**What we’re building (feature checklist)**

* **Core FP&A**: data consolidation, budgeting & forecasting, scenario analysis, dashboards, variance (BvA) with commentary & workflow. These mirror DataRails’ advertised capabilities and its Excel-friendly approach plus AI assistant (“Genius”). [Datarails+4Datarails+4Datarails+4](https://www.datarails.com/?utm_source=chatgpt.com)
* **Connectors**: pull Chart of Accounts, P&L, and (optionally) General Ledger from **QuickBooks Online** via OAuth 2.0 and Reports API. [Intuit Developer+2Intuit Developer+2](https://developer.intuit.com/app/developer/qbo/docs/develop/authentication-and-authorization/oauth-2.0?utm_source=chatgpt.com)  
  *Heads-up:* QBO UI vs API reports can differ; reconcile carefully. [Intuit Developer Support](https://help.developer.intuit.com/s/topic/0TOG00000004rkGOAQ/profit-and-loss?utm_source=chatgpt.com)
* **AI Copilot**: students can type “grow revenue 15% YoY, marketing = 12% of revenue, hire 2 SDRs in Q2” → the app turns that into a driver-based plan, long-range forecast, and talking-points/strategy notes. (In spirit of Datarails’ “Genius” chat.) [Datarails+1](https://www.datarails.com/solutions-fpna-genius/?utm_source=chatgpt.com)

**Primer for non-accountants (2 pages worth, max)**

**Budgets** = targets (by month) for revenue, COGS, OpEx, headcount, CAPEX, cash.  
**Actuals** = what really happened (from QBO).  
**Budget vs Actuals (BvA)** = Actual – Budget, plus % variance and “F/U” (Favorable/Unfavorable).  
**Driver-based planning** = use *drivers* (price, volume, conversion, headcount, % of revenue) to auto-build line items.  
**Forecasting** = near-term by drivers/time-series; long-term by growth curves and margins.  
**Scenarios** = Base / Best / Worst (toggle drivers, see impact).  
**Three statements** (nice-to-have): P&L (profit), Balance Sheet (what you own/owe), Cash Flow (cash in/out).

**Architecture (student-friendly)**

* **Frontend**: React (Next.js) + Tailwind + a chart lib (Recharts). Pages: *Data Import, Budget Builder, Forecasts, Scenarios, BvA, Dashboard, Approvals.*
* **Backend**: Python FastAPI.
* **Data**: Postgres (or SQLite to start).
* **Jobs**: Celery + Redis (optional) for recurring syncs.
* **AI**: a small “Plan DSL” (JSON) + parser; start rule-based then swap to an LLM call.
* **Auth**: OAuth to QBO; app auth = email+password (or Auth0). [Intuit Developer](https://developer.intuit.com/app/developer/qbo/docs/develop/authentication-and-authorization/oauth-2.0?utm_source=chatgpt.com)

# **Student Whiteboard Guide: Building the Next-Gen Budgeting & Forecasting Tool**

## **1. What Is Budgeting & Forecasting?**

* **Budget** = A **plan** for money (like a household deciding groceries, rent, savings for the year).
* **Forecast** = A **prediction** of what will actually happen, based on past data (e.g., sales trends, growth rates, economy).
* Businesses compare **Budget vs. Actuals (BvA)** → Did we overspend? Did revenue grow faster?

**Analogy**:

* Budget = *your workout plan* (I’ll run 5 miles each week).
* Actual = *what you really did* (you ran 3 miles).
* Forecast = *trainer predicts* how much you’ll run next week, given your habits.

## **2. Why Do Businesses Care?**

* **Keep cash safe** (don’t run out of money).
* **Spot problems early** (e.g., labor costs too high).
* **Plan growth** (hire more people? open new store?).
* **Report to investors/board**.

## **3. What Are We Building?**

We want to create **the best budgeting software for small businesses** (better than DataRails, Hyperion, NetSuite).

**Our tool will:**

* Automatically build budgets (e.g., for a Taco Bell restaurant).
* Predict the future with AI (machine learning, ARIMA).
* Explain differences in plain English:  
  “Costs went up because food prices rose 15%.”
* Compare multiple scenarios: Base, Best, Worst.
* Handle **departments, currencies, headcount/payroll, depreciation**.
* Make it **simple & visual** (charts, dashboards, wizard setup).

## **4. Project Flow (Whiteboard Sketch Idea)**

Draw boxes with arrows (like a pipeline):

**Step 1: Data In**

* Manual entry
* CSV/Excel
* QuickBooks
* Migration from DataRails, NetSuite, Hyperion

⬇️

**Step 2: Core Engine**

* Chart of Accounts (Revenue, COGS, Opex…)
* Budget creation (preset + manual + AI auto)
* Forecasting (AI/ML)
* Scenarios (Base, Best, Worst)
* Headcount & Payroll
* Multi-currency

⬇️

**Step 3: Analytics & AI**

* Budget vs. Actuals
* AI Narratives (“Profits down due to higher labor costs”)
* Goal Seek Solver (what changes needed to hit target EBITDA)

⬇️

**Step 4: Outputs**

* Dashboards (trend, YoY, waterfall, heatmaps)
* Board packs (auto PPT)
* Excel sync
* Migration-friendly exports

## **5. Deliverables Over 2 Semesters**

📅 **Semester 1 – Foundations**

* Build core budgeting engine + simple UI
* Preset generator (Taco Bell, childcare)
* Budget auto-generator (trend + growth)
* Executive dashboard (3 charts)
* AI narrative (rule-based)

📅 **Semester 2 – Advanced + AI**

* AI forecasting (ARIMA, ML regression)
* AI variance analysis (natural language)
* Scenario planning + solver
* Multi-currency + department hierarchy
* Excel add-in (like Hyperion Smart View)
* Board pack exporter + AI executive summary
* Usability features (wizard, tooltips, command palette)

## **6. Key Roles (Students’ Perspective)**

* **Backend team**: APIs for budgets, forecasts, scenarios, migrations.
* **Frontend team**: Dashboards, migration UI, setup wizard.
* **AI team**: Forecasting models, narrative generation, solver.
* **Testing/QA team**: pytest, UI testing, scenario verification.

