

MoneySplit

Complete Source Code Appendix

Generated: October 05, 2025 at 11:25

Table of Contents

1. DB/__init__.py
2. DB/reset.py
3. DB/setup.py
4. Logic/ProgramBackend.py
5. Logic/__init__.py
6. Logic/forecasting.py
7. Logic/pdf_generator.py
8. Logic/tax_calculator.py
9. Logic/validators.py
10. Menus/__init__.py
11. Menus/data_menu.py
12. Menus/db_menu.py
13. Menus/project_menu.py
14. Menus/report_menu.py
15. Menus/tax_menu.py
16. __init__.py
17. __main__.py
18. api/__init__.py
19. api/main.py
20. api/models.py
21. frontend/package.json
22. frontend/src/App.css
23. frontend/src/App.tsx
24. frontend/src/api/client.ts
25. frontend/src/pages/Dashboard.tsx
26. frontend/src/pages/Projects.tsx
27. requirements.txt
28. tests/__init__.py
29. tests/conftest.py
30. tests/test_api.py
31. tests/test_backend_logic.py
32. tests/test_database.py
33. tests/test_edge_cases.py

File 1: DB/___init___.py

1 |

File 2: DB/reset.py

```
1  """
2  Maintenance tool for MoneySplit DB.
3  - Backup DB tables to CSV
4  - Reset DB with optional backup
5  - Restore DB from CSV backup
6  - Reset only tax brackets
7  - Export tax bracket CSV template
8  """
9
10 from DB import setup
11 import csv
12 import os
13 from datetime import datetime
14
15 BACKUP_DIR = "backups"
16 os.makedirs(BACKUP_DIR, exist_ok=True)
17
18 def backup():
19     """Backup tax_records and people to timestamped CSVs."""
20     conn = setup.get_conn()
21     cursor = conn.cursor()
22
23     timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
24
25     # Backup tax_records
26     cursor.execute("SELECT * FROM tax_records")
27     rows = cursor.fetchall()
28     headers = [d[0] for d in cursor.description]
29     tax_file = os.path.join(BACKUP_DIR, f"tax_records_{timestamp}.csv")
30     with open(tax_file, "w", newline="") as f:
31         writer = csv.writer(f)
32         writer.writerow(headers)
33         writer.writerows(rows)
34
35     # Backup people
36     cursor.execute("SELECT * FROM people")
37     rows = cursor.fetchall()
38     headers = [d[0] for d in cursor.description]
39     people_file = os.path.join(BACKUP_DIR, f"people_{timestamp}.csv")
40     with open(people_file, "w", newline="") as f:
41         writer = csv.writer(f)
42         writer.writerow(headers)
43         writer.writerows(rows)
44
45     conn.close()
46     print(f"■ Backup complete → {tax_file}, {people_file}")
47
48
49 def reset():
50     """Reset DB after optional backup."""
51     confirm = input("■■■ This will DELETE ALL DATA. Type 'RESET' to confirm: ").strip()
52     if confirm != "RESET":
53         print("■ Reset canceled.")
54         return
55
56     choice = input("Do you want to create a backup before reset? (y/n): ").strip().lower()
57     if choice == "y":
58         backup()
59
60     setup.reset_db()
61
62
63 def reset_tax_brackets():
64     """Reset only the tax_brackets table, with auto-backup."""
65     confirm = input("■■■ This will DELETE ALL TAX BRACKETS. Type 'RESET' to confirm: ").strip()
66     if confirm != "RESET":
67         print("■ Reset canceled.")
68         return
69
70     conn = setup.get_conn()
71     cursor = conn.cursor()
72
73     # Backup current brackets before deleting
74     cursor.execute("SELECT * FROM tax_brackets")
```

```

75 |     rows = cursor.fetchall()
76 |     headers = [d[0] for d in cursor.description]
77 |
78 |     if rows: # Only back up if something exists
79 |         timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
80 |         backup_file = os.path.join(BACKUP_DIR, f"tax_brackets_{timestamp}.csv")
81 |         with open(backup_file, "w", newline="") as f:
82 |             writer = csv.writer(f)
83 |             writer.writerow(headers)
84 |             writer.writerows(rows)
85 |         print(f"■ Tax brackets backed up → {backup_file}")
86 |     else:
87 |         print("■ No existing tax brackets found to back up.")
88 |
89 |     # Delete and reseed defaults
90 |     cursor.execute("DELETE FROM tax_brackets")
91 |     conn.commit()
92 |     conn.close()
93 |
94 |     setup.seed_default_brackets()
95 |     print("■ Tax brackets reset and reseeded with defaults.")
96 |
97 | def restore_tax_brackets():
98 |     """Restore tax brackets from a CSV backup."""
99 |     filepath = input("Enter path to tax_brackets CSV backup: ").strip()
100 |
101 |     if not os.path.exists(filepath):
102 |         print("■ File not found.")
103 |         return
104 |
105 |     conn = setup.get_conn()
106 |     cursor = conn.cursor()
107 |
108 |     # Reset table before restoring
109 |     cursor.execute("DELETE FROM tax_brackets")
110 |
111 |     with open(filepath, "r") as f:
112 |         reader = csv.reader(f)
113 |         headers = next(reader) # skip header
114 |         for row in reader:
115 |             cursor.execute(f"""
116 |                 INSERT INTO tax_brackets ({','.join(headers)})
117 |                 VALUES ({','.join(['?']*len(headers))})
118 |                 """, row)
119 |
120 |     conn.commit()
121 |     conn.close()
122 |     print(f"■ Tax brackets restored from {filepath}")
123 |
124 |
125 | def restore():
126 |     """Restore DB from CSV backups in /backups/"""
127 |     tax_file = input("Enter path to tax_records CSV: ").strip()
128 |     people_file = input("Enter path to people CSV: ").strip()
129 |
130 |     if not os.path.exists(tax_file) or not os.path.exists(people_file):
131 |         print("■ One or both CSV files not found.")
132 |         return
133 |
134 |     setup.reset_db() # start fresh
135 |
136 |     conn = setup.get_conn()
137 |     cursor = conn.cursor()
138 |
139 |     # Restore tax_records
140 |     with open(tax_file, "r") as f:
141 |         reader = csv.reader(f)
142 |         headers = next(reader) # skip header
143 |         for row in reader:
144 |             cursor.execute(f"""
145 |                 INSERT INTO tax_records ({','.join(headers)})
146 |                 VALUES ({','.join(['?']*len(headers))})
147 |                 """, row)
148 |
149 |     # Restore people
150 |     with open(people_file, "r") as f:
151 |         reader = csv.reader(f)
152 |         headers = next(reader)

```

```

153         for row in reader:
154             cursor.execute(f"""
155                 INSERT INTO people ({','.join(headers)})
156                 VALUES ({','.join(['?']*len(headers))})
157                 """, row)
158
159     conn.commit()
160     conn.close()
161     print("■ Database restored from CSV.")
162
163
164 def export_tax_template():
165     """Export a blank tax bracket CSV template into backups/ folder."""
166     timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
167     template_file = os.path.join(BACKUP_DIR, f"tax_bracket_template_{timestamp}.csv")
168
169     headers = ["income_limit", "rate"]
170
171     with open(template_file, "w", newline="") as f:
172         writer = csv.writer(f)
173         writer.writerow(headers)
174         # add one example row so user sees format
175         writer.writerow([10000, 0.10])
176         writer.writerow([40000, 0.20])
177         writer.writerow([float("inf"), 0.30])
178
179     print(f"■ Tax bracket template exported → {template_file}")
180     print("■ Edit this CSV and then upload it back through the Tax Menu (Upload from CSV).")
181
182 def main():
183     while True:
184         print("\n=== DB Maintenance Tool ===")
185         print("1. Backup database to CSV")
186         print("2. Reset database ■■")
187         print("3. Restore database from CSV")
188         print("4. Reset tax brackets ■■")
189         print("5. Restore tax brackets from backup")
190         print("6. Export tax bracket CSV template")
191         print("7. Back to main menu")
192
193         choice = input("Choose an option (1-7): ").strip()
194
195         if choice == "1":
196             backup()
197         elif choice == "2":
198             reset()
199         elif choice == "3":
200             restore()
201         elif choice == "4":
202             reset_tax_brackets()
203         elif choice == "5":
204             restore_tax_brackets()
205         elif choice == "6":
206             export_tax_template()
207         elif choice == "7":
208             print("■ Returning to main menu.")
209             break
210         else:
211             print("■ Invalid choice. Please enter 1-7.")
212

```

File 3: DB/setup.py

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

```
1  """
2  Database setup and operations module.
3
4  AI ASSISTANCE DISCLOSURE:
5  The database schema design received AI consultation (ChatGPT/Claude).
6  - Prompts used: "Design a normalized database schema for tax records with people"
7  - Prompts used: "Should I use foreign keys? What's the best relationship structure?"
8  - AI helped recommend the 3-table design (tax_records, people, tax_brackets)
9  - AI suggested adding foreign key constraints and proper indexing
10
11 The final schema decisions were mine, but AI provided guidance on normalization,
12 relationships, and SQL best practices.
13 """
14 import json
15 import sqlite3
16 import sys
17 import csv
18 import os
19 from datetime import datetime
20
21 # Editable fields at project level
22 ALLOWED_FIELDS = {
23     "num_people", "revenue", "total_costs",
24     "tax_origin", "tax_option"
25     # other fields are derived → recalculated automatically
26 }
27
28 # -----
29 # Init
30 # -----
31
32 def _pb():
33     """Return already-loaded ProgramBackend module without re-importing it."""
34     m = sys.modules.get("MoneySplit.Logic.ProgramBackend")
35     if m is None:
36         raise RuntimeError("ProgramBackend not loaded; run a calculation first (option 1).")
37     return m
38
39
40 def get_conn():
41     """Get a SQLite connection with foreign keys enabled."""
42     # Use test database when running tests
43     db_name = os.environ.get('TEST_DB', 'example.db') if os.environ.get('TESTING') else 'example.db'
44     conn = sqlite3.connect(db_name)
45     conn.execute("PRAGMA foreign_keys = ON;")
46     return conn
47
48
49 def init_db():
50     """Initialize tax_records, people, and tax_brackets tables."""
51     conn = get_conn()
52     cursor = conn.cursor()
53
54     cursor.execute("""
55         CREATE TABLE IF NOT EXISTS tax_records (
56             id INTEGER PRIMARY KEY AUTOINCREMENT,
57             num_people INTEGER,
58             revenue REAL,
59             total_costs REAL,
60             group_income REAL,
61             individual_income REAL,
62             tax_origin TEXT,
63             tax_option TEXT,
64             tax_amount REAL,
65             net_income_per_person REAL,
66             net_income_group REAL,
67             created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
68         )
69     """)
70
71     cursor.execute("""
72         CREATE TABLE IF NOT EXISTS people (
```

```

73         id INTEGER PRIMARY KEY AUTOINCREMENT,
74         record_id INTEGER,
75         name TEXT NOT NULL,
76         work_share REAL,
77         gross_income REAL,
78         tax_paid REAL,
79         net_income REAL,
80         FOREIGN KEY (record_id) REFERENCES tax_records(id) ON DELETE CASCADE
81     )
82     """
83
84     cursor.execute("""
85         CREATE TABLE IF NOT EXISTS tax_brackets (
86             id INTEGER PRIMARY KEY AUTOINCREMENT,
87             country TEXT NOT NULL,
88             tax_type TEXT NOT NULL,
89             income_limit REAL NOT NULL,
90             rate REAL NOT NULL
91         )
92     """)
93
94     conn.commit()
95     conn.close()
96
97
98     def seed_default_brackets():
99         """Insert default US & Spain brackets if table is empty."""
100         conn = get_conn()
101         cursor = conn.cursor()
102         cursor.execute("SELECT COUNT(*) FROM tax_brackets")
103         count = cursor.fetchone()[0]
104
105         if count > 0:
106             conn.close()
107             return
108
109         defaults = [
110             # US Individual
111             ("US", "Individual", 10275, 0.10),
112             ("US", "Individual", 41775, 0.12),
113             ("US", "Individual", 89075, 0.22),
114             ("US", "Individual", 170050, 0.24),
115             ("US", "Individual", 215950, 0.32),
116             ("US", "Individual", 539900, 0.35),
117             ("US", "Individual", float("inf"), 0.37),
118             # US Business
119             ("US", "Business", float("inf"), 0.21),
120             # Spain Individual
121             ("Spain", "Individual", 12450, 0.19),
122             ("Spain", "Individual", 20200, 0.24),
123             ("Spain", "Individual", 35200, 0.30),
124             ("Spain", "Individual", 60000, 0.37),
125             ("Spain", "Individual", 300000, 0.45),
126             ("Spain", "Individual", float("inf"), 0.47),
127             # Spain Business
128             ("Spain", "Business", float("inf"), 0.25),
129         ]
130
131         cursor.executemany("""
132             INSERT INTO tax_brackets (country, tax_type, income_limit, rate)
133             VALUES (?, ?, ?, ?)
134         """, defaults)
135
136         conn.commit()
137         conn.close()
138
139
140     init_db()
141     seed_default_brackets()
142
143     # -----
144     # CRUD for tax_records
145     # -----
146
147     def save_to_db():
148         """Save the current ProgramBackend calculation to tax_records."""
149         pb = _pb()
150

```



```

151 | tax_origin = "US" if pb.tax_origin == 1 else "Spain"
152 | tax_option = "Individual" if pb.tax_option == 1 else "Business"
153 |
154 | if pb.tax_option == 1: # individual
155 |     net_income_per_person = pb.individual_income - pb.tax
156 |     net_income_group = net_income_per_person * pb.num_people
157 | else: # business
158 |     net_income_group = pb.group_income - pb.tax
159 |     net_income_per_person = net_income_group / pb.num_people
160 |
161 | conn = get_conn()
162 | cursor = conn.cursor()
163 |
164 | cursor.execute("""
165 |     INSERT INTO tax_records (
166 |         num_people, revenue, total_costs, group_income, individual_income,
167 |         tax_origin, tax_option, tax_amount,
168 |         net_income_per_person, net_income_group
169 |     ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
170 | """, (
171 |     pb.num_people,
172 |     pb.revenue,
173 |     pb.total_costs,
174 |     pb.group_income,
175 |     pb.individual_income,
176 |     tax_origin,
177 |     tax_option,
178 |     pb.tax,
179 |     net_income_per_person,
180 |     net_income_group
181 | ))
182 |
183 | record_id = cursor.lastrowid
184 | conn.commit()
185 | conn.close()
186 | return record_id
187 |
188 |
189 | def fetch_last_records(n=5):
190 |     conn = get_conn()
191 |     cursor = conn.cursor()
192 |     cursor.execute("""
193 |         SELECT id, tax_origin, tax_option,
194 |             revenue, total_costs,
195 |             tax_amount, net_income_group, net_income_per_person, created_at,
196 |             num_people, group_income, individual_income
197 |         FROM tax_records
198 |         ORDER BY created_at DESC
199 |         LIMIT ?
200 |     """, (n,))
201 |     rows = cursor.fetchall()
202 |     conn.close()
203 |     return rows
204 |
205 |
206 | def get_record_by_id(record_id: int):
207 |     conn = get_conn()
208 |     cursor = conn.cursor()
209 |     cursor.execute("""
210 |         SELECT id, tax_origin, tax_option,
211 |             revenue, total_costs,
212 |             tax_amount, net_income_group, net_income_per_person, created_at,
213 |             num_people, group_income, individual_income
214 |         FROM tax_records
215 |         WHERE id = ?
216 |     """, (record_id,))
217 |     row = cursor.fetchone()
218 |     conn.close()
219 |     return row
220 |
221 |
222 | def delete_record(record_id: int):
223 |     conn = get_conn()
224 |     cursor = conn.cursor()
225 |     cursor.execute("DELETE FROM tax_records WHERE id = ?", (record_id,))
226 |     conn.commit()
227 |     conn.close()
228 |     print(f"■■■ Record {record_id} and linked people deleted.")

```

```

229 |
230 |
231 | def update_record(record_id: int, field: str, new_value):
232 |     """Update a record by ID. Only base fields can be edited; derived fields recalculated."""
233 |     if field not in ALLOWED_FIELDS:
234 |         raise ValueError(f"Invalid field: {field}. Allowed: {'', '.join(sorted(ALLOWED_FIELDS))}")
235 |
236 |     conn = get_conn()
237 |     cursor = conn.cursor()
238 |     cursor.execute(f"UPDATE tax_records SET {field} = ? WHERE id = ?", (new_value, record_id))
239 |
240 |     # fetch values for recalculation
241 |     cursor.execute("""
242 |         SELECT num_people, revenue, total_costs, tax_origin, tax_option
243 |         FROM tax_records WHERE id = ?
244 |     """, (record_id,))
245 |     row = cursor.fetchone()
246 |
247 |     if row:
248 |         num_people, revenue, total_costs, origin, option = row
249 |         revenue, total_costs, num_people = float(revenue), float(total_costs), int(num_people)
250 |         income = revenue - total_costs
251 |         group_income = income
252 |         individual_income = income / num_people if num_people > 0 else 0
253 |
254 |         # Calculate tax using DB-driven tax brackets
255 |         if option == "Individual":
256 |             tax = calculate_tax_from_db(individual_income, origin, option)
257 |         else:
258 |             tax = calculate_tax_from_db(group_income, origin, option)
259 |
260 |         if option == "Individual":
261 |             net_income_per_person = individual_income - tax
262 |             net_income_group = net_income_per_person * num_people
263 |         else:
264 |             net_income_group = group_income - tax
265 |             net_income_per_person = net_income_group / num_people if num_people > 0 else 0
266 |
267 |         cursor.execute("""
268 |             UPDATE tax_records
269 |             SET group_income=?, individual_income=?, tax_amount=?,
270 |                 net_income_per_person=?, net_income_group=?
271 |             WHERE id=?
272 |             """, (group_income, individual_income, tax, net_income_per_person, net_income_group, record_id))
273 |
274 |     conn.commit()
275 |     conn.close()
276 |     print(f"■ Record {record_id} updated and recalculated ({field} → {new_value})")
277 |
278 | # -----
279 | # CRUD for people
280 | # -----
281 |
282 | def add_person(record_id: int, name: str, work_share: float,
283 |                gross_income: float, tax_paid: float, net_income: float):
284 |     conn = get_conn()
285 |     cursor = conn.cursor()
286 |     cursor.execute("""
287 |         INSERT INTO people (record_id, name, work_share, gross_income, tax_paid, net_income)
288 |         VALUES (?, ?, ?, ?, ?, ?)
289 |     """, (record_id, name, work_share, gross_income, tax_paid, net_income))
290 |     person_id = cursor.lastrowid
291 |     conn.commit()
292 |     conn.close()
293 |     print(f"■ Added person {name} (ID: {person_id}) to record {record_id}.")
294 |     return person_id
295 |
296 |
297 | def fetch_people_by_record(record_id: int):
298 |     conn = get_conn()
299 |     cursor = conn.cursor()
300 |     cursor.execute("""
301 |         SELECT id, name, work_share, gross_income, tax_paid, net_income
302 |         FROM people
303 |         WHERE record_id = ?
304 |     """, (record_id,))
305 |     rows = cursor.fetchall()
306 |     conn.close()

```

```

307 |         return rows
308 |
309 |
310 | def delete_person(person_id: int):
311 |     conn = get_conn()
312 |     cursor = conn.cursor()
313 |     cursor.execute("SELECT name FROM people WHERE id=?", (person_id,))
314 |     row = cursor.fetchone()
315 |     if not row:
316 |         print(f"■ No person found with ID {person_id}.")
317 |         conn.close()
318 |         return
319 |     name = row[0]
320 |     cursor.execute("DELETE FROM people WHERE id=?", (person_id,))
321 |     conn.commit()
322 |     conn.close()
323 |     print(f"■■ Deleted person {person_id} ({name}).")
324 |
325 |
326 | def fetch_records_by_person(name: str):
327 |     conn = get_conn()
328 |     cursor = conn.cursor()
329 |     cursor.execute("""
330 |         SELECT p.id, p.record_id, p.name, p.work_share,
331 |                p.gross_income, p.tax_paid, p.net_income, t.created_at
332 |         FROM people p
333 |         JOIN tax_records t ON p.record_id = t.id
334 |         WHERE LOWER(p.name) = LOWER(?)
335 |         ORDER BY t.created_at DESC
336 |         """, (name,))
337 |     rows = cursor.fetchall()
338 |     conn.close()
339 |     return rows
340 |
341 | # -----
342 | # DB Maintenance
343 | # -----
344 |
345 | def reset_db():
346 |     """■■ Drop and recreate all tables. Use for testing only."""
347 |     choice = input("Do you want to create a backup before reset? (y/n): ").strip().lower()
348 |     if choice == "y":
349 |         backup_db_to_csv()
350 |
351 |     conn = get_conn()
352 |     cursor = conn.cursor()
353 |
354 |     cursor.execute("DROP TABLE IF EXISTS people")
355 |     cursor.execute("DROP TABLE IF EXISTS tax_records")
356 |     cursor.execute("DROP TABLE IF EXISTS tax_brackets")
357 |
358 |     conn.commit()
359 |     conn.close()
360 |
361 |     print("■■ All tables dropped. Reinitializing...")
362 |     init_db()
363 |     seed_default_brackets()
364 |     print("■ Database reset complete.")
365 |
366 |
367 | def backup_db_to_csv():
368 |     """Backup tax_records and people tables into CSV files."""
369 |     conn = get_conn()
370 |     cursor = conn.cursor()
371 |
372 |     timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
373 |
374 |     # Backup tax_records
375 |     cursor.execute("SELECT * FROM tax_records")
376 |     rows = cursor.fetchall()
377 |     headers = [d[0] for d in cursor.description]
378 |     with open(f"backup_tax_records_{timestamp}.csv", "w", newline="") as f:
379 |         writer = csv.writer(f)
380 |         writer.writerow(headers)
381 |         writer.writerows(rows)
382 |
383 |     # Backup people
384 |     cursor.execute("SELECT * FROM people")

```

```

385 |     rows = cursor.fetchall()
386 |     headers = [d[0] for d in cursor.description]
387 |     with open(f"backup_people_{timestamp}.csv", "w", newline="") as f:
388 |         writer = csv.writer(f)
389 |         writer.writerow(headers)
390 |         writer.writerows(rows)
391 |
392 |     conn.close()
393 |     print(f"■ Backup complete → backup_tax_records_{timestamp}.csv, backup_people_{timestamp}.csv")
394 |
395 |
396 | # -----
397 | # Tax brackets management
398 | # -----
399 |
400 | def calculate_tax_from_db(income: float, country: str, tax_type: str) -> float:
401 |     """Generic tax calculator that fetches brackets from DB."""
402 |     brackets = get_tax_brackets(country, tax_type)
403 |     if not brackets:
404 |         raise ValueError(f"No tax brackets found for {country} {tax_type}")
405 |
406 |     tax = 0
407 |     prev = 0
408 |     for limit, rate in brackets:
409 |         if income > limit:
410 |             tax += (limit - prev) * rate
411 |             prev = limit
412 |         else:
413 |             tax += (income - prev) * rate
414 |             break
415 |     return tax
416 |
417 |
418 | def get_tax_brackets(country: str, tax_type: str, include_id: bool = False):
419 |     conn = get_conn()
420 |     cursor = conn.cursor()
421 |     if include_id:
422 |         cursor.execute("""
423 |             SELECT id, income_limit, rate
424 |             FROM tax_brackets
425 |             WHERE country=? AND tax_type=?
426 |             ORDER BY income_limit ASC
427 |             """, (country, tax_type))
428 |     else:
429 |         cursor.execute("""
430 |             SELECT income_limit, rate
431 |             FROM tax_brackets
432 |             WHERE country=? AND tax_type=?
433 |             ORDER BY income_limit ASC
434 |             """, (country, tax_type))
435 |     rows = cursor.fetchall()
436 |     conn.close()
437 |     return rows
438 |
439 |
440 | def add_tax_brackets_from_csv(country: str, tax_type: str, filepath: str):
441 |     """Upload multiple tax brackets from a CSV file with columns: income_limit, rate"""
442 |     if not os.path.exists(filepath):
443 |         print(f"■ File not found.")
444 |         return
445 |
446 |     conn = get_conn()
447 |     cursor = conn.cursor()
448 |     added, skipped = 0, 0
449 |
450 |     with open(filepath, newline="") as csvfile:
451 |         reader = csv.DictReader(csvfile)
452 |         if "income_limit" not in reader.fieldnames or "rate" not in reader.fieldnames:
453 |             print(f"■ CSV must contain 'income_limit' and 'rate'.")
454 |             conn.close()
455 |             return
456 |
457 |         for row in reader:
458 |             try:
459 |                 limit_raw = row.get("income_limit", "").strip()
460 |                 rate_raw = row.get("rate", "").strip()
461 |
462 |                 if not limit_raw or not rate_raw:

