Jad CHEBLY
BCSAI

Assignment 2 Report

Grade & What-If Tracker

Github Repository: https://github.com/jadchebly/SD_GradeAndWhatIfTracker.git

1. Code Quality and Testing Improvements

1.1 Refactoring for Maintainability

- **Separation of Concerns (SoC):**
  Database access was moved into a dedicated *repository layer*, and business logic into a *service layer*. Route handlers now only handle HTTP-level input/output.
- **Reduced Duplication:**
  Shared CRUD, validation, and error-handling logic is now centralized instead of repeated across endpoints.
- **Improved File Structure:**
  Backend modules are now cleanly separated:
  - `app.py` → routing, DI, wiring
  - `db.py` → database configuration
  - `services.py` → business logic
  - `models.py` → ORM models
  - `schemas.py` → Pydantic I/O
  - `calculations.py` → numeric logic for stats/what-if

1.2 SOLID Principles

- **Single Responsibility Principle**
  Each module handles only one concern.
  Example:
  - `app.py` configures the application & registers routes.
  - `db.py` handles DB connections only.
  - `calculations.py` handles all GPA/weight logic.
- **Open/Closed Principle**
  Adding new assessment types or stats features **no longer requires modifying core logic**, only adding new service methods.

- **Dependency Inversion Principle**
  Services depend on an abstract repository instead of the database session directly, improving testability.

1.3 Removal of Hard-coded Values

- Replaced hardcoded DB URLs with environment-driven configuration.
- Settings centralized in `settings.py` (Pydantic Settings).
- All sensitive values moved to GitHub & Railway secrets.

1.4 Input Validation

- Replaced ad-hoc checks with **strict Pydantic validation**, automatically checking:
  - weight ranges
  - score ranges
  - date formats
  - required fields

1.5 Testing Improvements

A major expansion of the test suite was completed:

Unit Tests Added

- `test_calculations_unit.py` — fully tests weighted grade logic
- `test_services_unit.py` — tests CRUD service behavior

API Integration Tests Added

Using FastAPI's `TestClient` + an **in-memory SQLite DB** fixture:

- CRUD tests (`test_api_assessments.py`)
- Validation tests (`test_api_validation.py`)
- Not-found behavior (`test_api_notfound.py`)
- Stats & edge cases (`test_api_stats.py`, `test_stats_edges.py`)

Testing Architecture Improvements

- Shared `override_get_db` in `conftest.py` ensures **DB isolation per test** using `StaticPool`.
- Schema reset before every test ensures deterministic behavior.

Coverage Requirement Met

- Coverage > **70%** achieved (validated in CI).

- Coverage report automatically generated as `coverage.xml`.

---

2. Continuous Integration Pipeline

A full CI pipeline was created at `.github/workflows/ci.yaml`.
The pipeline performs the following steps:

2.1 Install Dependencies

- Uses Python 3.11
- Installs backend requirements from `backend/requirements.txt`

2.2 Run Tests + Coverage

```
pytest --cov=backend --cov-report=xml --cov-fail-under=70
```

Pipeline **fails if coverage < 70%**, enforcing code quality.

2.3 Build Application

- Docker image is built using the project's `Dockerfile`.
- Ensures that the production container environment matches local development.

2.4 Continuous Deployment Trigger

- Deployment runs **only on pushes to `main`**, as required.
- GitHub Actions uses a Railway token stored securely in GitHub Secrets.

3. Deployment Automation (CD)

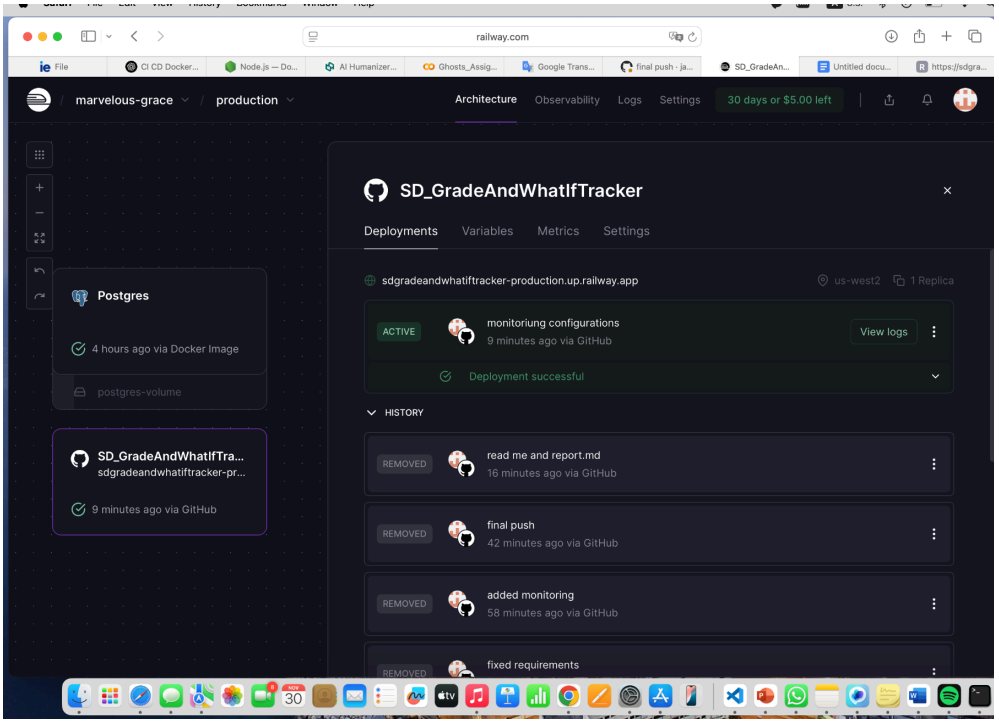
3.1 Dockerization

Your backend is containerized using a clean, production-ready `Dockerfile`:

- Installs dependencies
- Copies backend + frontend
- Exposes port 8000
- Runs the app via Uvicorn

This ensures reproducible deployments.

3.2 Deployment to Railway

- GitHub repo is linked directly to a Railway Production environment.
- Every push to `main` → triggers GitHub Actions → builds container → deploys to Railway.
- Railway uses auto-detected port **8000** from Uvicorn.
- Secrets such as `DATABASE_URL` and environment variables are configured in Railway, not committed.
- 

3.3 Secrets & Trigger Configuration

- *Only* the `main` branch deploys.
- Secrets stored securely under GitHub → Settings → Secrets → Actions:
  - `RAILWAY_TOKEN`
- `.env` values overwritten by Railway environment configuration.

4. Monitoring and Health Checks

Monitoring was added as required.

