calculation_loss(self):
|         """Test profit calculation with negative profit (loss)."""
|         revenue = 5000
|         costs = [4000, 2000]
|
|         profit = calculate_profit(revenue, costs)
|         assert profit == -1000
|
|     def test_profit_calculation_no_costs(self):
|         """Test profit calculation with no costs."""
|         revenue = 10000
|         costs = []
|
|         profit = calculate_profit(revenue, costs)
```

```
231 |         assert profit == 10000
232 |
```

File 32: tests/test_database.py

```
"""
Tests for database operations.

AI ASSISTANCE DISCLOSURE:
All tests in this file were generated with AI assistance (ChatGPT/Claude).
- Prompts used: "Write pytest tests for database CRUD operations"
- Prompts used: "Create tests for foreign key constraints and referential integrity"
- Prompts used: "Add tests for aggregation queries and complex SQL operations"
- AI helped generate test data, assertions, and database fixture management
"""

import pytest
import sqlite3
import os
from DB import setup


class TestDatabaseOperations:
    """Test database CRUD operations."""

    def test_insert_and_fetch_record(self):
        """Test inserting and fetching a tax record."""
        conn = setup.get_conn()
        cursor = conn.cursor()

        # Insert a test record
        cursor.execute("""
            INSERT INTO tax_records (num_people, revenue, total_costs, group_income,
                                     individual_income, tax_origin, tax_option, tax_amount,
                                     net_income_per_person, net_income_group)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        """, (2, 10000, 1500, 8500, 4250, "US", "Individual", 500, 3750, 8000))

        record_id = cursor.lastrowid
        conn.commit()

        # Fetch the record
        cursor.execute("SELECT * FROM tax_records WHERE id = ?", (record_id,))
        record = cursor.fetchone()

        assert record is not None
        assert record[2] == 10000  # revenue
        assert record[3] == 1500   # total_costs

        conn.close()

    def test_delete_record(self):
        """Test deleting a record."""
        conn = setup.get_conn()
        cursor = conn.cursor()

        # Insert
        cursor.execute("""
            INSERT INTO tax_records (num_people, revenue, total_costs, group_income,
                                     individual_income, tax_origin, tax_option, tax_amount,
                                     net_income_per_person, net_income_group)
            VALUES (1, 5000, 500, 4500, 4500, "US", "Individual", 450, 4050, 4050)
        """)
        record_id = cursor.lastrowid
        conn.commit()

        # Delete
        cursor.execute("DELETE FROM tax_records WHERE id = ?", (record_id,))
        conn.commit()

        # Verify deleted
        cursor.execute("SELECT * FROM tax_records WHERE id = ?", (record_id,))
        assert cursor.fetchone() is None

        conn.close()

    def test_update_record(self):
        """Test updating a record."""
        conn = setup.get_conn()
```

```
        cursor = conn.cursor()

        # Insert
        cursor.execute("""
            INSERT INTO tax_records (num_people, revenue, total_costs, group_income,
                                     individual_income, tax_origin, tax_option, tax_amount,
                                     net_income_per_person, net_income_group)
            VALUES (1, 5000, 500, 4500, 4500, "US", "Individual", 450, 4050, 4050)
        """)
        record_id = cursor.lastrowid
        conn.commit()

        # Update
        cursor.execute("UPDATE tax_records SET revenue = ? WHERE id = ?", (6000, record_id))
        conn.commit()

        # Verify updated
        cursor.execute("SELECT revenue FROM tax_records WHERE id = ?", (record_id,))
        updated_revenue = cursor.fetchone()[0]
        assert updated_revenue == 6000

        conn.close()


class TestPeopleTable:
    """Test people table operations."""

    def test_insert_person(self):
        """Test adding a person to a record."""
        conn = setup.get_conn()
        cursor = conn.cursor()

        # Create a tax record first
        cursor.execute("""
            INSERT INTO tax_records (num_people, revenue, total_costs, group_income,
                                     individual_income, tax_origin, tax_option, tax_amount,
                                     net_income_per_person, net_income_group)
            VALUES (1, 5000, 500, 4500, 4500, "US", "Individual", 450, 4050, 4050)
        """)
        record_id = cursor.lastrowid

        # Insert person
        cursor.execute("""
            INSERT INTO people (record_id, name, work_share, gross_income, tax_paid, net_income)
            VALUES (?, ?, ?, ?, ?, ?)
        """, (record_id, "Alice", 1.0, 4500, 450, 4050))

        conn.commit()