```

```

463 |                 skipped += 1
464 |                 continue
465 |
466 |                 income_limit = float("inf") if limit_raw.lower() == "inf" else float(limit_raw)
467 |                 rate = float(rate_raw)
468 |
469 |                 cursor.execute("""
470 |                     INSERT INTO tax_brackets (country, tax_type, income_limit, rate)
471 |                     VALUES (?, ?, ?, ?)
472 |                     """, (country, tax_type, income_limit, rate))
473 |                 added += 1
474 |             except Exception as e:
475 |                 print(f"■ Skipping row {row}: {e}")
476 |                 skipped += 1
477 |
478 |         conn.commit()
479 |         conn.close()
480 |         print(f"■ Imported {added} brackets for {country} {tax_type} ({skipped} skipped).")
481 |
482 |
483 | def update_tax_bracket(bracket_id: int, field: str, new_value):
484 |     allowed = {"country", "tax_type", "income_limit", "rate"}
485 |     if field not in allowed:
486 |         raise ValueError(f"Invalid field. Allowed: {'', ' '.join(allowed)}")
487 |     conn = get_conn()
488 |     cursor = conn.cursor()
489 |     cursor.execute(f"UPDATE tax_brackets SET {field}=? WHERE id=?", (new_value, bracket_id))
490 |     conn.commit()
491 |     conn.close()
492 |     print(f"■ Bracket {bracket_id} updated: {field} → {new_value}")
493 |
494 |
495 | def delete_tax_bracket(bracket_id: int):
496 |     conn = get_conn()
497 |     cursor = conn.cursor()
498 |     cursor.execute("DELETE FROM tax_brackets WHERE id=?", (bracket_id,))
499 |     conn.commit()
500 |     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 > 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) -> float:
31 |     return calculate_tax_from_db(income, "US", "Individual")
32 |
33 | def us_business_tax(income: float) -> float:
34 |     return calculate_tax_from_db(income, "US", "Business")
35 |
36 | def spain_individual_tax(income: float) -> float:
37 |     return calculate_tax_from_db(income, "Spain", "Individual")
38 |
39 | def spain_business_tax(income: float) -> 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) -> 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 > 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}")
```

```

73 |     print(f"Net income per person: ${individual_income - tax:,.2f}")
74 |     print(f"Total tax for all people: ${tax * num_people:,.2f}")
75 | else:
76 |     print(f"Effective tax rate: {(tax / group_income) * 100:,.2f}%")
77 |     print(f"\nBusiness income: ${group_income:,.2f}")
78 |     print(f"Business tax: ${tax:,.2f}")
79 |     print(f"Net Business income: ${group_income - tax:,.2f}")
80 |     print(f"\nNet income per person: {(group_income - tax) / num_people:,.2f}")
81 |
82 |
83 | # =====
84 | # Collect People Information
85 | # =====
86 | print("\nNow enter details for each person:")
87 |
88 | record_id = setup.save_to_db() # Save project-level data first
89 | total_work_share = 0.0
90 |
91 | people_shares = []
92 | people_data = [] # Store person ID and details
93 |
94 | for i in range(num_people):
95 |     name = validators.safe_string_input(f"Name of person {i + 1}: ", f"Person {i + 1} name")
96 |
97 |     # If only 1 person, auto-assign work_share = 1.0
98 |     if num_people == 1:
99 |         work_share = 1.0
100 |         print(f"■ {name} is the only person - automatically assigned 100% work share")
101 |     else:
102 |         work_share = validators.safe_float_input(f"Work share for {name} (0.0-1.0): ", "Work share")
103 |         try:
104 |             work_share = validators.validate_work_share(work_share)
105 |         except validators.ValidationError as e:
106 |             print(f"■ {e}")
107 |             work_share = validators.safe_float_input(f"Work share for {name} (0.0-1.0): ", "Work share")
108 |             work_share = validators.validate_work_share(work_share)
109 |
110 |     total_work_share += work_share
111 |     people_shares.append(work_share)
112 |
113 |     if tax_option == 1: # Individual
114 |         gross_income = individual_income * work_share * num_people
115 |         tax_paid = tax * work_share
116 |         net_income = gross_income - tax_paid
117 |     else: # Business → distributed after company tax
118 |         gross_income = group_income * work_share
119 |         tax_paid = tax * work_share
120 |         net_income = gross_income - tax_paid
121 |
122 |     person_id = setup.add_person(record_id, name, work_share, gross_income, tax_paid, net_income)
123 |     people_data.append({
124 |         'person_id': person_id,
125 |         'name': name,
126 |         'work_share': work_share,
127 |         'gross_income': gross_income,
128 |         'tax_paid': tax_paid,
129 |         'net_income': net_income
130 |     })
131 |
132 | # After all people are added - validate total work shares
133 | if num_people > 1: # Only validate if more than 1 person
134 |     try:
135 |         validators.validate_work_shares(people_shares)
136 |         print("■ Work shares add up to 1.0")
137 |     except validators.ValidationError as e:
138 |         print(f"■■ Warning: {e}")
139 |
140 | # Display summary of created people
141 | print(f"\n■ Project Summary (Record ID: {record_id}):")
142 | print(f"{'Person ID':&lt;10} | {'Name':&lt;15} | {'Work Share':&gt;12} | {'Gross Income':&gt;15} | {'Tax Paid':
&gt;12} | ...")
143 | print("-" * 90)
144 | for person in people_data:
145 |     print(f"{person['person_id']:&lt;10} | {person['name']:&lt;15} | {person['work_share']:&gt;12.2%} | ${person[
'gross_...
146 |
147 | # Export record_id for use by project_menu
148 | LAST_RECORD_ID = record_id

```


File 5: Logic/___init___.py

1 |

File 6: Logic/forecasting.py

```
1  """
2  Forecasting and prediction module for MoneySplit.
3  Provides revenue forecasting, tax optimization, and trend analysis.
4
5  AI ASSISTANCE DISCLOSURE:
6  This forecasting module was developed with AI assistance (ChatGPT/Claude).
7  - Prompts used: "Create revenue forecasting using scikit-learn LinearRegression"
8  - Prompts used: "Add tax optimization analysis comparing Individual vs Business"
9  - Prompts used: "Implement trend analysis with seasonality detection"
10
11  ACADEMIC BACKGROUND:
12  The machine learning concepts (linear regression, time-series forecasting) were
13  learned in Machine Learning course (2024). AI assistance was primarily used for:
14  - Writing the Python implementation code
15  - Structuring the data preprocessing pipeline
16  - Integrating sklearn with the database layer
17  The underlying ML theory and approach were based on coursework knowledge.
18  """
19  import numpy as np
20  from datetime import datetime, timedelta
21  from sklearn.linear_model import LinearRegression
22  from sklearn.preprocessing import PolynomialFeatures
23  import sys
24  import os
25  sys.path.insert(0, os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
26  from DB import setup
27
28
29  def get_historical_data():
30      """Fetch historical revenue data grouped by month."""
31      conn = setup.get_conn()
32      cursor = conn.cursor()
33
34      cursor.execute("""
35          SELECT strftime('%Y-%m', created_at) as month,
36                 SUM(revenue) as total_revenue,
37                 SUM(total_costs) as total_costs,
38                 SUM(net_income_group) as total_profit,
39                 COUNT(*) as num_projects,
40                 AVG(tax_amount * 100.0 / NULLIF(group_income, 0)) as avg_tax_rate
41          FROM tax_records
42          GROUP BY month
43          ORDER BY month
44      """)
45      rows = cursor.fetchall()
46      conn.close()
47
48      return rows
49
50
51  def forecast_revenue(months_ahead=3):
52      """
53      Forecast revenue using an enhanced algorithm.
54
55      Uses polynomial regression when enough data is available (better for non-linear trends).
56      Falls back to linear regression for smaller datasets.
57      Includes moving average smoothing to reduce noise.
58      """
59      historical = get_historical_data()
60
61      if len(historical) < 2:
62          return {
63              'success': False,
64              'message': 'Not enough historical data (need at least 2 months)',
65              'predictions': [],
66              'explanation': 'Create more projects across different months to enable AI forecasting.'
67          }
68
69      # Prepare data
70      revenues = np.array([row[1] for row in historical])
71      num_projects = np.array([row[4] for row in historical])
72
73      # Apply moving average smoothing for datasets with enough data
74      # IMPORTANT: Use 'valid' mode to avoid data leakage (no look-ahead bias)
```

```

75 | if len(revenues) >= 4:
76 |     # 3-point moving average - 'valid' mode loses 2 data points but prevents leakage
77 |     smoothed_revenues = np.convolve(revenues, np.ones(3)/3, mode='valid')
78 |     y = smoothed_revenues
79 |     # Adjust indices to match smoothed data length (starts from index 1)
80 |     months_indices = np.array([i+1 for i in range(len(smoothed_revenues))]).reshape(-1, 1)
81 | else:
82 |     y = revenues
83 |     months_indices = np.array([i for i in range(len(revenues))]).reshape(-1, 1)
84 |
85 | # Choose model based on data size
86 | if len(historical) >= 6:
87 |     # Polynomial regression for better curve fitting
88 |     poly_features = PolynomialFeatures(degree=2)
89 |     X_poly = poly_features.fit_transform(months_indices)
90 |     model = LinearRegression()
91 |     model.fit(X_poly, y)
92 |     r2_score = model.score(X_poly, y)
93 |     model_type = "Polynomial (curved trend)"
94 | else:
95 |     # Linear regression for smaller datasets
96 |     model = LinearRegression()
97 |     model.fit(months_indices, y)
98 |     r2_score = model.score(months_indices, y)
99 |     model_type = "Linear (straight trend)"
100 |
101 | # Enhanced confidence scoring
102 | if r2_score >= 0.8:
103 |     confidence = "High"
104 |     confidence_desc = "Very reliable - strong pattern detected"
105 | elif r2_score >= 0.6:
106 |     confidence = "Medium-High"
107 |     confidence_desc = "Reliable - clear pattern with minor variation"
108 | elif r2_score >= 0.4:
109 |     confidence = "Medium"
110 |     confidence_desc = "Moderate - pattern exists but with some variability"
111 | elif r2_score >= 0.2:
112 |     confidence = "Low-Medium"
113 |     confidence_desc = "Less reliable - high variability in data"
114 | else:
115 |     confidence = "Low"
116 |     confidence_desc = "Unreliable - very inconsistent data"
117 |
118 | # Predict future months
119 | predictions = []
120 | last_month = historical[-1][0] # Format: YYYY-MM
121 | last_date = datetime.strptime(last_month, '%Y-%m')
122 |
123 | for i in range(1, months_ahead + 1):
124 |     future_index = len(historical) + i - 1
125 |
126 |     if len(historical) >= 6:
127 |         future_x_poly = poly_features.transform([[future_index]])
128 |         predicted_revenue = model.predict(future_x_poly)[0]
129 |     else:
130 |         predicted_revenue = model.predict([[future_index]])[0]
131 |
132 |     # Prevent negative predictions
133 |     predicted_revenue = max(0, predicted_revenue)
134 |
135 |     # Calculate next month
136 |     next_month = last_date + timedelta(days=30*i)
137 |     month_str = next_month.strftime('%B %Y') # e.g., "November 2025"
138 |
139 |     # Calculate confidence interval (95%)
140 |     std_error = np.std(y - model.predict(X_poly if len(historical) >= 6 else months_indices))
141 |     lower_bound = max(0, predicted_revenue - 1.96 * std_error)
142 |     upper_bound = predicted_revenue + 1.96 * std_error
143 |
144 |     predictions.append({
145 |         'month': month_str,
146 |         'revenue': predicted_revenue,
147 |         'confidence': confidence,
148 |         'lower_bound': lower_bound,
149 |         'upper_bound': upper_bound,
150 |         'range': f"${lower_bound:,.0f} - ${upper_bound:,.0f}"
151 |     })
152 |

```

```

153 |     # Calculate trend with clearer description
154 |     if len(historical) >= 6:
155 |         slope = revenues[-1] - revenues[0]
156 |     else:
157 |         slope = model.coef_[0] if len(model.coef_) == 1 else model.coef_[1]
158 |
159 |     if slope >= 100:
160 |         trend = "Strongly Increasing"
161 |     elif slope > 0:
162 |         trend = "Growing"
163 |     elif slope < -100:
164 |         trend = "Declining"
165 |     else:
166 |         trend = "Stable"
167 |
168 |     trend_strength = abs(slope / len(historical)) if len(historical) > 0 else 0
169 |
170 |     # Generate plain English explanation
171 |     avg_revenue = np.mean(revenues)
172 |     last_revenue = revenues[-1]
173 |     growth_rate = ((last_revenue - revenues[0]) / revenues[0] * 100) if revenues[0] > 0 else 0
174 |
175 |     explanation = f"""■ What this means:
176 |
177 | Your business has {len(historical)} months of data. The AI analyzed this and found a {trend.lower()} pattern.
178 |
179 | • Average monthly revenue: ${avg_revenue:,.0f}
180 | • Last month: ${last_revenue:,.0f}
181 | • Overall growth: {growth_rate:+.1f}%
182 | • Confidence level: {confidence} ({confidence_desc})
183 | • Prediction model: {model_type}
184 |
185 | ■ Next month prediction: ${predictions[0]['revenue']:,.0f}
186 | (Range: {predictions[0]['range']})
187 |
188 | ■ The AI is {r2_score*100:.0f}% confident in this pattern. {
189 |     "This is very reliable!" if r2_score > 0.7 else
190 |     "Add more data over time for better accuracy." if r2_score < 0.5 else
191 |     "Predictions are reasonably accurate."
192 | }"""
193 |
194 |     return {
195 |         'success': True,
196 |         'predictions': predictions,
197 |         'trend': trend,
198 |         'trend_strength': trend_strength,
199 |         'r2_score': r2_score,
200 |         'confidence': confidence,
201 |         'confidence_description': confidence_desc,
202 |         'historical_avg': avg_revenue,
203 |         'model_slope': slope,
204 |         'growth_rate': growth_rate,
205 |         'model_type': model_type,
206 |         'explanation': explanation.strip(),
207 |         'data_quality': "Excellent" if len(historical) >= 10 else "Good" if len(historical) >= 6 else
"Fair"
208 |     }
209 |
210 |
211 | def tax_optimization_analysis():
212 |     """Analyze tax strategies and provide optimization recommendations."""
213 |     conn = setup.get_conn()
214 |     cursor = conn.cursor()
215 |
216 |     # Compare Individual vs Business tax for recent projects
217 |     cursor.execute("""
218 |         SELECT tax_option,
219 |             AVG(tax_amount) as avg_tax,
220 |             AVG(tax_amount * 100.0 / NULLIF(group_income, 0)) as avg_rate,
221 |             COUNT(*) as count,
222 |             AVG(revenue) as avg_revenue
223 |         FROM tax_records
224 |         GROUP BY tax_option
225 |     """)
226 |     tax_comparison = cursor.fetchall()
227 |
228 |     # Get most efficient strategy per country
229 |     cursor.execute("""

```

```

230 |         SELECT tax_origin, tax_option,
231 |               AVG(tax_amount * 100.0 / NULLIF(group_income, 0)) as avg_rate
232 |         FROM tax_records
233 |         GROUP BY tax_origin, tax_option
234 |         ORDER BY tax_origin, avg_rate
235 |     """
236 |     country_analysis = cursor.fetchall()
237 |
238 |     # Get overall rate before closing connection
239 |     cursor.execute("""
240 |         SELECT AVG(tax_amount * 100.0 / NULLIF(group_income, 0)) as overall_rate
241 |         FROM tax_records
242 |     """)
243 |     overall_rate = cursor.fetchone()[0] or 0
244 |
245 |     conn.close()
246 |
247 |     recommendations = []
248 |
249 |     # Analyze tax comparison
250 |     if len(tax_comparison) >= 2:
251 |         individual = next((t for t in tax_comparison if t[0] == 'Individual'), None)
252 |         business = next((t for t in tax_comparison if t[0] == 'Business'), None)
253 |
254 |         if individual and business:
255 |             if individual[2] <= business[2]:
256 |                 savings = business[1] - individual[1]
257 |                 recommendations.append(
258 |                     f"Individual tax is {business[2] - individual[2]:.1f}% lower on average. "
259 |                     f"Consider Individual tax for future projects (potential savings: ${savings:,.2f} per
project)"
260 |                 )
261 |             else:
262 |                 savings = individual[1] - business[1]
263 |                 recommendations.append(
264 |                     f"Business tax is {individual[2] - business[2]:.1f}% lower on average. "
265 |                     f"Consider Business tax for future projects (potential savings: ${savings:,.2f} per project)"
266 |                 )
267 |
268 |     # Country-specific recommendations
269 |     country_dict = {}
270 |     for row in country_analysis:
271 |         country = row[0]
272 |         if country not in country_dict:
273 |             country_dict[country] = []
274 |             country_dict[country].append((row[1], row[2]))
275 |
276 |     for country, strategies in country_dict.items():
277 |         if len(strategies) >= 2:
278 |             best_strategy = min(strategies, key=lambda x: x[1])
279 |             recommendations.append(
280 |                 f"For {country}: {best_strategy[0]} tax offers best rate ({best_strategy[1]:.1f}%)"
281 |             )
282 |
283 |     # General recommendations
284 |     if overall_rate >= 25:
285 |         recommendations.append(
286 |             f"Your average tax rate is {overall_rate:.1f}%. Consider reviewing tax brackets and deductions."
287 |         )
288 |
289 |     return {
290 |         'tax_comparison': [
291 |             {
292 |                 'type': t[0],
293 |                 'avg_tax': t[1],
294 |                 'avg_rate': t[2],
295 |                 'count': t[3],
296 |                 'avg_revenue': t[4]
297 |             } for t in tax_comparison
298 |         ],
299 |         'recommendations': recommendations if recommendations else ['Your tax strategy appears optimal based on
current ...
300 |     ]
301 |
302 |
303 | def break_even_analysis(revenue, costs):
304 |     """Calculate break-even point and margin of safety."""
305 |     profit = revenue - costs

```

```

306 |     margin = (profit / revenue * 100) if revenue > 0 else 0
307 |
308 |     return {
309 |         'revenue': revenue,
310 |         'costs': costs,
311 |         'profit': profit,
312 |         'profit_margin': margin,
313 |         'break_even_revenue': costs,
314 |         'margin_of_safety': revenue - costs,
315 |         'margin_of_safety_pct': margin
316 |     }
317 |
318 |
319 | def trend_analysis():
320 |     """Analyze trends in revenue, costs, and profitability."""
321 |     historical = get_historical_data()
322 |
323 |     if len(historical) < 3:
324 |         return {
325 |             'success': False,
326 |             'message': 'Need at least 3 months of data for trend analysis'
327 |         }
328 |
329 |     months = [row[0] for row in historical]
330 |     revenues = [row[1] for row in historical]
331 |     costs = [row[2] for row in historical]
332 |     profits = [row[3] for row in historical]
333 |     num_projects = [row[4] for row in historical]
334 |
335 |     # Calculate trends
336 |     revenue_trend = "increasing" if revenues[-1] > revenues[0] else "decreasing"
337 |     cost_trend = "increasing" if costs[-1] > costs[0] else "decreasing"
338 |     profit_trend = "increasing" if profits[-1] > profits[0] else "decreasing"
339 |
340 |     # Calculate growth rates
341 |     revenue_growth = ((revenues[-1] - revenues[0]) / revenues[0] * 100) if revenues[0] > 0 else 0
342 |     cost_growth = ((costs[-1] - costs[0]) / costs[0] * 100) if costs[0] > 0 else 0
343 |     profit_growth = ((profits[-1] - profits[0]) / profits[0] * 100) if profits[0] > 0 else 0
344 |
345 |     # Seasonality detection (simple moving average)
346 |     if len(revenues) >= 6:
347 |         avg_revenue = np.mean(revenues)
348 |         volatility = np.std(revenues) / avg_revenue if avg_revenue > 0 else 0
349 |         seasonality = "High seasonality detected" if volatility > 0.3 else "Low seasonality" if volatility
350 |     else:
351 |         seasonality = "Insufficient data"
352 |
353 |     return {
354 |         'success': True,
355 |         'months_analyzed': len(historical),
356 |         'revenue_trend': revenue_trend,
357 |         'cost_trend': cost_trend,
358 |         'profit_trend': profit_trend,
359 |         'revenue_growth': revenue_growth,
360 |         'cost_growth': cost_growth,
361 |         'profit_growth': profit_growth,
362 |         'seasonality': seasonality,
363 |         'avg_projects_per_month': np.mean(num_projects),
364 |         'current_month_projects': num_projects[-1] if num_projects else 0,
365 |         'insights': generate_insights(revenue_trend, cost_trend, profit_trend, revenue_growth, cost_growth)
366 |     }
367 |
368 |
369 | def generate_insights(rev_trend, cost_trend, profit_trend, rev_growth, cost_growth):
370 |     """Generate actionable insights from trend analysis."""
371 |     insights = []
372 |
373 |     if rev_trend == "increasing" and cost_trend == "increasing":
374 |         if cost_growth > rev_growth:
375 |             insights.append("■■■ Warning: Costs are growing faster than revenue. Review cost management.")
376 |         else:
377 |             insights.append("■ Good: Revenue growth is outpacing cost growth.")
378 |
379 |     if rev_trend == "decreasing":
380 |         insights.append("■■■ Revenue is declining. Consider marketing efforts or new revenue streams.")
381 |
382 |     if profit_trend == "decreasing":

```

```

383 |         insights.append("■■ Profitability is declining. Focus on cost reduction or pricing optimization.")
384 |
385 |     if profit_trend == "increasing":
386 |         insights.append("■ Profitability is improving. Maintain current strategy.")
387 |
388 |     if abs(rev_growth) < 5:
389 |         insights.append("■ Revenue is relatively stable. Consider growth strategies.")
390 |
391 |     return insights if insights else ["■ Business metrics are stable."]
392 |
393 |
394 | def comprehensive_forecast():
395 |     """Generate comprehensive forecast with all insights."""
396 |     revenue_forecast = forecast_revenue(3)
397 |     tax_optimization = tax_optimization_analysis()
398 |     trends = trend_analysis()
399 |
400 |     recommendations = []
401 |
402 |     # Add revenue forecast recommendations
403 |     if revenue_forecast['success']:
404 |         if revenue_forecast['trend'] == 'increasing':
405 |             recommendations.append(
406 |                 f"■ Revenue is trending upward (+${revenue_forecast['trend_strength']:.0f}/month). "
407 |                 f"Expected next month: ${revenue_forecast['predictions'][0]['revenue']:.2f}"
408 |             )
409 |         else:
410 |             recommendations.append(
411 |                 f"■ Revenue is trending downward (-${revenue_forecast['trend_strength']:.0f}/month). "
412 |                 "Consider strategies to reverse this trend."
413 |             )
414 |
415 |         if revenue_forecast['confidence'] == 'Low':
416 |             recommendations.append(
417 |                 "■■ Low forecast confidence. Predictions may be unreliable due to data volatility."
418 |             )
419 |
420 |     # Add tax optimization recommendations
421 |     recommendations.extend(tax_optimization['recommendations'])
422 |
423 |     # Add trend insights
424 |     if trends['success']:
425 |         recommendations.extend(trends['insights'])
426 |
427 |     return {
428 |         'revenue_forecast': revenue_forecast,
429 |         'tax_optimization': tax_optimization,
430 |         'trend_analysis': trends,
431 |         'recommendations': recommendations
432 |     }
433 |

```

File 7: Logic/pdf_generator.py

```
1  """
2  PDF Report Generator for MoneySplit.
3  Creates professional PDF reports for projects, summaries, and forecasts.
4  """
5  from reportlab.lib import colors
6  from reportlab.lib.pagesizes import letter, A4
7  from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
8  from reportlab.lib.units import inch
9  from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, Paragraph, Spacer, PageBreak
10 from reportlab.platypus import Image as RImage
11 from reportlab.lib.enums import TA_CENTER, TA_RIGHT, TA_LEFT
12 from datetime import datetime
13 import os
14
15
16 def generate_project_pdf(record_data, people_data, filepath="reports/project_report.pdf"):
17     """Generate PDF report for a single project/record."""
18     os.makedirs("reports", exist_ok=True)
19
20     doc = SimpleDocTemplate(filepath, pagesize=letter)
21     story = []
22     styles = getSampleStyleSheet()
23
24     # Custom styles
25     title_style = ParagraphStyle(
26         'CustomTitle',
27         parent=styles['Heading1'],
28         fontSize=24,
29         textColor=colors.HexColor('#2c3e50'),
30         spaceAfter=30,
31         alignment=TA_CENTER
32     )
33
34     heading_style = ParagraphStyle(
35         'CustomHeading',
36         parent=styles['Heading2'],
37         fontSize=16,
38         textColor=colors.HexColor('#34495e'),
39         spaceAfter=12,
40     )
41
42     # Title
43     story.append(Paragraph("■ MoneySplit Project Report", title_style))
44     story.append(Spacer(1, 0.2*inch))
45
46     # Project Info
47     story.append(Paragraph("Project Information", heading_style))
48
49     project_info = [
50         ['Record ID:', str(record_data[0])],
51         ['Date:', record_data[8]],
52         ['Country:', record_data[1]],
53         ['Tax Type:', record_data[2]],
54         ['Number of People:', str(record_data[9])],
55     ]
56
57     project_table = Table(project_info, colWidths=[2*inch, 4*inch])
58     project_table.setStyle(TableStyle([
59         ('BACKGROUND', (0, 0), (0, -1), colors.HexColor('#ecf0f1')),
60         ('TEXTCOLOR', (0, 0), (-1, -1), colors.black),
61         ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
62         ('FONTNAME', (0, 0), (0, -1), 'Helvetica-Bold'),
63         ('FONTSIZE', (0, 0), (-1, -1), 10),
64         ('BOTTOMPADDING', (0, 0), (-1, -1), 8),
65         ('GRID', (0, 0), (-1, -1), 1, colors.grey),
66     ]))
67     story.append(project_table)
68     story.append(Spacer(1, 0.3*inch))
69
70     # Financial Summary
71     story.append(Paragraph("Financial Summary", heading_style))
72
73     financial_data = [
74         ['Item', 'Amount'],
```



```

75 |         ['Revenue', f"${record_data[3]:.2f}"],
76 |         ['Total Costs', f"${record_data[4]:.2f}"],
77 |         ['Group Income', f"${record_data[10]:.2f}"],
78 |         ['Tax Amount', f"${record_data[5]:.2f}"],
79 |         ['Net Income (Group)', f"${record_data[6]:.2f}"],
80 |         ['Net Income (Per Person)', f"${record_data[7]:.2f}"],
81 |     ]
82 |
83 |     financial_table = Table(financial_data, colWidths=[3*inch, 3*inch])
84 |     financial_table.setStyle(TableStyle([
85 |         ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#3498db')),
86 |         ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
87 |         ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
88 |         ('ALIGN', (1, 1), (1, -1), 'RIGHT'),
89 |         ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
90 |         ('FONTSIZE', (0, 0), (-1, -1), 10),
91 |         ('BOTTOMPADDING', (0, 0), (-1, -1), 8),
92 |         ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
93 |         ('GRID', (0, 0), (-1, -1), 1, colors.grey),
94 |         ('BACKGROUND', (0, -1), (-1, -1), colors.HexColor('#2ecc71')),
95 |         ('TEXTCOLOR', (0, -1), (-1, -1), colors.whitesmoke),
96 |         ('FONTNAME', (0, -1), (-1, -1), 'Helvetica-Bold'),
97 |     ]))
98 |     story.append(financial_table)
99 |     story.append(Spacer(1, 0.3*inch))
100 |
101 |     # Team Breakdown
102 |     story.append(Paragraph("Team Breakdown", heading_style))
103 |
104 |     team_data = [['Name', 'Work Share', 'Gross Income', 'Tax Paid', 'Net Income']]
105 |     for person in people_data:
106 |         team_data.append([
107 |             person[1], # name
108 |             f"{person[2]*100:.1f}%", # work_share
109 |             f"${person[3]:.2f}", # gross_income
110 |             f"${person[4]:.2f}", # tax_paid
111 |             f"${person[5]:.2f}", # net_income
112 |         ])
113 |
114 |     team_table = Table(team_data, colWidths=[1.5*inch, 1.2*inch, 1.3*inch, 1.3*inch, 1.3*inch])
115 |     team_table.setStyle(TableStyle([
116 |         ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#3498db')),
117 |         ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
118 |         ('ALIGN', (0, 0), (0, -1), 'LEFT'),
119 |         ('ALIGN', (1, 0), (-1, -1), 'CENTER'),
120 |         ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
121 |         ('FONTSIZE', (0, 0), (-1, -1), 9),
122 |         ('BOTTOMPADDING', (0, 0), (-1, -1), 6),
123 |         ('GRID', (0, 0), (-1, -1), 1, colors.grey),
124 |         ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white, colors.lightgrey]),
125 |     ]))
126 |     story.append(team_table)
127 |
128 |     # Footer
129 |     story.append(Spacer(1, 0.5*inch))
130 |     footer_text = f"Generated by MoneySplit on {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}}"
131 |     footer_style = ParagraphStyle('Footer', parent=styles['Normal'], fontSize=8, textColor=colors.grey,
alignment=TA_CEN...)
132 |     story.append(Paragraph(footer_text, footer_style))
133 |
134 |     # Build PDF
135 |     doc.build(story)
136 |     return filepath
137 |
138 |
139 | def generate_summary_pdf(records, stats, filepath="reports/summary_report.pdf"):
140 |     """Generate PDF summary report for all records."""
141 |     os.makedirs("reports", exist_ok=True)
142 |
143 |     doc = SimpleDocTemplate(filepath, pagesize=letter)
144 |     story = []
145 |     styles = getSampleStyleSheet()
146 |
147 |     # Custom styles
148 |     title_style = ParagraphStyle(
149 |         'CustomTitle',
150 |         parent=styles['Heading1'],
151 |         fontSize=24,