4.1 Health Check Endpoint

Implemented:

`/health → { "ok": true }`

Used by:

- GitHub Actions (optional)
- Railway service health
- Local debugging

4.2 Prometheus Metrics

Prometheus-compatible metrics are exposed via:

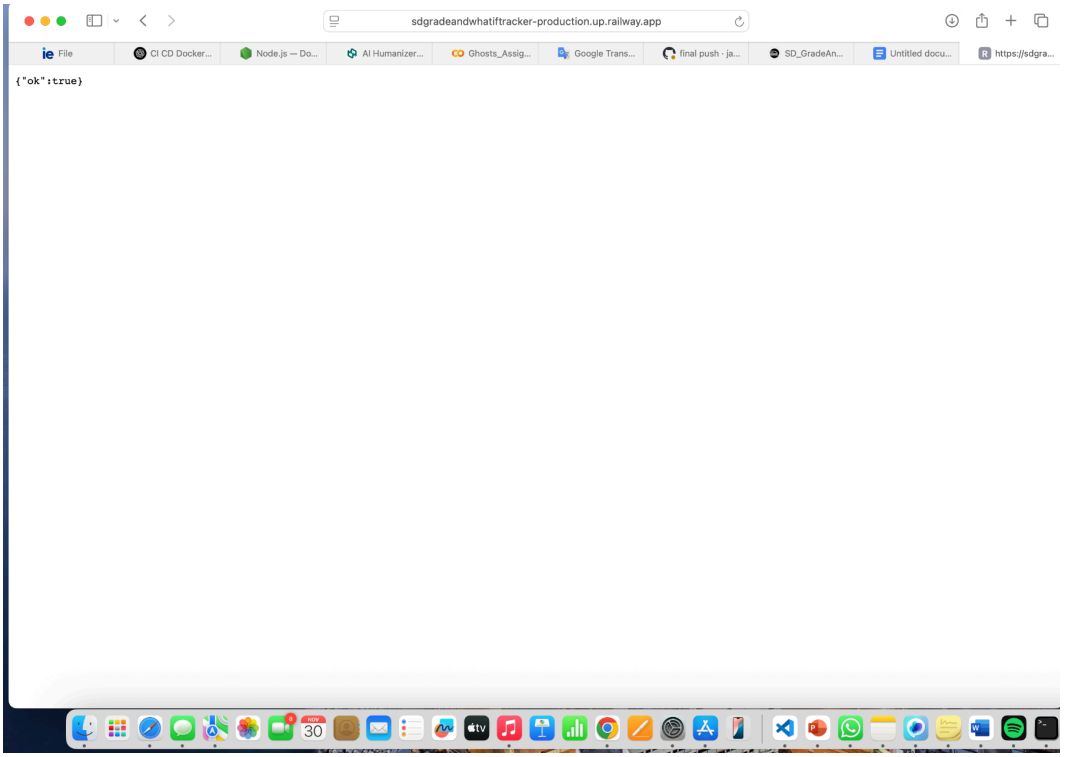
`/metrics`

Includes:

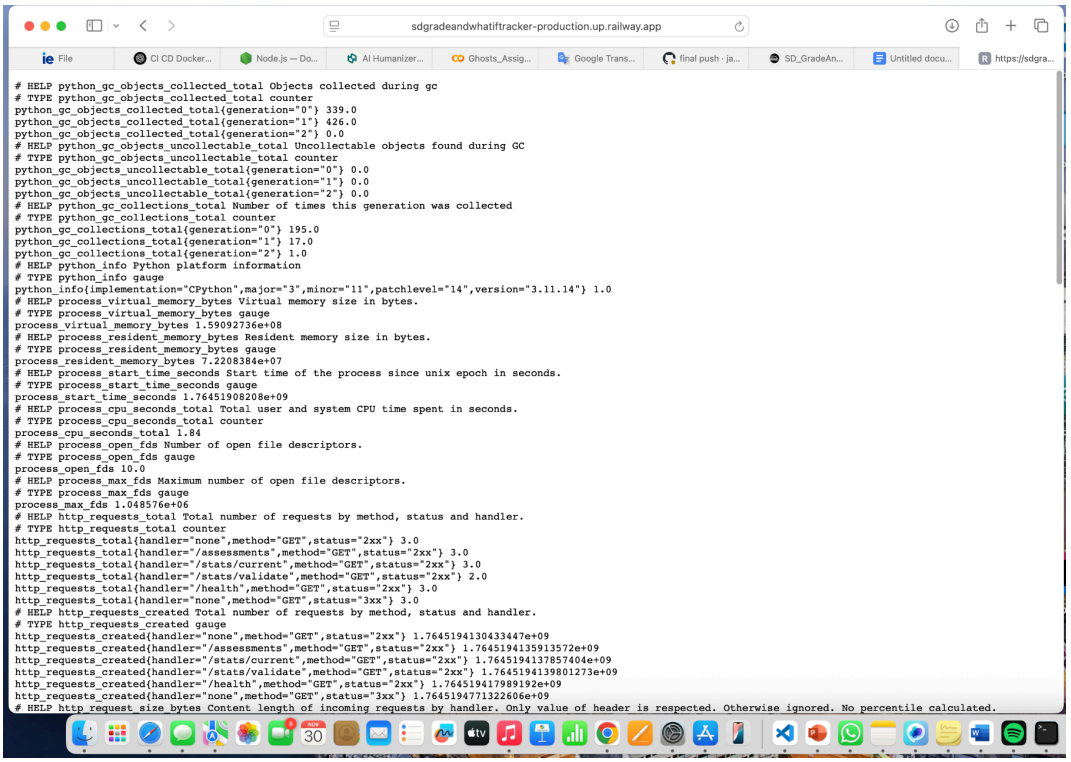
- request count
- request latency
- memory usage
- CPU usage
- handler-specific FastAPI metrics

Generated by:

- `prometheus_fastapi_instrumentator` *(primary)*
- `prometheus_client` *(fallback)*



Jad CHEBLY
BCSAI



Appendix:

````
# Project Codebase



## Root



###  tranformation.py

```py
import os

# Project root (change if needed)
PROJECT_ROOT = "."

# Output file
OUTPUT_FILE = "codebase.md"
````
````

````
# Extensions to include
INCLUDE_EXTS = (".py", ".html", ".css", ".js")

with open(OUTPUT_FILE, "w", encoding="utf-8") as out:
   out.write("# Project Codebase\n\n")
   for root, dirs, files in os.walk(PROJECT_ROOT):
       # Skip virtual envs or hidden folders
       if any(skip in root for skip in [".git", "__pycache__", ".venv",
"node_modules"]):
           continue

       # Write folder name
       rel_path = os.path.relpath(root, PROJECT_ROOT)
       if rel_path == ".":
           rel_path = "Root"
       out.write(f"\n\n## {rel_path}\n\n")

       for file in sorted(files):
           if file.endswith(INCLUDE_EXTS):
               filepath = os.path.join(root, file)
               out.write(f"\n###  {file}\n\n")
               out.write("```" + filepath.split(".")[-1] + "\n")  # syntax
highlight
               try:
                   with open(filepath, "r", encoding="utf-8") as f:
                       out.write(f.read())
               except Exception as e:
                   out.write(f" Could not read file: {e}")
               out.write("\n```\n")

print(f" Codebase exported to {OUTPUT_FILE}. Now run:")
print("   pandoc codebase.md -o codebase.pdf   # convert to PDF")
```