## **7. Why This Is Exciting 🚀**

* You’re building something **better than enterprise tools** — but for small businesses.
* Mix of **tech + finance + AI**.
* Opportunity to learn **real-world FP&A** concepts while practicing full-stack development.
* By the end: A small business could actually use your software to plan its future.

**Canonical tables (minimum):**

* company, user
* account (standardized categories), external\_account\_map (QBO→standard)
* actual\_monthly (per account, per month), budget\_monthly (same grain)
* assumption\_set (drivers), scenario, forecast\_monthly
* comment, approval\_event

**Milestone plan (8–10 weeks)**

1. **Week 1–2**: Data model + CSV import → BvA table.
2. **Week 3**: QuickBooks OAuth + P&L import (Reports API). [Intuit Developer](https://developer.intuit.com/app/developer/qbo/docs/workflows/run-reports?utm_source=chatgpt.com)
3. **Week 4**: Account mapping + rolling monthly actuals aggregation.
4. **Week 5**: Budget Builder (driver-based lines + manual overrides).
5. **Week 6**: Forecast engine (drivers + basic time-series).
6. **Week 7**: AI text→Plan DSL + strategy notes. (Start rules; optional LLM.) [Datarails](https://www.datarails.com/solutions-fpna-genius/?utm_source=chatgpt.com)
7. **Week 8**: Dashboards, scenario toggles, approvals & commentary.  
   (Stretch: headcount model, CAPEX→depreciation, cash flow.)

**QuickBooks integration (how + sample code)**

**Steps**

1. Create an Intuit developer app; get client ID/secret. [Intuit Developer](https://developer.intuit.com/app/developer/qbo/docs/develop/authentication-and-authorization/faq?utm_source=chatgpt.com)
2. Implement OAuth 2.0 (auth code flow). Store refresh token securely. [Intuit Developer](https://developer.intuit.com/app/developer/qbo/docs/develop/authentication-and-authorization/oauth-2.0?utm_source=chatgpt.com)
3. Call Reports API to fetch **Profit and Loss** by date range (monthly). [Intuit Developer](https://developer.intuit.com/app/developer/qbo/docs/workflows/run-reports?utm_source=chatgpt.com)
4. (Optional) Pull **General Ledger** if you need transaction-level detail; note community threads re: GL nuances. [Intuit Developer Support+1](https://help.developer.intuit.com/s/topic/0TOG00000004rFMOAY/report-api?utm_source=chatgpt.com)[Stack Overflow](https://stackoverflow.com/questions/23949224/extracting-general-ledger-report-data-via-quickbooks-online-v3-api?utm_source=chatgpt.com)

# backend/qbo\_client.py

import requests, pandas as pd

QBO\_BASE = "https://quickbooks.api.intuit.com/v3/company"

def qbo\_pnl(company\_id: str, bearer\_token: str, start: str, end: str) -> pd.DataFrame:

"""

Returns monthly Profit & Loss by Account

dates as YYYY-MM-DD; granularity via column 'ColData'

"""

url = f"{QBO\_BASE}/{company\_id}/reports/ProfitAndLoss"

params = {

"start\_date": start, "end\_date": end,

"accounting\_method": "Accrual", "summarize\_column\_by": "Month"

}

headers = {"Authorization": f"Bearer {bearer\_token}", "Accept": "application/json"}

r = requests.get(url, params=params, headers=headers, timeout=60)

r.raise\_for\_status()

data = r.json()["Rows"]["Row"] # shape documented under QBO Reports API

# Flatten rows → tidy frame (account, month, amount)

records = []

for row in data:

if row.get("Summary") or row.get("type") == "Section": # skip grand totals/sections

continue

account = row["ColData"][0]["value"]

for i, col in enumerate(row["ColData"][1:], start=1):

records.append({"account": account, "col\_idx": i, "amount": float(col.get("value", 0) or 0)})

df = pd.DataFrame(records)

# QBO returns month columns in order; attach proper month labels from Header

months = [c["ColTitle"] for c in r.json()["Columns"]["Column"][1:]] # e.g., "Jan-2025"

df["month"] = df["col\_idx"].apply(lambda i: months[i-1])

return df[["account", "month", "amount"]]

Tip: Validate that report totals match the QBO UI and note known mismatches some devs observe in community posts. Reconcile with GL if needed. [Intuit Developer Support](https://help.developer.intuit.com/s/topic/0TOG00000004rkGOAQ/profit-and-loss?utm_source=chatgpt.com)

**Budget-vs-Actuals engine (clear logic + code)**

# backend/bva.py

import pandas as pd

def compute\_bva(df\_actual: pd.DataFrame, df\_budget: pd.DataFrame, mapping: pd.DataFrame):

"""

df\_actual: columns [account, month, amount]

df\_budget: columns [account\_std, month, amount\_budget]

mapping: columns [account (QBO raw), account\_std]

"""

actual\_std = df\_actual.merge(mapping, on="account", how="left")

actual\_std = actual\_std.groupby(["account\_std", "month"], as\_index=False)["amount"].sum()

out = actual\_std.merge(df\_budget, on=["account\_std", "month"], how="outer").fillna(0)

out["variance"] = out["amount"] - out["amount\_budget"]

out["variance\_pct"] = out.apply(

lambda r: (r["variance"] / r["amount\_budget"]) if r["amount\_budget"] else 0, axis=1

)

out["fav\_unfav"] = out.apply(

lambda r: "F" if ((r["account\_std"].startswith("Revenue") and r["variance"]>=0) or

(not r["account\_std"].startswith("Revenue") and r["variance"]<=0)) else "U", axis=1

)

return out.sort\_values(["month","account\_std"])

**Driver-based budgeting & forecasting (starter)**

# backend/plan.py

from dataclasses import dataclass, field

from typing import Dict, List

@dataclass

class PlanDSL:

# simple, extend as needed

yoy\_growth\_pct: Dict[str, float] = field(default\_factory=dict) # {"Total Revenue": 0.15}

spend\_ratio: Dict[str, float] = field(default\_factory=dict) # {"Marketing": 0.12} (% of revenue)

hires: List[dict] = field(default\_factory=list) # [{"dept":"Sales","start":"2025-04","qty":2,"loaded\_cost":85000}]

churn\_mth: float = 0.02

seasonality: List[float] = field(default\_factory=lambda: [1]\*12) # multipliers Jan..Dec

def apply\_plan(budget\_base: dict, plan: PlanDSL) -> dict:

"""

budget\_base: {"2025-01":{"Revenue":{"Total":100000}, "Opex":{"Marketing":8000, ...}}, ...}

returns a new budget dict with drivers applied for 12–36 months

"""

out = {}

months = sorted(budget\_base.keys())

for i, m in enumerate(months):

prev = months[i-1] if i>0 else None

out[m] = {"Revenue": {}, "COGS": {}, "Opex": {}}

# Revenue growth

base\_rev = budget\_base[m]["Revenue"]["Total"]

growth = plan.yoy\_growth\_pct.get("Total Revenue", 0.0)

# naive monthly growth approximation

rev = base\_rev \* (1 + growth/12) \* plan.seasonality[i % 12]

out[m]["Revenue"]["Total"] = round(rev, 2)

# Marketing as % of revenue (example)

mkt\_pct = plan.spend\_ratio.get("Marketing", 0.0)

out[m]["Opex"]["Marketing"] = round(rev \* mkt\_pct, 2)

# (You’d fill in more drivers: headcount ramp to payroll, COGS as % of revenue, etc.)

return out

**AI Copilot: “text → plan” (rules first, LLM later)**

1. **Define the target JSON** (our *Plan DSL* above).
2. **Rules pass**: catch common phrases ("grow X% YoY", "set marketing to Y% of revenue", "hire N <role> in <month>").
3. **LLM pass (optional)**: if text is complex, send to an LLM and validate the JSON before applying (students can mock this).

# backend/ai\_parser.py

import re

from plan import PlanDSL

def parse\_text\_to\_plan(text: str) -> PlanDSL:

plan = PlanDSL()

if m := re.search(r"grow revenue\s+(\d+)%\s\*yoy", text, re.I):

plan.yoy\_growth\_pct["Total Revenue"] = float(m.group(1))/100.0

if m := re.search(r"marketing\s\*=\s\*(\d+(\.\d+)?)%\s\*of\s\*revenue", text, re.I):

plan.spend\_ratio["Marketing"] = float(m.group(1))/100.0

for hire in re.finditer(r"hire\s+(\d+)\s+(\w+)\w\*\s+in\s+([A-Za-z]{3,9})", text, re.I):

qty, role, month\_word = hire.groups()

plan.hires.append({"dept": role.title(), "start": f"2025-{month\_word[:3].title()}", "qty": int(qty), "loaded\_cost": 80000})

if m := re.search(r"churn\s+(\d+(\.\d+)?)%", text, re.I):

plan.churn\_mth = float(m.group(1))/100.0

return plan

# Optional LLM stub (students can integrate any provider)

def llm\_parse\_to\_plan(text: str) -> PlanDSL:

"""

1) call your LLM with a system prompt that outputs ONLY PlanDSL JSON

2) json.loads -> validate keys -> into PlanDSL

"""

raise NotImplementedError("Connect to your LLM of choice here.")

**Forecasting (simple & teachable)**

Start with **driver-based** (above). Add a **time-series** baseline (ARIMA from statsmodels) and average the two for months 1–6.

# backend/forecast.py

import pandas as pd

from statsmodels.tsa.arima.model import ARIMA

def ts\_forecast(monthly\_series: pd.Series, horizon=6):

model = ARIMA(monthly\_series, order=(1,1,1))

res = model.fit()

f = res.get\_forecast(steps=horizon)

return f.predicted\_mean.clip(lower=0)

**FastAPI skeleton (routes you need)**

# app.py

from fastapi import FastAPI, Depends

from typing import Dict

from qbo\_client import qbo\_pnl

from bva import compute\_bva

from ai\_parser import parse\_text\_to\_plan

from plan import apply\_plan, PlanDSL

app = FastAPI(title="Student FP&A")

@app.get("/health")

def health(): return {"ok": True}

@app.post("/qbo/pnl")

def fetch\_pnl(company\_id: str, token: str, start: str, end: str):

return qbo\_pnl(company\_id, token, start, end).to\_dict(orient="records")

@app.post("/plan/parse")

def plan\_parse(body: Dict[str, str]):

plan = parse\_text\_to\_plan(body["text"])

return plan.\_\_dict\_\_

@app.post("/budget/apply")

def budget\_apply(budget\_base: Dict, plan: Dict):

return apply\_plan(budget\_base, PlanDSL(\*\*plan))

**UI blueprint (what to build)**

* **Data Import**: Connect QBO → pick date range → preview P&L; upload CSV fallback. (Note Datarails leans into “Excel-friendly” mindset; you can mirror that with CSV/Excel upload.) [Datarails+1](https://www.datarails.com/?utm_source=chatgpt.com)
* **Budget Builder**: table (accounts × months), driver chips (e.g., “Marketing = 12% of Rev”), manual override cells, versioning & comments.
* **AI Tab**: chat box → shows parsed Plan DSL on the side → “Apply to Base” → creates Budget v1.
* **BvA**: heatmap and table (variance, % variance, F/U), inline commentary, export to CSV/PDF.
* **Scenarios**: toggle assumptions; compare Delta vs Base.
* **Approvals**: submit budget → manager approve/reject with notes.

**Testing & grading**

* Unit tests for: mapping, BvA math, parser rules, forecasting function.
* Golden datasets (CSV) with known outputs.
* End-to-end demo: import → plan text → budget → forecast → BvA dashboard.

