**Developer Guide: Main Inventory System**

**Table of Contents**

1. Introduction
2. Repository Structure
3. Database and Schema
4. Backend (PHP) Architecture
5. Frontend (HTML/JS) Architecture
6. Authentication & Authorization
7. Validation and Business Logic
8. Reporting & Views
9. Security Practices
10. Extending the System
11. Testing and Debugging
12. Deployment
13. Troubleshooting and Maintenance

**1. Introduction**

This guide is for developers maintaining or extending the Main Inventory System. The system tracks all IT and lab assets, their status, composite setups, employees, and storage, and provides reporting via DB views. The system is implemented in PHP (backend) and HTML/JavaScript (frontend) on a MariaDB database.

**2. Repository Structure**

/backend/ # PHP controllers, models, validation, DB connection

/frontend/ # HTML, CSS, JS, forms, dashboards, authentication views

/sql/ # Schema DDL, view definitions, sample data

/docs/ # Requirements, design, and user documentation

/tests/ # Unit/integration tests for critical logic

/exports/ # Generated CSV and report files (ignored in VCS)

**3. Database and Schema**

* Schema is in /sql/ and matches the provided MariaDB DDL.
* **Tables:**
  + accessories, graphicscards, keyboards, mice, minipc, monitors, motherboards, powersupplies, ramsticks, storage\_components, storage\_slots, employees, pcsetups
* **Views:**
  + component\_totals, disposed\_parts, stored\_components\_storage
* Primary keys are VARCHARs, often with custom prefixes (e.g., kb\_0001).
* pcsetups and storage\_components use FKs to ensure referential integrity.
* Employee emails are unique; passwords are currently simple (better security practice can be placed).

**4. Backend (PHP) Architecture**

* **MVC or Modular Structure:** Separate controllers for each component type, PC setups, employees, and storage.
* **Controllers:**
  + Handle CRUD, validation, business logic.
* **Models:**
  + Database interaction, record mapping.
* **Utilities:**
  + Authentication, CSV export, input validation.

**Example:**  
backend/controllers/KeyboardsController.php handles all keyboard asset logic.

**5. Frontend (HTML/JS) Architecture**

* **Single-page app (SPA) or multi-page forms.**
* **Forms:** For adding/editing assets, PC setups, employees.
* **Tables:** For listing/searching/filtering assets, setups, and employees.
* **Dashboard:** For reporting (component totals, disposed, storage).
* **Authentication UI:** Login/logout, session feedback.

**6. Authentication & Authorization**

* Employees authenticate with employee\_id and password.
* Session management via PHP server-side sessions or JWT if API-based.
* employee\_type governs access:
  + DBA: Full admin (CRUD on all)
  + FullTime/SoftwareDev: Asset management only
* Always authorize on backend, not just frontend.

**7. Validation and Business Logic**

* **Client-side:** Prevent obvious bad input, enforce required fields.
* **Server-side:**
  + Check all required fields, enum values, FKs.
  + Enforce unique IDs (PKs).
  + For setups/storage, ensure all referenced IDs exist.
  + Prevent deletion of assets if referenced in setups.

**8. Reporting & Views**

* Reports query DB views (component\_totals, disposed\_parts, stored\_components\_storage).
* Export as CSV: Backend generates a CSV string, headers force download.
* Reports are read-only; changes must be made via asset management.

**9. Security Practices**

* Use prepared statements for all DB access.
* Sanitize all user input.
* Store passwords (future implementation: hashed (bcrypt/Argon2).

**10. Extending the System**

* **New component:**
  + Create new table, model, controller, and frontend forms/views.
  + Update PC setup logic/reporting if needed.
* **New field:**
  + Add to table and all relevant forms/models/controllers.
* **Composite setups:**
  + Update setup model/controller to allow new component type.
* **Reports:**
  + Add new DB view and reporting UI as needed.

**11. Testing and Debugging**

* Use /tests/ for unit and integration tests.
* Test CRUD, authentication, business logic, and reporting.
* Use sample data in /sql/ for local development.
* Log errors and failed operations on backend.

**12. Deployment**

* Use secure hosting.
* Store DB credentials securely (env/config files, not in repo).
* Regularly update PHP and dependencies.
* Back up database regularly.
* Ensure HTTPS is enabled(If upgraded to internet connection).

**13. Troubleshooting and Maintenance**

* **DB errors:** Check logs, validate schema and FK constraints.
* **Performance:** Optimize queries, add indexes if needed.
* **User issues:** Check role permissions, data integrity.
* **Upgrades:** Merge code, apply DB migrations, test extensively.

**Questions or contributions?**  
See /docs/ for more or contact creator [cortinacort@gmail.com](mailto:cortinacort@gmail.com).