Technical Assignment Specification

Member Import & Dynamic Profile System

---

1. Purpose of This Assignment

Your task is to build a functional web application that allows users to:

- Upload CSV files containing member information
- Store and update member data
- Display a list of members on the start page
- Open a detailed profile page for each member
- View historical versions of member data to see how their information changed over time

This assignment evaluates:

- Backend modeling and extensibility
- Ability to work with dynamic data
- Frontend clarity and usability

If something is unclear, you may make reasonable assumptions and document them.

---

2. Overview of System Requirements

Your application must:

- Accept CSV uploads
- Store canonical and dynamic fields
- Update members when new CSVs are uploaded

- Display all members on the start page (canonical fields only)
- Display full details on the member profile page (canonical + dynamic fields)
- Preserve historical snapshots of member data when updates occur

The system must handle CSVs with different sets of fields.

---

3. CSV Format and Logic Requirements

3.1 Canonical Fields (Always Present)

Every CSV will always contain the following **required fields**:

- **id**
- **first_name**
- **last_name**
- **dob**

These must be stored in a fixed, stable structure.

3.2 Dynamic Fields (Additional Columns)

A CSV may contain any number of **extra fields** beyond the canonical ones.

Examples include:

- city
- status
- favorite_color
- membership_level
- department

- or any other additional field

Your system must:

- Detect all dynamic fields
- Store them correctly
- Preserve them across multiple uploads
- Display them on the member profile page

Dynamic fields are unpredictable and must be handled in a flexible, extensible way.

3.3 Identifying Duplicates / Same Member

Members are uniquely identified by the **id** field.

If two rows (even from different CSVs) use the same id:

- They represent the **same member**
- The system must **update**, not duplicate, the member

---

4. Required System Behavior

4.1 Start Page (Frontpage)

The start page must include:

1. **A CSV upload input**
    - Accepts .csv files
    - Sends the file to the backend for processing
2. **A list of all members**, showing only the canonical fields:
    - id

- first_name
- last_name
- dob

3. **Clickable member entries**
    - Selecting a member opens their profile page

4.2 Member Profile Page

Each member profile page must display:

Canonical fields

- id
- first_name
- last_name
- dob

Dynamic fields

All dynamic fields associated with the member across all CSV uploads must be shown.

These are additional data fields that appeared in any CSV where the member existed.

They must not be hidden, discarded, or filtered out.

Historical versions

Display all previous versions of the member's data, showing:

- Previous values for all fields (canonical + dynamic)
- Timestamp of when each version was created

4.3 CSV Uploading and Merging Behavior

This is the core of the assignment.

Rule 1: Same id = Update Member

If the CSV row contains an id already present in the database:

- Update the existing member

Rule 2: Always Keep Latest Values

If a field appears in multiple uploads:

- The **most recent upload** overrides earlier values
- Fields not overwritten must remain stored

Rule 3: Add All New Dynamic Fields

If a new CSV introduces dynamic fields that were not previously stored for a member:

- Add those fields
- Do not remove fields unless overwritten

---

5. Backend Data Handling Requirements

Your backend must support:

5.1 Canonical Data

Stored in a predictable, stable database structure.

5.2 Dynamic Data

Stored in a flexible structure that:

- Accepts any number of dynamically named fields
- Requires no schema changes
- Allows straightforward retrieval
- Supports merging across uploads

5.3 Merging Logic

Your backend must:

- Create new members if id is new
- Update existing members when id matches
- Add new dynamic fields
- Overwrite values with latest uploads
- Collect all fields into a complete final record for display

5.4 Historical Data Storage

Your backend must implement:

- Storage of previous member states when updates occur
- Timestamping of each historical version
- API endpoints to retrieve historical versions for a member

---

6. Assumption & Clarification Policy

If something is unclear or unspecified, you may:

- Make a reasonable assumption
- Implement based on that assumption
- Document it in your notes

These will be discussed during your project review.

---

7. Required Deliverables

7.1 Working Functionality

Your application must include:

- CSV upload and processing

- Database storage of canonical + dynamic fields
- Frontpage member list (canonical fields only)
- Profile page showing all fields (canonical + dynamic)
- Correct merging behavior for repeated CSV uploads
- 2 .csv files which can be used to test the application

7.2 Documentation (NOTES.md)

Your notes must include:

- Assumptions you made
- What you would improve with more time

---

8. Version History / Historical Uploads

Behavior:

- When a CSV is uploaded and a member with an existing id is updated:
    - The "current" member record should be overwritten with the latest data (canonical + dynamic).
    - But the previous state should still be accessible as history.
- There must be a way to:
    - See all historical versions of a member.
    - For each version, see:
        - The data at that time (canonical + dynamic).
        - A timestamp for when that version was created (e.g. upload time).

---

9. Evaluation Criteria

We will evaluate:

- Data model design and extensibility
- Correctness of CSV parsing and merging
- Accuracy of dynamic field handling
- UI simplicity and usability

The goal is to understand how you think, design, and build code—not to produce a perfect production-ready system.