CURRENT IMPLEMENTATION · VERIFIED 8 NOVEMBER 2026

Technical Requirements Document

Architecture, runtime, security, database schema, deployment, and verified development-database inventory.

01 · TECHNICAL OVERVIEW

1. Technical overview

SMROS is a server-rendered PHP application backed by MySQL/MariaDB. PHP pages process HTTP requests, validate form inputs, authorize the account, execute PDO queries, and render HTML through shared portal includes. Vanilla JavaScript and CSS add responsive navigation, meal filtering and quantity interactions, forms, charts, dialogs, and reduced-motion-aware animation. No separate API server or frontend build pipeline is declared.

The inspected local CLI runtime is PHP 8.1.25; the connected development database is MariaDB 10.4.32. A user-provided cPanel MultiPHP screenshot showed PHP 8.2 with PHP-FPM for the system domain, but the live app's PHP configuration, database connection, and deployment health were not independently verified.

02 · SYSTEM ARCHITECTURE

2. System architecture

1. **Browser:** requests PHP pages and receives server-rendered HTML. style.css defines the responsive portal UI; app.js provides progressive enhancements.
2. **Route/controller layer:** root pages contain shared request handlers; small student/ and admin/ wrappers set portal context and require role guards before including the appropriate view.
3. **Shared services:** includes/database.php initializes sessions and PDO; includes/functions.php implements account checks, CSRF, input/helper functions, and image handling; email_notifications.php integrates PHPMailer.
4. **Database:** InnoDB tables store users, categories, meals, pickup slots, orders, line items, status history, notification delivery state, login attempts, and password reset data.
5. **External SMTP:** authenticated SMTP is used for student password recovery/confirmation and Ready for Pickup mail. Email is optional for page operation but required for actual email delivery.

Students and administrators use one application and database but separate role-protected routes and navigation. The PRD/TRD Help pages require an authenticated active account.

03 · TECHNOLOGY STACK

3. Technology stack

Layer	Observed implementation
Server language	PHP 8 syntax; local CLI observed at PHP 8.1.25.
Database	MySQL/MariaDB, InnoDB, foreign keys, UTF-8/utf8mb4 PDO connection; local development server observed at MariaDB 10.4.32.
Web server	Apache under XAMPP for documented local development; cPanel/Apache-style PHP hosting is supported when configured.

Layer	Observed implementation
Frontend	HTML5, CSS, vanilla JavaScript; no SPA framework or frontend package/build manifest.
Composer libraries	phpmailer/phpmailer ^6.10 and dompdf/dompdf ^3.1; lock file and vendor/autoload.php are present.
Local tools observed	XAMPP PHP 8.1.25; cPanel screenshot shows PHP 8.2/PHP-FPM selection for the system domain.

Code uses PHP 8 language constructs such as union types, match, and null-safe access. Use PHP 8.0 or newer; PHP 8.1+ is recommended. The repository does not pin an exact production patch release or a required MariaDB/MySQL minor version.

04 · FRONTEND

4. Frontend technologies

- **HTML:** PHP-rendered forms, tables, dialogs, navigation, status timelines, and documentation tables.
- **CSS:** shared style.css supplies the cream and pastel SMROS theme, desktop/tablet/mobile portal layouts, compact/collapsible navigation, image ratios, and reduced-motion support. Print rules give documentation an A4-friendly Times New Roman, 12 pt, black layout and hide site chrome.
- **JavaScript:** app.js handles responsive sidebar behavior, meal search/category filtering, quantity and total updates, password visibility, demo autofill when enabled, reset-token browser flow, confirmation/loading states, image fallbacks, report print, and lightweight animation.
- **Accessibility:** controls use labels/ARIA where present, keyboard-visible focus, and prefers-reduced-motion checks; client-side validation supplements, but does not replace, server-side checks.
- **Assets:** logo and SVG meal placeholder are local. Meal photos are local files in uploads/meals/.

