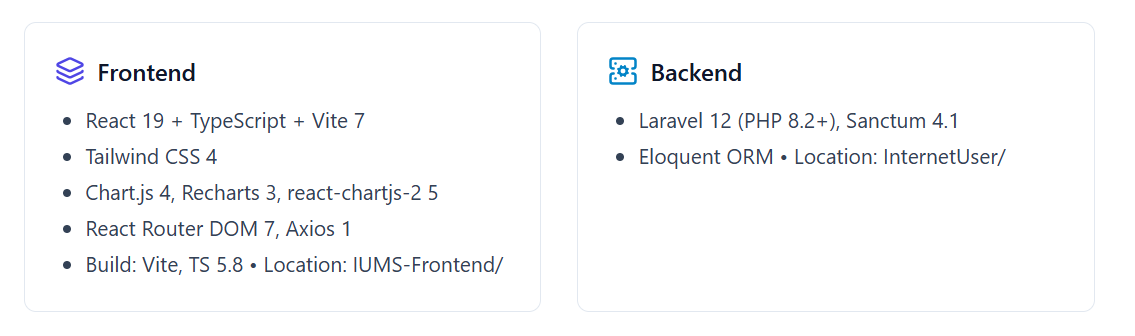
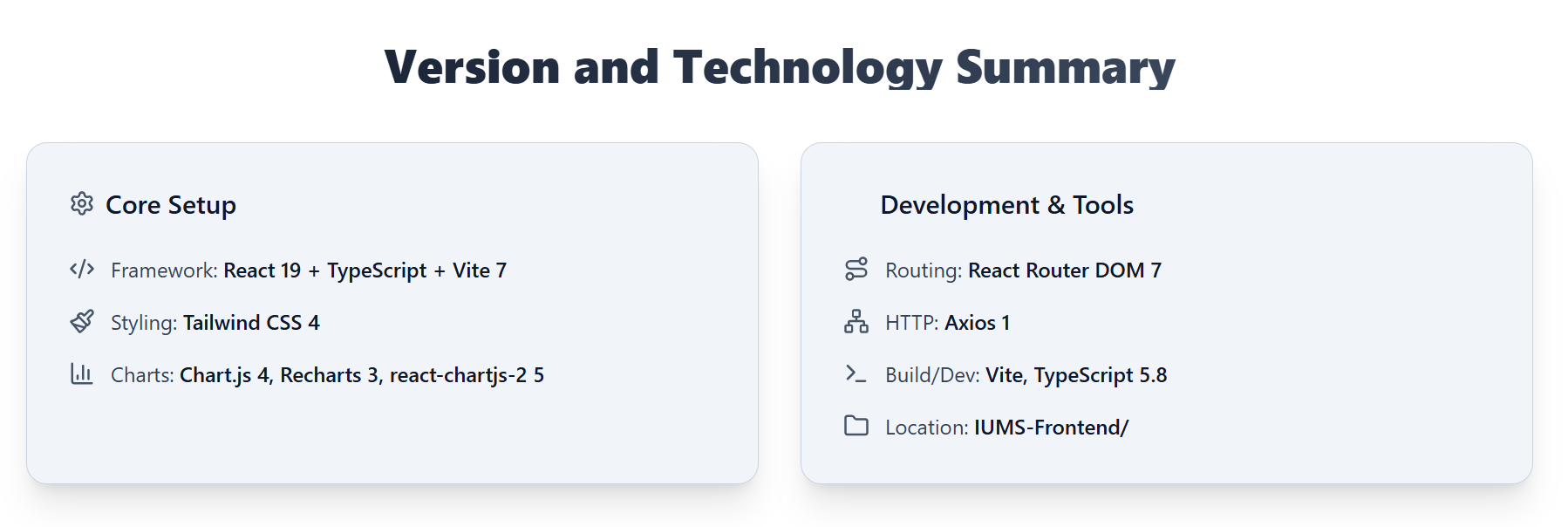


