Report AIML prod

Jyothiswaroop Reddy Kottala

Alt-Text & Search for

Albumy

1) GitHub Commit Link Final commit (permalink to the exact commit used for grading):

https://github.com/swaroop9494/fau-ml-prod-assignment1.git

2) Technical Description

Alt-Text: Automatic generation + manual override + regeneration

Where it happens: albumy/blueprints/main.py

1. **Auto-generate on upload**
   In the upload() route, once the image is saved and resized, the code generates alt-text and object labels using a local ML service:

```
image_path = os.path.join(current_app.config['ALBUMY_UPLOAD_PATH'], filename)
auto_alt_text = ml_service.generate_alt_text(image_path)
detected_objects = ml_service.detect_objects(image_path)
```

The returned auto_alt_text is stored on the Photo record; detected_objects are joined into a comma-separated string and also turned into **Tag** rows (created if missing, then attached to the photo). This makes them immediately visible in the UI and searchable.

2. **Manual editing**
   In edit_alt_text(photo_id), authors (or moderators) can directly update photo.alt_text via the AltTextForm, enabling human corrections.
3. **One-click regeneration**
   In regenerate_alt_text(photo_id), users can invoke the ML service again to replace the alt-text with a fresh caption.
4. **Accessibility fallback**
   In edit_description(photo_id), if a user saves a **description** while alt_text is empty, the code copies the description into alt_text so images never ship without an accessible alternative.

Relevant code (after you have the final commit):

- upload() — auto alt-text + auto tags

  https://github.com/swaroop9494/fau-ml-prod-assignment1.gitpy#L[upload-start]-L[upload-end]
- edit_alt_text() — manual override

  .../main.py

```
@main_bp.route('/photo/<int:photo_id>/alt-text', methods=['POST'])
@login_required
def edit_alt_text(photo_id):
    photo = Photo.query.get_or_404(photo_id)
    if current_user != photo.author and not current_user.can('MODERATE'):
        abort(403)

    form = AltTextForm()
    if form.validate_on_submit():
        photo.alt_text = form.alt_text.data
        db.session.commit()
        flash('Alt text updated.', 'success')

    flash_errors(form)
    return redirect(url_for('.show_photo', photo_id=photo_id))
```

- regenerate_alt_text() — one-click re-caption

```
@main_bp.route('/photo/<int:photo_id>/regenerate-alt-text',
methods=['POST'])
@login_required
def regenerate_alt_text(photo_id):
    photo = Photo.query.get_or_404(photo_id)
    if current_user != photo.author and not current_user.can('MODERATE'):
        abort(403)

    # Regenerate alt text using ML service
    image_path = os.path.join(current_app.config['ALBUMY_UPLOAD_PATH'],
photo.filename)
    auto_alt_text = ml_service.generate_alt_text(image_path)
    photo.alt_text = auto_alt_text
    db.session.commit()
    flash('Alt text regenerated automatically.', 'success')
    return redirect(url_for('.show_photo', photo_id=photo_id))
```

Search: Whooshee-backed with robust fallbacks

Where it happens: search() route in main.py.

The search endpoint supports three categories: **user**, **tag**, and **photo**.

1. **Primary search (Whooshee full-text):**

```
if category == 'user':
    pagination = User.query.whooshee_search(q).paginate(...)
elif category == 'tag':
    pagination = Tag.query.whooshee_search(q).paginate(...)
else:
    pagination = Photo.query.whooshee_search(q).paginate(...)
```

2. **Schema-mismatch resilience:**
   If Whooshee's index schema falls out of sync (e.g., you added a new field such as alt_text or detected_objects), a KeyError can occur. The handler:
- **Purges** the stale index subdir for the model,
- **Calls whooshee_ext.reindex()** to rebuild all indexes, then
- **Retries** the Whooshee query.

3. **Last-resort fallback (DB substring search):**
   If reindexing fails, it falls back to case-insensitive LIKE queries over Photo.description, Photo.alt_text, and Photo.detected_objects:

```
from sqlalchemy import or_
like = f"%{q}%"
Photo.query.filter(or_(
    Photo.description.ilike(like),
    Photo.alt_text.ilike(like),
    Photo.detected_objects.ilike(like)
))
```

4. **Indexing hygiene on upload:**
   During upload(), index updates are temporarily disabled while the new Photo is created. After commit, the app calls whooshee_ext.reindex() to ensure downstream search sees the new fields and schema.

Relevant code (after commit):

- search() — primary + fallback search logic
  https://github.com/swaroop9494/fau-ml-prod-assignment1.git
- py#L[search-start]-L[search-end]
- upload() — disable indexing during write + full reindex after
  .../main.py

```
from albumy.extensions import whooshee as whooshee_ext

        wh_cfg = current_app.extensions.get('whooshee', {})
```

```
        prev_enable = wh_cfg.get('enable_indexing', True)

        wh_cfg['enable_indexing'] = False
```

---

3) UI/UX Design Justification

We map each feature to class concepts (Automate / Prompt / Organize / Annotate / Hybrid), then justify by **forcefulness, frequency, value, cost**.

Alt-Text

- **Recommended Strategy: Hybrid (Automate + Annotate/Override)**
    - **Automate** on upload to minimize friction and ensure accessibility by default.
    - **Annotate/Override** so users can correct bias/errors; provide **Regenerate** for quick retries.
- **Forcefulness:** *Medium*. Captions appear automatically but are editable (not locked).
- **Frequency:** *High*. Every image.
- **Value:** *High*. Accessibility, searchability, and content understanding.
- **Cost:** *Moderate*. Inference time + occasional re-runs.
- **If implementation differs:** If you ever disabled manual editing, I'd add an **inline "Edit alt-text" affordance** near the image and a small **confidence chip** (e.g., "Low confidence") to nudge edits when needed.