No npm manifest, frontend build step, or JavaScript framework is declared.

05 · BACKEND

5. Backend technologies

- PHP handles route selection, form processing, validation, account authorization, database transactions, reports, and HTML rendering.
- PDO connects via a MySQL DSN with charset=utf8mb4, exception error mode, associative fetches, and emulated prepares disabled.
- Environment variables configure database, HTTPS enforcement, public base URL, SMTP, and development-only demo configuration. No application .env loader is declared.
- Composer autoloading provides PHPMailer and Dompdf. PHPMailer sends SMTP email; Dompdf generates the documentation PDF download.
- There is no general public JSON API; the reset-token validation request is an internal same-origin browser flow.

06 · DATABASE REQUIREMENTS

6. Database requirements

Use MySQL or MariaDB with InnoDB, foreign-key support, and a database/user with the required select, insert, update, and delete privileges. PDO connects using utf8mb4. Production must use a dedicated database and least-privilege account rather than the XAMPP root default.

The inspected database is smros_dev; the exact live/production schema may differ and should be queried or compared after backup before deployment or migration.

Connection variable	Purpose	Local default
SMROS_DB_HOST	Database server host.	localhost
SMROS_DB_NAME	Database name.	smros
SMROS_DB_USER	Database account.	root
SMROS_DB_PASS	Database account password.	Empty in a typical local XAMPP setup.

07 · VERIFIED DATABASE STRUCTURE

7. Database schema and relationships

The following catalog reflects information_schema for the local smros_dev database on 8 November 2026. Data types are those reported by MariaDB 10.4.32. “NN” means NOT NULL; defaults are shown where present. The database contains 12 tables and no separate payments or reservation table.

7.1 users — student and administrator accounts

Column	Type	Null/default/key
id	INT	NN; auto-increment; PK
student_no	VARCHAR(30)	NULL
name	VARCHAR(100)	NN
email	VARCHAR(120)	NN; UNIQUE
password_hash	VARCHAR(255)	NN
role	ENUM('student','admin')	NN; default 'student'
is_active	TINYINT(1)	NN; default 1
created_at	TIMESTAMP	NN; default CURRENT_TIMESTAMP

Indexes: PRIMARY(id), unique email. Parent of orders, reset records, notification recipients, and optional status-history actor references.

7.2 meal_categories — meal classification

Column	Type	Null/default/key
id	INT	NN; auto-increment; PK

Column	Type	Null/default/key
name	VARCHAR(50)	NN; UNIQUE
created_at	TIMESTAMP	NN; default CURRENT_TIMESTAMP

Indexes: PRIMARY(id), unique name. One category may relate to many meals; deleting a referenced category is restricted by the admin workflow, while the database meal FK uses ON DELETE SET NULL.

7.3 meals — menu, price, inventory, and image reference

Column	Type	Null/default/key
id	INT	NN; auto-increment; PK
name	VARCHAR(100)	NN
description	VARCHAR(255)	NULL
image_path	VARCHAR(255)	NULL
category_id	INT	NULL; FK → meal_categories.id
price	DECIMAL(8,2)	NN
stock	INT	NN; default 0
available	TINYINT(1)	NN; default 1

Indexes: PRIMARY(id), idx_meals_category_available(category_id,available). FK fk_meals_category references meal_categories(id) ON DELETE SET NULL. The stored image path is a generated filename, not a remote URL.

7.4 pickup_slots — reservation schedule and capacity

Column	Type	Null/default/key
id	INT	NN; auto-increment; PK
slot_date	DATE	NN
start_time	TIME	NN
end_time	TIME	NN
capacity	INT	NN; default 20

Indexes: PRIMARY(id), unique uq_pickup_slot(slot_date,start_time). One slot may have many orders; orders reference it with a restrictive FK.