        # Verify
        cursor.execute("SELECT * FROM people WHERE record_id = ?", (record_id,))
        person = cursor.fetchone()

        assert person is not None
        assert person[2] == "Alice"
        assert person[3] == 1.0

        conn.close()

    def test_multiple_people_per_record(self):
        """Test adding multiple people to one record."""
        conn = setup.get_conn()
        cursor = conn.cursor()

        # Create record
        cursor.execute("""
            INSERT INTO tax_records (num_people, revenue, total_costs, group_income,
                                     individual_income, tax_origin, tax_option, tax_amount,
                                     net_income_per_person, net_income_group)
            VALUES (2, 10000, 1000, 9000, 4500, "US", "Individual", 900, 4050, 8100)
        """)
        record_id = cursor.lastrowid

        # Insert multiple people
        people = [
            (record_id, "Alice", 0.6, 5400, 540, 4860),
            (record_id, "Bob", 0.4, 3600, 360, 3240)
        ]
```

```
|
|         cursor.executemany("""
|             INSERT INTO people (record_id, name, work_share, gross_income, tax_paid, net_income)
|             VALUES (?, ?, ?, ?, ?, ?)
|         """, people)
|
|         conn.commit()
|
|         # Verify
|         cursor.execute("SELECT COUNT(*) FROM people WHERE record_id = ?", (record_id,))
|         count = cursor.fetchone()[0]
|         assert count == 2
|
|         cursor.execute("SELECT SUM(work_share) FROM people WHERE record_id = ?", (record_id,))
|         total_share = cursor.fetchone()[0]
|         assert abs(total_share - 1.0) &lt; 0.01  # Should sum to 1.0
|
|         conn.close()
|
|
| class TestTaxBrackets:
|     """Test tax brackets table."""
|
|     def test_fetch_tax_brackets(self):
|         """Test fetching tax brackets."""
|         conn = setup.get_conn()
|         cursor = conn.cursor()
|
|         # Fetch US Individual brackets
|         cursor.execute("""
|             SELECT income_limit, rate FROM tax_brackets
|             WHERE country = 'US' AND tax_type = 'Individual'
|             ORDER BY income_limit
|         """)
|
|         brackets = cursor.fetchall()
|         assert len(brackets) &gt; 0
|
|         # First bracket should have lowest rate
|         assert brackets[0][1] &lt;= brackets[-1][1]
|
|         conn.close()
|
|     def test_tax_bracket_ordering(self):
|         """Test that tax brackets are ordered correctly."""
|         conn = setup.get_conn()
|         cursor = conn.cursor()
|
|         cursor.execute("""
|             SELECT income_limit FROM tax_brackets
|             WHERE country = 'US' AND tax_type = 'Individual'
|             ORDER BY income_limit
|         """)
|
|         limits = [row[0] for row in cursor.fetchall()]
|
|         # Verify ascending order
|         for i in range(len(limits) - 1):
|             assert limits[i] &lt; limits[i + 1]
|
|         conn.close()
|
|
| class TestDataIntegrity:
|     """Test data integrity and constraints."""
|
|     def test_foreign_key_constraint(self):
|         """Test that foreign key constraints are enforced."""
|         conn = setup.get_conn()
|         cursor = conn.cursor()
|
|         # Try to insert person with non-existent record_id
|         with pytest.raises(sqlite3.IntegrityError):
|             cursor.execute("""
|                 INSERT INTO people (record_id, name, work_share, gross_income, tax_paid, net_income)
|                 VALUES (?, ?, ?, ?, ?, ?)
|             """, (99999, "NonExistent", 1.0, 1000, 100, 900))
|             conn.commit()
```