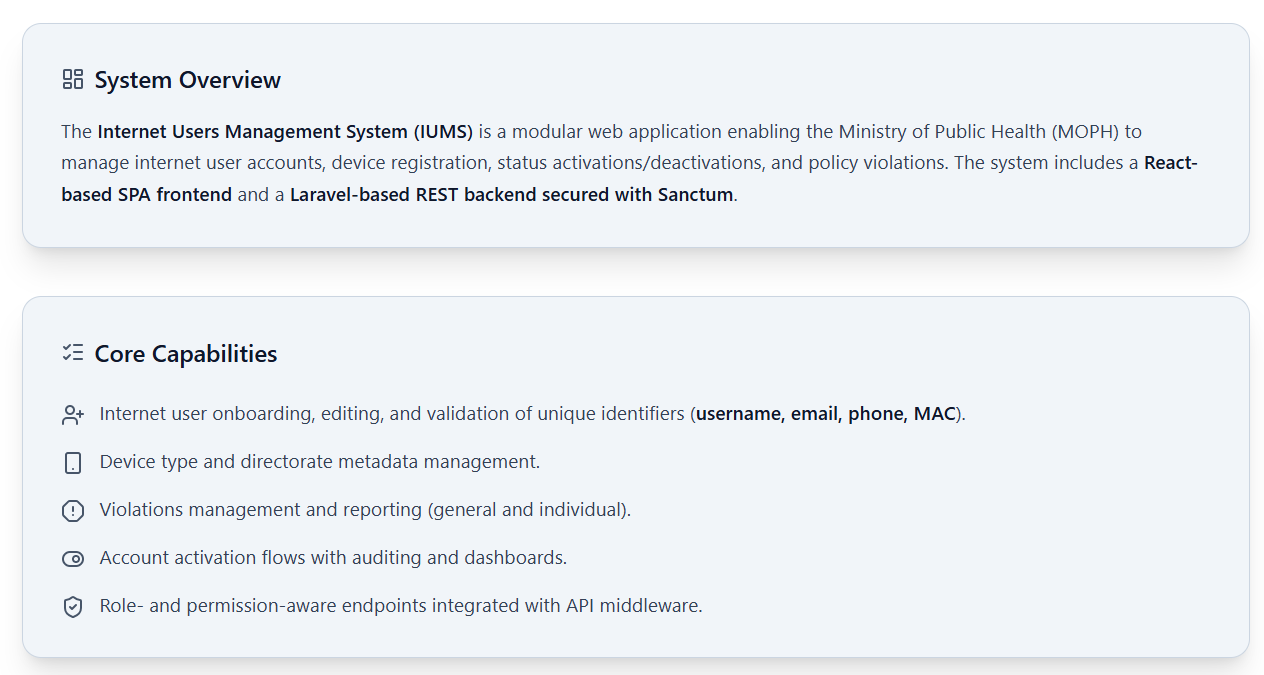
System Documentation



*Fardin Ibrahimi & Nesar Ahmad Awwab*

*Sep 16, 2025*

****



****

* Client (SPA): React app served by Vite, consuming REST endpoints and enforcing client-side routing and guarded routes. Source at IUMS-Frontend/src/.
* API Server: Laravel REST API under InternetUser/routes/api.php with business logic implemented in controllers and models under InternetUser/app/.
* Authentication: Token-based via Laravel Sanctum. Logged-in sessions guard protected routes and operations.
* Database: Managed by Eloquent models (InternetUser, Person, DeviceType, Directorate, EmploymentType, Violation, ViolationsType, AccountActivation, Group, Role, Permission).