7.5 orders — reservation/order header

Column	Type	Null/default/key
id	INT	NN; auto-increment; PK
user_id	INT	NN; FK → users.id
slot_id	INT	NN; FK → pickup_slots.id
status	ENUM('pending','preparing','ready','completed','cancelled')	NN; default 'pending'
total	DECIMAL(10,2)	NN; default 0.00
created_at	TIMESTAMP	NN; default CURRENT_TIMESTAMP
completed_at	TIMESTAMP	NULL

Indexes: PRIMARY(id), idx_orders_slot_status(slot_id,status), idx_orders_user_created(user_id,created_at), idx_orders_status_created(status,created_at), idx_orders_status_completed(status,completed_at). FKs to users and pickup_slots use the InnoDB default restrictive delete behavior.

7.6 order_items — order lines and price snapshots

Column	Type	Null/default/key
id	INT	NN; auto-increment; PK
order_id	INT	NN; FK → orders.id
meal_id	INT	NN; FK → meals.id
qty	INT	NN
unit_price	DECIMAL(8,2)	NN

Indexes: PRIMARY(id), idx_order_items_order(order_id), idx_order_items_meal(meal_id). FKs to orders and meals restrict deleting referenced history. unit_price preserves the order-time amount.

7.7 order_status_history — status transition records

Column	Type	Null/default/key
id	BIGINT	NN; auto-increment; PK
order_id	INT	NN; FK → orders.id
old_status	VARCHAR(20)	NULL for initial status
new_status	VARCHAR(20)	NN

Column	Type	Null/default/key
changed_by	INT	NULL; FK → users.id
changed_at	TIMESTAMP	NN; default CURRENT_TIMESTAMP

Indexes: PRIMARY(id), idx_order_status_history_order(order_id,changed_at), idx_order_status_history_admin(changed_by,changed_at). Order deletion cascades to its history; deleting a referenced actor sets changed_by to NULL.

7.8 order_notifications — email delivery attempts

Column	Type	Null/default/key
id	BIGINT	NN; auto-increment; PK
order_id	INT	NN; FK → orders.id
student_id	INT	NN; FK → users.id
recipient_email	VARCHAR(120)	NN
notification_type	VARCHAR(50)	NN
sending_status	ENUM('pending','sending','sent','failed')	NN; default 'pending'
attempt_count	INT UNSIGNED	NN; default 0
sent_at	DATETIME	NULL
last_attempt_at	DATETIME	NULL
error_message	VARCHAR(255)	NULL
created_at	TIMESTAMP	NN; default CURRENT_TIMESTAMP
updated_at	TIMESTAMP	NN; default CURRENT_TIMESTAMP; ON UPDATE CURRENT_TIMESTAMP

Indexes: PRIMARY(id), unique uq_order_notification_type(order_id,notification_type), idx_order_notifications_student(student_id,created_at), idx_order_notifications_status(sending_status,updated_at). Deleting an order cascades to its notifications; a referenced recipient user is restricted from deletion.

7.9 login_attempts — password/challenge throttling

Column	Type	Null/default/key
id	BIGINT	NN; auto-increment; PK

Column	Type	Null/default/key
ip_hash	CHAR(64)	NN; SHA-256 digest
attempted_at	DATETIME	NN

Indexes: PRIMARY(id), idx_login_attempts_ip_time(ip_hash,attempted_at). Old rows are removed by the login handler; this table has no user FK.

7.10 password_resets — legacy reset-token table

Column	Type	Null/default/key
id	INT	NN; auto-increment; PK
user_id	INT	NN; FK → users.id
token_hash	CHAR(64)	NN; UNIQUE
expires_at	DATETIME	NN
created_at	TIMESTAMP	NN; default CURRENT_TIMESTAMP

Indexes: PRIMARY(id), unique token hash, idx_password_resets_user(user_id), idx_password_resets_expiry(expires_at). User deletion cascades. This legacy table exists in the schema but the current reset flow uses password_reset_tokens.