**Gotchas students should know**

* **Dimensions** (class/location/customer) in QBO Reports can be tricky; start simple (account × month). [Intuit Developer](https://developer.intuit.com/app/developer/qbo/docs/workflows/run-reports?utm_source=chatgpt.com)[Intuit Developer Support](https://help.developer.intuit.com/s/topic/0TOG00000004rFMOAY/report-api?utm_source=chatgpt.com)
* **UI vs API** totals may differ—document reconciliation steps. [Intuit Developer Support](https://help.developer.intuit.com/s/topic/0TOG00000004rkGOAQ/profit-and-loss?utm_source=chatgpt.com)
* **Excel-friendly**: always allow CSV/Excel import/export to mirror DataRails’ ethos. [Datarails+1](https://www.datarails.com/?utm_source=chatgpt.com)

**Extras (if time permits)**

* Headcount module (roles, salary, start dates, benefits → payroll & cash).
* Cash flow builder (turn P&L + balance sheet drivers into cash).
* Workflow: notifications & approvals.
* Xero/Sage connectors after QBO.

If you’d like, I can package this into a starter repo (folders, FastAPI app, React skeleton, test data) so your students can clone and run on day 1.

**You said:**

how is the auto-budgets created. would it be on industry, what can the user specify and what should be automatically populated. One scenario for existing company is to use trends and then growth. Can the growth be on gdp, but user can update. What about new company. Suggestion to get the best budgeting and forecasting tool out.

**ChatGPT said:**

Thought for 6s

Awesome Q. Here’s a crisp blueprint for **auto-budget generation** that works for (1) existing companies with history and (2) new companies with little/no data — plus how to weave in **industry presets** and **macro drivers (e.g., GDP)** while keeping everything user-overrideable.

**Auto-Budget: Modes & Flow**

1. **Existing company (has actuals)**
   * **Baseline = trends** from the last 12–36 months (level, seasonality, mix).
   * **Drivers layered on**: user growth targets, headcount plan, price/volume, % of revenue spend, unit economics.
   * **Macro adjustors (optional)**: GDP, CPI, industry index → nudge growth/margins.
   * **Policy rules**: minimum/maximum caps, elasticity, lag effects (e.g., marketing → revenue with a 2-month lag).
   * **Finalize**: produce **Budget v1** (monthly, by account), auto-notes (“Revenue up 12% YoY driven by +2 AEs in Q2”).
2. **New company (no actuals)**
   * **Industry Pack** (selectable): SaaS, Retail, Services, Manufacturing, Childcare, etc.
   * **Wizard** prompts: pricing, expected volume, location count, staff roster, wage rates/benefits, COGS % or BOM, rent/utilities.
   * **Generate driver model** (price×volume, staffing → payroll, % of revenue for SG&A) with sensible defaults + seasonality for the industry.
   * **Macro adjustors** (optional): GDP for top-line, CPI for costs (wages, utilities), commodity index for COGS in manufacturing/food.
   * **Finalize**: **Budget v0** with editable assumptions.

**What’s Auto vs. User-Specified**

**User specifies (high-leverage inputs)**

* Revenue model: price & volume drivers, channels, conversion rates, ramp times.
* Strategic targets: “Grow total revenue 15% YoY”, “Marketing 12% of revenue”, “Hire 3 teachers in March”.
* Headcount: roles, start month, loaded cost; hiring freeze rules; attrition/churn.
* Capital items: CAPEX timing, depreciation life; financing/interest (if enabled).
* Macro links (toggle): choose which lines follow **GDP**, **CPI**, **industry index**—and add custom overrides.

**Auto-populate (smart defaults)**

* Seasonality profiles (by industry pack).
* Standard account mapping & %-of-revenue defaults for OpEx lines per industry.
* Payroll taxes & benefits as % of gross wages (editable).
* Depreciation schedules from CAPEX (straight-line default).
* Working-capital heuristics (DSO/DPO/Inventory days) if cash-flow enabled.
* Variance commentary suggestions (“Unfavorable Utilities variance due to 8% CPI energy trend”).

**Existing Company: “Trends → Growth” Recipe**

**Step-by-step**

1. **Decompose history**: for each revenue and major cost line, estimate
   * Level (last 3–6 mo avg)
   * **Seasonality** (12-month pattern)
   * **Trend** (YoY slope)
2. **Apply user growth**: convert “+15% YoY” to a **monthly growth factor** ((1+0.15)^(1/12)), multiply onto trend.
3. **Apply macro adjustors** (optional):
   * If revenue linked to GDP, scale growth by β\_GDP \* ΔGDP (β set by user or learned).
   * If costs linked to CPI, inflate monthly by the CPI forecast path.
4. **Apply drivers** (headcount, %-of-revenue, COGS %, unit costs).
5. **Constraints & lags**: hiring lead times, sales capacity caps, marketing→pipeline lag.
6. **Snap to accounts**: write monthly values into budget\_monthly (account\_std, month, amount).

**Pseudo-code**

def autobudget\_existing(actuals, assumptions, macros):

# 1) estimate level/seasonality/trend

prof = profile\_series(actuals) # returns {acct: {"level":..., "seasonality":[...], "trend":...}}

out = {}

for m in horizon\_months():

out[m] = {}

for acct in target\_accounts():

base = prof[acct]["level"] \* prof[acct]["seasonality"][month\_index(m)]

trend = apply\_trend(base, prof[acct]["trend"], m)

# 2) user growth

trend \*= monthly\_growth\_factor(assumptions, acct, m)

# 3) macro links

trend \*= macro\_multiplier(macros, assumptions, acct, m)

# 4) drivers (e.g., % of revenue, headcount)

val = apply\_drivers(trend, assumptions, acct, m)

# 5) caps/constraints

val = apply\_constraints(val, assumptions, acct, m)

out[m][acct] = round(val, 2)

return out

**Macro-Driven Growth (GDP/CPI) — Clean & Flexible**

* **Link types** per line item:
  + **Scale**: value \*= (1 + β \* Δindex) (e.g., revenue vs GDP, COGS vs commodity).
  + **Inflate**: value \*= (CPI\_path[m]/CPI\_base) for wages/utilities.
  + **Blend**: value = α \* driver\_based + (1-α) \* time\_series\_forecast.
* **User controls**:
  + Turn link **on/off** per line.
  + Set **β** (sensitivity) and **α** (blend weight).
  + Provide **custom macro series** (they can paste their own index).