```
|
|         conn.close()
|
|     def test_cascade_delete(self):
|         """Test that deleting a record requires deleting people first (FK constraint)."""
|         conn = setup.get_conn()
|         cursor = conn.cursor()
|
|         # Create record with person
|         cursor.execute("""
|             INSERT INTO tax_records (num_people, revenue, total_costs, group_income,
|                                      individual_income, tax_origin, tax_option, tax_amount,
|                                      net_income_per_person, net_income_group)
|             VALUES (1, 5000, 500, 4500, 4500, "US", "Individual", 450, 4050, 4050)
|         """)
|         record_id = cursor.lastrowid
|
|         cursor.execute("""
|             INSERT INTO people (record_id, name, work_share, gross_income, tax_paid, net_income)
|             VALUES (?, ?, ?, ?, ?, ?)
|         """, (record_id, "Test", 1.0, 4500, 450, 4050))
|
|         conn.commit()
|
|         # Delete people first, then record (FK constraint prevents direct delete)
|         cursor.execute("DELETE FROM people WHERE record_id = ?", (record_id,))
|         cursor.execute("DELETE FROM tax_records WHERE id = ?", (record_id,))
|         conn.commit()
|
|         # Verify both deleted
|         cursor.execute("SELECT COUNT(*) FROM people WHERE record_id = ?", (record_id,))
|         assert cursor.fetchone()[0] == 0
|
|         cursor.execute("SELECT COUNT(*) FROM tax_records WHERE id = ?", (record_id,))
|         assert cursor.fetchone()[0] == 0
|
|         conn.close()
|
|
| class TestDatabaseQueries:
|     """Test complex database queries."""
|
|     def test_aggregate_revenue_by_month(self):
|         """Test aggregating revenue by month."""
|         conn = setup.get_conn()
|         cursor = conn.cursor()
|
|         cursor.execute("""
|             SELECT strftime('%Y-%m', created_at) as month,
|                    SUM(revenue) as total_revenue,
|                    COUNT(*) as num_records
|             FROM tax_records
|             GROUP BY month
|             ORDER BY month
|         """)
|
|         results = cursor.fetchall()
|
|         if len(results) &gt; 0:
|             # Each result should have month, revenue, count
|             for row in results:
|                 assert len(row) == 3
|                 assert row[1] &gt; 0  # revenue should be positive
|                 assert row[2] &gt; 0  # count should be positive
|
|         conn.close()
|
|     def test_top_earners_query(self):
|         """Test querying top earners."""
|         conn = setup.get_conn()
|         cursor = conn.cursor()
|
|         cursor.execute("""
|             SELECT name,
|                    SUM(gross_income) as total_earned,
|                    COUNT(*) as projects
|             FROM people
|             GROUP BY name
```

```
            ORDER BY total_earned DESC
            LIMIT 5
        """)

        results = cursor.fetchall()

        if len(results) &gt; 0:
            # Results should be ordered by earnings (descending)
            for i in range(len(results) - 1):
                assert results[i][1] &gt;= results[i + 1][1]

        conn.close()

    def test_tax_rate_calculation(self):
        """Test calculating average tax rates."""
        conn = setup.get_conn()
        cursor = conn.cursor()

        cursor.execute("""
            SELECT tax_origin,
                   tax_option,
                   AVG(tax_amount * 100.0 / NULLIF(group_income, 0)) as avg_tax_rate,
                   COUNT(*) as count
            FROM tax_records
            WHERE group_income &gt; 0
            GROUP BY tax_origin, tax_option
        """)

        results = cursor.fetchall()

        for row in results:
            country, tax_type, avg_rate, count = row
            # Tax rate should be reasonable (0-50%)
            assert 0 &lt;= avg_rate &lt;= 50

        conn.close()

```

File 33: tests/test_edge_cases.py

```
"""
Edge case and boundary testing.

AI ASSISTANCE DISCLOSURE:
All tests in this file were generated with AI assistance (ChatGPT/Claude).
- Prompts used: "Create edge case tests for boundary values and invalid inputs"
- Prompts used: "Write tests for floating point precision and special characters"
- Prompts used: "Add tests for zero values, negative numbers, and extreme inputs"
- AI helped identify edge cases and create comprehensive boundary testing
"""

import pytest
from fastapi.testclient import TestClient
from api.main import app
from Logic.tax_calculator import calculate_tax, split_work_shares, calculate_profit
from Logic.validators import ValidationError, validate_work_shares, validate_tax_rate

client = TestClient(app)


class TestBoundaryValues:
    """Test boundary and extreme values."""

    def test_very_large_revenue(self):
        """Test handling of very large revenue values."""
        payload = {
            "num_people": 1,
            "revenue": 10000000,  # 10 million
            "costs": [1000000],
            "country": "US",
            "tax_type": "Individual",
            "people": [{"name": "Millionaire", "work_share": 1.0}]
        }

        response = client.post("/api/projects", json=payload)
        assert response.status_code == 201

    def test_very_small_revenue(self):
        """Test handling of very small revenue values."""
        payload = {
            "num_people": 1,
            "revenue": 0.01,  # 1 cent
            "costs": [0.001],
            "country": "US",
            "tax_type": "Individual",
            "people": [{"name": "Penny", "work_share": 1.0}]
        }

        response = client.post("/api/projects", json=payload)
        assert response.status_code == 201

    def test_zero_revenue(self):
        """Test handling of zero revenue."""
        payload = {
            "num_people": 1,
            "revenue": 0,
            "costs": [0],
            "country": "US",
            "tax_type": "Individual",
            "people": [{"name": "Zero", "work_share": 1.0}]
        }

        response = client.post("/api/projects", json=payload)
        # Should either accept or reject gracefully
        assert response.status_code in [201, 400, 422]

    def test_many_people(self):
        """Test handling of many people in a project."""
        num_people = 50
        payload = {
            "num_people": num_people,
            "revenue": 100000,
            "costs": [1000],
            "country": "US",
```