7.11 password_reset_tokens — active student reset flow

Column	Type	Null/default/key
id	BIGINT	NN; auto-increment; PK
student_id	INT	NN; FK → users.id
token_hash	CHAR(64)	NN; UNIQUE
expires_at	DATETIME	NN
used_at	DATETIME	NULL
created_at	TIMESTAMP	NN; default CURRENT_TIMESTAMP

Indexes: PRIMARY(id), unique token hash, idx_password_reset_tokens_student(student_id,used_at), idx_password_reset_tokens_expiry(expires_at). Deleting a user cascades to reset tokens.

7.12 password_reset_rate_limits — reset-request counters

Column	Type	Null/default/key
ip_hash	CHAR(64)	NN; PK
window_started_at	DATETIME	NN

Column	Type	Null/default/key
request_count	INT UNSIGNED	NN; default 0
updated_at	TIMESTAMP	NN; default CURRENT_TIMESTAMP; ON UPDATE CURRENT_TIMESTAMP

Indexes: PRIMARY(ip_hash). Stores a prefixed SHA-256 digest rather than a raw client IP and is not related by FK to users.

7.13 Relationships, indexes, and database-enforced rules

- meal_categories 1 → many meals; deleting a category sets meal category_id to NULL at the database layer.
- users 1 → many orders and pickup_slots 1 → many orders; order user/slot references are restrictive.
- orders 1 → many order_items; meals 1 → many order_items; both references are restrictive to preserve transaction history.
- orders 1 → many order_status_history and notifications; deleting an order cascades to these records. A deleted history actor is nulled.
- users 1 → many password_reset_tokens/password_resets with delete cascades; order-notification recipient users are restrictive.
- Unique constraints: user email, category name, reset token hashes, notification (order,type), pickup (date,start time). Secondary indexes are listed per table above.
- No database CHECK constraints were reported in the inspected schema; request validation and transaction logic enforce business rules.

7.14 Verified development database and seeded data

Exact SQL counts were queried from local smros_dev on 8 November 2026. The current CLI seeder defines 40 sample students, 2 sample administrators, 53 sample meals, 10 pickup slots, 45 sample orders, status history and, when supported, failed sample notification logs. It adds seven new categories because the existing Snacks category is reused. Sample user emails end in @smros.test; the seeder generates random password hashes and refuses production database names.

Table	Rows in smros_dev	Identified sample rows
users	48	42
meal_categories	10	7 added by sample seeder; three baseline categories
meals	60	53
pickup_slots	13	10
orders	51	45

Table	Rows in smros_dev	Identified sample rows
order_items	108	90
order_status_history	120	108
order_notifications	14	9
login_attempts	0	0
password_resets	0	0
password_reset_tokens	1	0 sample rows
password_reset_rate_limits	1	0 sample rows
Total	426	364 identified sample rows

The 364 sample-row total is 42 users + 7 inserted categories + 53 meals + 10 slots + 45 orders + 90 order items + 108 status-history rows + 9 sample notification logs. It is not a per-table target. The remaining rows include pre-existing/demo accounts and menu/order records plus operational reset-rate/token state. The seeded notifications are marked failed with an explicit “no email was sent” sample note.

Integrity checks found zero orphan orders, order items, or meal categories and zero order-total mismatches. The development database has 60/60 non-NULL meal image paths, 54 distinct filenames; this does not verify a separate production database.

There is no payments table. Reservation and pickup scheduling are represented by orders and pickup_slots. Counts describe the local development snapshot only and change when the database is modified.

08 · AUTHENTICATION

8. Authentication and authorization

- **Login:** email input is validated; PHP password_verify() checks the stored hash; only active student/admin roles are accepted. Successful login regenerates the session ID and redirects by stored role.
- **Registration:** uses a transaction and checks the users table before assigning the initial admin role; later public registrations are students. Database unique email prevents duplicates.
- **Authorization:** protected route wrappers call need('student') or need('admin'). The helper reloads account name, role, and active state from the database, clears inactive sessions, refreshes the session user data, and denies wrong-role requests with HTTP 403.
- **Bootstrap risk:** first-user administrator assignment is only appropriate for initial empty-database setup; production registration and initial admin creation need careful deployment controls.