```

```

152 |         textColor=colors.HexColor('#2c3e50'),
153 |         spaceAfter=30,
154 |         alignment=TA_CENTER
155 |     )
156 |
157 |     heading_style = ParagraphStyle(
158 |         'CustomHeading',
159 |         parent=styles['Heading2'],
160 |         fontSize=16,
161 |         textColor=colors.HexColor('#34495e'),
162 |         spaceAfter=12,
163 |     )
164 |
165 |     # Title
166 |     story.append(Paragraph("■ MoneySplit Summary Report", title_style))
167 |     story.append(Spacer(1, 0.2*inch))
168 |
169 |     # Overall Statistics
170 |     story.append(Paragraph("Overall Statistics", heading_style))
171 |
172 |     stats_data = [
173 |         ['Metric', 'Value'],
174 |         ['Total Records', str(stats['total_records'])],
175 |         ['Total Revenue', f"${stats['total_revenue']:.2f}"],
176 |         ['Total Costs', f"${stats['total_costs']:.2f}"],
177 |         ['Total Tax Paid', f"${stats['total_tax']:.2f}"],
178 |         ['Total Net Income', f"${stats['total_net_income']:.2f}"],
179 |         ['Average Tax Rate', f"${stats['average_tax_rate']:.2f}%"],
180 |         ['Unique People', str(stats['unique_people'])],
181 |     ]
182 |
183 |     stats_table = Table(stats_data, colWidths=[3*inch, 3*inch])
184 |     stats_table.setStyle(TableStyle([
185 |         ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#3498db')),
186 |         ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
187 |         ('ALIGN', (0, 0), (0, -1), 'LEFT'),
188 |         ('ALIGN', (1, 0), (1, -1), 'RIGHT'),
189 |         ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
190 |         ('FONTSIZE', (0, 0), (-1, -1), 10),
191 |         ('BOTTOMPADDING', (0, 0), (-1, -1), 8),
192 |         ('GRID', (0, 0), (-1, -1), 1, colors.grey),
193 |         ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.beige, colors.white]),
194 |     ]))
195 |     story.append(stats_table)
196 |     story.append(Spacer(1, 0.4*inch))
197 |
198 |     # Recent Records
199 |     story.append(Paragraph(f"Recent Records (Last {len(records)})", heading_style))
200 |
201 |     records_data = [['ID', 'Date', 'Country', 'Tax Type', 'Revenue', 'Tax', 'Net Income']]
202 |     for r in records:
203 |         records_data.append([
204 |             str(r[0]),
205 |             r[8][:10], # date only
206 |             r[1],
207 |             r[2],
208 |             f"${r[3]:.0f}",
209 |             f"${r[5]:.0f}",
210 |             f"${r[6]:.0f}",
211 |         ])
212 |
213 |     records_table = Table(records_data, colWidths=[0.5*inch, 1*inch, 1*inch, 1.2*inch, 1.2*inch, 1*inch, 1.
214 | 2*inch])
215 |     records_table.setStyle(TableStyle([
216 |         ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#3498db')),
217 |         ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
218 |         ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
219 |         ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
220 |         ('FONTSIZE', (0, 0), (-1, -1), 8),
221 |         ('BOTTOMPADDING', (0, 0), (-1, -1), 6),
222 |         ('GRID', (0, 0), (-1, -1), 1, colors.grey),
223 |         ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.white, colors.lightgrey]),
224 |     ]))
225 |     story.append(records_table)
226 |
227 |     # Footer
228 |     story.append(Spacer(1, 0.5*inch))
229 |     footer_text = f"Generated by MoneySplit on {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"

```

```

229 |     footer_style = ParagraphStyle('Footer', parent=styles['Normal'], fontSize=8, textColor=colors.grey,
alignment=TA_CEN...
230 |     story.append(Paragraph(footer_text, footer_style))
231 |
232 |     # Build PDF
233 |     doc.build(story)
234 |     return filepath
235 |
236 |
237 | def generate_forecast_pdf(forecast_data, filepath="reports/forecast_report.pdf"):
238 |     """Generate PDF forecast report with predictions."""
239 |     os.makedirs("reports", exist_ok=True)
240 |
241 |     doc = SimpleDocTemplate(filepath, pagesize=letter)
242 |     story = []
243 |     styles = getSampleStyleSheet()
244 |
245 |     # Custom styles
246 |     title_style = ParagraphStyle(
247 |         'CustomTitle',
248 |         parent=styles['Heading1'],
249 |         fontSize=24,
250 |         textColor=colors.HexColor('#2c3e50'),
251 |         spaceAfter=30,
252 |         alignment=TA_CENTER
253 |     )
254 |
255 |     heading_style = ParagraphStyle(
256 |         'CustomHeading',
257 |         parent=styles['Heading2'],
258 |         fontSize=16,
259 |         textColor=colors.HexColor('#34495e'),
260 |         spaceAfter=12,
261 |     )
262 |
263 |     # Title
264 |     story.append(Paragraph("■ MoneySplit Forecast Report", title_style))
265 |     story.append(Spacer(1, 0.2*inch))
266 |
267 |     # Predictions
268 |     story.append(Paragraph("Revenue Predictions (Next 3 Months)", heading_style))
269 |
270 |     predictions_data = [['Month', 'Predicted Revenue', 'Confidence']]
271 |     for pred in forecast_data['predictions']:
272 |         predictions_data.append([
273 |             pred['month'],
274 |             f"${pred['revenue']:.2f}",
275 |             pred['confidence']
276 |         ])
277 |
278 |     pred_table = Table(predictions_data, colWidths=[2*inch, 2.5*inch, 2*inch])
279 |     pred_table.setStyle(TableStyle([
280 |         ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#3498db')),
281 |         ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
282 |         ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
283 |         ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
284 |         ('FONTSIZE', (0, 0), (-1, -1), 10),
285 |         ('BOTTOMPADDING', (0, 0), (-1, -1), 8),
286 |         ('GRID', (0, 0), (-1, -1), 1, colors.grey),
287 |         ('ROWBACKGROUNDS', (0, 1), (-1, -1), [colors.lightblue, colors.white]),
288 |     ]))
289 |     story.append(pred_table)
290 |     story.append(Spacer(1, 0.3*inch))
291 |
292 |     # Recommendations
293 |     if 'recommendations' in forecast_data:
294 |         story.append(Paragraph("■ Recommendations", heading_style))
295 |         for rec in forecast_data['recommendations']:
296 |             story.append(Paragraph(f"• {rec}", styles['BodyText']))
297 |             story.append(Spacer(1, 0.1*inch))
298 |
299 |     # Footer
300 |     story.append(Spacer(1, 0.5*inch))
301 |     footer_text = f"Generated by MoneySplit on {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}}"
302 |     footer_style = ParagraphStyle('Footer', parent=styles['Normal'], fontSize=8, textColor=colors.grey,
alignment=TA_CEN...
303 |     story.append(Paragraph(footer_text, footer_style))
304 |

```

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

File 8: Logic/tax_calculator.py

```
1  """
2  Tax calculation module - testable functions without side effects.
3  """
4
5  from DB import setup
6
7
8  def calculate_tax(income: float, tax_brackets: list[tuple[float, float]]) -> float:
9      """
10     Calculate tax using progressive tax brackets.
11
12     Args:
13         income: The income amount to calculate tax for
14         tax_brackets: List of (limit, rate) tuples
15
16     Returns:
17         Total tax amount
18     """
19     tax = 0
20     prev = 0
21     for limit, rate in tax_brackets:
22         if income > limit:
23             tax += (limit - prev) * rate
24             prev = limit
25         else:
26             tax += (income - prev) * rate
27             break
28     return tax
29
30
31 def calculate_tax_from_db(income: float, country: str, tax_type: str) -> float:
32     """
33     Calculate tax by fetching brackets from database.
34
35     Args:
36         income: The income amount
37         country: Country name (e.g., "US", "Spain")
38         tax_type: Tax type ("Individual" or "Business")
39
40     Returns:
41         Total tax amount
42
43     Raises:
44         ValueError: If no tax brackets found for country/type
45     """
46     brackets = setup.get_tax_brackets(country, tax_type)
47     if not brackets:
48         raise ValueError(f"No tax brackets found for {country} {tax_type}")
49
50     return calculate_tax(income, brackets)
51
52
53 def split_work_shares(total_amount: float, work_shares: list[float]) -> list[float]:
54     """
55     Distribute total amount based on work shares.
56
57     Args:
58         total_amount: Total amount to distribute
59         work_shares: List of work share percentages (should sum to 1.0)
60
61     Returns:
62         List of distributed amounts
63     """
64     return [total_amount * share for share in work_shares]
65
66
67 def calculate_profit(revenue: float, costs: list[float]) -> float:
68     """
69     Calculate profit from revenue and costs.
70
71     Args:
72         revenue: Total revenue
73         costs: List of cost amounts
74     """
```

```
75 | Returns:
76 |     Profit amount (revenue - total costs)
77 |     """
78 |     return revenue - sum(costs)
79 |
```

File 9: Logic/validators.py

```
1  """
2  Input validation utilities for MoneySplit.
3  """
4
5  class ValidationError(Exception):
6      """Custom exception for validation errors."""
7      pass
8
9
10 def validate_positive_number(value: float, field_name: str) -> float:
11     """Validate that a number is positive."""
12     if value < 0:
13         raise ValidationError(f"{field_name} must be positive, got {value}")
14     return value
15
16
17 def validate_work_shares(shares: list[float]) -> None:
18     """Validate that work shares sum to 1.0 (with small tolerance)."""
19     total = sum(shares)
20     if abs(total - 1.0) > 0.01:
21         raise ValidationError(f"Work shares must sum to 1.0, got {total:.2f}")
22
23
24 def validate_work_share(share: float) -> float:
25     """Validate a single work share is between 0 and 1."""
26     if not 0 <= share <= 1:
27         raise ValidationError(f"Work share must be between 0 and 1, got {share}")
28     return share
29
30
31 def validate_non_empty_string(value: str, field_name: str) -> str:
32     """Validate that a string is not empty."""
33     value = value.strip()
34     if not value:
35         raise ValidationError(f"{field_name} cannot be empty")
36     return value
37
38
39 def validate_country(country: str) -> str:
40     """Validate country name (allows any non-empty string)."""
41     country = country.strip()
42     if not country:
43         raise ValidationError("Country cannot be empty")
44     return country
45
46
47 def validate_tax_type(tax_type: str) -> str:
48     """Validate tax type."""
49     valid_types = ["Individual", "Business"]
50     tax_type = tax_type.strip().title()
51     if tax_type not in valid_types:
52         raise ValidationError(f"Tax type must be one of {valid_types}, got '{tax_type}'")
53     return tax_type
54
55
56 def validate_tax_rate(rate: float) -> float:
57     """Validate tax rate is between 0 and 1."""
58     if not 0 <= rate <= 1:
59         raise ValidationError(f"Tax rate must be between 0 and 1, got {rate}")
60     return rate
61
62
63 def safe_float_input(prompt: str, field_name: str = "Value", allow_negative: bool = False) -> float:
64     """Safely get float input from user with validation."""
65     while True:
66         try:
67             value = float(input(prompt))
68             if not allow_negative:
69                 validate_positive_number(value, field_name)
70             return value
71         except ValueError:
72             print(f"■ Invalid number. Please enter a valid number for {field_name}.")
73         except ValidationError as e:
74             print(f"■ {e}")
```

```

75 |
76 |
77 | def safe_int_input(prompt: str, field_name: str = "Value", min_value: int = None, max_value: int = None) ->
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 < min_value:
83 |                 print(f"■ {field_name} must be at least {min_value}.")
84 |                 continue
85 |             if max_value is not None and value > 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") -> 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.csv"
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

```
1 | from DB import setup
2 | from Logic import validators
3 |
4 | def show_last_records(n=5):
5 |     print(f"\n=== Last {n} Saved Records ===")
6 |     records = setup.fetch_last_records(n)
7 |
8 |     if not records:
9 |         print("No records found.")
10 |         return
11 |
12 |     header = (
13 |         f"{'ID':<3} | {'Origin':<6} | {'Option':<10} | "
14 |         f"{'Revenue':>12} | {'Costs':>10} | {'Tax':>10} | "
15 |         f"{'Net Group':>12} | {'Net Person':>12} | {'Created At'}"
16 |     )
17 |     print(header)
18 |     print("-" * len(header))
19 |
20 |     for r in records:
21 |         id, origin, option, revenue, costs, tax, net_group, net_person, created = r
22 |         print(
23 |             f"{id:<3} | {origin:<6} | {option:<10} | "
24 |             f"{float(revenue):>12,.2f} | {float(costs):>10,.2f} | {float(tax):>10,.2f} | "
25 |             f"{float(net_group):>12,.2f} | {float(net_person):>12,.2f} | {created}"
26 |         )
27 |
28 |
29 | def show_people_for_record():
30 |     try:
31 |         record_id = int(input("Enter the ID of the record to view people: "))
32 |         people = setup.fetch_people_by_record(record_id)
33 |
34 |         if not people:
35 |             print(f"■ No people found for record {record_id}.")
36 |             return
37 |
38 |         print(f"\n=== People for Record {record_id} ===")
39 |         header = f"{'ID':<3} | {'Name':<10} | {'Work Share':>10} | {'Gross':>12} | {'Tax Paid':>12} | {'N...'"
40 |         print(header)
41 |         print("-" * len(header))
42 |
43 |         for p in people:
44 |             pid, name, work_share, gross, tax_paid, net_income = p
45 |             print(
46 |                 f"{pid:<3} | {name:<10} | {work_share:>10.2f} | "
47 |                 f"{gross:>12,.2f} | {tax_paid:>10,.2f} | {net_income:>12,.2f}"
48 |             )
49 |
50 |     except ValueError:
51 |         print("■ Invalid input. Please enter a number.")
52 |
53 |
54 | def delete_record_menu():
55 |     try:
56 |         record_id = int(input("Enter the ID of the record to delete: "))
57 |         record = setup.get_record_by_id(record_id)
58 |
59 |         if not record:
60 |             print(f"■ No record found with ID {record_id}.")
61 |             return
62 |
63 |         confirm = input(f"Are you sure you want to delete record {record_id} (and all linked people)? (y/n): ").
strip()....
64 |         if confirm == "y":
65 |             setup.delete_record(record_id)
66 |         else:
67 |             print("■ Deletion canceled.")
68 |
69 |     except ValueError:
70 |         print("■ Invalid ID. Please enter a number.")
71 |
72 |
```

```

73 | def update_record_menu():
74 |     try:
75 |         record_id = validators.safe_int_input("Enter the ID of the record to update: ", "Record ID", min_value=1)
76 |         record = setup.get_record_by_id(record_id)
77 |
78 |         if not record:
79 |             print(f"■ No record found with ID {record_id}.")
80 |             return
81 |
82 |         print("\nYou can update the following fields:")
83 |         for f in sorted(setup.ALLOWED_FIELDS):
84 |             print(f" - {f}")
85 |
86 |         field = validators.safe_string_input("\nEnter the field to update: ", "Field name")
87 |
88 |         if field not in setup.ALLOWED_FIELDS:
89 |             print(f"■ '{field}' is not editable. Allowed: {' '.join(sorted(setup.ALLOWED_FIELDS))}")
90 |             return
91 |
92 |         if field == "num_people":
93 |             new_value = validators.safe_int_input(f"Enter new value for {field}: ", field, min_value=1)
94 |         elif field in ["revenue", "total_costs"]:
95 |             new_value = validators.safe_float_input(f"Enter new value for {field}: ", field)
96 |         else:
97 |             new_value = validators.safe_string_input(f"Enter new value for {field}: ", field)
98 |
99 |         setup.update_record(record_id, field, new_value)
100 |
101 |     except validators.ValidationError as e:
102 |         print(f"■ {e}")
103 |
104 |
105 | def show_person_history():
106 |     name = input("Enter the person's name: ").strip()
107 |     records = setup.fetch_records_by_person(name)
108 |
109 |     if not records:
110 |         print(f"■ No records found for {name}.")
111 |         return
112 |
113 |     print(f"\n=== Records for {name} ===")
114 |     header = f"{'PersonID':&lt;8} | {'RecordID':&lt;8} | {'Work Share':&gt;10} | {'Gross':&gt;12} | {'Tax Paid':
&gt;10} | ..."
115 |     print(header)
116 |     print("-" * len(header))
117 |
118 |     total_gross = total_tax = total_net = 0
119 |
120 |     for r in records:
121 |         pid, record_id, pname, work_share, gross, tax_paid, net_income, created = r
122 |         print(
123 |             f"{pid:&lt;8} | {record_id:&lt;8} | {work_share:&gt;10.2f} | "
124 |             f"{gross:&gt;12,.2f} | {tax_paid:&gt;10,.2f} | {net_income:&gt;12,.2f} | {created}"
125 |         )
126 |         total_gross += gross
127 |         total_tax += tax_paid
128 |         total_net += net_income
129 |
130 |     print("\n--- Totals ---")
131 |     print(f"Total Gross: {total_gross:,.2f}")
132 |     print(f"Total Tax: {total_tax:,.2f}")
133 |     print(f"Total Net: {total_net:,.2f}")
134 |
135 |
136 | def update_person_menu():
137 |     try:
138 |         person_id = validators.safe_int_input("Enter the ID of the person to update: ", "Person ID", min_value=1)
139 |         print("\nYou can update the following fields:")
140 |         print(" - name")
141 |         print(" - work_share")
142 |
143 |         field = validators.safe_string_input("Enter the field to update: ", "Field name")
144 |
145 |         if field == "work_share":
146 |             new_value = validators.safe_float_input(f"Enter new value for {field} (0.0-1.0): ", field)
147 |             new_value = validators.validate_work_share(new_value)
148 |         elif field == "name":
149 |             new_value = validators.safe_string_input(f"Enter new value for {field}: ", field)

```

```

150 |         else:
151 |             print(f"■ Invalid field. Only 'name' and 'work_share' can be updated.")
152 |             return
153 |
154 |             setup.update_person(person_id, field, new_value)
155 |
156 | except validators.ValidationError as e:
157 |     print(f"■ {e}")
158 |
159 |
160 | def delete_person_menu():
161 |     try:
162 |         person_id = int(input("Enter the ID of the person to delete: "))
163 |         confirm = input(f"Are you sure you want to delete person {person_id}? (y/n): ").strip().lower()
164 |         if confirm == "y":
165 |             setup.delete_person(person_id)
166 |         else:
167 |             print("■ Deletion canceled.")
168 |     except ValueError:
169 |         print("■ Invalid input. Please enter a number.")
170 |
171 |
172 | def deduplicate_people_menu():
173 |     try:
174 |         record_id = int(input("Enter the record ID to deduplicate people: "))
175 |         setup.deduplicate_people(record_id)
176 |     except ValueError:
177 |         print("■ Invalid input. Please enter a number.")
178 |
179 |
180 | # --- Maintenance ---
181 | def reset_db_menu():
182 |     confirm = input("■■ This will DELETE ALL tax records and people. Type 'RESET' to confirm: ").strip()
183 |     if confirm == "RESET":
184 |         setup.reset_db()
185 |     else:
186 |         print("■ Reset canceled.")
187 |
188 |
189 | def reset_tax_brackets_menu():
190 |     confirm = input("■■ This will DELETE ALL tax brackets and restore defaults. Type 'RESET' to confirm: ").
strip()
191 |     if confirm == "RESET":
192 |         setup.reset_tax_brackets()
193 |     else:
194 |         print("■ Reset canceled.")
195 |
196 |
197 | def export_template_menu():
198 |     filepath = input("Enter filename for template (default: tax_template.csv): ").strip()
199 |     if not filepath:
200 |         filepath = "tax_template.csv"
201 |     setup.export_tax_template(filepath)
202 |
203 |
204 | def view_tax_brackets_menu():
205 |     country = input("Enter country (e.g. US, Spain): ").strip()
206 |     tax_type = input("Enter type (Individual/Business): ").strip().title()
207 |     rows = setup.get_tax_brackets(country, tax_type, include_id=True)
208 |     if not rows:
209 |         print("■ No brackets found.")
210 |     else:
211 |         print(f"\nBrackets for {country} {tax_type}:")
212 |         print(f"{'ID':<4} | {'Income Limit':>15} | {'Rate':>8}")
213 |         print("-" * 35)
214 |         for bid, limit, rate in rows:
215 |             limit_txt = "∞" if limit == float("inf") else f"{limit:,.0f}"
216 |             print(f"{bid:<4} | {limit_txt:>15} | {rate*100:>7.2f}%")
217 |
218 |
219 | # --- Advanced ---
220 | def clone_record_menu():
221 |     try:
222 |         record_id = int(input("Enter the ID of the record to clone: "))
223 |         setup.clone_record(record_id)
224 |     except ValueError:
225 |         print("■ Invalid input. Please enter a number.")
226 |

```

```

227 |
228 | def copy_people_menu():
229 |     try:
230 |         source_id = int(input("Enter source record ID: "))
231 |         target_id = int(input("Enter target record ID: "))
232 |         setup.copy_people(source_id, target_id)
233 |
234 |         # ■ Auto-update num_people in target
235 |         people = setup.fetch_people_by_record(target_id)
236 |         new_count = len(people)
237 |         setup.update_record(target_id, "num_people", new_count)
238 |
239 |         # ■ Run deduplication
240 |         removed = setup.deduplicate_people(target_id)
241 |         print(f"■ Target record {target_id} updated. num_people = {new_count}. "
242 |               f"Deduplicated {removed} duplicate(s).")
243 |
244 |     except ValueError:
245 |         print("■ Invalid input. Please enter numbers.")
246 |
247 |
248 | def advanced_options_menu():
249 |     while True:
250 |         print("\n=== Advanced DB Options ===")
251 |         print("1. Clone a record")
252 |         print("2. Copy people between records")
253 |         print("3. Back")
254 |
255 |         choice = input("Choose an option (1-3): ").strip()
256 |
257 |         if choice == "1":
258 |             clone_record_menu()
259 |         elif choice == "2":
260 |             copy_people_menu()
261 |         elif choice == "3":
262 |             break
263 |         else:
264 |             print("■ Invalid choice. Please enter 1-3.")
265 |
266 |
267 |     # --- Submenus ---
268 |     def search_records_menu():
269 |         print("\n=== Search Filters ===")
270 |         country = input("Country (leave blank to skip): ").strip() or None
271 |         tax_option = input("Tax option (Individual/Business, leave blank to skip): ").strip().title() or None
272 |         start_date = input("Start date (YYYY-MM-DD, leave blank to skip): ").strip() or None
273 |         end_date = input("End date (YYYY-MM-DD, leave blank to skip): ").strip() or None
274 |
275 |         rows = setup.search_records(country, tax_option, start_date, end_date)
276 |         if not rows:
277 |             print("■ No matching records found.")
278 |             return
279 |
280 |         print(f"\nFound {len(rows)} matching records:")
281 |         header = (
282 |             f"{'ID':<3} | {'Origin':<6} | {'Option':<10} | "
283 |             f"{'Revenue':>12} | {'Costs':>10} | {'Tax':>10} | "
284 |             f"{'Net Group':>12} | {'Net Person':>12} | {'Created At'}"
285 |         )
286 |         print(header)
287 |         print("-" * len(header))
288 |
289 |         for r in rows:
290 |             id, origin, option, revenue, costs, tax, net_group, net_person, created = r
291 |             print(
292 |                 f"{'id':<3} | {'origin':<6} | {'option':<10} | "
293 |                 f"{'float(revenue):>12,.2f} | {'float(costs):>10,.2f} | {'float(tax):>10,.2f} | "
294 |                 f"{'float(net_group):>12,.2f} | {'float(net_person):>12,.2f} | {'created'}"
295 |             )
296 |
297 |
298 |     def merge_records_menu():
299 |         try:
300 |             r1 = int(input("Enter the first record ID: "))
301 |             r2 = int(input("Enter the second record ID: "))
302 |             new_id = setup.merge_records(r1, r2)
303 |
304 |             # ■ Run deduplication

```

```

305         removed = setup.deduplicate_people(new_id)
306         print(f"■ Merged into record {new_id}. Deduplicated {removed} duplicate(s).")
307
308     except ValueError:
309         print("■ Invalid input. Please enter numbers.")
310
311 def records_menu():
312     while True:
313         print("\n=== Records Menu ■ ===")
314         print("1. View last 5 records")
315         print("2. Search records")
316         print("3. Clone record")
317         print("4. Merge records")
318         print("5. Update record by ID")
319         print("6. Delete record by ID")
320         print("7. Advanced options ■■")
321         print("8. Back")
322
323         choice = input("Choose an option (1-8): ").strip()
324         if choice == "1":
325             show_last_records(5)
326         elif choice == "2":
327             search_records_menu()
328         elif choice == "3":
329             clone_record_menu()
330         elif choice == "4":
331             merge_records_menu()
332         elif choice == "5":
333             update_record_menu()
334         elif choice == "6":
335             delete_record_menu()
336         elif choice == "7":
337             advanced_options_menu()
338         elif choice == "8":
339             break
340         else:
341             print("■ Invalid choice. Please enter 1-8.")
342
343
344 def people_menu():
345     while True:
346         print("\n=== People Menu ■ ===")
347         print("1. View people for a record")
348         print("2. View person history")
349         print("3. Update person by ID")
350         print("4. Delete person by ID")
351         print("5. Deduplicate people in record ■")
352         print("6. Back")
353
354         choice = input("Choose an option (1-6): ").strip()
355         if choice == "1":
356             show_people_for_record()
357         elif choice == "2":
358             show_person_history()
359         elif choice == "3":
360             update_person_menu()
361         elif choice == "4":
362             delete_person_menu()
363         elif choice == "5":
364             deduplicate_people_menu()
365         elif choice == "6":
366             break
367         else:
368             print("■ Invalid choice. Please enter 1-6.")
369
370
371 def maintenance_menu():
372     while True:
373         print("\n=== Maintenance Menu ■■ ===")
374         print("1. Reset database ■■■")
375         print("2. Reset tax brackets ■■■")
376         print("3. Export CSV template")
377         print("4. View tax brackets")
378         print("5. Global deduplication ■")
379         print("6. Back")
380
381         choice = input("Choose an option (1-6): ").strip()
382         if choice == "1":

```