**New Company: Industry Packs + Wizard**

**Industry preset examples (editable):**

* **SaaS**: 80–85% gross margin, COGS = hosting + support headcount; S&M 25–40% of revenue; R&D 15–25%; G&A 8–12%; ramp via ACV, win-rates, sales capacity.
* **Retail**: COGS 55–70% of revenue; rent per sq ft; staff per store-hour; strong holiday seasonality; card fees.
* **Services/Childcare**: revenue = capacity × utilization × price; staffing ratios; seasonality by school calendar; supplies as % of revenue.

**Wizard collects:**

* Units (seats, stores, AEs), pricing, utilization/volume.
* Hiring plan & wages/benefits (role library included).
* Rent/lease & key fixed costs; utilities; insurance; software tools.
* Chosen macro links (CPI for wages, local utility CPI for utilities, etc.)

**Make It the “Best” Budgeting/Forecasting Tool**

1. **Hybrid engine (Drivers ⨉ Time-Series ⨉ Macro)**
   * Per line, let users pick **Driver**, **TS**, or **Blend** with a slider.
   * Display contribution bars: what % of each number came from each component.
2. **Scenario fabric**
   * Global switches (Base / Best / Worst).
   * Shared assumption sets + deltas (“Best = +2 pts conversion, −1 mo ramp”).
   * One-click **Waterfall** from Base → Scenario.
3. **Learning loop**
   * Track monthly variance; **learn better β (macro sensitivity)**, better seasonality, and driver elasticities.
   * “Next year’s auto-budget” uses learned parameters unless the user locks them.
4. **Narrative AI**
   * Turn numbers into **board-ready commentary**: drivers, risks, KPI callouts.
   * Inline explainers: click any cell → “Why this value?” with the provenance (seasonality, macro, driver, manual override).
5. **Headcount-aware planning**
   * Roster view with **hiring dates, comp bands, benefits, payroll taxes**, and automatic mapping to departmental OpEx.
   * Capacity constraints in revenue engine (esp. sales & fulfillment).
6. **Approval workflow**
   * Versioning, submit/approve with redlines; cell-level comments & tasks.
   * Audit trail of all changes and overrides.
7. **Excel-friendly, API-first**
   * CSV/Excel import/export everywhere.
   * Connectors: QBO first; Xero/Netsuite later.
8. **Data model adds (to support the above)**
   * macro\_series(name, month, value)
   * line\_link(line\_id, macro\_name, beta, mode)
   * learned\_params(line\_id, seasonality[12], beta\_hat, elasticity)
   * assumption\_version(id, created\_by, created\_at) with diff tracking.

**Minimal Algorithms (copy-paste ready)**

**Monthly growth factor from annual target**

def monthly\_growth\_factor(assumptions, acct, month):

yoy = assumptions.get("yoy\_growth", {}).get(acct, assumptions.get("yoy\_growth", {}).get("Total Revenue", 0.0))

return (1.0 + yoy) \*\* (1/12)

**Macro multiplier (GDP/CPI link)**

def macro\_multiplier(macros, assumptions, acct, month):

link = assumptions.get("macro\_links", {}).get(acct) # e.g., {"type":"scale","series":"GDP","beta":0.6}

if not link: return 1.0

series = macros[link["series"]] # dict {"2026-01": idx\_val, ...}

base\_m = macros["base\_month"]

if link["type"] == "scale":

delta = (series[month] - series[base\_m]) / series[base\_m] # e.g., ΔGDP%

return 1 + link.get("beta", 0.5) \* delta

if link["type"] == "inflate": # e.g., CPI

return series[month] / series[base\_m]

return 1.0

**Blend driver & time-series**

def blended\_value(driver\_val, ts\_val, alpha=0.7):

return alpha \* driver\_val + (1 - alpha) \* ts\_val

**UX: Keep It Simple for Students/Users**

* **“AI Plan” Panel**: user types “grow 12% YoY; marketing 11% of revenue; add 2 AEs in April; CPI link wages” → parsed assumptions appear as chips with toggles and sliders.
* **“Why” Drawer** on any cell with the exact math (trend × seasonality × growth × macro × driver ± override).
* **Scenario Toggle** at the top; differences glow.

Done! I dropped a full **implementation pack** into the canvas with:

* backend code for **budget version diffs**, **YoY period compares**, **AI-assisted BvA** (root-cause + action suggestions), and **cell-level provenance** (“Why this number?”),
* starter **auto-budget** engines (existing company + new-company presets),
* and **React** components for a clean, modern UI (diff table, insights panel, “Why” drawer, variance pills).

erfect—added everything you asked for:

* **AI ON/OFF per line/cell** with priority **Manual > Driver > AI** and a tiny mode pill UI.
* **Multi-currency + consolidation:** FX tables, monthly conversion, group roll-ups across entities/countries.
* **Depreciation engine:** CAPEX → monthly straight-line schedules into Opex:Depreciation.
* **Per-account growth rules:** grow from prior year/month/avg with %; preview & apply.
* **Payroll increases:** enter % raise and a **start year** (global/entity/department scope).
* **Nested departments:** adjacency-list roll-ups so you can budget at any level (leaf → dept → entity → group).
* **Endpoints + React bits** to wire it all.

All set — I added a full **Python executive dashboard** (Plotly Dash) plus helper modules into the canvas:

* backend/charts.py — KPI computation + **trend, YoY, waterfall, heatmap** charts.
* backend/bva\_math.py — clean **BvA math** and filters for the dashboard.
* dashboard\_app.py — a **pure-Python Dash app** with scenario & year filters, KPI cards, BvA visuals, and an **AI narrative** panel (“why margins were down in Q2”). Also a **Generate Board Pack** button that calls the Python **board pack** code you already have.
* server.py — optional glue to mount **FastAPI + Dash** on one port.
* Notes on swapping CSVs for **DB queries** (SQLAlchemy/pandas) so it’s production-ready.

This complements everything we already built (auto-budgets, FX, depreciation, payroll increases, hierarchy rollups, Excel sync, QuickBooks import, version control, AI variance). If you want, I can also drop **unit tests** for the dashboard data layer and a **Dockerfile** for a one-command run.