## frontend


###  app.js

```js
// --- API base: use localhost to avoid IPv4/IPv6 mismatches ---
const API = ""; // same-origin (e.g., /assessments, /stats/current)
console.log("app.js v3; API base:", API || "(same-origin)");
````

Jad CHEBLY
BCSAI

```
// --- Grab elements explicitly (no relying on globals) ---
const els = {
 title: document.getElementById("title"),
 weight: document.getElementById("weight"),
 due: document.getElementById("due"),
 score: document.getElementById("score"),
 addBtn: document.getElementById("add"),
 tableBody: document.querySelector("#table tbody"),
 current: document.getElementById("current"),
 remaining: document.getElementById("remaining"),
 weightsMsg: document.getElementById("weightsMsg"),
 target: document.getElementById("target"),
 calcBtn: document.getElementById("calc"),
 answer: document.getElementById("answer"),
};

// --- Track editing state (Add vs Update) ---
let editingId = null;

function setEditingMode(assessment) {
 editingId = assessment.id;
 els.title.value = assessment.title;
 els.weight.value = assessment.weight_pct;
 els.due.value = assessment.due_date; // API is YYYY-MM-DD
 els.score.value = assessment.score_pct ?? "";
 els.addBtn.textContent = "Update";
 ensureCancelButton();
}

function clearEditingMode() {
 editingId = null;
 els.title.value = "";
 els.weight.value = "";
 els.due.value = "";
 els.score.value = "";
 els.addBtn.textContent = "Add / Update";
 removeCancelButton();
}

function ensureCancelButton() {
 if (document.getElementById("cancel-edit")) return;
 const btn = document.createElement("button");
 btn.id = "cancel-edit";
 btn.type = "button";
```

```
 btn.textContent = "Cancel";
 btn.style.marginLeft = ".5rem";
 btn.onclick = clearEditingMode;
 els.addBtn.insertAdjacentElement("afterend", btn);
}

function removeCancelButton() {
 const btn = document.getElementById("cancel-edit");
 if (btn) btn.remove();
}

async function fetchJSON(url, opts = {}) {
 const r = await fetch(url, {
   headers: { "Content-Type": "application/json" },
   ...opts,
 });
 if (!r.ok) {
   const msg = await r.text().catch(() => r.statusText);
   throw new Error(`${r.status} ${msg}`);
 }
 return r.status === 204 ? null : r.json();
}


async function load() {
 // List assessments
 const rows = await fetchJSON(`${API}/assessments`);
 els.tableBody.innerHTML = "";
 rows.forEach((r) => {
   const tr = document.createElement("tr");
   tr.setAttribute("data-id", r.id);
   tr.innerHTML = `
     <td>${r.title}</td>
     <td>${r.weight_pct}%</td>
     <td>${r.due_date}</td>
     <td>${(r.score_pct !== null && r.score_pct !== undefined) ? r.score_pct :
""}</td>
     <td>
       <button data-id="${r.id}" class="edit">Edit</button>
       <button data-id="${r.id}" class="del">Delete</button>
     </td>
   `;
   els.tableBody.appendChild(tr);
 });
```

```
  // Empty state
 if (rows.length === 0) {
   const tr = document.createElement("tr");
   tr.className = "empty";
   tr.innerHTML = `<td colspan="5">No assessments yet — add your first one above
</td>`;
   els.tableBody.appendChild(tr);
 }


 // Stats
 const stats = await fetchJSON(`${API}/stats/current`);
 els.current.textContent = stats.current_weighted.toFixed(2);
 els.remaining.textContent = stats.remaining_weight.toFixed(2);

 // Weight validation
 const v = await fetchJSON(`${API}/stats/validate`);
 els.weightsMsg.textContent = v.message;
}


// Create (Add / Update button)
els.addBtn.onclick = async () => {
 const payload = {
   title: els.title.value.trim(),
   weight_pct: Number(els.weight.value),
   due_date: els.due.value, // YYYY-MM-DD
   score_pct: els.score.value === "" ? null : Number(els.score.value),
 };

 if (!payload.title || !payload.due_date || Number.isNaN(payload.weight_pct)) {
   alert("Please fill Title, Weight and Due Date.");
   return;
 }

 if (editingId == null) {
   await fetchJSON(`${API}/assessments`, {
     method: "POST",
     body: JSON.stringify(payload),
   });
 } else {
   await fetchJSON(`${API}/assessments/${editingId}`, {
     method: "PUT",
     body: JSON.stringify(payload),
   });
```

```
 }

 await load();
 clearEditingMode();
};



// Delete via event delegation
document.querySelector("#table").onclick = async (e) => {
 if (e.target.classList.contains("del")) {
   const id = e.target.getAttribute("data-id");
   await fetch(`${API}/assessments/${id}`, { method: "DELETE" });
   await load();
 }
};
// Edit via event delegation
document.querySelector("#table").addEventListener("click", async (e) => {
 const btn = e.target.closest("button.edit");
 if (!btn) return;
 const id = Number(btn.dataset.id);
 const a = await fetchJSON(`${API}/assessments/${id}`);
 setEditingMode(a);
});



// What-if
els.calcBtn.onclick = async () => {
 const t = Number(els.target.value);
 if (Number.isNaN(t)) return (els.answer.textContent = "Enter a target %");
 const r = await fetchJSON(`${API}/stats/what-if?target=${t}`);
 els.answer.textContent =
   r.required_avg == null
     ? `No remaining work. Target ${r.target}% is ${r.attainable ? "already met" :
"not met"}.`
     : `You need an average of ${r.required_avg}% on remaining work.
(${r.attainable ? "attainable" : "not attainable"})`;
};

load();