09 · SESSION AND CSRF

9. Session and CSRF security

- PHP session strict mode and cookie-only session IDs are enabled.

- Session cookies set HttpOnly and SameSite=Lax; Secure is set when PHP detects HTTPS. HSTS is sent on HTTPS requests.
- Successful login calls session_regenerate_id(true). Protected state-changing forms carry a random session CSRF token and use hash_equals(); invalid/missing tokens receive HTTP 419.
- Login requires a random one-use arithmetic challenge with a five-minute expiry. Failed credentials/challenges are recorded using a SHA-256 digest of the remote IP; the threshold is 10 failures in 15 minutes, with old attempt rows cleaned up.
- HTTPS enforcement is configurable using SMROS_REQUIRE_HTTPS=1. The redirect validates the host/path and does not trust arbitrary forwarded-protocol headers.

10 · PASSWORDS AND RECOVERY

10. Password hashing and recovery

New and changed passwords are stored as PHP password_hash(..., PASSWORD_DEFAULT) hashes and verified with password_verify(). The application requires 8–72 bytes and at least three of lowercase, uppercase, numeric, and symbol character classes.

Only active student accounts can request a password reset. The request response is generic; requests are limited to five per IP digest per 15-minute window. A 32-byte cryptographically random token is delivered via a URL fragment; SHA-256(token) is stored with a 30-minute expiry. The token is single-use, locked/rechecked on completion, and consumed in the password-update transaction. Password reset and password-change confirmation email require working SMTP.

password_reset_tokens is the active flow. password_resets remains as a legacy schema table; the current request handlers do not use it.

11 · MEAL IMAGE STORAGE

11. Meal image upload and storage

- Database references use the existing nullable meals.image_path VARCHAR(255) column; no additional image table is used.
- Files are placed in uploads/meals/ with generated 32-character hexadecimal names and JPG, PNG, or WEBP extensions.
- Server validation requires a successful uploaded file, maximum 5 MiB, allowlisted Fileinfo MIME type matching the original extension and image metadata, positive dimensions no larger than 12,000 per side or 20 million pixels, and writable storage.
- Existing image is retained on edit when no replacement is uploaded. An explicit remove option can clear it. Image rendering rejects path traversal, symlinks, and missing files and uses the local SVG placeholder.
- Production must transfer the actual image files as well as compatible database image_path values and grant the PHP process only necessary write access to the upload directory. A database-only backup excludes image files.

12 · ORDER PROCESSING

12. Reservation and order processing

1. Before creating an order, the handler checks a session-scoped one-use random order token to prevent accidental duplicate form submission.
2. It validates each meal ID/quantity and pickup slot, then starts a transaction and locks the future pickup slot using SELECT ... FOR UPDATE.

3. The handler counts non-cancelled orders against slot capacity, inserts the order, then locks each available meal row and performs a conditional stock decrement.
4. It inserts order-item quantity and unit-price snapshots, computes the server-side total, and commits only after every item succeeds. Exceptions roll back the transaction.
5. Admin capacity edits lock the slot and cannot reduce capacity below active order count. Slot deletion is denied when order history references it.
6. Cancellations lock the order; student cancellation is limited to pending status. Cancellation is terminal and stock restoration updates from recorded order-item quantities in the same transactional workflow.

The schema has no separate payment transaction or reservation entity. The orders row is the reservation record.