```

383 |         reset_db_menu()
384 |     elif choice == "2":
385 |         reset_tax_brackets_menu()
386 |     elif choice == "3":
387 |         export_template_menu()
388 |     elif choice == "4":
389 |         view_tax_brackets_menu()
390 |     elif choice == "5":
391 |         setup.deduplicate_all_records()
392 |     elif choice == "6":
393 |         break
394 |     else:
395 |         print("■ Invalid choice. Please enter 1-6.")
396 |
397 |
398 |
399 | # --- Main ---
400 | def show_db_menu():
401 |     while True:
402 |         print("\n=== DB Menu ===")
403 |         print("1. Records ■")
404 |         print("2. People ■")
405 |         print("3. Maintenance ■■")
406 |         print("4. Back to main menu")
407 |
408 |         choice = input("Choose an option (1-4): ").strip()
409 |         if choice == "1":
410 |             records_menu()
411 |         elif choice == "2":
412 |             people_menu()
413 |         elif choice == "3":
414 |             maintenance_menu()
415 |         elif choice == "4":
416 |             break
417 |         else:
418 |             print("■ Invalid choice. Please enter 1-4.")
419 |

```


File 13: Menus/project_menu.py

```
1 | import sys, importlib
2 | from DB import setup
3 | from Menus import report_menu
4 |
5 |
6 | def run_new_project():
7 |     """Run a fresh MoneySplit calculation and save it to DB."""
8 |     importlib.invalidate_caches()
9 |     if "Logic.ProgramBackend" in sys.modules:
10 |         del sys.modules["Logic.ProgramBackend"]
11 |
12 |     # This import executes ProgramBackend.py (asks for inputs, runs calculation, and saves to DB)
13 |     pb = importlib.import_module("Logic.ProgramBackend")
14 |
15 |     # Get the record_id that was already saved in ProgramBackend
16 |     record_id = pb.LAST_RECORD_ID
17 |     print(f"\n■ Project results saved (record {record_id}).")
18 |     print("■ Calculation finished and stored in database.")
19 |
20 |     # Auto-report (summary + top contributors)
21 |     print("\n■ Auto-generated report:")
22 |     report_menu.show_report_menu(auto=True, record_id=record_id)
23 |
```

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} | {'net':&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():
```

```

71 | while True:
72 |     print("\n=== Reports ■ ===")
73 |     print("1. Summary Report")
74 |     print("2. Per-Person Report")
75 |     print("3. Record Statistics")
76 |     print("4. Back")
77 |
78 |     choice = input("Choose an option (1-4): ").strip()
79 |     if choice == "1":
80 |         summary_report()
81 |     elif choice == "2":
82 |         person_report()
83 |     elif choice == "3":
84 |         record_stats()
85 |     elif choice == "4":
86 |         break
87 |     else:
88 |         print("■ Invalid choice.")
89 |
90 | def export_to_csv(filename, headers, rows):
91 |     with open(filename, "w", newline="") as f:
92 |         writer = csv.writer(f)
93 |         writer.writerow(headers)
94 |         writer.writerows(rows)
95 |     print(f"■ Exported report → {filename}")
96 |
97 | def revenue_summary_report():
98 |     rows = setup.get_revenue_summary()
99 |     if not rows:
100 |         print("■ No data found.")
101 |         return
102 |
103 |     # Extract data
104 |     years = [row[0] for row in rows]
105 |     revenues = [row[1] for row in rows]
106 |     costs = [row[2] for row in rows]
107 |     net_incomes = [row[3] for row in rows]
108 |
109 |     # Create interactive chart
110 |     fig = make_subplots(
111 |         rows=2, cols=1,
112 |         subplot_titles=("Revenue & Costs by Year", "Net Income by Year"),
113 |         vertical_spacing=0.15
114 |     )
115 |
116 |     # Revenue and Costs
117 |     fig.add_trace(
118 |         go.Bar(name="Revenue", x=years, y=revenues, marker_color='rgb(55, 83, 109)'),
119 |         row=1, col=1
120 |     )
121 |     fig.add_trace(
122 |         go.Bar(name="Costs", x=years, y=costs, marker_color='rgb(219, 64, 82)'),
123 |         row=1, col=1
124 |     )
125 |
126 |     # Net Income
127 |     fig.add_trace(
128 |         go.Scatter(name="Net Income", x=years, y=net_incomes,
129 |             mode='lines+markers', marker_color='rgb(50, 171, 96)',
130 |             line=dict(width=3)),
131 |         row=2, col=1
132 |     )
133 |
134 |     fig.update_layout(
135 |         title_text="Revenue Summary by Year",
136 |         showlegend=True,
137 |         height=700
138 |     )
139 |     fig.update_xaxes(title_text="Year", row=2, col=1)
140 |     fig.update_yaxes(title_text="Amount ($)", row=1, col=1)
141 |     fig.update_yaxes(title_text="Net Income ($)", row=2, col=1)
142 |
143 |     # Save and open
144 |     filepath = "reports/revenue_summary.html"
145 |     os.makedirs("reports", exist_ok=True)
146 |     fig.write_html(filepath)
147 |     webbrowser.open('file://' + os.path.abspath(filepath))
148 |     print(f"■ Visualization opened in browser: {filepath}")

```

```

149 |
150 | # Also print text summary
151 | print("\n=== Revenue Summary (Text) ===")
152 | print(f"{'Year':<6} | {'Total Revenue':>15} | {'Total Costs':>15} | {'Net Income':>15}")
153 | print("-" * 60)
154 | for year, rev, cost, net in rows:
155 |     print(f"{year:<6} | {rev:>15.2f} | {cost:>15.2f} | {net:>15.2f}")
156 |
157 | choice = input("\nExport to CSV? (y/n): ").strip().lower()
158 | if choice == "y":
159 |     filename = input("Enter filename (default: report_revenue_summary.csv): ").strip()
160 |     if not filename:
161 |         filename = "report_revenue_summary.csv"
162 |     elif not filename.endswith(".csv"):
163 |         filename += ".csv"
164 |     headers = ["Year", "Total Revenue", "Total Costs", "Net Income"]
165 |     export_to_csv(filename, headers, rows)
166 |
167 | def top_people_report():
168 |     rows = setup.get_top_people()
169 |     if not rows:
170 |         print("■ No data found.")
171 |         return
172 |
173 |     # Extract data
174 |     names = [row[0] for row in rows]
175 |     gross_incomes = [row[1] for row in rows]
176 |     taxes_paid = [row[2] for row in rows]
177 |     net_incomes = [row[3] for row in rows]
178 |
179 |     # Create horizontal bar chart
180 |     fig = make_subplots(
181 |         rows=1, cols=2,
182 |         subplot_titles=("Net Income", "Gross Income vs Tax Paid"),
183 |         specs=[[{"type": "bar"}, {"type": "bar"}]]
184 |     )
185 |
186 |     # Net income chart
187 |     fig.add_trace(
188 |         go.Bar(name="Net Income", y=names, x=net_incomes, orientation='h',
189 |             marker_color='rgb(50, 171, 96)', text=net_incomes,
190 |             texttemplate='${text:,.0f}', textposition='outside'),
191 |         row=1, col=1
192 |     )
193 |
194 |     # Gross vs Tax chart
195 |     fig.add_trace(
196 |         go.Bar(name="Gross Income", y=names, x=gross_incomes, orientation='h',
197 |             marker_color='rgb(55, 83, 109)'),
198 |         row=1, col=2
199 |     )
200 |     fig.add_trace(
201 |         go.Bar(name="Tax Paid", y=names, x=taxes_paid, orientation='h',
202 |             marker_color='rgb(219, 64, 82)'),
203 |         row=1, col=2
204 |     )
205 |
206 |     fig.update_layout(
207 |         title_text="Top People by Net Income",
208 |         showlegend=True,
209 |         height=max(400, len(names) * 40),
210 |         barmode='group'
211 |     )
212 |     fig.update_xaxes(title_text="Net Income ($)", row=1, col=1)
213 |     fig.update_xaxes(title_text="Amount ($)", row=1, col=2)
214 |
215 |     # Save and open
216 |     filepath = "reports/top_people.html"
217 |     os.makedirs("reports", exist_ok=True)
218 |     fig.write_html(filepath)
219 |     webbrowser.open('file://' + os.path.abspath(filepath))
220 |     print(f"■ Visualization opened in browser: {filepath}")
221 |
222 |     # Also print text summary
223 |     headers = ["Name", "Total Gross", "Total Tax Paid", "Total Net Income"]
224 |     print("\n=== Top People by Net Income (Text) ===")
225 |     print(f"{headers[0]:<15} | {headers[1]:>15} | {headers[2]:>15} | {headers[3]:>15}")
226 |     print("-" * 70)

```

```

227 |     for name, gross, tax, net in rows:
228 |         print(f"{name:<15} | {gross:>15,.2f} | {tax:>15,.2f} | {net:>15,.2f}")
229 |
230 |     choice = input("\nExport to CSV? (y/n): ").strip().lower()
231 |     if choice == "y":
232 |         filename = input("Enter filename (default: report_top_people.csv): ").strip()
233 |         if not filename:
234 |             filename = "report_top_people.csv"
235 |         elif not filename.endswith(".csv"):
236 |             filename += ".csv"
237 |         export_to_csv(filename, headers, rows)
238 |
239 | def show_report_menu():
240 |     while True:
241 |         print("\n=== Reports Menu ■ ===")
242 |         print("1. Revenue summary by year")
243 |         print("2. Top people by net income")
244 |         print("3. Back")
245 |
246 |         choice = input("Choose an option (1-3): ").strip()
247 |         if choice == "1":
248 |             revenue_summary_report()
249 |         elif choice == "2":
250 |             top_people_report()
251 |         elif choice == "3":
252 |             break
253 |         else:
254 |             print("■ Invalid choice. Please enter 1-3.")
255 |
256 |
257 | def show_summary(record_id: int):
258 |     """Show summary for a given record."""
259 |     rec = setup.get_record_by_id(record_id)
260 |     if not rec:
261 |         print(f"■ Record {record_id} not found.")
262 |         return
263 |
264 |     rid, origin, option, revenue, costs, tax, net_group, net_person, created = rec
265 |     print(f"\n=== Summary for Record {rid} ({origin} {option}) ===")
266 |     print(f"Revenue:           {float(revenue):,.2f}")
267 |     print(f"Total Costs:           {float(costs):,.2f}")
268 |     print(f"Tax Paid:               {float(tax):,.2f}")
269 |     print(f"Net Group:              {float(net_group):,.2f}")
270 |     print(f"Net Per Person: {float(net_person):,.2f}")
271 |
272 |
273 | def show_top_contributors(record_id: int, top_n: int = 5):
274 |     """Show top contributors (by net income) for a record."""
275 |     people = setup.fetch_people_by_record(record_id)
276 |     if not people:
277 |         print(f"■ No people found for record {record_id}.")
278 |         return
279 |
280 |     ranked = sorted(people, key=lambda x: x[5], reverse=True) # net_income = idx 5
281 |     print(f"\n=== Top {min(top_n, len(ranked))} Contributors (by Net Income) ===")
282 |     for i, (pid, name, ws, gross, tax_paid, net) in enumerate(ranked[:top_n], start=1):
283 |         print(f"{i}. {name:<12} → Net {net:,.2f} (Gross {gross:,.2f}, Tax {tax_paid:,.2f})")
284 |
285 |
286 | def single_record_menu():
287 |     """Menu for single record reports."""
288 |     while True:
289 |         print("\n=== Single Record Reports ■ ===")
290 |         print("1. View summary for a record")
291 |         print("2. View top contributors for a record")
292 |         print("3. Export record to PDF ■")
293 |         print("4. Back")
294 |
295 |         choice = input("Choose an option (1-4): ").strip()
296 |
297 |         if choice == "1":
298 |             try:
299 |                 record_id = int(input("Enter record ID: "))
300 |                 show_summary(record_id)
301 |             except ValueError:
302 |                 print("■ Invalid input.")
303 |         elif choice == "2":
304 |             try:

```

```

305         record_id = int(input("Enter record ID: "))
306         show_top_contributors(record_id)
307     except ValueError:
308         print("■ Invalid input.")
309     elif choice == "3":
310         export_record_to_pdf()
311     elif choice == "4":
312         break
313     else:
314         print("■ Invalid choice. Please enter 1-4.")
315
316
317 def aggregate_reports_menu():
318     """Menu for aggregate reports across all data."""
319     while True:
320         print("\n=== Aggregate Reports ■ ===")
321         print("1. Revenue summary by year")
322         print("2. Top people (across all records)")
323         print("3. Tax strategy comparison")
324         print("4. Overall statistics")
325         print("5. Monthly trends ■")
326         print("6. Work distribution analysis ■")
327         print("7. Person performance timeline ■■")
328         print("8. Tax efficiency report ■")
329         print("9. Project profitability analysis ■")
330         print("10. Revenue forecasting & predictions ■")
331         print("11. Export summary to PDF ■")
332         print("12. Back")
333
334         choice = input("Choose an option (1-12): ").strip()
335
336         if choice == "1":
337             revenue_summary_report()
338         elif choice == "2":
339             top_people_report()
340         elif choice == "3":
341             tax_type_comparison_report()
342         elif choice == "4":
343             overall_statistics()
344         elif choice == "5":
345             monthly_trends_report()
346         elif choice == "6":
347             work_distribution_report()
348         elif choice == "7":
349             person_performance_timeline()
350         elif choice == "8":
351             tax_efficiency_report()
352         elif choice == "9":
353             project_profitability_report()
354         elif choice == "10":
355             show_forecast_report()
356         elif choice == "11":
357             export_summary_to_pdf()
358         elif choice == "12":
359             break
360         else:
361             print("■ Invalid choice. Please enter 1-12.")
362
363
364 def tax_type_comparison_report():
365     """Compare Individual vs Business tax strategies."""
366     conn = setup.get_conn()
367     cursor = conn.cursor()
368
369     cursor.execute("""
370         SELECT tax_origin, tax_option,
371                COUNT(*) as records,
372                AVG(revenue) as avg_revenue,
373                AVG(tax_amount) as avg_tax,
374                AVG(tax_amount * 100.0 / NULLIF(group_income, 0)) as avg_tax_rate,
375                SUM(net_income_group) as total_net
376         FROM tax_records
377         GROUP BY tax_origin, tax_option
378         ORDER BY tax_origin, tax_option
379     """)
380     rows = cursor.fetchall()
381     conn.close()
382

```

```

383 |     if not rows:
384 |         print("■ No data found.")
385 |         return
386 |
387 |     # Extract data
388 |     labels = [f"{row[0]} - {row[1]}" for row in rows]
389 |     avg_revenues = [row[3] for row in rows]
390 |     avg_taxes = [row[4] for row in rows]
391 |     avg_rates = [row[5] for row in rows]
392 |     total_nets = [row[6] for row in rows]
393 |
394 |     # Create comparison charts
395 |     fig = make_subplots(
396 |         rows=2, cols=2,
397 |         subplot_titles=("Average Tax Rate (%)", "Total Net Income",
398 |             "Average Revenue vs Tax", "Record Distribution"),
399 |         specs=[[{"type": "bar"}, {"type": "bar"}],
400 |             [{"type": "bar"}, {"type": "pie"}]]
401 |     )
402 |
403 |     # Avg tax rate
404 |     fig.add_trace(
405 |         go.Bar(x=labels, y=avg_rates, name="Avg Tax Rate",
406 |             marker_color='rgb(219, 64, 82)', text=avg_rates,
407 |             texttemplate='%{text:.2f}%', textposition='outside'),
408 |         row=1, col=1
409 |     )
410 |
411 |     # Total net income
412 |     fig.add_trace(
413 |         go.Bar(x=labels, y=total_nets, name="Total Net",
414 |             marker_color='rgb(50, 171, 96)', text=total_nets,
415 |             texttemplate='${text:,.0f}', textposition='outside'),
416 |         row=1, col=2
417 |     )
418 |
419 |     # Avg revenue vs tax
420 |     fig.add_trace(
421 |         go.Bar(x=labels, y=avg_revenues, name="Avg Revenue",
422 |             marker_color='rgb(55, 83, 109)'),
423 |         row=2, col=1
424 |     )
425 |     fig.add_trace(
426 |         go.Bar(x=labels, y=avg_taxes, name="Avg Tax",
427 |             marker_color='rgb(219, 64, 82)'),
428 |         row=2, col=1
429 |     )
430 |
431 |     # Record distribution pie
432 |     record_counts = [row[2] for row in rows]
433 |     fig.add_trace(
434 |         go.Pie(labels=labels, values=record_counts, textinfo='label+percent+value'),
435 |         row=2, col=2
436 |     )
437 |
438 |     fig.update_layout(
439 |         title_text="Tax Strategy Comparison",
440 |         showlegend=True,
441 |         height=800
442 |     )
443 |     fig.update_yaxes(title_text="Rate (%)", row=1, col=1)
444 |     fig.update_yaxes(title_text="Net Income ($)", row=1, col=2)
445 |     fig.update_yaxes(title_text="Amount ($)", row=2, col=1)
446 |
447 |     # Save and open
448 |     filepath = "reports/tax_strategy_comparison.html"
449 |     os.makedirs("reports", exist_ok=True)
450 |     fig.write_html(filepath)
451 |     webbrowser.open('file://' + os.path.abspath(filepath))
452 |     print(f"■ Visualization opened in browser: {filepath}")
453 |
454 |     # Also print text summary
455 |     print("\n== Tax Strategy Comparison (Text) ==")
456 |     print(f"{'Origin':&lt;8} | {'Strategy':&lt;10} | {'Records':&lt;7} | {'Avg Revenue':&gt;12} | {'Avg Tax':&gt;12} | {'Avg Rate':&gt;12} | {'Total Net':&gt;12}")
12} | {...
457 |     print("-" * 90)
458 |     for origin, option, cnt, avg_rev, avg_tax, avg_rate, total_net in rows:
459 |         print(f"{'origin':&lt;8} | {'option':&lt;10} | {'cnt':&lt;7} | {'avg_rev':&gt;12,.2f} | {'avg_tax':&gt;12,.2f} | {'avg_rate':&gt;12,.2f} | {'total_net':&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 |
28 |                    for i in range(n):
29 |                        limit = input(f"Bracket {i+1} income limit (number or 'inf'): ").strip()
30 |                        income_limit = float("inf") if limit.lower() == "inf" else validators.safe_float_input(
f"Re-ente...
31 |                        rate = validators.safe_float_input(f"Bracket {i+1} rate (0.0-1.0, e.g. 0.21): ", "Tax
rate")
32 |                        rate = validators.validate_tax_rate(rate)
33 |                        setup.add_tax_bracket(country, tax_type, income_limit, rate)
34 |                        print(f"■ Added {n} brackets for {country} {tax_type}")
35 |                except validators.ValidationError as e:
36 |                    print(f"■ {e}")
37 |
38 |            elif sub == "2": # CSV
39 |                try:
40 |                    country = validators.validate_country(input("Enter country (US/Spain): ").strip())
41 |                    tax_type = validators.validate_tax_type(input("Enter type (Individual/Business): ").strip())
42 |                    filepath = validators.safe_string_input("Enter path to CSV file: ", "File path")
43 |                    setup.add_tax_brackets_from_csv(country, tax_type, filepath)
44 |                except validators.ValidationError as e:
45 |                    print(f"■ {e}")
46 |
47 |            elif sub == "3":
48 |                continue
49 |
50 |        elif choice == "2": # Update
51 |            bracket_id = int(input("Enter bracket ID to update: "))
52 |            field = input("Which field (country, tax_type, income_limit, rate)? ").strip()
53 |            new_value = input("Enter new value: ").strip()
54 |            if field in ("income_limit", "rate") and new_value.lower() != "inf":
55 |                new_value = float(new_value)
56 |            elif new_value.lower() == "inf":
57 |                new_value = float("inf")
58 |            setup.update_tax_bracket(bracket_id, field, new_value)
59 |
60 |        elif choice == "3": # Delete
61 |            bracket_id = int(input("Enter bracket ID to delete: "))
62 |            setup.delete_tax_bracket(bracket_id)
63 |
64 |        elif choice == "4": # View
65 |            country = input("Enter country (e.g. US, Spain): ").strip()
66 |            tax_type = input("Enter type (Individual/Business): ").strip().title()
67 |            rows = setup.get_tax_brackets(country, tax_type, include_id=True)
68 |            if not rows:
69 |                print("■ No brackets found.")
70 |            else:
71 |                print(f"\nBrackets for {country} {tax_type}:")
72 |                print(f"{'ID':<4} | {'Income Limit':>15} | {'Rate':>8}")
```

```

72 |         print("-" * 35)
73 |         for bid, limit, rate in rows:
74 |             limit_txt = "∞" if limit == float("inf") else f"{limit:,.0f}"
75 |             print(f"{bid:<4} | {limit_txt:>15} | {rate*100:>7.2f}%")
76 |
77 |     elif choice == "5": # Back
78 |         break
79 |     else:
80 |         print("■ Invalid choice. Please enter 1-5.")
81 |
82 |
83 | def maintenance_menu():
84 |     while True:
85 |         print("\n=== Tax Maintenance ■■ ===")
86 |         print("1. Reset tax brackets ■■")
87 |         print("2. Export CSV template")
88 |         print("3. Back")
89 |
90 |         choice = input("Choose an option (1-3): ").strip()
91 |         if choice == "1":
92 |             setup.reset_tax_brackets()
93 |         elif choice == "2":
94 |             setup.export_tax_template()
95 |         elif choice == "3":
96 |             break
97 |         else:
98 |             print("■ Invalid choice. Please enter 1-3.")
99 |
100 |
101 | def show_tax_menu():
102 |     while True:
103 |         print("\n=== Tax Menu ===")
104 |         print("1. Manage Brackets ■")
105 |         print("2. Maintenance ■■")
106 |         print("3. Back to main menu")
107 |
108 |         choice = input("Choose an option (1-3): ").strip()
109 |         if choice == "1":
110 |             manage_brackets_menu()
111 |         elif choice == "2":
112 |             maintenance_menu()
113 |         elif choice == "3":
114 |             break
115 |         else:
116 |             print("■ Invalid choice. Please enter 1-3.")
117 |

```

File 16: __init__.py

1 |

File 17: __main__.py

```
1 | from Menus import project_menu, db_menu, tax_menu, report_menu
2 | from DB import reset as db_reset
3 | from DB import setup
4 |
5 | # Init DB + defaults
6 | setup.init_db()
7 | setup.seed_default_brackets()
8 |
9 | def main():
10 |     while True:
11 |         print("\n=== MoneySplit Main Menu ===")
12 |         print("1. New Project █")
13 |         print("2. Play with DB █")
14 |         print("3. Tax █")
15 |         print("4. Reports █")
16 |         print("5. DB Maintenance ███")
17 |         print("6. Exit █")
18 |
19 |         choice = input("Choose an option (1-6): ").strip()
20 |
21 |         if choice == "1":
22 |             project_menu.run_new_project()
23 |         elif choice == "2":
24 |             db_menu.show_db_menu()
25 |         elif choice == "3":
26 |             tax_menu.show_tax_menu()
27 |         elif choice == "4":
28 |             report_menu.show_report_menu()
29 |         elif choice == "5":
30 |             db_reset.main() # run the maintenance tool
31 |         elif choice == "6":
32 |             print("█ Exiting MoneySplit. Goodbye!")
33 |             break
34 |         else:
35 |             print("█ Invalid choice. Please enter 1-6.")
36 |
37 | if __name__ == "__main__":
38 |     main()
39 |
```