```

index.html

```html
```

Jad CHEBLY
BCSAI

```
<!doctype html>
<html lang="en">
<head>
 <meta charset="utf-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1" />
 <title>Grade & What-If Tracker</title>

 <!-- Minimal favicon so your backend logs stop showing 404 /favicon.ico -->
 <link rel="icon" href="data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 64 64'%3E%3Ccircle cx='32' cy='32'
r='28' fill='%23007aff'/%3E%3Ctext x='32' y='40' font-size='30'
text-anchor='middle' fill='white' font-family='Arial, Helvetica,
sans-serif'%3EG%3C/text%3E%3C/svg%3E" />

 <link rel="stylesheet" href="styles.css" />
</head>
<body>
 <main class="container">
 <h1>Grade &amp; What-If Tracker</h1>

 <!-- Assessment form -->
 <section class="card" aria-labelledby="add-edit">
   <h2 id="add-edit" class="sr-only">Add or Update an Assessment</h2>
   <div id="form" role="form" aria-describedby="form-hint">
     <label>
       Title
       <input id="title" name="title" placeholder="e.g., Midterm"
autocomplete="off" required />
     </label>

     <label>
       Weight %
       <input id="weight" name="weight" type="number" inputmode="decimal" min="0"
max="100" step="0.01"
              placeholder="e.g., 20" required />
     </label>

     <label>
       Due date
       <input id="due" name="due" type="date" required />
     </label>

     <label>
       Score % (optional)
```

```
      <input id="score" name="score" type="number" inputmode="decimal" min="0"
max="100" step="0.01"
             placeholder="e.g., 85" />
    </label>

    <!-- type=button so the page doesn't try to submit/reload -->
    <button id="add" type="button" aria-label="Add or update assessment">Add /
Update</button>
    <p id="form-hint" class="hint">Enter title, weight (0–100), due date, and
score if you have it.</p>
  </div>
 </section>

 <!-- Stats -->
 <section class="card" aria-labelledby="stats-title" id="stats">
  <h2 id="stats-title" class="sr-only">Current Stats</h2>
  <div>Current: <span id="current">0.00</span>%</div>
  <div>Remaining weight: <span id="remaining">100.00</span>%</div>
  <div id="weightsMsg" aria-live="polite"></div>
 </section>

 <!-- What-if -->
 <section class="card" aria-labelledby="whatif-title" id="whatif">
  <h2 id="whatif-title" class="sr-only">What-If Calculator</h2>
  <label>
    Target %
    <input id="target" name="target" type="number" inputmode="decimal" min="0"
max="100" step="0.01"
           placeholder="e.g., 85" />
  </label>
  <button id="calc" type="button">What do I need?</button>
  <div id="answer" aria-live="polite"></div>
 </section>

 <!-- Table -->
 <section class="card" aria-labelledby="assessments-title">
  <h2 id="assessments-title" class="sr-only">Assessments</h2>
  <table id="table" role="table">
    <thead>
      <tr>
        <th scope="col">Title</th>
        <th scope="col">Weight</th>
        <th scope="col">Due</th>
        <th scope="col">Score</th>
        <th scope="col" aria-label="Actions"></th>
```

```
      </tr>
    </thead>
    <tbody></tbody>
  </table>
</section>

<!-- Keep the script last (or add defer) so elements exist when JS runs -->
<script src="app.js?v=2" defer></script>


<!-- Optional: hide-only-for-screenreaders utility -->
<style>
  .sr-only {
    position: absolute !important;
    width: 1px; height: 1px;
    padding: 0; margin: -1px;
    overflow: hidden; clip: rect(0,0,1px,1px);
    white-space: nowrap; border: 0;
  }
  .hint { font-size: .9rem; color: #555; }
  #form label { display: inline-flex; flex-direction: column; margin-right:
.75rem; margin-bottom: .5rem; }
</style>
</main>
</body>
</html>


```

styles.css

```css
/* ===== Light, airy theme ===== */
:root{
 --bg: #f7f8fb;                 /* page bg (very light) */
 --bg-grad: radial-gradient(1200px 600px at 20% -10%, #eef2ff 0%, transparent 60%),
            radial-gradient(900px 500px at 120% 0%, #eaf7ff 0%, transparent 55%);
 --card: #ffffff;               /* card bg */
 --ink: #0f172a;                /* text */
 --muted: #64748b;              /* secondary text */
 --primary: #3b82f6;            /* soft blue */
 --primary-ink: #ffffff;        /* text on primary */
 --ring: rgba(59,130,246,.25);  /* focus ring */
 --border: #e6eaf2;             /* hairline borders */
```

```
 --shadow: 0 8px 24px rgba(2,6,23,.08), 0 1px 1px rgba(2,6,23,.04);
}

@media (prefers-color-scheme: dark){
 :root{
   --bg: #0f172a;
   --bg-grad: radial-gradient(1200px 600px at 20% -10%, rgba(59,130,246,.12) 0%,
transparent 60%),
              radial-gradient(900px 500px at 120% 0%, rgba(20,184,166,.10) 0%,
transparent 55%);
   --card:#0b1220;
   --ink:#e5e7eb;
   --muted:#9aa4b2;
   --border:#1f2937;
   --shadow: 0 10px 30px rgba(0,0,0,.25);
 }
}

/* ===== Page ===== */
html,body{height:100%}
body{
 margin:0;
 font: 15px/1.6 system-ui, -apple-system, Segoe UI, Roboto, "Helvetica Neue",
Arial, "Noto Sans", "Apple Color Emoji","Segoe UI Emoji";
 color:var(--ink);
 background: var(--bg), var(--bg-grad);
 background-blend-mode: normal, soft-light;
 -webkit-font-smoothing:antialiased;
 -moz-osx-font-smoothing:grayscale;
}

.container{
 max-width: 960px;
 margin: 48px auto 96px;
 padding: 0 20px;
}

/* ===== Headings ===== */
h1{
 font-size: clamp(28px, 2.2vw + 12px, 40px);
 line-height: 1.1;
 margin: 0 0 16px;
 color: var(--ink);
 letter-spacing: -0.02em;
}
```

```
/* ===== Card ===== */
.card{
 background: var(--card);
 border: 1px solid var(--border);
 border-radius: 18px;
 padding: 18px 18px 14px;
 box-shadow: var(--shadow);
 backdrop-filter: blur(3px);
 margin: 16px 0;
}

.card:first-of-type{ margin-top: 8px }

/* ===== Form Grid ===== */
#form{
 display:grid;
 grid-template-columns: repeat(4, minmax(0,1fr));
 gap: 12px;
 align-items: end;
}
#form label{
 display:flex; flex-direction:column; gap:6px;
 font-weight:600; color:var(--muted);
}
input{
 border:1px solid var(--border);
 background:#ffffff;
 color:var(--ink);
 border-radius:12px;
 padding:.55rem .7rem;
 outline:none;
 transition: box-shadow .15s ease, border-color .15s ease, background .2s ease;
}
input:hover{ background:#fbfcff; }
input:focus{
 border-color: var(--primary);
 box-shadow: 0 0 0 4px var(--ring);
 background:#ffffff;
}

/* Buttons */
button{
 cursor:pointer;
 border:1px solid var(--border);
```

```
 background:#ffffff;
 color:#0f172a;
 border-radius:12px;
 padding:.55rem .9rem;
 font-weight:600;
 transition: transform .05s ease, box-shadow .15s ease, background .2s ease, color
.2s ease, border-color .2s ease;
 box-shadow: 0 1px 1px rgba(2,6,23,.04);
}
button:hover{ transform: translateY(-1px); background:#fafcff; }
button:active{ transform: translateY(0); }

/* Primary actions */
#add, #calc{
 background: var(--primary);
 color: var(--primary-ink);
 border-color: transparent;
 box-shadow: 0 8px 18px rgba(59,130,246,.28);
}
#add:hover, #calc:hover{
 box-shadow: 0 10px 22px rgba(59,130,246,.34);
}

/* Secondary / table action buttons */
button.edit{
 background: #f3f7ff;
 color: #2563eb;
 border-color: #c7d8ff;
}
button.edit:hover{
 background: #ecf3ff;
}
button.del{
 background: #fff5f5;
 color: #ef4444;
 border-color: #ffd4d4;
}
button.del:hover{
 background: #ffecec;
}

/* Cancel edit pill (if present) */
#cancel-edit{ margin-left:.5rem; opacity:.9 }

/* ===== Stats & What-if layout ===== */
```

```
#stats{
 display:grid;
 grid-template-columns: 1fr 1fr;
 gap: 6px 14px;
 align-items:center;
}
#weightsMsg{ grid-column: 1 / -1; color: var(--muted); }

#whatif{
 display:grid;
 grid-template-columns: 1fr auto;
 gap: 12px;
 align-items:end;
}
#whatif label{ display:flex; flex-direction:column; gap:6px; color:var(--muted);
font-weight:600 }
#answer{ margin-top: 8px; grid-column: 1 / -1 }

/* ===== Table ===== */
table{
 width:100%;
 border-collapse:separate;
 border-spacing:0;
 overflow:hidden;
 border-radius: 14px;
 border:1px solid var(--border);
 background: var(--card);
}
thead th{
 text-align:left;
 padding:12px 14px;
 background: linear-gradient(180deg, rgba(99,102,241,.06), transparent);
 font-weight:700;
 color: var(--muted);
 border-bottom:1px solid var(--border);
}
tbody td{
 padding:14px;
 border-top:1px solid var(--border);
}
tbody tr:hover{
 background: rgba(2,6,23,.03);
}

/* Empty state row */
```

```
tr.empty td{
 text-align:center;
 color:var(--muted);
 padding:22px;
}

/* ===== Utilities ===== */
.hint{ font-size:.92rem; color:var(--muted); margin:.25rem 0 0 }
.sr-only{
 position:absolute !important; width:1px; height:1px;
 padding:0; margin:-1px; overflow:hidden; clip: rect(0,0,1px,1px);
 white-space:nowrap; border:0;
}

/* ===== Responsive tweaks ===== */
@media (max-width: 760px){
 #form{ grid-template-columns: 1fr 1fr; }
 #stats{ grid-template-columns: 1fr; }
 #whatif{ grid-template-columns: 1fr; }
}