13 · OUTBOUND EMAIL

13. Email and SMTP configuration

Composer-managed PHPMailer uses authenticated SMTP. Apache/PHP must receive these environment variables:

- SMROS_SMTP_HOST, SMROS_SMTP_PORT, SMROS_SMTP_ENCRYPTION
- SMROS_SMTP_USERNAME, SMROS_SMTP_PASSWORD, SMROS_SMTP_FROM

Code accepts port 587 with tls (STARTTLS) or port 465 with ssl (implicit TLS), requires OpenSSL, validates host/email configuration, and uses a 15-second mail connection timeout. SMTP credentials belong in protected server configuration, never PHP source, repository files, or public web-root files.

Ready mail is queued in order_notifications after order status commits. Delivery attempts are claimed as sending, then recorded as sent or failed. Admin retry is limited to failed ready-order emails with fewer than five attempts. A successful provider acceptance cannot assure inbox delivery or exactly-once delivery.

Password reset and password-changed messages use the same SMTP configuration. SMROS_BASE_URL must be the public HTTPS application base URL in production; HTTP reset URLs are accepted only for localhost development hosts.

14 · SCHEMA EVOLUTION

14. Database migrations

- schema.sql**:** creates the initial database/schema and includes current tables, constraints, indexes, initial categories, and starter meals. Import into a new database after adjusting the database name as needed.
- migrate.sql**:** consolidated upgrade script for an existing smros database. It conditionally adds meal image/category columns, account active state, order completion time, indexes, foreign keys, notification and history tables, password reset tokens, and rate-limit storage while retaining data.
- migrations/001_ready_email_notifications.sql**:** creates order notification delivery tracking.
- migrations/002_admin_email_and_activity.sql**:** adds notification last-attempt tracking and order status history.
- migrations/003_student_password_reset_tokens.sql**:** creates active reset-token and IP rate-limit tables.

Back up before applying DDL. The consolidated script includes the versioned schema additions; do not apply the consolidated script and equivalent versioned scripts as separate setup steps. The

repository does not contain a migration-version ledger table; operators track which script path was used.

15 · SECURITY REQUIREMENTS

15. Security requirements

- Keep live database and SMTP credentials out of source control and web-accessible files; configure least-privilege production database users and protected environment settings.
- Use prepared PDO statements, server-side validation, escaped output, role guards, and CSRF validation for state-changing requests.
- Keep PHP session cookies secure under HTTPS; install a valid certificate and enable SMROS_REQUIRE_HTTPS=1 for production.
- Keep APP_ENV unset or set to production on the live site. Demo account exposure and demo seeders require development-only environment controls and exact smros_dev database checks.
- Do not allow public upload execution in uploads/; preserve validation and least-privilege filesystem permissions. Keep backups containing hashes/tokens private.
- Review cPanel/PHP error logs server-side; never expose stack traces, database errors, SMTP values, reset tokens, or credentials to public users.

16 · RUNTIME REQUIREMENTS

16. System requirements

Requirement	Current implementation need
PHP	PHP 8.0+ syntax; PHP 8.1+ recommended. Enable PDO/pdo_mysql, Fileinfo for image validation, OpenSSL for HTTPS/SMTP, and DOM plus mbstring for Dompdf/Composer runtime dependencies. Zip is not required by the application at runtime.
Database	MySQL/MariaDB with InnoDB, foreign keys, utf8mb4 support; local observed MariaDB 10.4.32.
Web server	Apache/PHP handler configured for the correct document root; XAMPP is the documented local setup. cPanel must map the intended domain to the project root.
Composer	Install locked dependencies; bundled vendor tree currently provides PHPMailer ^6.10 and Dompdf ^3.1.
Uploads	PHP can write to uploads/meals/; set upload_max_filesize ≥ 5M and post_max_size ≥ 6M.
Browser	Current desktop/mobile browser supporting standard HTML, CSS Grid/Flexbox, and JavaScript for enhanced controls.

The codebase does not specify production capacity, CPU/memory sizing, concurrency, service availability, or automated performance tests; determine and verify these for the chosen host.