Added! I embedded a full **multidimensional OLAP layer**—Anaplan/Hyperion/Cognos–style—into the canvas, including:

* A Python **cube engine** (cube\_core.py) with dimensions, hierarchies, sparse storage, roll-ups, allocations, writeback, versioning & what-ifs.
* **Driver rules**, **FX handling**, **intercompany eliminations**, and **cost allocations** modules.
* End-to-end **examples**, MDX-like slicing, and exercises for students.
* Clear hooks to your existing BvA/auto-budget/Dash dashboards, Excel sync, and board packs.

Boom — I added:

* A full **pytest test suite** (math, FX, depreciation, growth, payroll, hierarchy, cube, API, and dashboard smoke tests).
* **Dockerfile** for the backend (FastAPI + Dash) and a **docker-compose** that brings up Postgres + the app.
* A handy **Makefile**, **.env example**, a **seed loader script**, and a **GitHub Actions** CI workflow that runs tests on push.
* A “**one-command demo**” section so students can run the whole thing quickly.

Dialed in for usability and SMBs. I added to the canvas:

* A polished **design system & UX patterns** (chips, mode pills, command palette, mobile-first, tooltips, empty states, keyboard shortcuts, dark mode).
* A complete **Quick Service Restaurant (QSR) preset** targeted at a small Taco Bell–style shop:
  + Python generator presets\_qsr.py (traffic × avg ticket; COGS/labor/royalty/ad %; rent/utilities; seasonality),
  + FastAPI endpoint /presets/qsr,
  + A one-screen **QSR Wizard (React)** to produce a ready budget in ~5 minutes.
* A **15-minute setup script** and **Excel template workflow** so even non-finance users can create a budget fast.
* UI “easy wins”: assumption chips (click-to-edit), provenance “Why?”, colorful variance pills, sticky totals, toasts, undo/redo.
* This file will run with:
* pip install fastapi uvicorn dash plotly pandas numpy statsmodels python-pptx
* python fpna\_app.py
* → Opens API at http://127.0.0.1:8000 and dashboard at http://127.0.0.1:8050.

**. Student Guide (Accompanying)**

Sections will include:

1. **Setup** (Python install, dependencies).
2. **Run the app**.
3. **Explore endpoints** (/budget/autogen, /bva/analyze, /presets/qsr).
4. **Use the dashboard** (KPI cards, charts).
5. **Generate your first Taco Bell budget** (wizard endpoint).
6. **Change assumptions** (growth %, labor %).
7. **Compare scenarios** (Base vs Best vs Worst).
8. **Export a board pack**.
9. **AI narrative**: run “Why were margins down in Q2?” example.
10. **Stretch goals**: add your own preset (e.g., SaaS business).

just updated your canvas with:

* #58–65: **How Hyperion’s Excel linkups work** and our **parity plan**.
* Drop-in **API endpoints** for your single-file app:
  + POST /excel/retrieve — ad-hoc pulls by POV (supports simple wildcards like Revenue:\*)
  + POST /excel/submit — writeback from a named range (Smart View–style)
  + GET /excel/expand\_children — stub for hierarchy expand/zoom
* Three Excel client options with code:
  1. **Power Query (M)** for no-code retrieve
  2. **Office Script (TypeScript)** to submit a named range
  3. **xlwings (Python)** for a lightweight local add-in (retrieve + submit)
* Security/locks guidance, and a **student mini-lab** to build the Excel round-trip.

I added a full **Office-JS Excel Web Add-in** to the canvas that mirrors Hyperion Smart View:

* **Manifest** (manifest.xml) for sideloading
* **Task pane UI** (taskpane.html) with POV pickers (Year, Scenario, Accounts), buttons (Retrieve/Submit/Refresh/Expand)
* **Functions hook** (functions.html) so you can open the task pane from the ribbon
* Calls your existing endpoints:
  + POST /excel/retrieve (ad-hoc pulls with wildcards like Revenue:\*)
  + POST /excel/submit (writeback from a sheet range)
  + GET /excel/expand\_children (hierarchy expand stub)

I also included:

* HTTPS dev server instructions (so Excel trusts the add-in),
* Sideload steps for Windows/Mac Excel and Excel web,
* Notes on hardening (auth, validation, locks).

I bundled **everything** (full code + full student guide) and added **auth tokens** across the API and the Excel add-in.

**What you’re getting**

* fpna\_app.py — single-file app with:
  + FastAPI + Plotly Dash executive dashboard
  + QSR preset, auto-budget, scenarios, BvA with AI narrative
  + Board pack exporter (pptx)
  + **Excel endpoints** (/excel/retrieve, /excel/submit, /excel/expand\_children)
  + **Bearer token auth** (APP\_TOKEN) enforced on protected endpoints
* STUDENT\_GUIDE.md — complete step-by-step guide (setup, auth, API calls, dashboard, Excel linkups, scripts)
* excel\_addin/ — Office-JS **Excel Web Add-in**:
  + manifest.xml, taskpane.html, functions.html
  + Task pane includes a **token input**; sends Authorization: Bearer <token>

**Download**

Download the full bundle (zip)

unzip fpna\_singlefile\_with\_excel\_auth.zip

cd fpna\_singlefile\_pack

pip install fastapi uvicorn dash plotly pandas numpy python-pptx

export APP\_TOKEN=secret123 # optional; recommended

python fpna\_app.py

# API: http://127.0.0.1:8000/docs

# Dashboard: <http://127.0.0.1:8000/dashboard>

**Project Guide — “AI-First Budgeting & Forecasting Platform”**

**1. What are we building?**

Think about how families plan their money — income, groceries, bills, savings. Businesses do the same thing, but on a bigger scale, called **budgets and forecasts**.

We’re building **software that helps businesses plan their money automatically**. Instead of crunching spreadsheets, the software will:

* Create **budgets** (planned income/expenses),
* Compare with **actual results** (real income/expenses),
* Show **graphs and dashboards** so owners see what’s working,
* Use **AI** to explain what happened (“Costs went up because food prices rose 15%”) and predict the future (“If you add two new employees in June, profit may fall by 8%”).