Search

- **Recommended Strategy: Organize + Annotate (with a light Prompt)**
    - **Organize:** Use facets/filters for **Tags**, **Author**, **Time**; visually cluster results (grid for photos) and show key metadata (alt-text snippet, tags).
    - **Annotate:** Show **highlighted matches** from alt-text/description and detected objects to explain why a result surfaced.
    - **Prompt (light):** When a search returns zero/low results, suggest **auto-tagging** or query refinements.
- **Forcefulness:** *Low*. Search is user-initiated.
- **Frequency:** *Medium*. Depends on user goal.
- **Value:** *High*. Better findability = higher retention.
- **Cost:** *Low→Medium*. Depends on index size and whether you add image embeddings later.
- **If implementation differs:** If only keyword search is present, I'd add **chips** for tags and quick toggles ("Only my photos", "With people", "High confidence") for faster narrowing.

---

4) Harms & Risks

1. **Bias & Mislabeling**
    - Example: Gendered or occupational bias in captions (e.g., “woman in lab coat → nurse” vs “doctor”).
    - **Mitigations:**
        - Keep manual **Edit/Override** in the UI (already implemented).
        - Store a **moderation audit trail** (who changed what) to learn from corrections.
        - Show **confidence** and encourage edits on low-confidence captions.
        - Periodically retrain or switch to models with fairness audits; add curated post-processing rules (e.g., avoid inferring gender/occupation without strong visual evidence).
2. **Offensive/Unsafe Content**
    - Risk: Generated tags/captions may contain slurs or graphic terms.
    - **Mitigations:**
        - Add a **content moderation filter** before persisting or displaying generated text.
        - Provide a **“Report caption”** action (symmetry with report_photo/report_comment) to crowdsource issues.
3. **Exclusion / Accessibility Gaps**
    - Risk: Missing alt-text harms screen-reader users.
    - **Mitigations:**
        - You already added a **description→alt-text fallback**; keep that.
        - Consider an **upload gate** that warns if both caption and description are empty (soft prompt, not a hard block).
4. **Privacy**
    - Risk: Detected objects may reveal sensitive context (e.g., “child”, “address”).
    - **Mitigations:**
        - Provide per-photo **visibility** controls for tags/captions.
        - Allow **tag redaction** and **per-field privacy** (show to owner only).

---

5) Production Challenges

Goal: Scale to millions of users while controlling **latency**, **cost**, and **relevance**.

1. **Issue: Inference Cost & Throughput**
    - Alt-text on upload can be expensive at scale (spikes during bulk imports).
    - **Solution:**
        - **Asynchronous pipelines** (queue + workers), **batching**, and **result caching** (content hash of image).
        - Introduce a **tiered model strategy**: cheap fast model on upload; allow on-demand “High Quality” regen with a stronger (costly) model.
2. **Issue: Search Index Scaling**

    - Whooshee (SQLite/FS-backed) is great for small/medium apps but becomes a bottleneck for sharding, concurrency, and schema evolution.
    - **Solution:**
        - Migrate to a dedicated engine like **OpenSearch/Elasticsearch** for full-text and facets; use **rolling reindex** patterns and versioned indices for zero-downtime schema changes.
        - If adding semantic search, keep a **dual index**: BM25 for precision + ANN (FAISS/Milvus/OpenSearch KNN) for recall.
3. **Issue: Hot Reindex / Schema Drift**
    - The current code repairs schema mismatch at query time by purging & reindexing. That's resilient for dev, but risky in prod (heavy I/O, race conditions).
    - **Solution:**
        - Move to **managed background reindex jobs** on deploy/migration; block only the affected collections; use **blue/green** index directories.
4. **Issue: Data Store & Fan-out**
    - Tags + many-to-many relationships + notifications will stress the DB.
    - **Solution:**
        - Add **read replicas**, **partitioning** by user/org, and **write-behind** for denormalized aggregates (e.g., tag counts).
        - Cache hot queries (top tags, recent photos) via **Redis** with TTL + cache invalidation hooks on writes.
5. **Issue: Observability & Quality**
    - Silent caption errors degrade trust.
    - **Solution:**
        - Track **caption edit rate**, **report rate**, **search click-through**, and **time-to-first-result**.
        - Build a **feedback loop** that samples low-confidence or frequently corrected captions for evaluation.

Appendix — Pointers to the Key Functions (for graders)

- @main_bp.route('/upload') → auto alt-text + auto tags; indexing disabled then reindex() after commit.
- @main_bp.route('/search') → Whooshee search with robust reindex & SQL fallback.
- @main_bp.route('/photo/<id>/alt-text') → manual alt-text edit.
- @main_bp.route('/photo/<id>/regenerate-alt-text') → re-caption via ML service.
- @main_bp.route('/photo/<id>/tags/auto') → auto-tagging endpoint.
- @main_bp.route('/photo/<id>/description') → description→alt-text fallback for accessibility.

What I'd hand-tune if there were another sprint

- Add **inline confidence badges** next to alt-text.
- Expose **filters** in search (chips for tags/authors/time).
- Swap “reindex on failure” for **migrate-then-switch** strategy.
- Consider **semantic search** once you have enough data (keep keyword as the primary for precision).