```

.pytest_cache

.pytest_cache/v

.pytest_cache/v/cache

tests

conftest.py

```py
```

```python
# tests/conftest.py
from fastapi.testclient import TestClient
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.pool import StaticPool
import pytest

# Import your app + SQLAlchemy Base + get_db dependency
from backend.app import app
from backend.models import Base
from backend.app import get_db  # if get_db lives in app.py, change to: from
backend.app import get_db

# --- Single shared in-memory SQLite across all connections/threads ---
engine = create_engine(
    "sqlite://",                    # note: no '///' - this uses a memory DB shared
by StaticPool
    connect_args={"check_same_thread": False},
    poolclass=StaticPool,
)

TestingSessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

# Create a fresh schema before each test (so tests don't bleed into each other)
@pytest.fixture(autouse=True)
def _create_schema():
    Base.metadata.drop_all(bind=engine)
    Base.metadata.create_all(bind=engine)
    yield

# Override the app's get_db dependency so routes use our test session
def override_get_db():
    db = TestingSessionLocal()
    try:
        yield db
    finally:
        db.close()

app.dependency_overrides[get_db] = override_get_db

# Provide a test client
@pytest.fixture
def client():
    return TestClient(app)
```

```
```

test_api_assessments.py

```py
# tests/test_api_assessments.py
from datetime import date

def make_assessment(client, title, weight, due, score=None):
    payload = {"title": title, "weight_pct": weight, "due_date": due}
    if score is not None:
        payload["score_pct"] = score
    r = client.post("/assessments", json=payload)
    assert r.status_code == 200, r.text
    return r.json()

def test_crud_flow(client):
    # Create
    created = make_assessment(client, "Midterm", 20.0, "2025-11-01")

    # Read one
    r = client.get(f"/assessments/{created['id']}")
    assert r.status_code == 200
    got = r.json()
    assert got["title"] == "Midterm"

    # Update
    update = dict(created, title="Midterm (updated)", score_pct=85.0)
    r = client.put(f"/assessments/{created['id']}", json=update)
    assert r.status_code == 200
    updated = r.json()
    assert updated["title"].endswith("(updated)")
    assert updated["score_pct"] == 85.0

    # List
    r = client.get("/assessments")
    assert r.status_code == 200
    rows = r.json()
    assert any(row["title"].endswith("(updated)") for row in rows)

    # Delete
    r = client.delete(f"/assessments/{created['id']}")
    assert r.status_code in (200, 204)

    # Verify gone
```

```
    r = client.get("/assessments")
    assert all(row["id"] != created["id"] for row in r.json())

```

test_api_notfound.py

```py
# tests/test_api_notfound.py

def test_get_missing_returns_404(client):
    r = client.get("/assessments/999999")
    assert r.status_code == 404

def test_put_missing_returns_404(client):
    r = client.put("/assessments/999999", json={
        "id": 999999,
        "title": "Nope",
        "weight_pct": 10.0,
        "due_date": "2025-01-01",
        "score_pct": None
    })
    assert r.status_code == 404

def test_delete_missing_returns_404(client):
    r = client.delete("/assessments/999999")
    assert r.status_code == 404

```

test_api_stats.py

```py
# tests/test_api_stats.py

def seed(client):
    """Create a stable set of rows for stats tests."""
    client.post("/assessments",
json={"title":"A1","weight_pct":30.0,"due_date":"2025-10-01","score_pct":90.0})  #
contributes 27
    client.post("/assessments",
json={"title":"A2","weight_pct":30.0,"due_date":"2025-11-01","score_pct":80.0})  #
contributes 24
```

```
    client.post("/assessments",
json={"title":"Final","weight_pct":40.0,"due_date":"2025-12-01","score_pct":None})
# remaining 40

def test_current_and_remaining(client):
    seed(client)
    r = client.get("/stats/current")
    assert r.status_code == 200
    stats = r.json()
    # 0.3*90 + 0.3*80 = 27 + 24 = 51 ; remaining = 40
    assert round(stats["current_weighted"], 2) == 51.00
    assert round(stats["remaining_weight"], 2) == 40.00

def test_validate_weights(client):
    seed(client)  # <-- important
    r = client.get("/stats/validate")
    assert r.status_code == 200
    v = r.json()
    assert round(v["total_weight"], 2) == 100.00

def test_what_if(client):
    seed(client)  # <-- important
    r = client.get("/stats/what-if", params={"target": 70})
    assert r.status_code == 200
    w = r.json()
    # With the seeded data: completed = 51, remaining = 40 → (70 - 51)*100/40 = 47.5
    assert round(w["required_avg"], 2) == 47.50