File 18: api/__init__.py

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

File 19: api/main.py

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

```
1  """
2  | MoneySplit FastAPI Application
3  | RESTful API for commission-based income splitting with tax calculations.
4  |
5  | AI ASSISTANCE DISCLOSURE:
6  | This API was developed with AI assistance (ChatGPT/Claude).
7  | - Prompts used: "Create FastAPI endpoints for CRUD operations on tax records"
8  | - Prompts used: "Add forecasting endpoints with scikit-learn integration"
9  | - Prompts used: "Generate PDF export endpoints using ReportLab"
10 | - AI helped structure the API following REST principles and FastAPI best practices
11 | """
12 | from fastapi import FastAPI, HTTPException, Query
13 | from fastapi.responses import HTMLResponse, FileResponse
14 | from fastapi.middleware.cors import CORSMiddleware
15 | from typing import List, Optional
16 | import sys
17 | import os
18 |
19 | # Add parent directory to path for imports
20 | sys.path.insert(0, os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
21 |
22 | from DB import setup
23 | from Logic import pdf_generator, forecasting
24 | from api.models import (
25 |     ProjectCreate, ProjectCreateResponse, RecordResponse,
26 |     RecordWithPeople, PersonResponse, TaxBracketCreate,
27 |     TaxBracketResponse, RecordUpdate, MessageResponse
28 | )
29 |
30 | app = FastAPI(
31 |     title="MoneySplit API",
32 |     description="RESTful API for commission-based income splitting with tax calculations",
33 |     version="1.0.0"
34 | )
35 |
36 | # Enable CORS for frontend integration
37 | app.add_middleware(
38 |     CORSMiddleware,
39 |     allow_origins=["*"],
40 |     allow_credentials=True,
41 |     allow_methods=["*"],
42 |     allow_headers=["*"],
43 | )
44 |
45 |
46 | # ===== Project/Record Endpoints =====
47 |
48 | @app.post("/api/projects", response_model=ProjectCreateResponse, status_code=201)
49 | async def create_project(project: ProjectCreate):
50 |     """Create a new project with people and calculate taxes."""
51 |     try:
52 |         # Calculate totals
53 |         total_costs = sum(project.costs)
54 |         income = project.revenue - total_costs
55 |         group_income = income
56 |         individual_income = income / project.num_people if project.num_people > 0 else 0
57 |
58 |         # Calculate tax
59 |         if project.tax_type == "Individual":
60 |             tax = setup.calculate_tax_from_db(individual_income, project.country, project.tax_type)
61 |         else:
62 |             tax = setup.calculate_tax_from_db(group_income, project.country, project.tax_type)
63 |
64 |         # Calculate net incomes
65 |         if project.tax_type == "Individual":
66 |             net_income_per_person = individual_income - tax
67 |             net_income_group = net_income_per_person * project.num_people
68 |         else:
69 |             net_income_group = group_income - tax
70 |             net_income_per_person = net_income_group / project.num_people if project.num_people > 0 else 0
71 |
72 |         # Save to database
```

```

73 |         conn = setup.get_conn()
74 |         cursor = conn.cursor()
75 |         cursor.execute("""
76 |             INSERT INTO tax_records (
77 |                 num_people, revenue, total_costs, group_income, individual_income,
78 |                 tax_origin, tax_option, tax_amount,
79 |                 net_income_per_person, net_income_group
80 |             ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
81 |         """, (
82 |             project.num_people,
83 |             project.revenue,
84 |             total_costs,
85 |             group_income,
86 |             individual_income,
87 |             project.country,
88 |             project.tax_type,
89 |             tax,
90 |             net_income_per_person,
91 |             net_income_group
92 |         ))
93 |         record_id = cursor.lastrowid
94 |
95 |         # Save people
96 |         for person in project.people:
97 |             if project.tax_type == "Individual":
98 |                 gross_income = individual_income * person.work_share * project.num_people
99 |                 tax_paid = tax * person.work_share
100 |                 net_income = gross_income - tax_paid
101 |             else:
102 |                 gross_income = group_income * person.work_share
103 |                 tax_paid = tax * person.work_share
104 |                 net_income = gross_income - tax_paid
105 |
106 |             cursor.execute("""
107 |                 INSERT INTO people (record_id, name, work_share, gross_income, tax_paid, net_income)
108 |                 VALUES (?, ?, ?, ?, ?, ?)
109 |             """, (record_id, person.name, person.work_share, gross_income, tax_paid, net_income))
110 |
111 |         conn.commit()
112 |         conn.close()
113 |
114 |         return ProjectCreateResponse(
115 |             record_id=record_id,
116 |             message="Project created successfully",
117 |             summary={
118 |                 "revenue": project.revenue,
119 |                 "total_costs": total_costs,
120 |                 "tax_amount": tax,
121 |                 "net_income_group": net_income_group,
122 |                 "net_income_per_person": net_income_per_person
123 |             }
124 |         )
125 |     except Exception as e:
126 |         raise HTTPException(status_code=400, detail=str(e))
127 |
128 |
129 | @app.get("/api/records", response_model=List[RecordResponse])
130 | async def get_records(limit: int = Query(10, ge=1, le=100)):
131 |     """Get recent records (default: last 10)."""
132 |     records = setup.fetch_last_records(limit)
133 |     return [
134 |         RecordResponse(
135 |             id=r[0], tax_origin=r[1], tax_option=r[2],
136 |             revenue=r[3], total_costs=r[4], tax_amount=r[5],
137 |             net_income_group=r[6], net_income_per_person=r[7],
138 |             created_at=r[8],
139 |             num_people=r[9], group_income=r[10], individual_income=r[11]
140 |         ) for r in records
141 |     ]
142 |
143 |
144 | @app.get("/api/records/{record_id}", response_model=RecordWithPeople)
145 | async def get_record(record_id: int):
146 |     """Get a specific record with its people."""
147 |     record = setup.get_record_by_id(record_id)
148 |     if not record:
149 |         raise HTTPException(status_code=404, detail=f"Record {record_id} not found")
150 |

```

```

151 |     people = setup.fetch_people_by_record(record_id)
152 |
153 |     return RecordWithPeople(
154 |         id=record[0], tax_origin=record[1], tax_option=record[2],
155 |         revenue=record[3], total_costs=record[4], tax_amount=record[5],
156 |         net_income_group=record[6], net_income_per_person=record[7],
157 |         created_at=record[8],
158 |         num_people=record[9], group_income=record[10], individual_income=record[11],
159 |         people=[
160 |             PersonResponse(
161 |                 id=p[0], name=p[1], work_share=p[2],
162 |                 gross_income=p[3], tax_paid=p[4], net_income=p[5]
163 |             ) for p in people
164 |         ]
165 |     )
166 |
167 |
168 | @app.put("/api/records/{record_id}", response_model=MessageResponse)
169 | async def update_record(record_id: int, update: RecordUpdate):
170 |     """Update a record field."""
171 |     try:
172 |         setup.update_record(record_id, update.field, update.value)
173 |         return MessageResponse(message=f"Record {record_id} updated successfully")
174 |     except Exception as e:
175 |         raise HTTPException(status_code=400, detail=str(e))
176 |
177 |
178 | @app.delete("/api/records/{record_id}", response_model=MessageResponse)
179 | async def delete_record(record_id: int):
180 |     """Delete a record and its people."""
181 |     record = setup.get_record_by_id(record_id)
182 |     if not record:
183 |         raise HTTPException(status_code=404, detail=f"Record {record_id} not found")
184 |
185 |     setup.delete_record(record_id)
186 |     return MessageResponse(message=f"Record {record_id} deleted successfully")
187 |
188 |
189 | # ===== Tax Bracket Endpoints =====
190 |
191 | @app.get("/api/tax-brackets", response_model=List[TaxBracketResponse])
192 | async def get_tax_brackets(country: str, tax_type: str):
193 |     """Get tax brackets for a country and type."""
194 |     import math
195 |     brackets = setup.get_tax_brackets(country, tax_type, include_id=True)
196 |     return [
197 |         TaxBracketResponse(
198 |             id=b[0],
199 |             income_limit=999999999 if math.isinf(b[1]) else b[1], # Convert inf to large number for JSON
200 |             rate=b[2],
201 |             country=country,
202 |             tax_type=tax_type
203 |         ) for b in brackets
204 |     ]
205 |
206 |
207 | @app.post("/api/tax-brackets", response_model=MessageResponse, status_code=201)
208 | async def create_tax_bracket(bracket: TaxBracketCreate):
209 |     """Add a new tax bracket."""
210 |     try:
211 |         setup.add_tax_bracket(bracket.country, bracket.tax_type, bracket.income_limit, bracket.rate)
212 |         return MessageResponse(message="Tax bracket created successfully")
213 |     except Exception as e:
214 |         raise HTTPException(status_code=400, detail=str(e))
215 |
216 |
217 | @app.delete("/api/tax-brackets/{bracket_id}", response_model=MessageResponse)
218 | async def delete_tax_bracket(bracket_id: int):
219 |     """Delete a tax bracket."""
220 |     try:
221 |         setup.delete_tax_bracket(bracket_id)
222 |         return MessageResponse(message=f"Tax bracket {bracket_id} deleted successfully")
223 |     except Exception as e:
224 |         raise HTTPException(status_code=400, detail=str(e))
225 |
226 |
227 | # ===== People Endpoints =====
228 |

```



```

229 | @app.get("/api/people/{person_id}", response_model=PersonResponse)
230 | async def get_person(person_id: int):
231 |     """Get a specific person by ID."""
232 |     conn = setup.get_conn()
233 |     cursor = conn.cursor()
234 |     cursor.execute("SELECT id, name, work_share, gross_income, tax_paid, net_income FROM people WHERE id=?", (
person_id,...
235 |     person = cursor.fetchone()
236 |     conn.close()
237 |
238 |     if not person:
239 |         raise HTTPException(status_code=404, detail=f"Person {person_id} not found")
240 |
241 |     return PersonResponse(
242 |         id=person[0], name=person[1], work_share=person[2],
243 |         gross_income=person[3], tax_paid=person[4], net_income=person[5]
244 |     )
245 |
246 |
247 | @app.get("/api/people/history/{name}", response_model=List[dict])
248 | async def get_person_history(name: str):
249 |     """Get all records for a person by name."""
250 |     records = setup.fetch_records_by_person(name)
251 |     return [
252 |         {
253 |             "person_id": r[0],
254 |             "record_id": r[1],
255 |             "name": r[2],
256 |             "work_share": r[3],
257 |             "gross_income": r[4],
258 |             "tax_paid": r[5],
259 |             "net_income": r[6],
260 |             "created_at": r[7]
261 |         } for r in records
262 |     ]
263 |
264 |
265 | # ===== Report Endpoints =====
266 |
267 | @app.get("/api/reports/revenue-summary")
268 | async def revenue_summary():
269 |     """Get revenue summary by year."""
270 |     rows = setup.get_revenue_summary()
271 |     return [
272 |         {
273 |             "year": r[0],
274 |             "total_revenue": r[1],
275 |             "total_costs": r[2],
276 |             "net_income": r[3]
277 |         } for r in rows
278 |     ]
279 |
280 |
281 | @app.get("/api/reports/top-people")
282 | async def top_people(limit: int = Query(10, ge=1, le=50)):
283 |     """Get top people by net income."""
284 |     rows = setup.get_top_people(limit)
285 |     return [
286 |         {
287 |             "name": r[0],
288 |             "total_gross": r[1],
289 |             "total_tax_paid": r[2],
290 |             "total_net": r[3]
291 |         } for r in rows
292 |     ]
293 |
294 |
295 | @app.get("/api/reports/statistics")
296 | async def overall_statistics():
297 |     """Get overall database statistics."""
298 |     conn = setup.get_conn()
299 |     cursor = conn.cursor()
300 |
301 |     cursor.execute("""
302 |         SELECT COUNT(*),
303 |             COALESCE(SUM(revenue), 0),
304 |             COALESCE(SUM(total_costs), 0),
305 |             COALESCE(SUM(tax_amount), 0),

```

```

306 |             COALESCE(SUM(net_income_group), 0),
307 |             COALESCE(AVG(CASE
308 |                 WHEN group_income > 0 THEN tax_amount * 100.0 / group_income
309 |                 ELSE 0
310 |             END), 0)
311 |         FROM tax_records
312 |     """
313 |     result = cursor.fetchone()
314 |     total_records = result[0] or 0
315 |     total_rev = result[1] or 0
316 |     total_costs = result[2] or 0
317 |     total_tax = result[3] or 0
318 |     total_net = result[4] or 0
319 |     avg_rate = result[5] or 0
320 |
321 |     cursor.execute("SELECT COUNT(*), COUNT(DISTINCT name) FROM people")
322 |     people_result = cursor.fetchone()
323 |     total_people_entries = people_result[0] or 0
324 |     unique_people = people_result[1] or 0
325 |
326 |     conn.close()
327 |
328 |     return {
329 |         "total_records": total_records,
330 |         "total_revenue": float(total_rev),
331 |         "total_costs": float(total_costs),
332 |         "total_tax": float(total_tax),
333 |         "total_net_income": float(total_net),
334 |         "average_tax_rate": float(avg_rate),
335 |         "total_people_entries": total_people_entries,
336 |         "unique_people": unique_people
337 |     }
338 |
339 |
340 | # ===== Visualization Endpoints =====
341 |
342 | def create_stunning_html(plotly_fig, title, emoji, description):
343 |     """Wrap Plotly figure in world-class premium HTML template."""
344 |     plot_config = {'displayModeBar': True, 'displaylogo': False}
345 |     plotly_html = plotly_fig.to_html(include_plotlyjs='cdn', full_html=False, config=plot_config)
346 |
347 |     return f"""
348 | <!DOCTYPE html>
349 | <html lang="en">
350 | <head>
351 |     <meta charset="UTF-8">
352 |     <meta name="viewport" content="width=device-width, initial-scale=1.0">
353 |     <title>{title} - MoneySplit Analytics</title>
354 |     <link rel="preconnect" href="https://fonts.googleapis.com">
355 |     <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
356 |     <link href="https://fonts.googleapis.com/css2?family=Inter:wght@300;400;500;600;700;800;900&
family=Poppins:wg...
357 |     <style>
358 |         :root {{
359 |             --bg-primary: #0a0a0f;
360 |             --bg-secondary: #13131a;
361 |             --bg-tertiary: #1a1a24;
362 |             --accent-primary: #6366f1;
363 |             --accent-secondary: #8b5cf6;
364 |             --accent-tertiary: #ec4899;
365 |             --text-primary: #ffffff;
366 |             --text-secondary: #a1a1aa;
367 |             --border-subtle: rgba(255, 255, 255, 0.08);
368 |             --glow-purple: rgba(139, 92, 246, 0.5);
369 |             --glow-pink: rgba(236, 72, 153, 0.5);
370 |         }}
371 |
372 |         * {{
373 |             margin: 0;
374 |             padding: 0;
375 |             box-sizing: border-box;
376 |         }}
377 |
378 |         html {{
379 |             scroll-behavior: smooth;
380 |         }}
381 |
382 |         body {{

```

```

383 |         font-family: 'Inter', -apple-system, BlinkMacSystemFont, sans-serif;
384 |         background: var(--bg-primary);
385 |         color: var(--text-primary);
386 |         min-height: 100vh;
387 |         padding: 0;
388 |         position: relative;
389 |         overflow-x: hidden;
390 |         line-height: 1.6;
391 |         -webkit-font-smoothing: antialiased;
392 |         -moz-osx-font-smoothing: grayscale;
393 |     }}
394 |
395 |     /* ULTRA-PREMIUM MESH GRADIENT BACKGROUND */
396 |     body::before {{
397 |         content: '';
398 |         position: fixed;
399 |         top: 0;
400 |         left: 0;
401 |         width: 100%;
402 |         height: 100%;
403 |         z-index: 0;
404 |         background:
405 |             radial-gradient(ellipse 80% 50% at 50% -20%, rgba(139, 92, 246, 0.15), transparent),
406 |             radial-gradient(ellipse 60% 40% at 10% 40%, rgba(99, 102, 241, 0.1), transparent),
407 |             radial-gradient(ellipse 60% 40% at 90% 60%, rgba(236, 72, 153, 0.1), transparent);
408 |         pointer-events: none;
409 |     }}
410 |
411 |     /* ANIMATED GRID PATTERN */
412 |     body::after {{
413 |         content: '';
414 |         position: fixed;
415 |         top: 0;
416 |         left: 0;
417 |         width: 100%;
418 |         height: 100%;
419 |         z-index: 0;
420 |         background-image:
421 |             linear-gradient(rgba(255, 255, 255, 0.02) 1px, transparent 1px),
422 |             linear-gradient(90deg, rgba(255, 255, 255, 0.02) 1px, transparent 1px);
423 |         background-size: 50px 50px;
424 |         mask-image: radial-gradient(ellipse 100% 100% at 50% 50%, black 40%, transparent 100%);
425 |         pointer-events: none;
426 |     }}
427 |
428 |     .container {{
429 |         max-width: 1600px;
430 |         margin: 0 auto;
431 |         padding: 60px 40px;
432 |         position: relative;
433 |         z-index: 1;
434 |     }}
435 |
436 |     @media (max-width: 768px) {{
437 |         .container {{
438 |             padding: 40px 20px;
439 |         }}
440 |     }}
441 |
442 |     /* NAVIGATION BAR */
443 |     .nav {{
444 |         position: fixed;
445 |         top: 0;
446 |         left: 0;
447 |         right: 0;
448 |         z-index: 1000;
449 |         backdrop-filter: blur(20px) saturate(180%);
450 |         background: rgba(10, 10, 15, 0.7);
451 |         border-bottom: 1px solid var(--border-subtle);
452 |     }}
453 |
454 |     .nav-content {{
455 |         max-width: 1600px;
456 |         margin: 0 auto;
457 |         padding: 16px 40px;
458 |         display: flex;
459 |         align-items: center;
460 |         justify-content: space-between;

```

```

461 |     }}
462 |
463 |     .logo {{
464 |         display: flex;
465 |         align-items: center;
466 |         gap: 12px;
467 |         font-family: 'Poppins', sans-serif;
468 |         font-size: 20px;
469 |         font-weight: 700;
470 |         background: linear-gradient(135deg, var(--accent-primary), var(--accent-secondary));
471 |         -webkit-background-clip: text;
472 |         -webkit-text-fill-color: transparent;
473 |     }}
474 |
475 |     .badge {{
476 |         padding: 4px 12px;
477 |         border-radius: 12px;
478 |         font-size: 11px;
479 |         font-weight: 600;
480 |         text-transform: uppercase;
481 |         letter-spacing: 0.5px;
482 |         background: linear-gradient(135deg, var(--accent-primary), var(--accent-secondary));
483 |         color: white;
484 |     }}
485 |
486 |     /* HERO SECTION */
487 |     .header {{
488 |         margin-top: 80px;
489 |         margin-bottom: 48px;
490 |         text-align: center;
491 |         animation: fadeInDown 0.6s cubic-bezier(0.16, 1, 0.3, 1);
492 |     }}
493 |
494 |     @keyframes fadeInDown {{
495 |         from {{
496 |             opacity: 0;
497 |             transform: translateY(-20px);
498 |         }}
499 |         to {{
500 |             opacity: 1;

```

File 20: api/models.py

```
1  """
2  Pydantic models for API request/response validation.
3  """
4  from pydantic import BaseModel, Field, field_validator
5  from typing import List, Optional
6  from datetime import datetime
7
8
9  # ===== Request Models =====
10
11  class PersonInput(BaseModel):
12      name: str = Field(..., min_length=1, description="Person's name")
13      work_share: float = Field(..., ge=0, le=1, description="Work share (0.0 to 1.0)")
14
15
16  class ProjectCreate(BaseModel):
17      num_people: int = Field(..., gt=0, description="Number of people")
18      revenue: float = Field(..., ge=0, description="Total revenue")
19      costs: List[float] = Field(..., description="List of costs")
20      country: str = Field(..., min_length=1, description="Country (e.g., US, Spain, UK, etc.)")
21      tax_type: str = Field(..., pattern="^(Individual|Business)$", description="Tax type: Individual or Business")
22      people: List[PersonInput] = Field(..., description="List of people with work shares")
23
24      @field_validator('people')
25      @classmethod
26      def validate_people_count(cls, v, info):
27          if 'num_people' in info.data and len(v) != info.data['num_people']:
28              raise ValueError(f"Expected {info.data['num_people']} people, got {len(v)}")
29          return v
30
31      @field_validator('people')
32      @classmethod
33      def validate_work_shares(cls, v):
34          total_share = sum(person.work_share for person in v)
35          if abs(total_share - 1.0) > 0.01:
36              raise ValueError(f"Work shares must sum to 1.0, got {total_share:.2f}")
37          return v
38
39
40  class TaxBracketCreate(BaseModel):
41      country: str = Field(..., min_length=1, description="Country name")
42      tax_type: str = Field(..., pattern="^(Individual|Business)$")
43      income_limit: float = Field(..., ge=0)
44      rate: float = Field(..., ge=0, le=1)
45
46
47  class RecordUpdate(BaseModel):
48      field: str = Field(..., pattern="^(num_people|revenue|total_costs|tax_origin|tax_option)$")
49      value: str | int | float
50
51
52  # ===== Response Models =====
53
54  class PersonResponse(BaseModel):
55      id: int
56      name: str
57      work_share: float
58      gross_income: float
59      tax_paid: float
60      net_income: float
61
62      class Config:
63          from_attributes = True
64
65
66  class RecordResponse(BaseModel):
67      id: int
68      num_people: int
69      revenue: float
70      total_costs: float
71      group_income: float
72      individual_income: float
73      tax_origin: str
74      tax_option: str
```

```
75 |         tax_amount: float
76 |         net_income_per_person: float
77 |         net_income_group: float
78 |         created_at: str
79 |
80 |         class Config:
81 |             from_attributes = True
82 |
83 |
84 | class RecordWithPeople(RecordResponse):
85 |     people: List[PersonResponse] = []
86 |
87 |
88 | class TaxBracketResponse(BaseModel):
89 |     id: int
90 |     country: str
91 |     tax_type: str
92 |     income_limit: float
93 |     rate: float
94 |
95 |     class Config:
96 |         from_attributes = True
97 |
98 |
99 | class ProjectCreateResponse(BaseModel):
100 |     record_id: int
101 |     message: str
102 |     summary: dict
103 |
104 |
105 | class MessageResponse(BaseModel):
106 |     message: str
107 |     details: Optional[dict] = None
108 |
```

File 21: frontend/package.json

```
1 | {
2 |   "name": "frontend",
3 |   "version": "0.1.0",
4 |   "private": true,
5 |   "dependencies": {
6 |     "@tanstack/react-query": "^5.90.2",
7 |     "@testing-library/dom": "^10.4.1",
8 |     "@testing-library/jest-dom": "^6.9.1",
9 |     "@testing-library/react": "^16.3.0",
10 |    "@testing-library/user-event": "^13.5.0",
11 |    "@types/jest": "^27.5.2",
12 |    "@types/node": "^16.18.126",
13 |    "@types/react": "^19.2.0",
14 |    "@types/react-dom": "^19.2.0",
15 |    "axios": "^1.12.2",
16 |    "lucide-react": "^0.544.0",
17 |    "react": "^19.2.0",
18 |    "react-dom": "^19.2.0",
19 |    "react-router-dom": "^7.9.3",
20 |    "react-scripts": "5.0.1",
21 |    "recharts": "^3.2.1",
22 |    "typescript": "^4.9.5",
23 |    "web-vitals": "^2.1.4"
24 |  },
25 |   "scripts": {
26 |     "start": "react-scripts start",
27 |     "build": "react-scripts build",
28 |     "test": "react-scripts test",
29 |     "eject": "react-scripts eject"
30 |   },
31 |   "eslintConfig": {
32 |     "extends": [
33 |       "react-app",
34 |       "react-app/jest"
35 |     ]
36 |   },
37 |   "browserslist": {
38 |     "production": [
39 |       ">0.2%",
40 |       "not dead",
41 |       "not op_mini all"
42 |     ],
43 |     "development": [
44 |       "last 1 chrome version",
45 |       "last 1 firefox version",
46 |       "last 1 safari version"
47 |     ]
48 |   }
49 | }
```

File 22: frontend/src/App.css

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

```
1 | @import url('https://fonts.googleapis.com/css2?family=Inter:wght@300;400;500;600;700;800;900&display=swap');
2 | @import url('https://fonts.googleapis.com/css2?family=Poppins:wght@700;800;900&display=swap');
3 |
4 | * {
5 |   margin: 0;
6 |   padding: 0;
7 |   box-sizing: border-box;
8 | }
9 |
10 | body {
11 |   font-family: 'Inter', -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', sans-serif;
12 |   -webkit-font-smoothing: subpixel-antialiased;
13 |   -moz-osx-font-smoothing: auto;
14 |   background: #1a1625;
15 |   background: linear-gradient(135deg, #1a1625 0%, #2d2640 50%, #1f1b2e 100%);
16 |   min-height: 100vh;
17 |   overflow-x: hidden;
18 |   position: relative;
19 |   transform: translateZ(0);
20 |   backface-visibility: hidden;
21 |   perspective: 1000px;
22 | }
23 |
24 | /* STATIC BACKGROUND - No Animation */
25 | body::before {
26 |   content: '';
27 |   position: fixed;
28 |   width: 100%;
29 |   height: 100%;
30 |   top: 0;
31 |   left: 0;
32 |   z-index: 0;
33 |   background:
34 |     radial-gradient(circle at 20% 80%, rgba(102, 126, 234, 0.15) 0%, transparent 50%),
35 |     radial-gradient(circle at 80% 20%, rgba(118, 75, 162, 0.15) 0%, transparent 50%),
36 |     radial-gradient(circle at 40% 40%, rgba(99, 102, 241, 0.15) 0%, transparent 50%),
37 |     radial-gradient(circle at 60% 80%, rgba(138, 116, 249, 0.15) 0%, transparent 50%);
38 |   pointer-events: none;
39 | }
40 |
41 | @keyframes rotate {
42 |   0% { transform: rotate(0deg); }
43 |   100% { transform: rotate(360deg); }
44 | }
45 |
46 | /* FLOATING GRADIENT ORBS */
47 | body::after {
48 |   content: '';
49 |   position: fixed;
50 |   width: 100%;
51 |   height: 100%;
52 |   top: 0;
53 |   left: 0;
54 |   z-index: 0;
55 |   background:
56 |     radial-gradient(circle at 15% 90%, rgba(102, 126, 234, 0.35) 0%, transparent 25%),
57 |     radial-gradient(circle at 85% 10%, rgba(118, 75, 162, 0.35) 0%, transparent 25%),
58 |     radial-gradient(circle at 50% 50%, rgba(138, 116, 249, 0.25) 0%, transparent 30%);
59 |   animation: float 20s ease-in-out infinite;
60 |   pointer-events: none;
61 | }
62 |
63 | @keyframes float {
64 |   0%, 100% {
65 |     transform: translateY(0) ;
66 |     opacity: 0.8;
67 |   }
68 |   33% {
69 |     transform: translateY(-30px) ;
70 |     opacity: 1;
71 |   }
72 |   66% {
```



```

73 |     transform: translateY(30px) ;
74 |     opacity: 0.9;
75 |   }
76 | }
77 |
78 | .app {
79 |   display: flex;
80 |   min-height: 100vh;
81 |   position: relative;
82 |   z-index: 1;
83 | }
84 |
85 | /* STARS/PARTICLES EFFECT */
86 | .app::before {
87 |   content: '';
88 |   position: fixed;
89 |   top: 0;
90 |   left: 0;
91 |   width: 100%;
92 |   height: 100%;
93 |   background-image:
94 |     radial-gradient(2px 2px at 20px 30px, rgba(255, 255, 255, 0.3), transparent),
95 |     radial-gradient(2px 2px at 60px 70px, rgba(255, 255, 255, 0.2), transparent),
96 |     radial-gradient(1px 1px at 50px 50px, rgba(255, 255, 255, 0.4), transparent),
97 |     radial-gradient(1px 1px at 130px 80px, rgba(255, 255, 255, 0.3), transparent),
98 |     radial-gradient(2px 2px at 90px 10px, rgba(255, 255, 255, 0.5), transparent);
99 |   background-size: 200px 200px;
100 |   animation: sparkle 4s ease-in-out infinite;
101 |   pointer-events: none;
102 |   z-index: 0;
103 | }
104 |
105 | @keyframes sparkle {
106 |   0%, 100% { opacity: 1; }
107 |   50% { opacity: 0.5; }
108 | }
109 |
110 | /* PREMIUM 3D SIDEBAR */
111 | .sidebar {
112 |   width: 280px;
113 |   background: rgba(20, 18, 35, 0.98);
114 |   color: #ffffff;
115 |   padding: 32px 24px;
116 |   display: flex;
117 |   flex-direction: column;
118 |   box-shadow:
119 |     4px 0 30px rgba(0, 0, 0, 0.4),
120 |     inset -1px 0 0 rgba(255, 255, 255, 0.08);
121 |   position: relative;
122 |   z-index: 10;
123 |   border-right: 1px solid rgba(138, 116, 249, 0.2);
124 |   animation: slideInLeft 0.8s cubic-bezier(0.16, 1, 0.3, 1);
125 |   transition: transform 0.4s cubic-bezier(0.4, 0, 0.2, 1);
126 | }
127 |
128 | .sidebar.collapsed {
129 |   transform: translateX(-100%);
130 | }
131 |
132 | @keyframes slideInLeft {
133 |   from {
134 |     transform: translateX(-100%);
135 |     opacity: 0;
136 |   }
137 |   to {
138 |     transform: translateX(0);
139 |     opacity: 1;
140 |   }
141 | }
142 |
143 | .sidebar::before {
144 |   content: '';
145 |   position: absolute;
146 |   top: 0;
147 |   left: 0;
148 |   right: 0;
149 |   height: 6px;
150 |   background: linear-gradient(90deg,

