System Architecture Document

Table of Contents

---

1. System Overview

1.1 Project Description

The Moral Decisions project supports “AI + Human Exploration of Daily Moral Decisions” via two interactive websites:

- **Moral Profile Website:** Scenario-based moral mini-games (inspired by online forums) to help users explore common moral dilemmas.
- **Opinion Survey Website:** Collects real moral choices and generates personalized feedback reports based on decision patterns.

The goal is to leverage AI–human collaboration to build a data-driven platform for everyday, nuanced moral reasoning, creating a scalable foundation for ethical AI research.

Tech stack and deployment highlights (from the LandingSite repo and linked resources):

Deployment: Azure Web App (Docker supported) Frontend: Next.js (with API Routes; index page may directly access DB to simplify backend) Backend: NestJS (separate from frontend; interacts via REST API) API: Two key endpoints Deliver survey questions by studyId Receive survey responses by prolificId (documentation also mentions “profilicId”; use actual implementation in code) Version Control: Git (GitHub) Dev environment: Unified via server + Docker Testing: Postman for API testing Audience and impact:

Users: Global participants Beneficiaries: Researchers in AI ethics, moral psychology, and social computing Client: Ziyu Chen (Computational Media Lab) Reference repository: https://github.com/24-S1-2-C-Moral-Decisions/LandingSite

1.2 Project Objectives and Scope

Objectives

- Build a scalable AI–human interaction platform that covers everyday (non-extreme/non-idealized) moral situations.
- Collect high-quality moral decision data to form datasets and trends suitable for research.
- Provide personalized feedback to help users understand their moral reasoning patterns.

Scope

- Frontend interaction (Next.js) and server-side APIs (Next.js API Routes + a separate NestJS backend).
- Two core APIs: deliver surveys by studyId; receive responses by prolificId (documentation also mentions the spelling “profilicId”).
- Direct database access by Next.js for index page scenarios to reduce extra backend hops.
- Azure WebApp deployment, availability maintenance, logging, and basic monitoring.

Out of scope (deferred to future phases or research tooling)

- Advanced analysis workbench and visualizations (can be built later on exported data).
- Complex identity systems and full RBAC (current focus is anonymous participation/external IDs).
- High-compliance anonymization/de-identification pipelines (to be added per research ethics and compliance needs).

1.3 Note on Project History

This is a continuation project inherited from previous team.

---

2. Architecture Diagrams

2.1 High-Level System Architecture

```
+-------------------+           HTTPS          +--------------------------
--+
|  End Users        |  <----------------------> |  Azure WebApp (Next.js)
|
|  (Browser/Mobile)|                            |  - UI (Moral Profile &
|
+-------------------+                            |    Opinion Survey)
|
                                                 |  - Next.js API Routes
|
                                                 |    (partial reads/index)
|
                                                 +------------+-----------
-+
                                                              |
                                        REST (JSON)           | Direct DB
(specific pages)
                                                              v
                                                 +------------+-----------
-+
                                                 | Azure WebApp (NestJS)
|
                                                 | - Separate backend
|
                                                 | - Two core APIs
|
                                                 +------------+-----------
-+
                                                         |
                                                         | DB
Driver/ORM
                                                         v
                                              +-------------------------

--+
                                              | Managed Database

|
                                              | (PostgreSQL or MongoDB)
|
                                              +-------------------------

--+
```

- Next.js serves the frontend and some light API routes (especially for index data reads).
- NestJS acts as a separate backend layer handling main business APIs and logic.
- The database stores core data for studies, questions, and responses.

2.2 Component Interactions and Data Flow

(1) Page access User -> Next.js: request home/survey pages (SSR/CSR)

(2) Fetch survey Next.js UI -> API: GET /api/survey?studyId= API -> DB: query study and questions API -> Next.js UI: return survey JSON

(3) Submit responses Next.js UI -> API: POST /api/response {prolificId, studyId, answers, ...} API -> DB: write response, timestamp, compute necessary derived fields

(4) Personalized feedback Next.js UI -> API: (A) compute on-the-fly and return (B) read from cache/derived table and return API -> Next.js UI: return feedback JSON -> UI renders report

(5) Research data export (offline/back-office) Admin/Research tools -> DB: export by time window/stratified sampling

2.3 Frontend-Backend-Database Relationships

[Frontend (Next.js)]