```

test_api_validation.py

```py
# tests/test_api_validation.py

import pytest

def post(client, payload):
    return client.post("/assessments", json=payload)

def base_payload(**overrides):
    data = {
        "title": "Any",
        "weight_pct": 20.0,
        "due_date": "2025-01-10",
```

```
        "score_pct": None,
    }
    data.update(overrides)
    return data

@pytest.mark.parametrize("bad_weight", [-1, 101, 1000])
def test_post_rejects_invalid_weight_range(client, bad_weight):
    r = post(client, base_payload(weight_pct=bad_weight))
    assert r.status_code == 422

@pytest.mark.parametrize("bad_score", [-5, 105, 1000])
def test_post_rejects_invalid_score_range(client, bad_score):
    r = post(client, base_payload(score_pct=bad_score))
    assert r.status_code == 422

@pytest.mark.parametrize("bad_date", ["", "not-a-date", "2025/01/01",
"13-40-9999"])
def test_post_rejects_invalid_date_format(client, bad_date):
    r = post(client, base_payload(due_date=bad_date))
    assert r.status_code == 422

def test_post_requires_title(client):
    r = post(client, {"weight_pct": 10.0, "due_date": "2025-01-01"})
    assert r.status_code == 422

def test_put_rejects_bad_updates(client):
    # create a good row
    created = post(client, base_payload(title="X")).json()
    # try to set invalid score on update
    bad = dict(created, score_pct=1000)
    r = client.put(f"/assessments/{created['id']}", json=bad)
    assert r.status_code == 422

```

test_calculations_unit.py

```py
from datetime import date

from backend import calculations, schemas


class Obj:
    """Lightweight helper to mimic Assessment rows."""
```

```
    def __init__(self, weight_pct, score_pct):
        self.weight_pct = weight_pct
        self.score_pct = score_pct


def test_current_stats_mixes_completed_and_pending():
    rows = [
        Obj(30.0, 90.0),  # contributes 27
        Obj(20.0, 50.0),  # contributes 10
        Obj(50.0, None),
    ]
    stats = calculations.current_stats(rows)
    assert isinstance(stats, schemas.CurrentStats)
    assert stats.current_weighted == 37.0
    assert stats.weight_done == 50.0
    assert stats.remaining_weight == 50.0


def test_what_if_with_remaining_work():
    rows = [Obj(40.0, 80.0), Obj(60.0, None)]
    result = calculations.what_if(rows, target=75.0)
    assert isinstance(result, schemas.WhatIf)
    # Need ~71.67 on remaining 60% to reach target
    assert result.required_avg == 71.67
    assert result.attainable is True


def test_what_if_when_no_remaining_work():
    rows = [Obj(50.0, 80.0), Obj(50.0, 90.0)]
    result = calculations.what_if(rows, target=85.0)
    assert result.required_avg is None
    assert result.attainable is True


def test_validate_weights_messages():
    rows = [Obj(40.0, 80.0), Obj(30.0, None)]
    res = calculations.validate_weights(rows)
    assert isinstance(res, schemas.Validation)
    assert res.total_weight == 70.0
    assert res.is_exactly_100 is False
    assert "You can still add" in res.message

```

test_services_unit.py

```py
from datetime import date

from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.pool import StaticPool

from backend import models, schemas, services


def _session():
    engine = create_engine(
        "sqlite://",
        connect_args={"check_same_thread": False},
        poolclass=StaticPool,
    )
    models.Base.metadata.create_all(bind=engine)
    return sessionmaker(bind=engine)()


def test_service_crud_flow():
    session = _session()
    repo = services.AssessmentRepository(session)
    svc = services.AssessmentService(repo)

    payload = schemas.AssessmentIn(
        title="Midterm",
        weight_pct=20.0,
        due_date=date(2025, 11, 1),
        score_pct=None,
    )
    created = svc.create_assessment(payload)
    assert created.id is not None
    assert created.title == "Midterm"

    fetched = svc.get_assessment(created.id)
    assert fetched.id == created.id

    updated = svc.update_assessment(
        created.id, schemas.AssessmentUpdate(score_pct=85.0)
    )
    assert updated.score_pct == 85.0
```

```
    all_rows = svc.list_assessments()
    assert len(all_rows) == 1

    svc.delete_assessment(created.id)
    assert svc.list_assessments() == []


```

test_stats_edges.py

```py
# tests/test_stats_edges.py

def test_stats_on_empty_db(client):
    # current: no rows -> current=0, remaining = 100 (you can still add 100%)
    r = client.get("/stats/current")
    assert r.status_code == 200
    s = r.json()
    assert s["current_weighted"] == 0
    assert s["remaining_weight"] == 100

    # what-if on empty DB: required avg equals the target, attainable true
    r = client.get("/stats/what-if", params={"target": 70})
    assert r.status_code == 200
    wi = r.json()
    assert round(wi["required_avg"], 2) == 70.00
    assert wi["attainable"] is True

def test_stats_all_completed(client):
    # seed: everything graded already
    client.post("/assessments",
json={"title":"A1","weight_pct":50.0,"due_date":"2025-01-01","score_pct":80.0})
    client.post("/assessments",
json={"title":"A2","weight_pct":50.0,"due_date":"2025-02-01","score_pct":90.0})

    r = client.get("/stats/current")
    s = r.json()
    assert round(s["current_weighted"], 2) == 85.00  # 0.5*80 + 0.5*90

    # nothing remaining → required_avg is None; attainable depends on target
    r = client.get("/stats/what-if", params={"target": 90})
    wi = r.json()
    assert wi["required_avg"] is None
    assert wi["attainable"] is (85.00 >= 90.0)  # False
```

```
def test_unattainable_target_with_remaining(client):
   # seed: completed 10% at 50 → current=5; remaining=90
   client.post("/assessments",
json={"title":"A1","weight_pct":10.0,"due_date":"2025-01-01","score_pct":50.0})
   client.post("/assessments",
json={"title":"Big","weight_pct":90.0,"due_date":"2025-02-01","score_pct":None})

   # target 99 overall → required avg will be > 100 → unattainable
   r = client.get("/stats/what-if", params={"target": 99})
   wi = r.json()
   assert wi["required_avg"] > 100.0
   assert wi["attainable"] is False

def test_what_if_when_no_remaining_work(client):
   # All weights sum to 100 and all are scored -> remaining = 0
   client.post("/assessments",
json={"title":"A1","weight_pct":50.0,"due_date":"2025-01-01","score_pct":80.0})
   client.post("/assessments",
json={"title":"A2","weight_pct":50.0,"due_date":"2025-02-01","score_pct":90.0})

   r = client.get("/stats/current")
   s = r.json()
   assert s["remaining_weight"] == 0

   r = client.get("/stats/what-if", params={"target": 85})
   wi = r.json()
   # nothing left to earn -> required_avg is None; attainable depends on current >=
target
   assert wi["required_avg"] is None
   assert wi["attainable"] is True  # current is exactly 85 in this seed