```

```

151 |         #667eea 0%,
152 |         #764ba2 25%,
153 |         #f093fb 50%,
154 |         #764ba2 75%,
155 |         #667eea 100%);
156 |     background-size: 200% 100%;
157 |     animation: shimmer 3s linear infinite;
158 | }
159 |
160 | @keyframes shimmer {
161 |     0% { background-position: 0% 0%; }
162 |     100% { background-position: 200% 0%; }
163 | }
164 |
165 | .sidebar::after {
166 |     content: '';
167 |     position: absolute;
168 |     top: 0;
169 |     left: 0;
170 |     right: 0;
171 |     bottom: 0;
172 |     background: linear-gradient(180deg, rgba(102, 126, 234, 0.05) 0%, transparent 100%);
173 |     pointer-events: none;
174 | }
175 |
176 | .sidebar h1 {
177 |     font-family: 'Poppins', sans-serif;
178 |     font-size: 28px;
179 |     font-weight: 900;
180 |     margin-bottom: 56px;
181 |     background: linear-gradient(135deg, #667eea 0%, #764ba2 50%, #f093fb 100%);
182 |     -webkit-background-clip: text;
183 |     -webkit-text-fill-color: transparent;
184 |     background-clip: text;
185 |     display: flex;
186 |     align-items: center;
187 |     gap: 8px;
188 |     letter-spacing: -0.5px;
189 |     position: relative;
190 |     z-index: 1;
191 |     animation: titleGlow 3s ease-in-out infinite;
192 | }
193 |
194 | @keyframes titleGlow {
195 |     0%, 100% {
196 |         filter: drop-shadow(0 0 20px rgba(102, 126, 234, 0.6));
197 |     }
198 |     50% {
199 |         filter: drop-shadow(0 0 40px rgba(118, 75, 162, 0.8));
200 |     }
201 | }
202 |
203 | .logo-emoji {
204 |     font-size: 32px;
205 |     filter: drop-shadow(0 4px 16px rgba(102, 126, 234, 0.8));
206 |     animation: bounce 2s ease-in-out infinite;
207 |     display: inline-block;
208 |     line-height: 1;
209 | }
210 |
211 | @keyframes bounce {
212 |     0%, 100% {
213 |         transform: translateY(0) ;
214 |     }
215 |     50% {
216 |         transform: translateY(-8px) ;
217 |     }
218 | }
219 |
220 | .nav-menu {
221 |     display: flex;
222 |     flex-direction: column;
223 |     gap: 8px;
224 | }
225 |
226 | .nav-item {
227 |     padding: 16px 20px;
228 |     border-radius: 12px;

```

```

229 | cursor: pointer;
230 | transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
231 | display: flex;
232 | align-items: center;
233 | gap: 14px;
234 | text-decoration: none;
235 | color: rgba(255, 255, 255, 0.65);
236 | font-weight: 500;
237 | font-size: 15px;
238 | position: relative;
239 | overflow: hidden;
240 | background: rgba(255, 255, 255, 0.03);
241 | border: 1px solid rgba(255, 255, 255, 0.05);
242 | }
243 |
244 | .nav-item::before {
245 |   content: '';
246 |   position: absolute;
247 |   left: 0;
248 |   top: 0;
249 |   bottom: 0;
250 |   width: 5px;
251 |   background: linear-gradient(180deg, #667eea 0%, #764ba2 100%);
252 |   transform: scaleY(0);
253 |   transition: transform 0.4s cubic-bezier(0.175, 0.885, 0.32, 1.275);
254 |   box-shadow: 0 0 20px rgba(102, 126, 234, 0.8);
255 | }
256 |
257 | .nav-item::after {
258 |   content: '';
259 |   position: absolute;
260 |   inset: 0;
261 |   background: linear-gradient(135deg, rgba(102, 126, 234, 0.2), rgba(118, 75, 162, 0.2));
262 |   opacity: 0;
263 |   transition: opacity 0.4s ease;
264 |   border-radius: 18px;
265 | }
266 |
267 | .nav-item:hover {
268 |   background: linear-gradient(135deg, rgba(102, 126, 234, 0.18) 0%, rgba(118, 75, 162, 0.18) 100%);
269 |   color: rgba(255, 255, 255, 0.95);
270 |   transform: translateX(8px);
271 |   box-shadow:
272 |     0 8px 24px rgba(102, 126, 234, 0.25),
273 |     inset 0 1px 0 rgba(255, 255, 255, 0.08);
274 | }
275 |
276 | .nav-item:hover::before {
277 |   transform: scaleY(1);
278 | }
279 |
280 | .nav-item:hover::after {
281 |   opacity: 1;
282 | }
283 |
284 | .nav-item.active {
285 |   background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
286 |   color: white;
287 |   font-weight: 700;
288 |   box-shadow:
289 |     0 12px 40px rgba(102, 126, 234, 0.6),
290 |     0 0 60px rgba(102, 126, 234, 0.4),
291 |     inset 0 1px 0 rgba(255, 255, 255, 0.2);
292 |   transform: translateX(8px);
293 |   animation: activeGlow 2s ease-in-out infinite;
294 | }
295 |
296 | @keyframes activeGlow {
297 |   0%, 100% {
298 |     box-shadow:
299 |       0 12px 40px rgba(102, 126, 234, 0.6),
300 |       0 0 60px rgba(102, 126, 234, 0.4),
301 |       inset 0 1px 0 rgba(255, 255, 255, 0.2);
302 |   }
303 |   50% {
304 |     box-shadow:
305 |       0 12px 40px rgba(118, 75, 162, 0.8),
306 |       0 0 80px rgba(118, 75, 162, 0.6),

```

```

307 |         inset 0 1px 0 rgba(255, 255, 255, 0.3);
308 |     }
309 | }
310 |
311 | .nav-item.active::before {
312 |     transform: scaleY(1);
313 |     background: white;
314 | }
315 |
316 | .nav-item span:first-child {
317 |     font-size: 28px;
318 |     filter: drop-shadow(0 4px 8px rgba(0, 0, 0, 0.3));
319 |     transition: transform 0.3s ease;
320 | }
321 |
322 | .nav-item:hover span:first-child,
323 | .nav-item.active span:first-child {
324 |     transform: rotate(5deg);
325 | }
326 |
327 | /* Menu Toggle Button */
328 | .menu-toggle {
329 |     position: fixed;
330 |     top: 20px;
331 |     left: 20px;
332 |     z-index: 100;
333 |     width: 50px;
334 |     height: 50px;
335 |     border-radius: 16px;
336 |     background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
337 |     border: none;
338 |     cursor: pointer;
339 |     display: flex;
340 |     align-items: center;
341 |     justify-content: center;
342 |     box-shadow:
343 |         0 10px 30px rgba(102, 126, 234, 0.5),
344 |         0 0 40px rgba(102, 126, 234, 0.3);
345 |     transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
346 | }
347 |
348 | .menu-toggle:hover {
349 |     transform: translateY(-2px);
350 |     box-shadow:
351 |         0 15px 40px rgba(102, 126, 234, 0.7),
352 |         0 0 60px rgba(102, 126, 234, 0.5);
353 | }
354 |
355 | .menu-toggle svg {
356 |     width: 24px;
357 |     height: 24px;
358 |     stroke: white;
359 |     stroke-width: 2.5;
360 |     fill: none;
361 | }
362 |
363 | /* Top Right Logo */
364 | .top-logo {
365 |     position: fixed;
366 |     top: 24px;
367 |     right: 32px;
368 |     z-index: 100;
369 |     display: flex;
370 |     align-items: center;
371 |     gap: 12px;
372 |     padding: 14px 24px;
373 |     background: rgba(20, 18, 35, 0.95);
374 |     border-radius: 20px;
375 |     border: 2px solid rgba(138, 116, 249, 0.3);
376 |     box-shadow:
377 |         0 10px 30px rgba(0, 0, 0, 0.3),
378 |         0 0 40px rgba(102, 126, 234, 0.2);
379 | }
380 |
381 | .top-logo-emoji {
382 |     font-size: 28px;
383 |     filter: drop-shadow(0 4px 12px rgba(102, 126, 234, 0.6));
384 | }

```

```

385 |
386 | .top-logo-text {
387 |     font-family: 'Poppins', sans-serif;
388 |     font-size: 20px;
389 |     font-weight: 800;
390 |     background: linear-gradient(135deg, #667eea 0%, #764ba2 50%, #f093fb 100%);
391 |     -webkit-background-clip: text;
392 |     -webkit-text-fill-color: transparent;
393 |     background-clip: text;
394 |     letter-spacing: -0.5px;
395 | }
396 |
397 | /* Main Content */
398 | .main-content {
399 |     flex: 1;
400 |     padding: 40px;
401 |     overflow-y: auto;
402 |     position: relative;
403 |     z-index: 1;
404 | }
405 |
406 | .page-header {
407 |     margin-bottom: 56px;
408 |     animation: fadeInDown 0.8s cubic-bezier(0.16, 1, 0.3, 1);
409 |     position: relative;
410 | }
411 |
412 | .page-header::after {
413 |     content: '';
414 |     position: absolute;
415 |     bottom: -20px;
416 |     left: 0;
417 |     width: 120px;
418 |     height: 6px;
419 |     background: linear-gradient(90deg, #667eea 0%, #764ba2 50%, #f093fb 100%);
420 |     border-radius: 3px;
421 |     box-shadow: 0 0 30px rgba(102, 126, 234, 0.8);
422 |     animation: widthGrow 0.8s ease 0.4s both;
423 | }
424 |
425 | @keyframes widthGrow {
426 |     from { width: 0; }
427 |     to { width: 120px; }
428 | }
429 |
430 | .page-header h2 {
431 |     font-family: 'Poppins', sans-serif;
432 |     font-size: 56px;
433 |     font-weight: 900;
434 |     color: white;
435 |     margin-bottom: 16px;
436 |     letter-spacing: -2px;
437 |     text-shadow:
438 |         0 4px 20px rgba(0, 0, 0, 0.4),
439 |         0 0 60px rgba(102, 126, 234, 0.5);
440 |     animation: titleBounce 1s ease;
441 | }
442 |
443 | @keyframes titleBounce {
444 |     0%, 100% { transform: translateY(0); }
445 |     50% { transform: translateY(-10px); }
446 | }
447 |
448 | .page-header p {
449 |     color: rgba(255, 255, 255, 0.9);
450 |     font-size: 20px;
451 |     font-weight: 600;
452 |     text-shadow: 0 2px 8px rgba(0, 0, 0, 0.3);
453 |     letter-spacing: 0.3px;
454 | }
455 |
456 | /* PREMIUM 3D CARDS */
457 | .card {
458 |     background: rgba(15, 12, 41, 0.85);
459 |     border-radius: 32px;
460 |     padding: 40px;
461 |     box-shadow:
462 |         0 30px 80px rgba(0, 0, 0, 0.5),

```

```

463 |     inset 0 1px 0 rgba(255, 255, 255, 0.1),
464 |     0 0 60px rgba(102, 126, 234, 0.2);
465 | margin-bottom: 32px;
466 | border: 2px solid rgba(102, 126, 234, 0.3);
467 | animation: fadeInUp 0.8s cubic-bezier(0.16, 1, 0.3, 1);
468 | transition: all 0.5s cubic-bezier(0.175, 0.885, 0.32, 1.275);
469 | position: relative;
470 | overflow: visible;
471 | transform-style: preserve-3d;
472 | word-wrap: break-word;
473 | overflow-wrap: break-word;
474 | }
475 |
476 | .card::before {
477 |     content: '';
478 |     position: absolute;
479 |     top: 0;
480 |     left: 0;
481 |     right: 0;
482 |     bottom: 0;
483 |     background: radial-gradient(circle at 20% 50%, rgba(102, 126, 234, 0.15), transparent 60%);
484 |     opacity: 0;
485 |     transition: opacity 0.5s ease;
486 |     pointer-events: none;
487 | }
488 |
489 | .card:hover {
490 |     transform: translateY(-8px) ;
491 |     box-shadow:
492 |         0 40px 100px rgba(0, 0, 0, 0.6),
493 |         0 0 100px rgba(102, 126, 234, 0.4),
494 |         inset 0 1px 0 rgba(255, 255, 255, 0.15);
495 |     border-color: rgba(102, 126, 234, 0.5);
496 | }
497 |
498 | .card:hover::before {
499 |     opacity: 1;
500 | }

```

File 23: frontend/src/App.tsx

```
1  /**
2   * MoneySplit React Frontend Application
3   *
4   * AI ASSISTANCE DISCLOSURE:
5   * This entire frontend was developed with AI assistance (ChatGPT/Claude).
6   * - Prompts used: "Create a React TypeScript app with sidebar navigation for MoneySplit"
7   * - Prompts used: "Build Dashboard page with statistics cards and recent projects table"
8   * - Prompts used: "Create Projects page with form for creating tax calculation projects"
9   * - Prompts used: "Add responsive sidebar with gradient purple theme"
10  * - AI helped with React hooks, state management, TypeScript types, and CSS styling
11  */
12  import React, { useState } from 'react';
13  import './App.css';
14  import Dashboard from './pages/Dashboard';
15  import Projects from './pages/Projects';
16  import Reports from './pages/Reports';
17  import RecordsManagement from './pages/RecordsManagement';
18  import TaxBracketsManagement from './pages/TaxBracketsManagement';
19
20  type Page = 'dashboard' | 'projects' | 'reports' | 'records' | 'tax-brackets';
21
22  function App() {
23    const [currentPage, setCurrentPage] = useState<Page>('dashboard');
24    const [sidebarCollapsed, setSidebarCollapsed] = useState(false);
25
26    const renderPage = () => {
27      switch (currentPage) {
28        case 'dashboard':
29          return <Dashboard />;
30        case 'projects':
31          return <Projects />;
32        case 'reports':
33          return <Reports />;
34        case 'records':
35          return <RecordsManagement />;
36        case 'tax-brackets':
37          return <TaxBracketsManagement />;
38        default:
39          return <Dashboard />;
40      }
41    };
42
43    return (
44      <div className="app">
45        <button
46          className="menu-toggle"
47          onClick={() => setSidebarCollapsed(!sidebarCollapsed)}
48          aria-label="Toggle menu"
49        />
50        <svg viewBox="0 0 24 24">
51          {sidebarCollapsed ? (
52            <path d="M3 12h18M3 6h18M3 18h18" strokeLinecap="round" strokeLinejoin="round" />
53          ) : (
54            <path d="M6 18L18 6M6 6L18 18" strokeLinecap="round" strokeLinejoin="round" />
55          )}
56        </svg>
57      </button>
58      <div className="top-logo">
59        <span className="top-logo-emoji">■</span>
60        <span className="top-logo-text">MoneySplit</span>
61      </div>
62      <aside className={`sidebar ${sidebarCollapsed ? 'collapsed' : ''}`}>
63        <h1></h1>
64        <span className="logo-emoji">■</span>
65        <span>MoneySplit</span>
66        <h1></h1>
67        <nav className="nav-menu">
68          <div
69            className={`nav-item ${currentPage === 'dashboard' ? 'active' : ''}`}
70            onClick={() => setCurrentPage('dashboard')}
71          >
72            <span>■</span>
73            <span>Dashboard</span>
74          </div>
75          <div
76            className={`nav-item ${currentPage === 'projects' ? 'active' : ''}`}
77            onClick={() => setCurrentPage('projects')}
78          >
79            <span>■</span>
80            <span>Projects</span>
81          </div>
82          <div
83            className={`nav-item ${currentPage === 'reports' ? 'active' : ''}`}
84            onClick={() => setCurrentPage('reports')}
85          >
86            <span>■</span>
87            <span>Reports</span>
88          </div>
89          <div
90            className={`nav-item ${currentPage === 'records' ? 'active' : ''}`}
91            onClick={() => setCurrentPage('records')}
92          >
93            <span>■</span>
94            <span>Records</span>
95          </div>
96          <div
97            className={`nav-item ${currentPage === 'tax-brackets' ? 'active' : ''}`}
98            onClick={() => setCurrentPage('tax-brackets')}
99          >
100             <span>■</span>
101             <span>Tax Brackets</span>
102           </div>
103         </nav>
104       </aside>
105       <div>
106         <div>
107           <div>
108             <div>
109               <div>
110                 <div>
111                   <div>
112                     <div>
113                       <div>
114                         <div>
115                           <div>
116                             <div>
117                               <div>
118                                 <div>
119                                   <div>
120                                     <div>
121                                       <div>
122                                         <div>
123                                           <div>
124                                             <div>
125                                             </div>
126                                           </div>
127                                         </div>
128                                       </div>
129                                     </div>
130                                   </div>
131                                 </div>
132                               </div>
133                             </div>
134                           </div>
135                         </div>
136                       </div>
137                     </div>
138                   </div>
139                 </div>
140               </div>
141             </div>
142           </div>
143         </div>
144       </div>
145     );
146  }
```

```

75 |         &gt;
76 |         <span>■</span>
77 |         <span>New Project</span>
78 |     </div>
79 | <div>
80 |     className={`nav-item ${currentPage === 'reports' ? 'active' : ''}`}
81 |     onClick={() => setCurrentPage('reports')}
82 | <div>
83 |     <span>■</span>
84 |     <span>Reports</span>
85 | </div>
86 | <div>
87 |     className={`nav-item ${currentPage === 'records' ? 'active' : ''}`}
88 |     onClick={() => setCurrentPage('records')}
89 | <div>
90 |     <span>■</span>
91 |     <span>Manage Records</span>
92 | </div>
93 | <div>
94 |     className={`nav-item ${currentPage === 'tax-brackets' ? 'active' : ''}`}
95 |     onClick={() => setCurrentPage('tax-brackets')}
96 | <div>
97 |     <span>■</span>
98 |     <span>Tax Brackets</span>
99 | </div>
100 | </nav>
101 | </aside>
102 | <main className="main-content">
103 |     {renderPage()}
104 | </main>
105 | </div>
106 | );
107 | }
108 |
109 | export default App;
110 |

```


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

```
1 | import axios from 'axios';
2 |
3 | const API_BASE_URL = process.env.REACT_APP_API_URL || 'http://localhost:8000/api';
4 |
5 | export const apiClient = axios.create({
6 |   baseURL: API_BASE_URL,
7 |   headers: {
8 |     'Content-Type': 'application/json',
9 |   },
10 | });
11 |
12 | // Types
13 | export interface Person {
14 |   name: string;
15 |   work_share: number;
16 | }
17 |
18 | export interface ProjectCreate {
19 |   num_people: number;
20 |   revenue: number;
21 |   costs: number[];
22 |   country: string;
23 |   tax_type: 'Individual' | 'Business';
24 |   people: Person[];
25 | }
26 |
27 | export interface Record {
28 |   id: number;
29 |   num_people: number;
30 |   revenue: number;
31 |   total_costs: number;
32 |   group_income: number;
33 |   individual_income: number;
34 |   tax_origin: string;
35 |   tax_option: string;
36 |   tax_amount: number;
37 |   net_income_per_person: number;
38 |   net_income_group: number;
39 |   created_at: string;
40 | }
41 |
42 | export interface PersonResponse {
43 |   id: number;
44 |   name: string;
45 |   work_share: number;
46 |   gross_income: number;
47 |   tax_paid: number;
48 |   net_income: number;
49 | }
50 |
51 | export interface RecordWithPeople extends Record {
52 |   people: PersonResponse[];
53 | }
54 |
55 | export interface Statistics {
56 |   total_records: number;
57 |   total_revenue: number;
58 |   total_costs: number;
59 |   total_tax: number;
60 |   total_net_income: number;
61 |   average_tax_rate: number;
62 |   total_people_entries: number;
63 |   unique_people: number;
64 | }
65 |
66 | export interface Forecast {
67 |   success: boolean;
68 |   predictions: {
69 |     month: string;
70 |     revenue: number;
71 |     confidence: string;
72 |     lower_bound: number;
73 |     upper_bound: number;
74 |     range: string;
```

```

75 |     }[];
76 |     trend: string;
77 |     trend_strength: number;
78 |     r2_score: number;
79 |     confidence: string;
80 |     confidence_description: string;
81 |     historical_avg: number;
82 |     model_slope: number;
83 |     growth_rate: number;
84 |     model_type: string;
85 |     explanation: string;
86 |     data_quality: string;
87 |     recommendations?: string[];
88 | }
89 |
90 | // API Functions
91 | export interface RecordUpdate {
92 |     field: string;
93 |     value: string | number;
94 | }
95 |
96 | export interface TaxBracket {
97 |     id: number;
98 |     country: string;
99 |     tax_type: string;
100 |     income_limit: number;
101 |     rate: number;
102 | }
103 |
104 | export interface TaxBracketCreate {
105 |     country: string;
106 |     tax_type: string;
107 |     income_limit: number;
108 |     rate: number;
109 | }
110 |
111 | export const projectsApi = {
112 |     create: (data: ProjectCreate) => apiClient.post('/projects', data),
113 |     getRecords: (limit = 10) => apiClient.get<Record[]>(`/records?limit=${limit}`),
114 |     getRecord: (id: number) => apiClient.get<RecordWithPeople>(`/records/${id}`),
115 |     updateRecord: (id: number, update: RecordUpdate) => apiClient.put(`/records/${id}`, update),
116 |     deleteRecord: (id: number) => apiClient.delete(`/records/${id}`),
117 | };
118 |
119 | export const taxBracketsApi = {
120 |     getTaxBrackets: (country: string, taxType: string) => apiClient.get<TaxBracket[]>(`/tax-brackets?country=${co...
121 |     createTaxBracket: (data: TaxBracketCreate) => apiClient.post('/tax-brackets', data),
122 |     deleteTaxBracket: (id: number) => apiClient.delete(`/tax-brackets/${id}`),
123 | };
124 |
125 | export const reportsApi = {
126 |     getStatistics: () => apiClient.get<Statistics>('/reports/statistics'),
127 |     getRevenueSummary: () => apiClient.get('/reports/revenue-summary'),
128 |     getTopPeople: (limit = 10) => apiClient.get(`/reports/top-people?limit=${limit}`),
129 | };
130 |
131 | export const forecastApi = {
132 |     getRevenueForecast: (months = 3) => apiClient.get<Forecast>(`/forecast/revenue?months=${months}`),
133 |     getComprehensive: () => apiClient.get('/forecast/comprehensive'),
134 |     getTaxOptimization: () => apiClient.get('/forecast/tax-optimization'),
135 |     getTrends: () => apiClient.get('/forecast/trends'),
136 | };
137 |
138 | export const exportApi = {
139 |     exportRecordPDF: (id: number) => `${API_BASE_URL}/export/record/${id}/pdf`,
140 |     exportSummaryPDF: () => `${API_BASE_URL}/export/summary/pdf`,
141 |     exportForecastPDF: () => `${API_BASE_URL}/export/forecast/pdf`,
142 | };
143 |
144 | export const visualizationApi = {
145 |     getRevenueSummary: () => `${API_BASE_URL}/visualizations/revenue-summary`,
146 |     getMonthlyTrends: () => `${API_BASE_URL}/visualizations/monthly-trends`,
147 |     getWorkDistribution: () => `${API_BASE_URL}/visualizations/work-distribution`,
148 |     getTaxComparison: () => `${API_BASE_URL}/visualizations/tax-comparison`,
149 |     getPersonPerformance: (name: string) => `${API_BASE_URL}/visualizations/person-performance/${name}`,
150 |     getProjectProfitability: () => `${API_BASE_URL}/visualizations/project-profitability`,
151 | };

```


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

```
1 | import React, { useEffect, useState } from 'react';
2 | import { reportsApi, projectsApi, Statistics, Record } from '../api/client';
3 |
4 | // Format dollar amounts - round down to nearest dollar
5 | const formatCurrency = (amount: number): string => {
6 |   return Math.floor(amount).toLocaleString();
7 | };
8 |
9 | // Determine font size based on string length
10 | const getValueLength = (value: string | number): string => {
11 |   const strValue = String(value);
12 |   if (strValue.length > 10) return 'very-long';
13 |   if (strValue.length > 7) return 'long';
14 |   return 'normal';
15 | };
16 |
17 | const Dashboard: React.FC = () => {
18 |   const [stats, setStats] = useState<Statistics | null>(null);
19 |   const [recentRecords, setRecentRecords] = useState<Record[]>([]);
20 |   const [loading, setLoading] = useState(true);
21 |
22 |   useEffect(() => {
23 |     const fetchData = async () => {
24 |       try {
25 |         const [statsRes, recordsRes] = await Promise.all([
26 |           reportsApi.getStatistics(),
27 |           projectsApi.getRecords(5),
28 |         ]);
29 |         setStats(statsRes.data);
30 |         setRecentRecords(recordsRes.data);
31 |       } catch (error) {
32 |         console.error('Error fetching dashboard data:', error);
33 |       } finally {
34 |         setLoading(false);
35 |       }
36 |     };
37 |
38 |     fetchData();
39 |   }, []);
40 |
41 |   if (loading) {
42 |     return (
43 |       <div className="loading">
44 |         <div className="spinner"></div>
45 |       </div>
46 |     );
47 |   }
48 |
49 |   return (
50 |     <div>
51 |       <div className="page-header">
52 |         <h2>Dashboard</h2>
53 |         <p>Overview of your commission splitting business</p>
54 |       </div>
55 |
56 |       <div className="stats-grid">
57 |         <div className="stat-card">
58 |           <div style={{ fontSize: '36px', marginBottom: '8px' }}></div>
59 |           <h4>Total Revenue</h4>
60 |           <div className="stat-value" data-length={stats ? getValueLength(`${formatCurrency(stats.
total_revenue)}`) ...
61 |             ${stats ? formatCurrency(stats.total_revenue) : 0}
62 |           </div>
63 |           <div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}>
64 |             All time earnings
65 |           </div>
66 |         </div>
67 |         <div className="stat-card">
68 |           <div style={{ fontSize: '36px', marginBottom: '8px' }}></div>
69 |           <h4>Total Projects</h4>
70 |           <div className="stat-value" data-length={getValueLength(stats?.total_records || 0)}>
71 |             {stats?.total_records || 0}
72 |           </div>
73 |           <div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}>
```

```

74 |         Completed projects
75 |         </div>
76 |     </div>
77 |     <div className="stat-card">
78 |         <div style={{ fontSize: '36px', marginBottom: '8px' }}>■</div>
79 |         <h4>Tax Paid</h4>
80 |         <div className="stat-value" data-length={stats ? getValueLength(`${formatCurrency(stats.
total_tax)})` : 'n...
81 |             ${stats ? formatCurrency(stats.total_tax) : 0}
82 |         </div>
83 |         <div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}>
84 |             Total contributions
85 |         </div>
86 |     </div>
87 |     <div className="stat-card">
88 |         <div style={{ fontSize: '36px', marginBottom: '8px' }}>■</div>
89 |         <h4>Net Income</h4>
90 |         <div className="stat-value" data-length={stats ? getValueLength(`${formatCurrency(stats.
total_net_income)})` : 'n...
91 |             ${stats ? formatCurrency(stats.total_net_income) : 0}
92 |         </div>
93 |         <div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}>
94 |             After tax profits
95 |         </div>
96 |     </div>
97 | </div>
98 |
99 | <div className="card">
100 |     <h3 style={{ marginBottom: '24px' }}>■ Recent Projects</h3>
101 |     {recentRecords.length > 0 ? (
102 |         <div style={{ overflowX: 'auto' }}>
103 |             <table className="table">
104 |                 <thead>
105 |                     <tr>
106 |                         <th>ID</th>
107 |                         <th>Date</th>
108 |                         <th>Country</th>
109 |                         <th>Tax Type</th>
110 |                         <th>Revenue</th>
111 |                         <th>Net Income</th>
112 |                     </tr>
113 |                 </thead>
114 |                 <tbody>
115 |                     {recentRecords.map((record) => (
116 |                         <tr key={record.id}>
117 |                             <td>
118 |                                 <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 |                                 }}>
126 |                                     #{record.id}
127 |                                 </span>
128 |                             </td>
129 |                             <td>{new Date(record.created_at).toLocaleDateString()}</td>
130 |                             <td>
131 |                                 <span style={{ fontWeight: 600 }}>{record.tax_origin}</span>
132 |                             </td>
133 |                             <td>
134 |                                 <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 |                                 }}>
146 |                                     {record.tax_option}
147 |                                 </span>
148 |                             </td>
149 |                             <td style={{ fontWeight: 600, color: '#2d3748' }}>