- UI rendering (SSR/CSR/ISR)
- Light APIs (API Routes, direct DB reads for specific pages) | | JSON/HTTPS v [Backend (NestJS)]
- Business APIs (survey delivery, response intake)
- Validation, rate limiting, basic auditing | | DB connection (pooling/retries) v [Database]
- Core business data (studies, questions, responses, optional feedback cache)
- Indexing and backup strategy

---

3. Technology Stack

3.1 Frontend Technologies

- Framework: Next.js (React-based)
- Rendering: SSR/CSR; ISR or caching where appropriate

- Data fetching: fetch/React Query/SWR (one of these, per actual code)
- Styling/build: per repository implementation (e.g., CSS Modules/Tailwind)
- Version: follow package.json; align with Node LTS

3.2 Backend Technologies

- Runtime: Node.js (LTS 18/20 per package.json/deployment)
- Framework: NestJS (with Express or Fastify adapter)
- Validation/serialization: class-validator/class-transformer (typical Nest usage; confirm in code)
- Logging: platform logs (Azure) + application logs

3.3 Database

- Database Type: MongoDB
- Version: [Version number]
- Structure: see Chapter 4 (logical model). Next.js may directly access DB for index-page reads; other operations go through NestJS APIs.

3.4 Deployment and Infrastructure

- Hosting: Azure WebApp (Linux), container-supported (Docker)
- Infrastructure:
  - Environment variables (PORT, DATABASE_URL, etc.)
  - Horizontal scaling / plan SKU upgrades
  - Platform and container logs (/home/LogFiles); optional Application Insights
- CI/CD:
  - Git for version control (GitHub)
  - Deploy via Azure Portal/CLI or GitHub Actions (if configured)
  - Postman for API regression and smoke testing

---

4. Database Overview

4.1 List of Databases

Primary App Database (single instance): Stores studies, questions, responses, and optional feedback cache/audit records.

4.2 Database Purposes

- studies: survey definitions and metadata (studyId, version, publish status, start/end)
- questions: question bank (scoped to a study or shared)
- responses: response data (prolificId, studyId, answers, duration, device/UA metadata)
- feedback_cache (optional): cached personalized feedback results to speed reads
- audit_logs (optional): key event auditing (survey creation, data export, retry anomalies) Note: Actual table/collection splits, field names, and indexing should follow implementation.

4.3 Database Technologies

- Engine: PostgreSQL or MongoDB (one of them, per deployment)
- Access patterns:
- Next.js: direct connection for index-page reads (read-only/limited writes)
- NestJS: business API reads/writes
- Configuration: via DATABASE_URL and connection pool settings Note: For detailed connection, mutation, and content conventions, see the Database Documentation in the repository.

Note: For detailed database connection, modification, and content information, see the Database Documentation.

---

5. API Structure

5.1 API Overview

Protocol: HTTPS + JSON Two core API categories:

- Survey delivery: return survey and questions by studyId
- Response intake: record user responses by prolificId (or profilicId) Additional common/suggested endpoints:
- Health check: GET /api/health
- Feedback retrieval (if implemented): GET /api/feedback?prolificId=&studyId=

Endpoint locations may be:

- Next.js API Routes (/pages/api or app routes)
- Separate NestJS service (with a prefix like /api/...)

5.2 Main API Routes

GET /api/survey?studyId=

- Description: Fetch the survey configuration and question list for a given study.

POST /api/response

- Description: Submit a user’s survey response.

GET /api/health

- Description: Health check returning service and dependency status.

Notes:

- Index-page data retrieval may connect directly to the database from Next.js to reduce extra backend hops.
- Other business interactions should go through the NestJS APIs for authorization, auditing, and extensibility.

5.3 Authentication Approach

Participant endpoints: typically anonymous or identified by an external platform ID (prolificId), no login required. Admin/researcher endpoints (e.g., export, corrections, replays): adopt at least one of the following (combinable):

- Network-level access restrictions (Azure Access Restrictions/IP allowlisting)
- API Key (custom header, e.g., X-API-Key) stored in secure secret stores (Key Vault/env vars)
- Basic auth or OAuth (if integrating with a researcher portal) Security baseline:
- HTTPS everywhere
- Input validation and rate limiting
- Audit logging (retain traces for key admin operations)
- Privacy and compliance (collect the minimum necessary anonymous identifiers and metadata)

Quality Checklist

- ✅ Clear diagrams showing system components
- ✅ Accurate representation of current system
- ✅ Professional formatting
- ✅ All sections completed
- ✅ Reviewed by team