```

backend

__init__.py

```py

```

app.py

```py
from typing import NoReturn
from pathlib import Path

from fastapi import FastAPI, Depends, HTTPException, Response
from fastapi.middleware.cors import CORSMiddleware
from fastapi.staticfiles import StaticFiles
from sqlalchemy.orm import Session

from . import calculations, models, schemas, services
from .db import SessionLocal, engine
from .settings import Settings, settings

NOT_FOUND_DETAIL = "Assessment not found"


# ----------------------------
# Database Dependency
# ----------------------------
def get_db():
    session = SessionLocal()
    try:
        yield session
    finally:
        session.close()


def get_assessment_service(
    session: Session = Depends(get_db),
) -> services.AssessmentService:
    repository = services.AssessmentRepository(session)
    return services.AssessmentService(repository)


def get_settings() -> Settings:
    return settings


def _raise_not_found(err: Exception) -> NoReturn:
    raise HTTPException(status_code=404, detail=NOT_FOUND_DETAIL) from err


# ----------------------------
# Application Factory
```

```python
# ------------------------------
def create_app(app_settings: Settings = settings) -> FastAPI:
    app = FastAPI(
        title=app_settings.app_title,
        version=app_settings.app_version,
    )

    # CORS
    app.add_middleware(
        CORSMiddleware,
        allow_origins=list(app_settings.allowed_origins),
        allow_credentials=True,
        allow_methods=["*"],
        allow_headers=["*"],
    )

    # Auto-create tables (SQLite or Postgres)
    if app_settings.auto_create_tables:
        @app.on_event("startup")
        def _create_tables() -> None:
            models.Base.metadata.create_all(bind=engine)

    # ------------------------------
    # Monitoring: Prometheus Instrumentation
    # ------------------------------
    try:
        from prometheus_fastapi_instrumentator import Instrumentator

        Instrumentator().instrument(app).expose(app)
    except Exception:
        # Safe fallback — tests may not have this package installed
        pass

    # Register API routes
    _register_routes(app)

    return app


# ------------------------------
# Routes / API Endpoints
# ------------------------------
def _register_routes(app: FastAPI) -> None:

    # ---- Health Check ------------------------------------------------
```

```python
@app.get("/health")
def health():
    return {"ok": True}

# ---- Basic Metrics Endpoint (backup) -------------------------------
# Instrumentator already exposes /metrics, but this is a fallback
try:
    from prometheus_client import generate_latest, CONTENT_TYPE_LATEST

    @app.get("/metrics")
    def metrics():
        data = generate_latest()
        return Response(content=data, media_type=CONTENT_TYPE_LATEST)
except Exception:
    pass

# ---- CRUD: Assessments ---------------------------------------------
@app.post("/assessments", response_model=schemas.AssessmentOut)
def create_assessment(
    payload: schemas.AssessmentIn,
    service: services.AssessmentService = Depends(get_assessment_service),
):
    return service.create_assessment(payload)

@app.get("/assessments", response_model=list[schemas.AssessmentOut])
def list_assessments(
    service: services.AssessmentService = Depends(get_assessment_service),
):
    return service.list_assessments()

@app.get("/assessments/{aid}", response_model=schemas.AssessmentOut)
def get_assessment(
    aid: int,
    service: services.AssessmentService = Depends(get_assessment_service),
):
    try:
        return service.get_assessment(aid)
    except services.AssessmentNotFound as err:
        _raise_not_found(err)

@app.put("/assessments/{aid}", response_model=schemas.AssessmentOut)
def update_assessment(
    aid: int,
    payload: schemas.AssessmentUpdate,
    service: services.AssessmentService = Depends(get_assessment_service),
```

```
    ):
        try:
            return service.update_assessment(aid, payload)
        except services.AssessmentNotFound as err:
            _raise_not_found(err)

    @app.delete("/assessments/{aid}")
    def delete_assessment(
        aid: int,
        service: services.AssessmentService = Depends(get_assessment_service),
    ):
        try:
            service.delete_assessment(aid)
            return {"ok": True}
        except services.AssessmentNotFound as err:
            _raise_not_found(err)

    # ---- Stats: current / what-if / validate ----------------------------
    @app.get("/stats/current", response_model=schemas.CurrentStats)
    def current_stats(
        service: services.AssessmentService = Depends(get_assessment_service),
    ):
        return calculations.current_stats(service.list_for_stats())

    @app.get("/stats/what-if", response_model=schemas.WhatIf)
    def what_if(
        target: float,
        service: services.AssessmentService = Depends(get_assessment_service),
    ):
        return calculations.what_if(service.list_for_stats(), target)

    @app.get("/stats/validate", response_model=schemas.Validation)
    def validate_weights(
        service: services.AssessmentService = Depends(get_assessment_service),
    ):
        return calculations.validate_weights(service.list_for_stats())

    # ---- Serve Frontend -------------------------------------------------
    frontend_dir = Path(__file__).resolve().parents[1] / "frontend"

    if frontend_dir.exists():
        app.mount("/", StaticFiles(directory=str(frontend_dir), html=True),
name="frontend")
```

```
# ----------------------------
# ASGI Server Entrypoint
# ----------------------------
app = create_app()



```

calculations.py

```py
from typing import Iterable, Protocol

from . import schemas

# rows are objects with: weight_pct (float), score_pct (float|None)
class AssessmentScore(Protocol):
    weight_pct: float
    score_pct: float | None


def _split(rows: Iterable[AssessmentScore]) -> tuple[list[AssessmentScore], float]:
    scored = [r for r in rows if getattr(r, "score_pct", None) is not None]
    weight_done = sum(float(r.weight_pct) for r in scored)
    return scored, weight_done


def current_stats(rows: Iterable[AssessmentScore]) -> schemas.CurrentStats:
    scored, weight_done = _split(rows)
    completed = sum(float(r.weight_pct) * float(r.score_pct) for r in scored)
    current_weighted = (completed / 100.0) if weight_done > 0 else 0.0
    remaining = max(0.0, 100.0 - weight_done)
    return schemas.CurrentStats(
        current_weighted=round(current_weighted, 2),
        weight_done=round(weight_done, 2),
        remaining_weight=round(remaining, 2),
    )


def what_if(rows: Iterable[AssessmentScore], target: float) -> schemas.WhatIf:
    stats = current_stats(rows)
    rem = stats.remaining_weight
    if rem == 0:
        return schemas.WhatIf(
```

```
            target=target,
            required_avg=None,
            attainable=stats.current_weighted >= target,
        )
    req = (target - stats.current_weighted) * 100.0 / rem
    return schemas.WhatIf(
        target=target,
        required_avg=round(req, 2),
        attainable=0 <= req <= 100,
    )


def validate_weights(rows: Iterable[AssessmentScore]) -> schemas.Validation:
    total = round(sum(float(r.weight_pct) for r in rows), 2)
    is_exact = abs(total - 100.0) < 1e-6
    if is_exact:
        msg = "Weights sum to 100%."
    elif total < 100.0:
        msg = f"Weights sum to {total}%. You can still add {round(100.0 - total,
2)}%."
    else:
        msg = f"Weights exceed 100% (total {total}%). Consider reducing some
weights."
    return schemas.Validation(
        total_weight=total,
        is_exactly_100=bool(is_exact),
        message=msg,
    )