```
            "tax_type": "Individual",
            "people": [
                {"name": f"Person{i}", "work_share": 1.0/num_people}
                for i in range(num_people)
            ]
        }

        response = client.post("/api/projects", json=payload)
        assert response.status_code == 201

    def test_single_person_project(self):
        """Test single person project."""
        payload = {
            "num_people": 1,
            "revenue": 5000,
            "costs": [500],
            "country": "US",
            "tax_type": "Individual",
            "people": [{"name": "Solo", "work_share": 1.0}]
        }

        response = client.post("/api/projects", json=payload)
        assert response.status_code == 201


class TestInvalidInputs:
    """Test invalid input handling."""

    def test_negative_num_people(self):
        """Test negative number of people."""
        payload = {
            "num_people": -1,
            "revenue": 5000,
            "costs": [500],
            "country": "US",
            "tax_type": "Individual",
            "people": []
        }

        response = client.post("/api/projects", json=payload)
        assert response.status_code == 422  # Validation error

    def test_empty_people_list(self):
        """Test empty people list."""
        payload = {
            "num_people": 2,
            "revenue": 5000,
            "costs": [500],
            "country": "US",
            "tax_type": "Individual",
            "people": []
        }

        response = client.post("/api/projects", json=payload)
        assert response.status_code in [400, 422]

    def test_mismatched_num_people(self):
        """Test mismatch between num_people and actual people list."""
        payload = {
            "num_people": 5,
            "revenue": 5000,
            "costs": [500],
            "country": "US",
            "tax_type": "Individual",
            "people": [
                {"name": "Person1", "work_share": 0.5},
                {"name": "Person2", "work_share": 0.5}
            ]  # Only 2 people but num_people=5
        }

        response = client.post("/api/projects", json=payload)
        assert response.status_code in [400, 422]

    def test_invalid_country(self):
        """Test invalid country code."""
        payload = {
            "num_people": 1,
            "revenue": 5000,
```

```
            "costs": [500],
            "country": "",  # Empty country
            "tax_type": "Individual",
            "people": [{"name": "Test", "work_share": 1.0}]
        }

        response = client.post("/api/projects", json=payload)
        assert response.status_code in [400, 422]

    def test_invalid_tax_type(self):
        """Test invalid tax type."""
        payload = {
            "num_people": 1,
            "revenue": 5000,
            "costs": [500],
            "country": "US",
            "tax_type": "Corporate",  # Invalid type
            "people": [{"name": "Test", "work_share": 1.0}]
        }

        response = client.post("/api/projects", json=payload)
        assert response.status_code in [400, 422]


class TestWorkShareEdgeCases:
    """Test work share edge cases."""

    def test_work_shares_sum_to_zero(self):
        """Test work shares that sum to zero."""
        with pytest.raises(ValidationError):
            validate_work_shares([0.0, 0.0])

    def test_work_shares_sum_slightly_off(self):
        """Test work shares that sum to almost 1.0."""
        # Should pass with small tolerance
        validate_work_shares([0.33, 0.33, 0.34])  # Sums to 1.0
        validate_work_shares([0.333, 0.333, 0.334])  # Sums to 1.0

    def test_work_shares_very_unequal(self):
        """Test very unequal work shares."""
        payload = {
            "num_people": 2,
            "revenue": 10000,
            "costs": [1000],
            "country": "US",
            "tax_type": "Individual",
            "people": [
                {"name": "Leader", "work_share": 0.99},
                {"name": "Helper", "work_share": 0.01}
            ]
        }

        response = client.post("/api/projects", json=payload)
        assert response.status_code == 201

    def test_all_work_to_one_person(self):
        """Test when one person does all work."""
        profit = 10000
        work_shares = [1.0, 0.0, 0.0]

        distribution = split_work_shares(profit, work_shares)

        assert distribution[0] == 10000
        assert distribution[1] == 0
        assert distribution[2] == 0


class TestCostsEdgeCases:
    """Test costs edge cases."""