The goal: **make budgeting as easy as clicking a few buttons** — even for a small business like a Taco Bell franchise, a daycare, or a coffee shop.

**2. Why does this matter?**

* Today, small businesses use **Excel** or expensive tools like **DataRails** or **Hyperion**.
* These tools are either **too simple** (just spreadsheets) or **too complex** (designed for huge corporations).
* Our platform is **just right**:
  + Easy enough for a small shop to set up in **5 minutes**,
  + Powerful enough to handle multiple departments, currencies, and scenarios,
  + Smart enough to let owners focus on **decisions, not data entry**.

**3. How AI makes it smarter**

AI in our platform will do **four big things**:

1. **AI Budgets**
   * New company? AI uses industry templates (restaurants, childcare, SaaS) to create a starter budget.
   * Existing company? AI studies past months and auto-generates next year’s forecast, adjusting for growth, GDP, or inflation.
2. **AI Forecasting**
   * Every month, when new data comes in, AI re-forecasts the rest of the year automatically.
   * Example: “Sales dropped 8% in July, so projected year-end revenue will be $1.2M, not $1.3M.”
3. **AI Analysis**
   * AI explains **why numbers changed**.
   * Example: “Q2 margin declined because labor overtime was $12k higher than budget, and beef costs rose 15% vs. plan.”
4. **AI Recommendations**
   * AI suggests actions:
     + “Switch suppliers to save 5% on food costs,”
     + “Stagger staffing on weekdays to cut overtime,”
     + “If you invest $20k in marketing, you can grow sales by 10%.”

**4. Why we’re better than DataRails, Hyperion, Cognos, etc.**

* **DataRails**: Good for Excel users, but limited AI, limited scenarios, and no instant industry presets.  
  ➝ We add **AI forecasting, variance explanations, board packs, and 5-minute setup.**
* **Hyperion/Essbase**: Super powerful, but requires consultants and months of setup.  
  ➝ We give **Hyperion power in a student-friendly package** with modern UI and one-file setup.
* **Cognos/Anaplan**: Enterprise-grade, but too expensive and complex.  
  ➝ We’re **affordable for SMBs**, still with multi-currency, scenarios, and department rollups.

**Our positioning:**  
👉 *“As powerful as enterprise FP&A tools, but as easy to use as an iPhone app.”*

**5. Features that make this the best**

* **Graphs and Dashboards**: Revenue, costs, profit, margins, variance waterfalls, heatmaps.
* **AI Forecasting**: Auto-update forecasts when actuals arrive.
* **Scenario Planning**: Base/Best/Worst cases with one-click comparisons.
* **Multi-currency**: Convert USD, EUR, INR — consolidate across countries.
* **Headcount/Payroll**: Add employees, contractors, and payroll increases.
* **Depreciation & Rent**: Auto-schedules for CAPEX and fixed costs.
* **Excel Sync**: Retrieve/Submit data like Hyperion Smart View.
* **Board Packs**: Generate PowerPoint decks with AI narratives instantly.
* **Usability**: Wizards, templates, tooltips — easy for small business owners.

**6. Deliverables — Over 2 Semesters**

**📅 Semester 1 (Foundations)**  
Goal: Build the **core budgeting engine** + simple UI.

* Set up **accounts and actuals** (revenue, COGS, expenses).
* Build **preset generator** (e.g., Taco Bell QSR, childcare).
* Implement **auto-budget from actuals** (trend + growth).
* Create **executive dashboard** with 3 charts: Trend, YoY, Waterfall.
* Build API endpoints for budgets, scenarios, and BvA.
* Add **AI narrative (rule-based)** to explain variances.
* Deliverable: Working single-file app + dashboards, QSR preset.

**📅 Semester 2 (Advanced + AI)**  
Goal: Add **AI forecasting, advanced features, and integrations.**

* Add **AI forecasting** (ARIMA, Prophet, ML regression).
* AI variance analysis with natural language explanation.
* **Scenario planning** with Base/Best/Worst and sensitivity toggles.
* Add **multi-currency** and **department hierarchies**.
* Add **Excel add-in** for retrieve/submit (Smart View-style).
* Build **board pack exporter** with AI executive summary.
* Add usability: setup wizard, command palette, tooltips.
* Deliverable: Full SMB budgeting platform — **better than DataRails**.

pdated your canvas with:

1. **fpna\_app.py — v2 (Advanced + AI)**

* New features now built-in:
  + **AI forecasting** endpoint with **ARIMA**, **Prophet** (optional), and a **simple ML regressor** → POST /forecast/ai
  + **Scenario sensitivity** endpoint to apply targeted deltas (e.g., “COGS +10% in Q3”) → POST /scenario/sensitivity
  + **Multi-currency** (monthly FX rates) with auto-conversion to functional currency
  + **Department hierarchies** (Sales/Ops → HQ) and rollups
  + Improved **AI variance narrative** (quarter callouts + top drivers)
  + **Setup Wizard** endpoint to create presets for QSR and Childcare → POST /wizard/create
  + Everything from before: dashboards (Trend, YoY, Waterfall, Heatmap), Excel retrieve/submit, board packs, scenarios, BvA

1. **Student Guide — Two-Semester Edition (v2)**

* Read-aloud roadmap that maps 1:1 to your **Semester 1** and **Semester 2** goals.
* Clear “what’s pre-built” list, so teams can immediately focus on:
  + **UI polish** (assumption drawer, command palette, tooltips)
  + **Testing** (math utilities, FX, API, UI flows)
  + **Creative AI** (extra narratives, causal drivers, anomaly detection)

pip install fastapi uvicorn dash plotly pandas numpy python-pptx statsmodels scikit-learn