```

```

150 |             ${formatCurrency(record.revenue)}
151 |         </td>>
152 |         <td style={{
153 |             fontWeight: 700,
154 |             background: 'linear-gradient(135deg, #667eea 0%, #764ba2 100%)',
155 |             WebkitBackgroundClip: 'text',
156 |             WebkitTextFillColor: 'transparent'
157 |         }}>
158 |             ${formatCurrency(record.net_income_group)}
159 |         </td>>
160 |     </tr>>
161 | ))}
162 | </tbody>>
163 | </table>>
164 | </div>>
165 | ) : (
166 |     <div style={{
167 |         textAlign: 'center',
168 |         padding: '48px 24px',
169 |         color: '#718096'
170 |     }}>
171 |         <div style={{ fontSize: '64px', marginBottom: '16px' }}>&#9632;</div>
172 |         <p style={{ fontSize: '18px', fontWeight: 500 }}>No projects yet</p>
173 |         <p style={{ fontSize: '14px', marginTop: '8px' }}>Create your first project to get started!</p>
174 |     </div>
175 | )}
176 | </div>
177 |
178 | <div className="stats-grid">
179 |     <div className="stat-card">
180 |         <div style={{ fontSize: '36px', marginBottom: '8px' }}>&#9632;</div>
181 |         <h4>Average Tax Rate</h4>
182 |         <div className="stat-value">{stats?.average_tax_rate.toFixed(1)}</div>
183 |         <div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}>
184 |             Effective rate
185 |         </div>
186 |     </div>
187 |     <div className="stat-card">
188 |         <div style={{ fontSize: '36px', marginBottom: '8px' }}>&#9632;</div>
189 |         <h4>Unique People</h4>
190 |         <div className="stat-value">{stats?.unique_people || 0}</div>
191 |         <div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}>
192 |             Team members
193 |         </div>
194 |     </div>
195 |     <div className="stat-card">
196 |         <div style={{ fontSize: '36px', marginBottom: '8px' }}>&#9632;</div>
197 |         <h4>Total Costs</h4>
198 |         <div className="stat-value" data-length={stats ? getValueLength(`${formatCurrency(stats.
total_costs)}`) : ...
199 |             ${stats ? formatCurrency(stats.total_costs) : 0}
200 |         </div>
201 |         <div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}>
202 |             Business expenses
203 |         </div>
204 |     </div>
205 |     <div className="stat-card">
206 |         <div style={{ fontSize: '36px', marginBottom: '8px' }}>&#9632;</div>
207 |         <h4>Profit Margin</h4>
208 |         <div className="stat-value" data-length="normal">
209 |             {stats
210 |               ? Math.floor((stats.total_net_income / stats.total_revenue) * 100)
211 |               : 0}%
212 |         </div>
213 |         <div style={{ fontSize: '12px', color: '#718096', marginTop: '8px', fontWeight: 500 }}>
214 |             Net profit ratio
215 |         </div>
216 |     </div>
217 | </div>
218 | </div>
219 | );
220 | };
221 |
222 | export default Dashboard;
223 |

```


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

```
1 | import React, { useState } from 'react';
2 | import { projectsApi, ProjectCreate, Person } from '../api/client';
3 |
4 | const Projects: React.FC = () => {
5 |   const [numPeople, setNumPeople] = useState<number>(2);
6 |   const [revenue, setRevenue] = useState<string>('');
7 |   const [costs, setCosts] = useState<string>('');
8 |   const [country, setCountry] = useState<string>('US');
9 |   const [taxType, setTaxType] = useState<'Individual' | 'Business'>('Individual');
10 |   const [people, setPeople] = useState<Person[]>([
11 |     { name: '', work_share: 0.5 },
12 |     { name: '', work_share: 0.5 },
13 |   ]);
14 |   const [successMessage, setSuccessMessage] = useState<string>('');
15 |   const [errorMessage, setErrorMessage] = useState<string>('');
16 |   const [loading, setLoading] = useState(false);
17 |
18 |   const handleNumPeopleChange = (value: number) => {
19 |     setNumPeople(value);
20 |     const newPeople: Person[] = [];
21 |     const sharePerPerson = 1 / value;
22 |
23 |     for (let i = 0; i < value; i++) {
24 |       newPeople.push({
25 |         name: people[i]?.name || '',
26 |         work_share: sharePerPerson,
27 |       });
28 |     }
29 |     setPeople(newPeople);
30 |   };
31 |
32 |   const handlePersonChange = (index: number, field: keyof Person, value: string | number) => {
33 |     const newPeople = [...people];
34 |     newPeople[index] = { ...newPeople[index], [field]: value };
35 |     setPeople(newPeople);
36 |   };
37 |
38 |   const handleSubmit = async (e: React.FormEvent) => {
39 |     e.preventDefault();
40 |     setSuccessMessage('');
41 |     setErrorMessage('');
42 |     setLoading(true);
43 |
44 |     try {
45 |       const costsArray = costs.split(',').map(c => parseFloat(c.trim()));
46 |
47 |       const data: ProjectCreate = {
48 |         num_people: numPeople,
49 |         revenue: parseFloat(revenue),
50 |         costs: costsArray,
51 |         country,
52 |         tax_type: taxType,
53 |         people,
54 |       };
55 |
56 |       const response = await projectsApi.create(data);
57 |       setSuccessMessage(`Project created successfully! Record ID: ${response.data.record_id}`);
58 |
59 |       // Reset form
60 |       setRevenue('');
61 |       setCosts('');
62 |       handleNumPeopleChange(2);
63 |     } catch (error: any) {
64 |       setErrorMessage(error.response?.data?.detail || 'Error creating project');
65 |     } finally {
66 |       setLoading(false);
67 |     }
68 |   };
69 |
70 |   return (
71 |     <div>
72 |       <div className="page-header">
73 |         <h2>■ New Project</h2>
74 |         <p>Create a new commission splitting project</p>
```



```

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

```

```

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

```

File 27: requirements.txt

```
1 | # Core dependencies
2 | plotly==6.3.0
3 |
4 | # API dependencies
5 | fastapi==0.116.1
6 | uvicorn==0.35.0
7 | pydantic==2.11.7
8 |
9 | # PDF generation
10 | reportlab==4.4.4
11 |
12 | # Machine Learning & Data Science
13 | scikit-learn>=1.0.0
14 | numpy>=1.20.0
15 | pandas>=2.0.0
16 |
17 | # Python version requirement
18 | # Python >= 3.8
19 |
```

File 28: tests/__init__.py

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

File 29: tests/conftest.py

```
1 | """Pytest configuration and fixtures."""
2 |
3 | import pytest
4 | import os
5 | import sys
6 | import sqlite3
7 |
8 | # Add parent directory to path for imports
9 | sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))
10 |
11 | # Use same database as production for school project
12 | # os.environ['TESTING'] = '1'
13 | # os.environ['TEST_DB'] = 'test_example.db'
14 |
15 |
16 | @pytest.fixture(scope="session", autouse=True)
17 | def test_database():
18 |     """Use production database for school project."""
19 |     # No test database setup - using production database
20 |     yield "example.db"
21 |
22 |
23 | @pytest.fixture
24 | def sample_project_data():
25 |     """Provide sample project data for tests."""
26 |     return {
27 |         "num_people": 2,
28 |         "revenue": 10000,
29 |         "costs": [1000, 500, 300],
30 |         "country": "US",
31 |         "tax_type": "Individual",
32 |         "people": [
33 |             {"name": "Alice", "work_share": 0.6},
34 |             {"name": "Bob", "work_share": 0.4}
35 |         ]
36 |     }
37 |
38 |
39 | @pytest.fixture
40 | def sample_tax_brackets():
41 |     """Provide sample tax brackets for tests."""
42 |     return [
43 |         {'min': 0, 'max': 11000, 'rate': 0.10},
44 |         {'min': 11000, 'max': 44725, 'rate': 0.12},
45 |         {'min': 44725, 'max': 95375, 'rate': 0.22},
46 |         {'min': 95375, 'max': 182100, 'rate': 0.24}
47 |     ]
48 |
```

File 30: tests/test_api.py

```
1  """
2  Integration tests for FastAPI endpoints.
3
4  AI ASSISTANCE DISCLOSURE:
5  All tests in this file were generated with AI assistance (ChatGPT/Claude).
6  - Prompts used: "Write pytest tests for FastAPI CRUD endpoints"
7  - Prompts used: "Create integration tests for forecasting and export endpoints"
8  - Prompts used: "Add edge case tests for invalid inputs and error handling"
9  - AI helped structure test cases, assertions, and test data fixtures
10 """
11
12 import pytest
13 from fastapi.testclient import TestClient
14 from api.main import app
15
16 client = TestClient(app)
17
18
19 class TestProjectEndpoints:
20     """Test project CRUD endpoints."""
21
22     def test_create_project_success(self):
23         """Test successful project creation."""
24         payload = {
25             "num_people": 2,
26             "revenue": 10000,
27             "costs": [1000, 500],
28             "country": "US",
29             "tax_type": "Individual",
30             "people": [
31                 {"name": "Alice", "work_share": 0.6},
32                 {"name": "Bob", "work_share": 0.4}
33             ]
34         }
35
36         response = client.post("/api/projects", json=payload)
37
38         assert response.status_code == 201 # FastAPI returns 201 for created
39         data = response.json()
40         assert "record_id" in data
41         assert data["message"] == "Project created successfully"
42
43     def test_create_project_invalid_work_shares(self):
44         """Test project creation with invalid work shares."""
45         payload = {
46             "num_people": 2,
47             "revenue": 10000,
48             "costs": [1000],
49             "country": "US",
50             "tax_type": "Individual",
51             "people": [
52                 {"name": "Alice", "work_share": 0.7},
53                 {"name": "Bob", "work_share": 0.7} # Sums to 1.4
54             ]
55         }
56
57         response = client.post("/api/projects", json=payload)
58         assert response.status_code == 422 # FastAPI validation error
59
60     def test_create_project_negative_revenue(self):
61         """Test project creation with negative revenue."""
62         payload = {
63             "num_people": 1,
64             "revenue": -5000,
65             "costs": [1000],
66             "country": "US",
67             "tax_type": "Individual",
68             "people": [
69                 {"name": "Alice", "work_share": 1.0}
70             ]
71         }
72
73         response = client.post("/api/projects", json=payload)
74         assert response.status_code == 422 # Validation error
```

```

75 |
76 | def test_get_recent_records(self):
77 |     """Test fetching recent records."""
78 |     response = client.get("/api/records")
79 |
80 |     assert response.status_code == 200
81 |     data = response.json()
82 |     assert isinstance(data, list)
83 |
84 | def test_get_record_by_id(self):
85 |     """Test fetching specific record by ID."""
86 |     # First create a record
87 |     payload = {
88 |         "num_people": 1,
89 |         "revenue": 5000,
90 |         "costs": [500],
91 |         "country": "US",
92 |         "tax_type": "Individual",
93 |         "people": [
94 |             {"name": "Test User", "work_share": 1.0}
95 |         ]
96 |     }
97 |
98 |     create_response = client.post("/api/projects", json=payload)
99 |     record_id = create_response.json()["record_id"]
100 |
101 |     # Fetch the record
102 |     response = client.get(f"/api/records/{record_id}")
103 |
104 |     assert response.status_code == 200
105 |     data = response.json()
106 |     assert data["id"] == record_id # API uses "id" not "record_id"
107 |
108 | def test_get_nonexistent_record(self):
109 |     """Test fetching a non-existent record."""
110 |     response = client.get("/api/records/999999")
111 |
112 |     assert response.status_code == 404
113 |
114 |
115 | class TestReportEndpoints:
116 |     """Test report and statistics endpoints."""
117 |
118 |     def test_get_statistics(self):
119 |         """Test overall statistics endpoint."""
120 |         response = client.get("/api/reports/statistics")
121 |
122 |         assert response.status_code == 200
123 |         data = response.json()
124 |         assert "total_revenue" in data
125 |         assert "total_records" in data # API uses "total_records" not "total_projects"
126 |         assert "total_tax" in data # API uses "total_tax" not "total_tax_paid"
127 |
128 |     def test_get_revenue_summary(self):
129 |         """Test revenue summary by year."""
130 |         response = client.get("/api/reports/revenue-summary")
131 |
132 |         assert response.status_code == 200
133 |         data = response.json()
134 |         assert isinstance(data, list)
135 |
136 |     def test_get_top_people(self):
137 |         """Test top contributors endpoint."""
138 |         response = client.get("/api/reports/top-people")
139 |
140 |         assert response.status_code == 200
141 |         data = response.json()
142 |         assert isinstance(data, list)
143 |         if len(data) > 0:
144 |             assert "name" in data[0]
145 |             assert "total_gross" in data[0] # API uses "total_gross" not "total_earned"
146 |
147 |
148 | class TestForecastingEndpoints:
149 |     """Test forecasting and ML endpoints."""
150 |
151 |     def test_revenue_forecast_default(self):
152 |         """Test revenue forecasting with default months."""

```

```

153 |         response = client.get("/api/forecast/revenue")
154 |
155 |         assert response.status_code == 200
156 |         data = response.json()
157 |         assert "predictions" in data
158 |         # May not have enough data, so just check structure exists
159 |
160 |     def test_revenue_forecast_custom_months(self):
161 |         """Test revenue forecasting with custom months."""
162 |         response = client.get("/api/forecast/revenue?months=6")
163 |
164 |         assert response.status_code == 200
165 |         data = response.json()
166 |         assert "predictions" in data
167 |         # May be empty if insufficient data
168 |
169 |     def test_comprehensive_forecast(self):
170 |         """Test comprehensive forecast endpoint."""
171 |         response = client.get("/api/forecast/comprehensive")
172 |
173 |         assert response.status_code == 200
174 |         data = response.json()
175 |         assert "revenue_forecast" in data
176 |         assert "recommendations" in data # API uses "recommendations" directly
177 |
178 |     def test_tax_optimization(self):
179 |         """Test tax optimization recommendations."""
180 |         response = client.get("/api/forecast/tax-optimization")
181 |
182 |         assert response.status_code == 200
183 |         data = response.json()
184 |         assert "tax_comparison" in data # API structure is different
185 |         assert "recommendations" in data
186 |
187 |     def test_trends_analysis(self):
188 |         """Test trend analysis endpoint."""
189 |         response = client.get("/api/forecast/trends")
190 |
191 |         assert response.status_code == 200
192 |         data = response.json()
193 |         # May not have enough data, check for success or message
194 |         assert "success" in data or "message" in data
195 |
196 |
197 |     class TestVisualizationEndpoints:
198 |         """Test visualization endpoints."""
199 |
200 |         def test_revenue_summary_visualization(self):
201 |             """Test revenue summary visualization."""
202 |             response = client.get("/api/visualizations/revenue-summary")
203 |
204 |             assert response.status_code == 200
205 |             assert "text/html" in response.headers["content-type"]
206 |             assert b"Revenue Summary" in response.content
207 |
208 |         def test_monthly_trends_visualization(self):
209 |             """Test monthly trends visualization."""
210 |             response = client.get("/api/visualizations/monthly-trends")
211 |
212 |             assert response.status_code == 200
213 |             assert "text/html" in response.headers["content-type"]
214 |
215 |         def test_work_distribution_visualization(self):
216 |             """Test work distribution visualization."""
217 |             response = client.get("/api/visualizations/work-distribution")
218 |
219 |             assert response.status_code == 200
220 |             assert "text/html" in response.headers["content-type"]
221 |
222 |         def test_tax_comparison_visualization(self):
223 |             """Test tax comparison visualization."""
224 |             response = client.get("/api/visualizations/tax-comparison")
225 |
226 |             assert response.status_code == 200
227 |             assert "text/html" in response.headers["content-type"]
228 |
229 |         def test_profitability_visualization(self):
230 |             """Test project profitability visualization."""

```



```

231 |         response = client.get("/api/visualizations/project-profitability")
232 |
233 |         assert response.status_code == 200
234 |         assert "text/html" in response.headers["content-type"]
235 |
236 |
237 | class TestPDFExportEndpoints:
238 |     """Test PDF export endpoints."""
239 |
240 |     def test_export_record_pdf(self):
241 |         """Test exporting a record as PDF."""
242 |         # First create a record
243 |         payload = {
244 |             "num_people": 1,
245 |             "revenue": 5000,
246 |             "costs": [500],
247 |             "country": "US",
248 |             "tax_type": "Individual",
249 |             "people": [
250 |                 {"name": "PDF Test User", "work_share": 1.0}
251 |             ]
252 |         }
253 |
254 |         create_response = client.post("/api/projects", json=payload)
255 |         record_id = create_response.json()["record_id"]
256 |
257 |         # Export as PDF
258 |         response = client.get(f"/api/export/record/{record_id}/pdf")
259 |
260 |         assert response.status_code == 200
261 |         assert response.headers["content-type"] == "application/pdf"
262 |
263 |     def test_export_summary_pdf(self):
264 |         """Test exporting summary as PDF."""
265 |         response = client.get("/api/export/summary/pdf")
266 |
267 |         assert response.status_code == 200
268 |         assert response.headers["content-type"] == "application/pdf"
269 |
270 |     def test_export_forecast_pdf(self):
271 |         """Test exporting forecast as PDF."""
272 |         response = client.get("/api/export/forecast/pdf")
273 |
274 |         assert response.status_code == 200
275 |         assert response.headers["content-type"] == "application/pdf"
276 |
277 |
278 | class TestHealthCheck:
279 |     """Test health check endpoint."""
280 |
281 |     def test_root_endpoint(self):
282 |         """Test API root endpoint."""
283 |         response = client.get("/")
284 |
285 |         assert response.status_code == 200
286 |         # Root may return HTML, not JSON
287 |         assert response.text is not None
288 |

```

File 31: tests/test_backend_logic.py

```
1  """
2  Unit tests for backend business logic.
3
4  AI ASSISTANCE DISCLOSURE:
5  All tests in this file were generated with AI assistance (ChatGPT/Claude).
6  - Prompts used: "Create unit tests for tax calculation functions"
7  - Prompts used: "Write tests for work share distribution logic"
8  - Prompts used: "Add parameterized tests for different income levels"
9  - AI helped create comprehensive test coverage for all business logic functions
10 """
11
12 import pytest
13 from Logic.tax_calculator import calculate_tax, calculate_tax_from_db, split_work_shares, calculate_profit
14 from Logic.validators import (
15     validate_positive_number,
16     validate_work_shares,
17     validate_work_share,
18     validate_non_empty_string,
19     validate_country,
20     validate_tax_type,
21     validate_tax_rate,
22     ValidationError
23 )
24
25
26 class TestTaxCalculations:
27     """Test tax calculation logic."""
28
29     def test_calculate_progressive_tax(self):
30         """Test progressive tax calculation with mock brackets."""
31         # Mock tax brackets: 10% up to 10k, 20% above 10k
32         from unittest.mock import patch
33
34         mock_brackets = [(10000, 0.10), (50000, 0.20)]
35
36         with patch('Logic.tax_calculator.setup.get_tax_brackets', return_value=mock_brackets):
37             # Income of 15000: (10000 * 0.10) + (5000 * 0.20) = 1000 + 1000 = 2000
38             tax = calculate_tax_from_db(15000, "US", "Individual")
39             assert tax == 2000.0
40
41     def test_calculate_tax_single_bracket(self):
42         """Test tax calculation within single bracket."""
43         from unittest.mock import patch
44
45         mock_brackets = [(10000, 0.10), (50000, 0.20)]
46
47         with patch('Logic.tax_calculator.setup.get_tax_brackets', return_value=mock_brackets):
48             # Income of 5000: entirely in first bracket
49             tax = calculate_tax_from_db(5000, "US", "Individual")
50             assert tax == 500.0 # 5000 * 0.10
51
52     def test_calculate_tax_zero_income(self):
53         """Test tax calculation for zero income."""
54         from unittest.mock import patch
55
56         mock_brackets = [(10000, 0.10)]
57
58         with patch('Logic.tax_calculator.setup.get_tax_brackets', return_value=mock_brackets):
59             tax = calculate_tax_from_db(0, "US", "Individual")
60             assert tax == 0.0
61
62     def test_calculate_tax_direct(self):
63         """Test direct tax calculation without DB."""
64         brackets = [(10000, 0.10), (50000, 0.20)]
65
66         # Income of 15000: (10000 * 0.10) + (5000 * 0.20) = 1000 + 1000 = 2000
67         tax = calculate_tax(15000, brackets)
68         assert tax == 2000.0
69
70         # Income of 5000: entirely in first bracket
71         tax = calculate_tax(5000, brackets)
72         assert tax == 500.0
73
74
```

```

75 | class TestWorkShareDistribution:
76 |     """Test work share distribution logic."""
77 |
78 |     def test_equal_split_two_people(self):
79 |         """Test equal work share split between two people."""
80 |         profit = 10000
81 |         work_shares = [0.5, 0.5]
82 |
83 |         distribution = split_work_shares(profit, work_shares)
84 |
85 |         assert len(distribution) == 2
86 |         assert distribution[0] == 5000.0
87 |         assert distribution[1] == 5000.0
88 |
89 |     def test_unequal_split_three_people(self):
90 |         """Test unequal work share split."""
91 |         profit = 10000
92 |         work_shares = [0.5, 0.3, 0.2]
93 |
94 |         distribution = split_work_shares(profit, work_shares)
95 |
96 |         assert len(distribution) == 3
97 |         assert distribution[0] == 5000.0
98 |         assert distribution[1] == 3000.0
99 |         assert distribution[2] == 2000.0
100 |
101 |
102 | class TestValidators:
103 |     """Test input validation functions."""
104 |
105 |     def test_validate_positive_number_valid(self):
106 |         """Test positive number validation with valid value."""
107 |         result = validate_positive_number(10000, "Revenue")
108 |         assert result == 10000
109 |
110 |     def test_validate_positive_number_zero(self):
111 |         """Test positive number validation with zero."""
112 |         result = validate_positive_number(0, "Revenue")
113 |         assert result == 0
114 |
115 |     def test_validate_positive_number_negative(self):
116 |         """Test positive number validation with negative value."""
117 |         with pytest.raises(ValidationError):
118 |             validate_positive_number(-5000, "Revenue")
119 |
120 |     def test_validate_work_shares_valid(self):
121 |         """Test work shares validation with valid shares."""
122 |         # Should not raise exception
123 |         validate_work_shares([0.5, 0.5])
124 |         validate_work_shares([0.6, 0.4])
125 |         validate_work_shares([0.33, 0.33, 0.34])
126 |
127 |     def test_validate_work_shares_invalid_sum(self):
128 |         """Test work shares validation with invalid sum."""
129 |         with pytest.raises(ValidationError):
130 |             validate_work_shares([0.6, 0.6]) # Sums to 1.2
131 |
132 |         with pytest.raises(ValidationError):
133 |             validate_work_shares([0.3, 0.3]) # Sums to 0.6
134 |
135 |     def test_validate_work_share_valid(self):
136 |         """Test single work share validation."""
137 |         assert validate_work_share(0.5) == 0.5
138 |         assert validate_work_share(0.0) == 0.0
139 |         assert validate_work_share(1.0) == 1.0
140 |
141 |     def test_validate_work_share_invalid(self):
142 |         """Test single work share validation with invalid values."""
143 |         with pytest.raises(ValidationError):
144 |             validate_work_share(-0.1)
145 |
146 |         with pytest.raises(ValidationError):
147 |             validate_work_share(1.5)
148 |
149 |     def test_validate_non_empty_string_valid(self):
150 |         """Test non-empty string validation."""
151 |         assert validate_non_empty_string("Alice", "Name") == "Alice"
152 |         assert validate_non_empty_string(" Bob ", "Name") == "Bob"