    def test_costs_exceed_revenue(self):
        """Test when costs exceed revenue (negative profit)."""
        payload = {
            "num_people": 1,
            "revenue": 5000,
            "costs": [6000],  # Costs &gt; revenue
            "country": "US",
            "tax_type": "Individual",
```

```
            "people": [{"name": "Loss", "work_share": 1.0}]
        }

        response = client.post("/api/projects", json=payload)
        # Should handle negative profit scenario
        assert response.status_code in [201, 400]

    def test_many_small_costs(self):
        """Test many small cost items."""
        costs = [0.01] * 1000  # 1000 costs of 1 cent each

        payload = {
            "num_people": 1,
            "revenue": 20,
            "costs": costs,
            "country": "US",
            "tax_type": "Individual",
            "people": [{"name": "Penny", "work_share": 1.0}]
        }

        response = client.post("/api/projects", json=payload)
        assert response.status_code == 201

    def test_zero_costs(self):
        """Test project with zero costs."""
        payload = {
            "num_people": 1,
            "revenue": 5000,
            "costs": [],  # No costs
            "country": "US",
            "tax_type": "Individual",
            "people": [{"name": "NoCost", "work_share": 1.0}]
        }

        response = client.post("/api/projects", json=payload)
        assert response.status_code == 201


class TestTaxCalculationEdgeCases:
    """Test tax calculation edge cases."""

    def test_income_at_bracket_boundary(self):
        """Test income exactly at bracket boundary."""
        brackets = [(10000, 0.10), (50000, 0.20)]
        tax = calculate_tax(10000, brackets)  # Exactly at boundary

        # 10000 at 10% = 1000
        assert abs(tax - 1000) &lt; 0.01

    def test_zero_income_tax(self):
        """Test tax on zero income."""
        brackets = [(10000, 0.10), (50000, 0.20)]
        tax = calculate_tax(0, brackets)

        assert tax == 0

    def test_income_in_top_bracket(self):
        """Test very high income in top bracket."""
        brackets = [(10000, 0.10), (50000, 0.20), (100000, 0.30)]
        tax = calculate_tax(500000, brackets)

        # Should use all brackets
        assert tax &gt; 0


class TestAPIEdgeCases:
    """Test API edge cases."""

    def test_get_nonexistent_record_various_ids(self):
        """Test getting records with various nonexistent IDs."""
        for record_id in [0, -1, 999999, "abc"]:
            response = client.get(f"/api/records/{record_id}")
            assert response.status_code in [404, 422]

    def test_concurrent_project_creation(self):
        """Test creating multiple projects rapidly."""
        payload = {
            "num_people": 1,
```

```
            "revenue": 5000,
            "costs": [500],
            "country": "US",
            "tax_type": "Individual",
            "people": [{"name": "Concurrent", "work_share": 1.0}]
        }

        # Create 10 projects rapidly
        responses = []
        for i in range(10):
            payload["people"][0]["name"] = f"Person{i}"
            response = client.post("/api/projects", json=payload)
            responses.append(response)

        # All should succeed
        for response in responses:
            assert response.status_code == 201

    def test_special_characters_in_names(self):
        """Test special characters in person names."""
        special_names = [
            "José García",
            "■■",
            "Müller",
            "O'Brien",
            "Jean-Pierre",
            "■■■■■■■■■"
        ]

        for name in special_names:
            payload = {
                "num_people": 1,
                "revenue": 5000,
                "costs": [500],
                "country": "US",
                "tax_type": "Individual",
                "people": [{"name": name, "work_share": 1.0}]
            }

            response = client.post("/api/projects", json=payload)
            assert response.status_code == 201


class TestCalculationAccuracy:
    """Test calculation accuracy."""

    def test_floating_point_precision(self):
        """Test floating point precision in calculations."""
        profit = 10000.33
        work_shares = [0.333, 0.333, 0.334]

        distribution = split_work_shares(profit, work_shares)

        # Sum should equal original (within floating point tolerance)
        total = sum(distribution)
        assert abs(total - profit) &lt; 0.01

    def test_profit_calculation_precision(self):
        """Test profit calculation precision."""
        revenue = 10000.99
        costs = [1000.33, 500.22, 300.11]

        profit = calculate_profit(revenue, costs)

        expected = revenue - sum(costs)
        assert abs(profit - expected) &lt; 0.01
```