# Optional Prophet:

# pip install prophet

export APP\_TOKEN=secret123 # optional, for protected endpoints

python fpna\_app.py

# API: http://127.0.0.1:8000/docs

# Dashboard: <http://127.0.0.1:8000/dashboard>

**What’s inside**

* **backend/** (FastAPI)
  + app.py — budgets, scenarios, GDP-driven presets, AI forecasts (ARIMA/ML), AI variance explanations, board pack export
  + requirements.txt
* **frontend/** (React + Vite + Recharts)
  + index.html, vite.config.js, package.json
  + src/App.jsx — main app shell (Dashboard, Wizard, Scenarios)
  + src/components/Dashboard.jsx — AI narrative, variance drivers chart, hotspots table, forecasting panel
  + src/components/Wizard.jsx — 5-minute setup for a Taco-Bell–style budget (with GDP growth)
  + src/components/Scenarios.jsx — clone Best/Worst, apply sensitivities (e.g., “Labor +8% from June”)
* **STUDENT\_GUIDE.md** — runbook + roadmap + stretch ideas

**Quick start**

Backend:

cd backend

python -m venv .venv && source .venv/bin/activate # Windows: .venv\Scripts\activate

pip install -r requirements.txt

uvicorn app:api --reload --port 8000

Frontend:

cd frontend

npm install

npm run dev

Open:

* API docs: http://127.0.0.1:8000/docs
* App UI: http://127.0.0.1:5173

**Highlights 5-minute setup**: Wizard builds a full budget for 12 months (with seasonality + GDP growth).

* **AI forecasting**: ARIMA or ML autoregression for 12-month predictions.
* **AI explanations**: Automatic narrative for BvA (e.g., “Costs went up because food prices rose ~15%.”).
* **Scenario planning**: Clone Base/Best/Worst and apply targeted sensitivities (e.g., COGS +10% in Q3).
* **Multi-department + FX**: Sales/Ops/HQ with functional-currency conversion.
* **Executive outputs**: Generate a Board Pack PPT with KPIs and summary.

**Student focus (what’s left to do)**

* Get software to work and iron out issues/bugs. Test functionality in the software
* **UI polish**: real waterfall & heatmap, KPI cards, tooltips, dark mode.
* **Testing**: unit tests for math/FX, API smoke tests, UI e2e (Playwright).
* **Creative AI**: causal drivers (weather/promos), anomaly detection, smarter narratives, chat-style “budget copilot”.
* **Integrations**: QuickBooks importer, Excel add-in (use the add-in we built earlier), CSV import/export.

V3 file:

**What’s new in v3 (multi-file)**

* **Backend (FastAPI)** — /backend/app.py
  + **Headcount endpoints**
    - GET /headcount/list — view roster
    - POST /headcount/upsert — add/update an employee (dept, start, salary, FTE, raise month/% , benefits%, taxes%)
    - POST /headcount/bake\_to\_budget — auto-generate monthly **Opex:Labor / PayrollTaxes / Benefits**for a fiscal year and merge into a scenario.
  + **Goal Seek solver**
    - POST /solver/goal\_seek — set target\_ebitda and (optionally) apply: true. Coarse grid search suggests % changes to **Revenue / COGS / Labor** to get closest to target, and can apply to the scenario in one click.
* **Frontend (React + Vite + Recharts)**
  + New **Headcount** page: maintain roster, then “Bake to Budget”.
  + New **Solver** page: enter target EBITDA, see suggested levers, optionally apply.
* **Tests (pytest)**
  + tests/test\_v3.py validates headcount bake pipeline and solver call.

**How to run**

Backend:

cd backend

python -m venv .venv && source .venv/bin/activate # Windows: .venv\Scripts\activate

pip install -r requirements.txt

uvicorn app:api --reload --port 8000

Frontend:

cd frontend

npm install

npm run dev

Open:

* API docs → http://127.0.0.1:8000/docs
* App UI → http://127.0.0.1:5173

**Student Guide**

The ZIP includes **STUDENT\_GUIDE.md** with a quick walkthrough:

* Create a 2027 Base plan in the Wizard.
* Add employees in **Headcount** (e.g., Ops hire in June, 5% raise in Sep).
* **Bake to Budget** → see payroll flow into Opex.
* Use **Solver** to target EBITDA (e.g., $500k) and apply suggested % changes.

V3.1

* backend/app.py — **complete** FastAPI backend with:
  + /migrate/datarails, /migrate/netsuite, /migrate/hyperion
  + /migrate/{system}/csv (CSV uploads with options JSON)
  + Dry-run reporting, mapping overrides, and idempotent upserts to **Actuals** or a **Budget scenario**
  + QSR preset generator (/preset/qsr) so you can test BvA endpoints quickly
  + Minimal BvA endpoint for sanity checks
* backend/requirements.txt
* backend/tests/test\_migrations.py (you can extend)
* frontend/README.txt
* MIGRATION\_GUIDE.md

cd backend

python -m venv .venv && source .venv/bin/activate # Windows: .venv\Scripts\activate

pip install -r requirements.txt

uvicorn app:api --reload --port 8000

# Docs: http://127.0.0.1:8000/docs

## **Quick smoke test**

1. Create a budget to test against:

curl -X POST http://127.0.0.1:8000/preset/qsr -H "Content-Type: application/json" \

-d '{"fiscal\_year":2027, "gdp\_growth":0.02}'

1. Dry-run a DataRails import to **actuals**:

curl -X POST http://127.0.0.1:8000/migrate/datarails -H "Content-Type: application/json" -d '{

"rows":[

{"Date":"2024-03-01","Account":"Sales - Food","Dept":"Sales","Amount":12000,"Currency":"USD"},

{"Date":"2024-03-01","Account":"Wages","Dept":"Ops","Amount":4000,"Currency":"USD"}

],

"options": {"to":"actuals","dry\_run": true}

}'

1. Apply a NetSuite import to a **budget**:

curl -X POST http://127.0.0.1:8000/migrate/netsuite -H "Content-Type: application/json" -d '{

"rows":[

{"Tran Date":"2027-01-31","Account":"Sales - Beverage","Department":"Sales","Amount":5000,"Currency":"USD"},

{"Tran Date":"2027-01-31","Account":"COGS - Paper","Department":"Ops","Amount":-300,"Currency":"USD"}

],

"options": {"to":"budget","scenario":"2027:Imported","dry\_run": false}

}'