17 · DEVELOPMENT ENVIRONMENT

17. Development environment

1. Install XAMPP with Apache, MySQL/MariaDB, PHP 8+, PDO MySQL, Fileinfo, OpenSSL, and Composer.
2. Place project files in C:\xampp\htdocs\smros; start Apache and MySQL. Import schema.sql for a fresh database or back up then use the appropriate migration for an existing database.
3. Configure SMROS_DB_HOST, SMROS_DB_NAME, SMROS_DB_USER, and SMROS_DB_PASS. The code defaults to localhost/smros/root/empty-password for local XAMPP.
4. Install Composer libraries with composer install when needed. For demo/sample seeders, set APP_ENV=development and SMROS_DB_NAME=smros_dev; seeders also verify the connected DB. Never run these against production.
5. Set SMTP values only in protected local Apache configuration when testing real email. Without SMTP, the portals remain usable but outbound emails fail.

Documented local PHP CLI observation: 8.1.25. Development sample data should not be copied wholesale into a production database.

18 · PRODUCTION DEPLOYMENT

18. Production deployment requirements

1. Use the hosting portal/cPanel Domains page to confirm the intended domain document root. Upload the complete application tree without an extra nested project folder; include the matching Composer vendor/ files or run Composer on the host.
2. Use cPanel MultiPHP to select PHP 8.0+ (PHP 8.1/8.2 is appropriate) and verify extensions, especially pdo_mysql, fileinfo, and openssl. Configure upload limits and PHP-FPM/runtime values for the application.
3. Create a production database and database user in cPanel MySQL Database Wizard. Import schema/migrations only after a backup and inspect the target schema. Set exact hosting-prefixed credentials in the PHP/server environment, not source defaults.
4. Set the public HTTPS SMROS_BASE_URL, configure valid TLS, make PHP recognize HTTPS, and set SMROS_REQUIRE_HTTPS=1. Do not enable development mode or sample/demo data.
5. Set SMTP variables in protected server configuration and verify using admin SMTP diagnostics. Keep credentials outside document root and do not put real passwords in .htaccess unless host policy specifically secures it.
6. Upload the local meal images into the matching uploads/meals/ path and ensure production meal rows reference the corresponding filenames. The local development DB update does not update production.
7. Set ownership/permissions so PHP can write only required upload and session/log locations; test login, student/admin access, order/stock flow, mail, reports, and backups after deployment.

PHP 8.2/PHP-FPM appears in the user-provided cPanel version screenshot for the system domain; hosting availability and the live database/application configuration were not verified during this documentation review.

19 · BACKUP AND RECOVERY

19. Backup and recovery

The admin portal provides a database JSON download and a merge-restore workflow. The restore accepts SMROS JSON backups up to 25 MB, remaps related IDs, inserts missing records, leaves existing records untouched, and runs transactionally. A downloaded backup includes sensitive

account hashes and password-reset data; store it securely. It contains database records only, not PHP files or meal images.

Back up the database and uploads/meals/ separately using the hosting provider’s tools or a tested operational process. Validate restores against a compatible schema in a non-production environment first. No automated scheduled backup service or recovery-time/point objective is defined by the project.

20. Testing and maintenance

- The repository provides PHP syntax lint guidance (php -l) and CLI seeders that verify selected foreign-key relationships, sample order totals, and status coverage.
- No dedicated automated unit/integration test suite or CI workflow was identified in the project source tree reviewed.
- For release validation, manually test student/admin login and role guards, registration, profile/password update, meal/category CRUD, image upload/replace/remove, search/filter, order placement/cancellation/stock restoration, slot capacity, all status transitions, email success/failure/retry, password reset, CSV reports, and backup/merge restore.
- Database verification for this revision queried all 12 table row counts and schema metadata, confirmed no orphan order/item/category relationships, checked all order totals against line items, and checked development meal image-path coverage.
- After changes, lint PHP files, review cPanel/PHP logs, verify migration state and permissions, and keep a fresh backup before schema changes.