****

* Frontend
* IUMS-Frontend/src/App.tsx — Top-level routing configuration and page composition.
* IUMS-Frontend/src/main.tsx — Application bootstrap and mount point.
* IUMS-Frontend/src/components/ — Reusable UI components (tables, charts, sidebar, etc.).
* IUMS-Frontend/src/internetUsers/ — Feature pages and flows for user CRUD, reactivation, violations, reports, and step-by-step forms.
* IUMS-Frontend/src/systemUsers/ — Auth forms and protected route utilities.
* IUMS-Frontend/src/site/ — Public pages (Home, About, Contact) and layout components (Navbar, Footer, Hero).
* IUMS-Frontend/src/Auth/AuthContext.tsx — Authentication context (session state, role/permission checks).
* IUMS-Frontend/src/config.ts — API base config.
* Backend
* InternetUser/routes/api.php — All REST endpoints for auth, internet users, groups, violations, reports, device types, and account activation.
* InternetUser/app/Http/Controllers/ — Controllers for API domains (InternetUserController, ViolationController, AuthController, etc.).
* InternetUser/app/Models/ — Eloquent models (InternetUser.php, Violation.php, ViolationsType.php, DeviceType.php, Directorate.php, EmploymentType.php, Group.php, AccountActivation.php).
* InternetUser/config/auth.php — Sanctum and guard configuration.
* InternetUser/bootstrap/app.php — App bootstrap.
* InternetUser/composer.json — Backend dependencies.

****

* User Management
* Create, update, list, and delete internet users via endpoints and UI screens:
* Frontend screens: IUMS-Frontend/src/internetUsers/AllUsers.tsx, editModal.tsx, AddInternetUsers.tsx, steps/step1.tsx–step4.tsx
* API: GET/POST/PUT/DELETE /api/internet; GET /api/internet-user-edit/{id}
* Status controls:
* API: PUT /api/users/{id}/status for activation/deactivation
* UI: IUMS-Frontend/src/internetUsers/reActivation.tsx and AllReactivations.tsx
* Uniqueness validations:
* API: /api/check-username, /api/check-email-of-internet-users, /api/check-phone-of-internet-user, /api/check-mac-address
* Violations Management
* Violation Types: CRUD of types used for classifying violations
* API: GET/POST/PUT/DELETE /api/violation
* UI: IUMS-Frontend/src/internetUsers/addviolationType.tsx, allviolationtypes.tsx
* Violations on Users:
  + API: POST /api/violationOnaUser, GET /api/allViolationsFromUsers, GET /api/getSpecifiedUserForViolation
* UI: IUMS-Frontend/src/internetUsers/addViolationOnaUser.tsx, AllViolationsFromUsers.tsx, employeeViolationForm.tsx
* Important Data Model Note:
  + The backend `Violation` model (InternetUser/app/Models/Violation.php) exposes fillable fields only for `internet\_user\_id` and `comment`. It does not contain a `violation\_type\_id` nor fields for violation date or count. `ViolationsType` is related via a hasMany relationship, but not persisted directly on `Violation`. Frontend submission and display must align with this backend shape unless the backend is extended. (See “Known Limitations & Alignment Considerations”.)
* Reporting
* General and individual reports:
  + API: GET /api/reports/general, GET /api/reports/individual
* UI: IUMS-Frontend/src/internetUsers/reports.tsx, dashboard.tsx; charts at IUMS-Frontend/src/components/deputyMinistrySummaryChart.tsx, groupTypePieChart.tsx
* Reference Data Management
* Employment Type, Directorate, Device Type:
  + API: GET /api/employment-type, GET /api/directorate, GET /api/device-types, GET /api/group-count
* UI components integrate these via dropdowns/searchable combos (e.g., IUMS-Frontend/src/components/APISearchableComboBox.tsx, ApiDropDown.tsx)
* Authentication and Authorization
* Auth endpoints:
  + POST /api/login, POST /api/logout, POST /api/register
  + GET /api/profile, PUT /api/update-profile/{id}
  + GET /api/user for system users administration
* Middleware:
  + Routes are protected with Sanctum and custom middleware: e.g., check.access:<PermissionName> on sensitive endpoints (create/update/delete)
* Frontend guard utilities:
  + IUMS-Frontend/src/components/AdminRoute.tsx and PrivateRoute.tsx
  + IUMS-Frontend/src/components/RoleChecker.tsx
* Account Activation
  + API: POST /api/account/activate, GET /api/all-reactivation, GET/PUT/DELETE /api/account-activation/{id}
  + UI: IUMS-Frontend/src/internetUsers/reActivation.tsx, AllReactivations.tsx

