SafeNutriKids Parental Data Access Module

Technical and Integration Guidelines for GDPR-Oriented Education Platforms

1. Purpose and scope

The parental data access module provides a narrowly scoped service that manages parental registration, child profile registration, consent recording and withdrawal, parental access to stored child data, and audit logging of parental actions. The module is designed for integration into child-centred education and nutrition platforms that operate under the General Data Protection Regulation and related data protection frameworks. It does not handle authentication, user interface rendering, or identity proofing. Instead, it assumes that these elements are provided by the host platform, and focuses on providing a transparent and auditable interface for data access and rectification by parents or legal guardians.

The scope of the service is deliberately limited to avoid uncontrolled growth of responsibilities. The module manages only the information that is strictly necessary for its functions, such as pseudonymised identifiers, parental contact information, consent scopes, and structured representations of child data. It delegates all decisions regarding retention periods, legal bases for processing, and institutional policies to the integrating organisation, which configures and governs the deployment environment.

2. Architectural overview

The module is implemented as a stateless FastAPI-based microservice that exposes a RESTful HTTP interface. Each API call is self-contained and includes all information needed to process the request. Persistent state is stored in a backing data store, which is abstracted in the reference implementation as in-memory dictionaries and which must be replaced by a database in any production deployment.

The service is intended to sit behind an API gateway or reverse proxy that handles transport layer security, authentication, rate limiting and traffic monitoring. The platform that embeds the module authenticates parents and passes only the identifiers that are required for authorisation decisions. The module does not issue tokens and does not store passwords or credentials. It only evaluates whether a parent is allowed to perform a particular operation on a specific child profile, based on the stored relationships and consent records.

The typical deployment pattern is a containerised service managed by an orchestration platform. Horizontal scaling is possible because the application layer remains stateless and all stateful data are managed in an external database. Logging is performed in a structured manner and can be shipped to a central log management system to support audits and incident investigations.

3. Data model and semantics

The module models five core entities. The Parent entity stores a unique identifier, an e-mail address, the full name that the parent chooses to provide, and a creation timestamp. This entity is meant to be linked to the platform's own account model but does not need to replicate it in full. The Child entity stores a unique identifier, the identifier of the associated parent, a pseudonym or screen name that is suitable for display, optional non-identifying demographic attributes such as birth year, and a creation timestamp. Legal names and direct identifiers of the child remain solely in the host platform and are not persisted by this module.

The Consent entity records the relationship between a parent, a child, and a set of consent scopes. The scope field is a list of machine-readable labels, such as "profile", "analytics", or "personalised_recommendations", that can be interpreted by other components of the platform. The consent has a status, either granted or withdrawn, together with timestamps for creation and last update. The data model allows several consent records per parent–child pair to capture evolving scopes across time, although the typical integration will consider only the currently active record.

The ChildData entity is represented internally as a dictionary that maps field names to arbitrary values. This provides flexibility for different domains, such as educational performance indicators or nutrition-related preferences. The module does not impose a schema but it is advisable that the host platform maintains a clear data dictionary that specifies which keys are used and what data types are expected. Finally, the AccessLog entity captures parental actions, including the parent identifier, the child identifier, the type of action (view, rectify, export), the affected field when applicable, and a timestamp. These logs provide the basis for later audits and for responding to supervisory authority enquiries.

4. API behaviour and integration patterns

The service exposes a small set of endpoints that correspond directly to parental data rights. A registration endpoint accepts parental contact details and returns an internal identifier that the platform can map to its own user account. A child registration endpoint creates a new child profile associated with a parent and initialises an empty data record. A consent endpoint records that a given parent has granted consent for specific processing scopes on a given child. A withdrawal endpoint changes the status of a specific consent record to withdrawn.

Data access endpoints operate strictly within the constraints of these relationships and consents. A data view endpoint accepts a parent identifier and a child identifier and returns the stored data for the child, provided that there is an active consent record connecting this parent and child. A data rectification endpoint accepts a set of key–value updates and applies them to the child's record, again only if an active consent exists and the parent–child relationship is valid. Every successful access or rectification operation triggers the generation of an access log entry. An additional endpoint returns the full list of access log entries, which can be filtered or exported by the integrating platform.

The module assumes that all parent and child identifiers presented to the API have been authenticated and validated by the host platform. In practice, the education platform will issue authenticated requests to the module on behalf of the parent and will never expose the internal identifiers directly to the user interface. The platform is responsible for enforcing rate limits, preventing enumeration of identifiers, and protecting against cross-site request forgery in its own client interfaces.