```

db.py

```py
import os
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base
from sqlalchemy.engine.url import make_url


def _connect_args_from_url(db_url: str) -> dict:
    """Detect correct connect_args based on backend."""
    url = make_url(db_url)

    # SQLite requires this for FastAPI multithreading
```

```
    if url.get_backend_name() == "sqlite":
        return {"check_same_thread": False}

    # Postgres / MySQL / others do not need connect_args
    return {}


# Load DATABASE_URL safely
raw_url = os.getenv("DATABASE_URL")

if raw_url:
    # Railway provides postgresql:// which needs to be adapted
    DATABASE_URL = raw_url.replace(
        "postgresql://",
        "postgresql+psycopg2binary://"
    )
else:
    # Fallback: used in GitHub Actions tests
    DATABASE_URL = "sqlite:///./test.db"


# Create engine with proper connect args
engine = create_engine(
    DATABASE_URL,
    connect_args=_connect_args_from_url(DATABASE_URL)
)

# Session factory
SessionLocal = sessionmaker(
    autocommit=False,
    autoflush=False,
    bind=engine
)

# Base class for models
Base = declarative_base()
```

models.py

```py
from sqlalchemy.orm import declarative_base
from sqlalchemy import Column, Integer, String, Float, Date
```

```
Base = declarative_base()

class Assessment(Base):
   __tablename__ = "assessments"

   id = Column(Integer, primary_key=True, index=True)
   title = Column(String, nullable=False)
   weight_pct = Column(Float, nullable=False)       # e.g., 20.0
   due_date = Column(Date, nullable=False)
   score_pct = Column(Float, nullable=True)         # None until graded




#10452

```

schemas.py

```py
from pydantic import BaseModel, Field
from datetime import date
from typing import Optional

# --------- Assessment I/O models ----------

class AssessmentBase(BaseModel):
   title: str
   weight_pct: float = Field(ge=0, le=100)
   due_date: date
   score_pct: Optional[float] = Field(default=None, ge=0, le=100)

class AssessmentIn(AssessmentBase):
   pass

class AssessmentUpdate(BaseModel):
   title: Optional[str] = None
   weight_pct: Optional[float] = Field(default=None, ge=0, le=100)
   due_date: Optional[date] = None
   score_pct: Optional[float] = Field(default=None, ge=0, le=100)

class AssessmentOut(AssessmentBase):
   id: int
   class Config:
```

```
        orm_mode = True

# --------- Stats response models ----------

class CurrentStats(BaseModel):
    current_weighted: float
    weight_done: float
    remaining_weight: float

class WhatIf(BaseModel):
    target: float
    required_avg: Optional[float]    # None if no remaining work
    attainable: bool

class Validation(BaseModel):
    total_weight: float
    is_exactly_100: bool
    message: str
```

services.py

```py
from __future__ import annotations

from typing import Iterable

from sqlalchemy.orm import Session

from . import models, schemas


class AssessmentNotFound(Exception):
    """Raised when an assessment row cannot be located."""

    def __init__(self, assessment_id: int) -> None:
        super().__init__(f"Assessment {assessment_id} not found")
        self.assessment_id = assessment_id


class AssessmentRepository:
    """Persistence boundary for assessments."""

    def __init__(self, session: Session) -> None:
        self._session = session
```

```
    def list(self, ordered: bool = True) -> list[models.Assessment]:
        query = self._session.query(models.Assessment)
        if ordered:
            query = query.order_by(models.Assessment.due_date)
        return query.all()

    def get(self, assessment_id: int) -> models.Assessment | None:
        return self._session.get(models.Assessment, assessment_id)

    def save(self, assessment: models.Assessment) -> models.Assessment:
        self._session.add(assessment)
        self._session.commit()
        self._session.refresh(assessment)
        return assessment

    def delete(self, assessment: models.Assessment) -> None:
        self._session.delete(assessment)
        self._session.commit()


class AssessmentService:
    """Encapsulates CRUD operations for assessments."""

    def __init__(self, repository: AssessmentRepository) -> None:
        self._repository = repository

    def list_assessments(self, ordered: bool = True) -> list[models.Assessment]:
        return self._repository.list(ordered)

    def get_assessment(self, assessment_id: int) -> models.Assessment:
        assessment = self._repository.get(assessment_id)
        if assessment is None:
            raise AssessmentNotFound(assessment_id)
        return assessment

    def create_assessment(self, payload: schemas.AssessmentIn) -> models.Assessment:
        assessment = models.Assessment(**payload.dict())
        return self._repository.save(assessment)

    def update_assessment(
        self, assessment_id: int, payload: schemas.AssessmentUpdate
    ) -> models.Assessment:
        assessment = self.get_assessment(assessment_id)
        for field, value in payload.dict(exclude_unset=True).items():
```

```
            setattr(assessment, field, value)
        return self._repository.save(assessment)

    def delete_assessment(self, assessment_id: int) -> None:
        assessment = self.get_assessment(assessment_id)
        self._repository.delete(assessment)

    def list_for_stats(self) -> Iterable[models.Assessment]:
        """Internal helper to keep stats queries consistent."""
        return self.list_assessments(ordered=False)
```

settings.py

```py
from typing import List

from pydantic_settings import BaseSettings


class Settings(BaseSettings):
    """Application configuration with environment overrides."""

    app_title: str = "Grade & What-If Tracker"
    app_version: str = "1.0"
    database_url: str = "sqlite:///./grades.db"
    allowed_origins: List[str] = [
        "http://127.0.0.1:5500",
        "http://localhost:5500",
    ]
    auto_create_tables: bool = True

    class Config:
        env_prefix = "GRADEAPP_"
        env_file = ".env"


# Singleton settings instance for the app
settings = Settings()
```

backend/.pytest_cache

Jad CHEBLY
BCSAI

```
## backend/.pytest_cache/v

## backend/.pytest_cache/v/cache
```