****

* Entry and Composition
* main.tsx initializes React and mounts the app.
* App.tsx defines the routing map for public and private sections, composing:
  + Public site pages: IUMS-Frontend/src/site/Home.tsx, About.tsx, Contact.tsx
  + System users pages: LoginForm.tsx, RegisterForm.tsx, Settings.tsx, SystemUsersPage.tsx
  + Internet users module pages: dashboard.tsx, AllUsers.tsx, AddInternetUsers.tsx, reActivation.tsx, reports.tsx, violations pages
* UI/UX Components
* Navigation and layout: IUMS-Frontend/src/site/SiteNavbar.tsx, SiteFooter.tsx, SiteHero.tsx, components/Sidebar.tsx, components/Header.tsx
* Feedback utils: components/Spinner.tsx, components/ProgressBar.tsx, components/scrollToTop.tsx
* Data displays: components/userRow.tsx, APISearchableComboBox.tsx, react-select usage in forms
* Charts: components/deputyMinistrySummaryChart.tsx, groupTypePieChart.tsx using Chart.js/Recharts
* Forms and Flows
* Multi-step user creation: IUMS-Frontend/src/internetUsers/steps/step1.tsx–step4.tsx, shared inputs in steps/InputField.tsx, steps/selectfield.tsx
* Modals: IUMS-Frontend/src/internetUsers/editModal.tsx
* Auth and Access Control
* AuthContext.tsx centralizes auth state and exposes utilities to guards: AdminRoute.tsx, PrivateRoute.tsx, RoleChecker.tsx
* Styling
* TailwindCSS 4 used via @tailwindcss/vite plugin, styles at IUMS-Frontend/src/index.css and utility classes inline

****

* Routing
* InternetUser/routes/api.php — All protected API endpoints, wrapped in Route::middleware('auth:sanctum') where needed and custom permission checks (check.access).
* InternetUser/routes/web.php — Minimal web route returning welcome view (SPA is separate).
* Controllers (InternetUser/app/Http/Controllers/)
* AuthController.php — Login, logout, register, profile, system user admin, email checks.
* api/app/InternetUser/InternetUserController — Users CRUD, validations, status updates, counts, reports, forms.
* api/app/Violation/ViolationController — Store violations on users, list violations, fetch specific user data for the violation form.
* api/app/Violation/ViolationTypeController — Manage violation types.
* api/app/Directorate/DirectorateController, api/app/EmploymentType/EmploymentTypeController, api/app/Device\_type/DeviceTypeController — Reference data APIs.
* api/app/Account/AccountActivationController — Account activation lifecycle and listing.
* Api/Group/GroupController — Group listing and aggregated counts.
* Models (InternetUser/app/Models/)
  + InternetUser.php — Domain model for internet users.
  + Person.php — Person record linked to users.
  + DeviceType.php, Directorate.php, DirectorateType.php, EmploymentType.php — Reference tables.
  + AccountActivation.php — Activation record tracking.
  + Group.php — Group entity supporting counts by type.
  + Role.php, permission.php, RolePermission.php — RBAC structures.
  + Violation.php — User violations entity.
  + ViolationsType.php — Master list of violation types.