5. Deployment and configuration

In the reference implementation the module is written in Python using FastAPI and Pydantic. A typical deployment uses a production-grade ASGI server such as Uvicorn or Hypercorn, behind a reverse proxy that terminates transport layer security and routes requests. The service should be packaged as a container image with all dependencies pinned to specific versions to ensure reproducibility.

The configuration should be externalised through environment variables or configuration files. At minimum, the deployment should specify the database connection string, log level, and security-related settings such as allowed origins for cross-origin requests. Database migrations should be handled through a dedicated tool so that changes in the data model can be applied predictably across environments.

It is recommended to maintain separate configurations for development, testing, staging, and production environments. Test and staging deployments should use synthetic or fully anonymised data and should not contain any live child information. Continuous integration pipelines can run automated tests and static analysis tools upon each modification of the codebase and create versioned container images when tests succeed.

6. Security, privacy, and data protection practices

The parental data access module is intended to support the technical implementation of parental rights and obligations under applicable data protection legislation. In order to use it responsibly, the integrating organisation must adopt specific operational measures. All communication with the service must take place over encrypted channels that use current best practices for protocol configuration. Authentication and authorisation must be provided by the host platform, and parents must be identified and verified through procedures that meet the institution’s legal obligations.

The data model should be deployed in a way that supports data minimisation. Child entities should only contain pseudonyms or screen names, birth year or approximate age, and strictly necessary linkage identifiers. Any mapping to legal identities or school registers should remain in separate, more restricted systems. Database-level access should be limited to technical personnel with a clear operational role, and query logs should be regularly reviewed for anomalous patterns.

Access logs generated by the module should be collected into an immutable or append-only storage system so that they can provide credible evidence in the event of disputes or regulatory investigations. Retention periods for logs and child data should be implemented through scheduled jobs or database lifecycle management policies that align with institutional rules. Regular privacy and security audits should be carried out to verify that the module behaves as specified and that integrations do not introduce insecure shortcuts.

7. Alignment with the DRG4Food Toolbox

The module is designed to be compatible with the privacy audit frameworks and user-centric design patterns promoted by the **DRG4Food Toolbox**. The logging model is intentionally simple and machine readable, which facilitates integration with automated privacy auditing tools that consume event streams. Consent scopes are expressed as explicit, structured labels that can be mapped to threat models and risk assessments used elsewhere in the **DRG4Food** ecosystem.

The module provides a clear separation between user interface design and backend enforcement. This separation allows different user interface patterns recommended by the toolbox, such as child-friendly explanatory screens for data use, to be implemented independently while relying on a stable and verifiable backend for enforcement. Impact assessment templates and fairness validation tools from the toolbox can reference the behaviour of the module as a concrete mitigation measure, for instance by documenting how parents can inspect and correct nutrition-related profiles of their children.

8. Testing and validation

Robust testing is essential before this module is deployed in environments that process children’s data. Unit tests should cover all endpoints under normal and exceptional conditions, including attempts to access data without valid consent, use of incorrect parent identifiers, and withdrawal of consent followed by access attempts. Integration tests should confirm that the host platform correctly propagates authenticated parent identities and that errors are handled gracefully in the user interface.

Scenario-based tests should model realistic parental journeys. Examples include registering multiple children under a single parent, recording and withdrawing consent for specific processing scopes, rectifying incorrect allergy information, and exporting access logs for a single family. Synthetic datasets that reflect diverse family structures and use cases can be generated for this purpose. Periodic regression testing is advisable whenever the module or its dependencies are updated.

9. Open source packaging and governance

Released as open source, the module should be packaged with a clear repository structure. Source code can be placed under a src directory, with automated tests under tests and narrative documentation under docs. A machine-readable licence file should be included, specifying an OSI-approved licence such as the MIT Licence or Apache Licence 2.0. A configuration file for dependency management should list all required packages with version constraints.

Governance of the open source project should include contribution guidelines that explain how external contributors can propose changes, how issues and security vulnerabilities should be reported, and how release versions are managed. Semantic versioning is recommended so that downstream users can anticipate the impact of upgrades. An explicit roadmap that indicates planned extensions, such as support for additional consent scopes or alternative storage backends, will help the wider **DRG4Food** community to align their own developments with the module's evolution.