```

```

153 |
154 | def test_validate_non_empty_string_invalid(self):
155 |     """Test non-empty string validation with empty string."""
156 |     with pytest.raises(ValidationError):
157 |         validate_non_empty_string("", "Name")
158 |
159 |     with pytest.raises(ValidationError):
160 |         validate_non_empty_string(" ", "Name")
161 |
162 | def test_validate_country_valid(self):
163 |     """Test country validation."""
164 |     assert validate_country("US") == "US"
165 |     assert validate_country("Spain") == "Spain"
166 |
167 | def test_validate_country_invalid(self):
168 |     """Test country validation with empty string."""
169 |     with pytest.raises(ValidationError):
170 |         validate_country("")
171 |
172 | def test_validate_tax_type_valid(self):
173 |     """Test tax type validation."""
174 |     assert validate_tax_type("Individual") == "Individual"
175 |     assert validate_tax_type("Business") == "Business"
176 |     assert validate_tax_type("individual") == "Individual" # Case insensitive
177 |
178 | def test_validate_tax_type_invalid(self):
179 |     """Test tax type validation with invalid type."""
180 |     with pytest.raises(ValidationError):
181 |         validate_tax_type("Corporate")
182 |
183 | def test_validate_tax_rate_valid(self):
184 |     """Test tax rate validation."""
185 |     assert validate_tax_rate(0.10) == 0.10
186 |     assert validate_tax_rate(0.0) == 0.0
187 |     assert validate_tax_rate(1.0) == 1.0
188 |
189 | def test_validate_tax_rate_invalid(self):
190 |     """Test tax rate validation with invalid values."""
191 |     with pytest.raises(ValidationError):
192 |         validate_tax_rate(-0.1)
193 |
194 |     with pytest.raises(ValidationError):
195 |         validate_tax_rate(1.5)
196 |
197 |
198 | class TestProfitCalculations:
199 |     """Test profit calculation logic."""
200 |
201 |     def test_profit_calculation_basic(self):
202 |         """Test basic profit calculation."""
203 |         revenue = 10000
204 |         costs = [1000, 500, 300]
205 |
206 |         profit = calculate_profit(revenue, costs)
207 |         assert profit == 8200
208 |
209 |     def test_profit_calculation_no_profit(self):
210 |         """Test profit calculation with zero profit."""
211 |         revenue = 5000
212 |         costs = [3000, 2000]
213 |
214 |         profit = calculate_profit(revenue, costs)
215 |         assert profit == 0
216 |
217 |     def test_profit_calculation_loss(self):
218 |         """Test profit calculation with negative profit (loss)."""
219 |         revenue = 5000
220 |         costs = [4000, 2000]
221 |
222 |         profit = calculate_profit(revenue, costs)
223 |         assert profit == -1000
224 |
225 |     def test_profit_calculation_no_costs(self):
226 |         """Test profit calculation with no costs."""
227 |         revenue = 10000
228 |         costs = []
229 |
230 |         profit = calculate_profit(revenue, costs)

```

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

File 32: tests/test_database.py

```
1  """
2  Tests for database operations.
3
4  AI ASSISTANCE DISCLOSURE:
5  All tests in this file were generated with AI assistance (ChatGPT/Claude).
6  - Prompts used: "Write pytest tests for database CRUD operations"
7  - Prompts used: "Create tests for foreign key constraints and referential integrity"
8  - Prompts used: "Add tests for aggregation queries and complex SQL operations"
9  - AI helped generate test data, assertions, and database fixture management
10 """
11
12 import pytest
13 import sqlite3
14 import os
15 from DB import setup
16
17
18 class TestDatabaseOperations:
19     """Test database CRUD operations."""
20
21     def test_insert_and_fetch_record(self):
22         """Test inserting and fetching a tax record."""
23         conn = setup.get_conn()
24         cursor = conn.cursor()
25
26         # Insert a test record
27         cursor.execute("""
28             INSERT INTO tax_records (num_people, revenue, total_costs, group_income,
29                                     individual_income, tax_origin, tax_option, tax_amount,
30                                     net_income_per_person, net_income_group)
31             VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
32             """, (2, 10000, 1500, 8500, 4250, "US", "Individual", 500, 3750, 8000))
33
34         record_id = cursor.lastrowid
35         conn.commit()
36
37         # Fetch the record
38         cursor.execute("SELECT * FROM tax_records WHERE id = ?", (record_id,))
39         record = cursor.fetchone()
40
41         assert record is not None
42         assert record[2] == 10000 # revenue
43         assert record[3] == 1500 # total_costs
44
45         conn.close()
46
47     def test_delete_record(self):
48         """Test deleting a record."""
49         conn = setup.get_conn()
50         cursor = conn.cursor()
51
52         # Insert
53         cursor.execute("""
54             INSERT INTO tax_records (num_people, revenue, total_costs, group_income,
55                                     individual_income, tax_origin, tax_option, tax_amount,
56                                     net_income_per_person, net_income_group)
57             VALUES (1, 5000, 500, 4500, 4500, "US", "Individual", 450, 4050, 4050)
58             """)
59         record_id = cursor.lastrowid
60         conn.commit()
61
62         # Delete
63         cursor.execute("DELETE FROM tax_records WHERE id = ?", (record_id,))
64         conn.commit()
65
66         # Verify deleted
67         cursor.execute("SELECT * FROM tax_records WHERE id = ?", (record_id,))
68         assert cursor.fetchone() is None
69
70         conn.close()
71
72     def test_update_record(self):
73         """Test updating a record."""
74         conn = setup.get_conn()
```

```

75 |         cursor = conn.cursor()
76 |
77 |         # Insert
78 |         cursor.execute("""
79 |             INSERT INTO tax_records (num_people, revenue, total_costs, group_income,
80 |                                     individual_income, tax_origin, tax_option, tax_amount,
81 |                                     net_income_per_person, net_income_group)
82 |             VALUES (1, 5000, 500, 4500, 4500, "US", "Individual", 450, 4050, 4050)
83 |         """)
84 |         record_id = cursor.lastrowid
85 |         conn.commit()
86 |
87 |         # Update
88 |         cursor.execute("UPDATE tax_records SET revenue = ? WHERE id = ?", (6000, record_id))
89 |         conn.commit()
90 |
91 |         # Verify updated
92 |         cursor.execute("SELECT revenue FROM tax_records WHERE id = ?", (record_id,))
93 |         updated_revenue = cursor.fetchone()[0]
94 |         assert updated_revenue == 6000
95 |
96 |         conn.close()
97 |
98 |
99 | class TestPeopleTable:
100 |     """Test people table operations."""
101 |
102 |     def test_insert_person(self):
103 |         """Test adding a person to a record."""
104 |         conn = setup.get_conn()
105 |         cursor = conn.cursor()
106 |
107 |         # Create a tax record first
108 |         cursor.execute("""
109 |             INSERT INTO tax_records (num_people, revenue, total_costs, group_income,
110 |                                     individual_income, tax_origin, tax_option, tax_amount,
111 |                                     net_income_per_person, net_income_group)
112 |             VALUES (1, 5000, 500, 4500, 4500, "US", "Individual", 450, 4050, 4050)
113 |         """)
114 |         record_id = cursor.lastrowid
115 |
116 |         # Insert person
117 |         cursor.execute("""
118 |             INSERT INTO people (record_id, name, work_share, gross_income, tax_paid, net_income)
119 |             VALUES (?, ?, ?, ?, ?, ?)
120 |         """, (record_id, "Alice", 1.0, 4500, 450, 4050))
121 |
122 |         conn.commit()
123 |
124 |         # Verify
125 |         cursor.execute("SELECT * FROM people WHERE record_id = ?", (record_id,))
126 |         person = cursor.fetchone()
127 |
128 |         assert person is not None
129 |         assert person[2] == "Alice"
130 |         assert person[3] == 1.0
131 |
132 |         conn.close()
133 |
134 |     def test_multiple_people_per_record(self):
135 |         """Test adding multiple people to one record."""
136 |         conn = setup.get_conn()
137 |         cursor = conn.cursor()
138 |
139 |         # Create record
140 |         cursor.execute("""
141 |             INSERT INTO tax_records (num_people, revenue, total_costs, group_income,
142 |                                     individual_income, tax_origin, tax_option, tax_amount,
143 |                                     net_income_per_person, net_income_group)
144 |             VALUES (2, 10000, 1000, 9000, 4500, "US", "Individual", 900, 4050, 8100)
145 |         """)
146 |         record_id = cursor.lastrowid
147 |
148 |         # Insert multiple people
149 |         people = [
150 |             (record_id, "Alice", 0.6, 5400, 540, 4860),
151 |             (record_id, "Bob", 0.4, 3600, 360, 3240)
152 |         ]

```

```

153 |
154 |         cursor.executemany("""
155 |             INSERT INTO people (record_id, name, work_share, gross_income, tax_paid, net_income)
156 |             VALUES (?, ?, ?, ?, ?, ?)
157 |             """, people)
158 |
159 |         conn.commit()
160 |
161 |         # Verify
162 |         cursor.execute("SELECT COUNT(*) FROM people WHERE record_id = ?", (record_id,))
163 |         count = cursor.fetchone()[0]
164 |         assert count == 2
165 |
166 |         cursor.execute("SELECT SUM(work_share) FROM people WHERE record_id = ?", (record_id,))
167 |         total_share = cursor.fetchone()[0]
168 |         assert abs(total_share - 1.0) < 0.01 # Should sum to 1.0
169 |
170 |         conn.close()
171 |
172 |
173 | class TestTaxBrackets:
174 |     """Test tax brackets table."""
175 |
176 |     def test_fetch_tax_brackets(self):
177 |         """Test fetching tax brackets."""
178 |         conn = setup.get_conn()
179 |         cursor = conn.cursor()
180 |
181 |         # Fetch US Individual brackets
182 |         cursor.execute("""
183 |             SELECT income_limit, rate FROM tax_brackets
184 |             WHERE country = 'US' AND tax_type = 'Individual'
185 |             ORDER BY income_limit
186 |             """)
187 |
188 |         brackets = cursor.fetchall()
189 |         assert len(brackets) > 0
190 |
191 |         # First bracket should have lowest rate
192 |         assert brackets[0][1] <= brackets[-1][1]
193 |
194 |         conn.close()
195 |
196 |     def test_tax_bracket_ordering(self):
197 |         """Test that tax brackets are ordered correctly."""
198 |         conn = setup.get_conn()
199 |         cursor = conn.cursor()
200 |
201 |         cursor.execute("""
202 |             SELECT income_limit FROM tax_brackets
203 |             WHERE country = 'US' AND tax_type = 'Individual'
204 |             ORDER BY income_limit
205 |             """)
206 |
207 |         limits = [row[0] for row in cursor.fetchall()]
208 |
209 |         # Verify ascending order
210 |         for i in range(len(limits) - 1):
211 |             assert limits[i] < limits[i + 1]
212 |
213 |         conn.close()
214 |
215 |
216 | class TestDataIntegrity:
217 |     """Test data integrity and constraints."""
218 |
219 |     def test_foreign_key_constraint(self):
220 |         """Test that foreign key constraints are enforced."""
221 |         conn = setup.get_conn()
222 |         cursor = conn.cursor()
223 |
224 |         # Try to insert person with non-existent record_id
225 |         with pytest.raises(sqlite3.IntegrityError):
226 |             cursor.execute("""
227 |                 INSERT INTO people (record_id, name, work_share, gross_income, tax_paid, net_income)
228 |                 VALUES (?, ?, ?, ?, ?, ?)
229 |                 """, (99999, "NonExistent", 1.0, 1000, 100, 900))
230 |             conn.commit()

```



```

231 |
232 |         conn.close()
233 |
234 |     def test_cascade_delete(self):
235 |         """Test that deleting a record requires deleting people first (FK constraint)."""
236 |         conn = setup.get_conn()
237 |         cursor = conn.cursor()
238 |
239 |         # Create record with person
240 |         cursor.execute("""
241 |             INSERT INTO tax_records (num_people, revenue, total_costs, group_income,
242 |                                     individual_income, tax_origin, tax_option, tax_amount,
243 |                                     net_income_per_person, net_income_group)
244 |             VALUES (1, 5000, 500, 4500, 4500, "US", "Individual", 450, 4050, 4050)
245 |         """)
246 |         record_id = cursor.lastrowid
247 |
248 |         cursor.execute("""
249 |             INSERT INTO people (record_id, name, work_share, gross_income, tax_paid, net_income)
250 |             VALUES (?, ?, ?, ?, ?, ?)
251 |             """, (record_id, "Test", 1.0, 4500, 450, 4050))
252 |
253 |         conn.commit()
254 |
255 |         # Delete people first, then record (FK constraint prevents direct delete)
256 |         cursor.execute("DELETE FROM people WHERE record_id = ?", (record_id,))
257 |         cursor.execute("DELETE FROM tax_records WHERE id = ?", (record_id,))
258 |         conn.commit()
259 |
260 |         # Verify both deleted
261 |         cursor.execute("SELECT COUNT(*) FROM people WHERE record_id = ?", (record_id,))
262 |         assert cursor.fetchone()[0] == 0
263 |
264 |         cursor.execute("SELECT COUNT(*) FROM tax_records WHERE id = ?", (record_id,))
265 |         assert cursor.fetchone()[0] == 0
266 |
267 |         conn.close()
268 |
269 |
270 | class TestDatabaseQueries:
271 |     """Test complex database queries."""
272 |
273 |     def test_aggregate_revenue_by_month(self):
274 |         """Test aggregating revenue by month."""
275 |         conn = setup.get_conn()
276 |         cursor = conn.cursor()
277 |
278 |         cursor.execute("""
279 |             SELECT strftime('%Y-%m', created_at) as month,
280 |                    SUM(revenue) as total_revenue,
281 |                    COUNT(*) as num_records
282 |             FROM tax_records
283 |             GROUP BY month
284 |             ORDER BY month
285 |         """)
286 |
287 |         results = cursor.fetchall()
288 |
289 |         if len(results) > 0:
290 |             # Each result should have month, revenue, count
291 |             for row in results:
292 |                 assert len(row) == 3
293 |                 assert row[1] > 0 # revenue should be positive
294 |                 assert row[2] > 0 # count should be positive
295 |
296 |         conn.close()
297 |
298 |     def test_top_earners_query(self):
299 |         """Test querying top earners."""
300 |         conn = setup.get_conn()
301 |         cursor = conn.cursor()
302 |
303 |         cursor.execute("""
304 |             SELECT name,
305 |                    SUM(gross_income) as total_earned,
306 |                    COUNT(*) as projects
307 |             FROM people
308 |             GROUP BY name

```

```

309 |         ORDER BY total_earned DESC
310 |         LIMIT 5
311 |     """
312 |
313 |     results = cursor.fetchall()
314 |
315 |     if len(results) > 0:
316 |         # Results should be ordered by earnings (descending)
317 |         for i in range(len(results) - 1):
318 |             assert results[i][1] >= results[i + 1][1]
319 |
320 |     conn.close()
321 |
322 | def test_tax_rate_calculation(self):
323 |     """Test calculating average tax rates."""
324 |     conn = setup.get_conn()
325 |     cursor = conn.cursor()
326 |
327 |     cursor.execute("""
328 |         SELECT tax_origin,
329 |                tax_option,
330 |                AVG(tax_amount * 100.0 / NULLIF(group_income, 0)) as avg_tax_rate,
331 |                COUNT(*) as count
332 |         FROM tax_records
333 |         WHERE group_income > 0
334 |         GROUP BY tax_origin, tax_option
335 |     """)
336 |
337 |     results = cursor.fetchall()
338 |
339 |     for row in results:
340 |         country, tax_type, avg_rate, count = row
341 |         # Tax rate should be reasonable (0-50%)
342 |         assert 0 <= avg_rate <= 50
343 |
344 |     conn.close()
345 |

```

File 33: tests/test_edge_cases.py

```
1  """
2  | Edge case and boundary testing.
3  |
4  | AI ASSISTANCE DISCLOSURE:
5  | All tests in this file were generated with AI assistance (ChatGPT/Claude).
6  | - Prompts used: "Create edge case tests for boundary values and invalid inputs"
7  | - Prompts used: "Write tests for floating point precision and special characters"
8  | - Prompts used: "Add tests for zero values, negative numbers, and extreme inputs"
9  | - AI helped identify edge cases and create comprehensive boundary testing
10 | """
11 |
12 | import pytest
13 | from fastapi.testclient import TestClient
14 | from api.main import app
15 | from Logic.tax_calculator import calculate_tax, split_work_shares, calculate_profit
16 | from Logic.validators import ValidationError, validate_work_shares, validate_tax_rate
17 |
18 | client = TestClient(app)
19 |
20 |
21 | class TestBoundaryValues:
22 |     """Test boundary and extreme values."""
23 |
24 |     def test_very_large_revenue(self):
25 |         """Test handling of very large revenue values."""
26 |         payload = {
27 |             "num_people": 1,
28 |             "revenue": 10000000, # 10 million
29 |             "costs": [1000000],
30 |             "country": "US",
31 |             "tax_type": "Individual",
32 |             "people": [{"name": "Millionaire", "work_share": 1.0}]
33 |         }
34 |
35 |         response = client.post("/api/projects", json=payload)
36 |         assert response.status_code == 201
37 |
38 |     def test_very_small_revenue(self):
39 |         """Test handling of very small revenue values."""
40 |         payload = {
41 |             "num_people": 1,
42 |             "revenue": 0.01, # 1 cent
43 |             "costs": [0.001],
44 |             "country": "US",
45 |             "tax_type": "Individual",
46 |             "people": [{"name": "Penny", "work_share": 1.0}]
47 |         }
48 |
49 |         response = client.post("/api/projects", json=payload)
50 |         assert response.status_code == 201
51 |
52 |     def test_zero_revenue(self):
53 |         """Test handling of zero revenue."""
54 |         payload = {
55 |             "num_people": 1,
56 |             "revenue": 0,
57 |             "costs": [0],
58 |             "country": "US",
59 |             "tax_type": "Individual",
60 |             "people": [{"name": "Zero", "work_share": 1.0}]
61 |         }
62 |
63 |         response = client.post("/api/projects", json=payload)
64 |         # Should either accept or reject gracefully
65 |         assert response.status_code in [201, 400, 422]
66 |
67 |     def test_many_people(self):
68 |         """Test handling of many people in a project."""
69 |         num_people = 50
70 |         payload = {
71 |             "num_people": num_people,
72 |             "revenue": 100000,
73 |             "costs": [1000],
74 |             "country": "US",
```

```

75         "tax_type": "Individual",
76         "people": [
77             {"name": f"Person{i}", "work_share": 1.0/num_people}
78             for i in range(num_people)
79         ]
80     }
81
82     response = client.post("/api/projects", json=payload)
83     assert response.status_code == 201
84
85     def test_single_person_project(self):
86         """Test single person project."""
87         payload = {
88             "num_people": 1,
89             "revenue": 5000,
90             "costs": [500],
91             "country": "US",
92             "tax_type": "Individual",
93             "people": [{"name": "Solo", "work_share": 1.0}]
94         }
95
96         response = client.post("/api/projects", json=payload)
97         assert response.status_code == 201
98
99
100     class TestInvalidInputs:
101         """Test invalid input handling."""
102
103     def test_negative_num_people(self):
104         """Test negative number of people."""
105         payload = {
106             "num_people": -1,
107             "revenue": 5000,
108             "costs": [500],
109             "country": "US",
110             "tax_type": "Individual",
111             "people": []
112         }
113
114         response = client.post("/api/projects", json=payload)
115         assert response.status_code == 422 # Validation error
116
117     def test_empty_people_list(self):
118         """Test empty people list."""
119         payload = {
120             "num_people": 2,
121             "revenue": 5000,
122             "costs": [500],
123             "country": "US",
124             "tax_type": "Individual",
125             "people": []
126         }
127
128         response = client.post("/api/projects", json=payload)
129         assert response.status_code in [400, 422]
130
131     def test_mismatched_num_people(self):
132         """Test mismatch between num_people and actual people list."""
133         payload = {
134             "num_people": 5,
135             "revenue": 5000,
136             "costs": [500],
137             "country": "US",
138             "tax_type": "Individual",
139             "people": [
140                 {"name": "Person1", "work_share": 0.5},
141                 {"name": "Person2", "work_share": 0.5}
142             ] # Only 2 people but num_people=5
143         }
144
145         response = client.post("/api/projects", json=payload)
146         assert response.status_code in [400, 422]
147
148     def test_invalid_country(self):
149         """Test invalid country code."""
150         payload = {
151             "num_people": 1,
152             "revenue": 5000,

```

```

153         "costs": [500],
154         "country": "", # Empty country
155         "tax_type": "Individual",
156         "people": [{"name": "Test", "work_share": 1.0}]
157     }
158
159     response = client.post("/api/projects", json=payload)
160     assert response.status_code in [400, 422]
161
162     def test_invalid_tax_type(self):
163         """Test invalid tax type."""
164         payload = {
165             "num_people": 1,
166             "revenue": 5000,
167             "costs": [500],
168             "country": "US",
169             "tax_type": "Corporate", # Invalid type
170             "people": [{"name": "Test", "work_share": 1.0}]
171         }
172
173         response = client.post("/api/projects", json=payload)
174         assert response.status_code in [400, 422]
175
176
177     class TestWorkShareEdgeCases:
178         """Test work share edge cases."""
179
180         def test_work_shares_sum_to_zero(self):
181             """Test work shares that sum to zero."""
182             with pytest.raises(ValidationError):
183                 validate_work_shares([0.0, 0.0])
184
185         def test_work_shares_sum_slightly_off(self):
186             """Test work shares that sum to almost 1.0."""
187             # Should pass with small tolerance
188             validate_work_shares([0.33, 0.33, 0.34]) # Sums to 1.0
189             validate_work_shares([0.333, 0.333, 0.334]) # Sums to 1.0
190
191         def test_work_shares_very_unequal(self):
192             """Test very unequal work shares."""
193             payload = {
194                 "num_people": 2,
195                 "revenue": 10000,
196                 "costs": [1000],
197                 "country": "US",
198                 "tax_type": "Individual",
199                 "people": [
200                     {"name": "Leader", "work_share": 0.99},
201                     {"name": "Helper", "work_share": 0.01}
202                 ]
203             }
204
205             response = client.post("/api/projects", json=payload)
206             assert response.status_code == 201
207
208         def test_all_work_to_one_person(self):
209             """Test when one person does all work."""
210             profit = 10000
211             work_shares = [1.0, 0.0, 0.0]
212
213             distribution = split_work_shares(profit, work_shares)
214
215             assert distribution[0] == 10000
216             assert distribution[1] == 0
217             assert distribution[2] == 0
218
219
220     class TestCostsEdgeCases:
221         """Test costs edge cases."""
222
223         def test_costs_exceed_revenue(self):
224             """Test when costs exceed revenue (negative profit)."""
225             payload = {
226                 "num_people": 1,
227                 "revenue": 5000,
228                 "costs": [6000], # Costs > revenue
229                 "country": "US",
230                 "tax_type": "Individual",

```

```

231 |         "people": [{"name": "Loss", "work_share": 1.0}]
232 |     }
233 |
234 |     response = client.post("/api/projects", json=payload)
235 |     # Should handle negative profit scenario
236 |     assert response.status_code in [201, 400]
237 |
238 | def test_many_small_costs(self):
239 |     """Test many small cost items."""
240 |     costs = [0.01] * 1000 # 1000 costs of 1 cent each
241 |
242 |     payload = {
243 |         "num_people": 1,
244 |         "revenue": 20,
245 |         "costs": costs,
246 |         "country": "US",
247 |         "tax_type": "Individual",
248 |         "people": [{"name": "Penny", "work_share": 1.0}]
249 |     }
250 |
251 |     response = client.post("/api/projects", json=payload)
252 |     assert response.status_code == 201
253 |
254 | def test_zero_costs(self):
255 |     """Test project with zero costs."""
256 |     payload = {
257 |         "num_people": 1,
258 |         "revenue": 5000,
259 |         "costs": [], # No costs
260 |         "country": "US",
261 |         "tax_type": "Individual",
262 |         "people": [{"name": "NoCost", "work_share": 1.0}]
263 |     }
264 |
265 |     response = client.post("/api/projects", json=payload)
266 |     assert response.status_code == 201
267 |
268 |
269 | class TestTaxCalculationEdgeCases:
270 |     """Test tax calculation edge cases."""
271 |
272 |     def test_income_at_bracket_boundary(self):
273 |         """Test income exactly at bracket boundary."""
274 |         brackets = [(10000, 0.10), (50000, 0.20)]
275 |         tax = calculate_tax(10000, brackets) # Exactly at boundary
276 |
277 |         # 10000 at 10% = 1000
278 |         assert abs(tax - 1000) < 0.01
279 |
280 |     def test_zero_income_tax(self):
281 |         """Test tax on zero income."""
282 |         brackets = [(10000, 0.10), (50000, 0.20)]
283 |         tax = calculate_tax(0, brackets)
284 |
285 |         assert tax == 0
286 |
287 |     def test_income_in_top_bracket(self):
288 |         """Test very high income in top bracket."""
289 |         brackets = [(10000, 0.10), (50000, 0.20), (100000, 0.30)]
290 |         tax = calculate_tax(500000, brackets)
291 |
292 |         # Should use all brackets
293 |         assert tax > 0
294 |
295 |
296 | class TestAPIEdgeCases:
297 |     """Test API edge cases."""
298 |
299 |     def test_get_nonexistent_record_various_ids(self):
300 |         """Test getting records with various nonexistent IDs."""
301 |         for record_id in [0, -1, 999999, "abc"]:
302 |             response = client.get(f"/api/records/{record_id}")
303 |             assert response.status_code in [404, 422]
304 |
305 |     def test_concurrent_project_creation(self):
306 |         """Test creating multiple projects rapidly."""
307 |         payload = {
308 |             "num_people": 1,

```

```

309         "revenue": 5000,
310         "costs": [500],
311         "country": "US",
312         "tax_type": "Individual",
313         "people": [{"name": "Concurrent", "work_share": 1.0}]
314     }
315
316     # Create 10 projects rapidly
317     responses = []
318     for i in range(10):
319         payload["people"][0]["name"] = f"Person{i}"
320         response = client.post("/api/projects", json=payload)
321         responses.append(response)
322
323     # All should succeed
324     for response in responses:
325         assert response.status_code == 201
326
327     def test_special_characters_in_names(self):
328         """Test special characters in person names."""
329         special_names = [
330             "José García",
331             "■",
332             "Müller",
333             "O'Brien",
334             "Jean-Pierre",
335             "■■■■■■■■"
336         ]
337
338         for name in special_names:
339             payload = {
340                 "num_people": 1,
341                 "revenue": 5000,
342                 "costs": [500],
343                 "country": "US",
344                 "tax_type": "Individual",
345                 "people": [{"name": name, "work_share": 1.0}]
346             }
347
348             response = client.post("/api/projects", json=payload)
349             assert response.status_code == 201
350
351
352     class TestCalculationAccuracy:
353         """Test calculation accuracy."""
354
355         def test_floating_point_precision(self):
356             """Test floating point precision in calculations."""
357             profit = 10000.33
358             work_shares = [0.333, 0.333, 0.334]
359
360             distribution = split_work_shares(profit, work_shares)
361
362             # Sum should equal original (within floating point tolerance)
363             total = sum(distribution)
364             assert abs(total - profit) < 0.01
365
366         def test_profit_calculation_precision(self):
367             """Test profit calculation precision."""
368             revenue = 10000.99
369             costs = [1000.33, 500.22, 300.11]
370
371             profit = calculate_profit(revenue, costs)
372
373             expected = revenue - sum(costs)
374             assert abs(profit - expected) < 0.01
375

```