* Auth and System Users
* POST /api/login
* POST /api/logout
* POST /api/register (check.access: CreateUsers)
* GET /api/profile (check.access: ViewUsers)
* GET /api/user
* PUT /api/user/{id} (check.access: UpdateSystemData)
* DELETE /api/user/{id} (check.access: DeleteSystemData)
* PUT /api/update-profile/{id} (check.access: UpdateUsers)
* POST /api/check-email
* Internet Users and Validations
* GET /api/internet
* POST /api/internet (check.access: AddSystemData)
* PUT /api/internet/{id} (check.access: UpdateSystemData)
* DELETE /api/internet/{id} (check.access: DeleteSystemData)
* GET /api/internet-user-edit/{id} (check.access: UpdateSystemData)
* POST /api/check-username
* POST /api/check-email-of-internet-users
* POST /api/check-phone-of-internet-user
* POST /api/check-mac-address
* PUT /api/users/{id}/status (check.access: UpdateSystemData)
* GET /api/total-users
* Reference Data and Groups
* GET /api/employment-type
* GET /api/directorate
* GET /api/device-types
* GET /api/groups
* GET /api/group-count
* GET /api/employment-type-counts
* Violations
* GET /api/violation
* POST /api/violation (check.access: AddSystemData)
* PUT /api/violation/{id} (check.access: UpdateSystemData)
* DELETE /api/violation/{id} (check.access: DeleteSystemData)
* POST /api/violationOnaUser (check.access: AddSystemData)
* GET /api/allViolationsFromUsers
* GET /api/getSpecifiedUserForViolation
* GET /api/violation-form
* Activation
* POST /api/account/activate (check.access: AddSystemData)
* GET /api/all-reactivation
* GET /api/account-activation/{id}/edit
* PUT /api/account-activation/{id} (check.access: UpdateSystemData)
* DELETE /api/account-activation/{id} (check.access: DeleteSystemData)
* GET /api/internet-users-deactivated
* Reports
* GET /api/reports/general
* GET /api/reports/individual



* Authentication
* Sanctum token authentication protects all operational endpoints in a Route::middleware('auth:sanctum') group.
* Authorization
* Fine-grained access enforced via custom middleware check.access:<PermissionName>, e.g., CreateUsers, UpdateSystemData, DeleteSystemData, ViewUsers.
* Frontend Guarding
* PrivateRoute.tsx and AdminRoute.tsx enforce route-level gating. RoleChecker.tsx provides conditional rendering based on roles/permissions.
* Sensitive Operations
* Create/Update/Delete on users, violation types, and account activation are permission-gated.

****

* InternetUser (InternetUser/app/Models/InternetUser.php)
* Core user profile, device, and status information. Used across CRUD, activation, reporting.
* Violation and ViolationsType
* ViolationsType is a catalog of possible violations.
* Violation records are stored with fields `internet\_user\_id` and `comment` only in the base model. There is no persisted foreign key to `ViolationsType` by default.
* Implication: Frontend should submit `internet\_user\_id` and `comment`, and if a violation “type” is needed for analytics or display, it must be derived via related collections or additional backend support.
* AccountActivation
* Tracks activation events for audit and dashboard reporting.
* Reference Entities
* DeviceType, Directorate, EmploymentType, Group underpin dropdowns and aggregated reports.



****

* Frontend
* Axios interceptors (if configured) handle 401/403 flows; components like AccessDenied.tsx provide UX feedback for unauthorized actions.
* Forms validate required inputs and uniqueness via API endpoints.
* Backend
* Controllers validate request payloads and return standardized JSON responses.
* Sanctum and middleware guard sensitive endpoints; invalid tokens yield 401/403.

****

* SPA with Vite delivers highly optimized static assets and HMR in dev for rapid iteration.
* Charting libraries are lazy-utilized in analytics views.
* Use pagination for GET /api/internet and aggregated queries as needed.
* Consider database indexing on high-cardinality fields (username, email, MAC, group\_id) for scale.

****

* Consistent layout with `SiteNavbar`, `SiteFooter`, and `Sidebar`.
* Clear user feedback with `Spinner`, `ProgressBar`, and form validation messaging.
* Charts complement tabular data, aiding diverse consumption preferences.

****

* Backend
* PHPUnit configured (phpunit/phpunit ^11.5) for unit and feature tests.
* Recommended: cover authentication, permission middleware, user CRUD, activation flows, and violations endpoints.
* Frontend
* Recommended: add React Testing Library and Vitest for component and integration tests around forms, routers, and guards.

****

* Frontend
* Build artifacts in IUMS-Frontend/dist/ can be served via Nginx/Apache or a CDN.
* Backend
* Laravel deployed to PHP 8.2 environment with proper web server configuration.
* Ensure Sanctum stateful domains and CORS configured for SPA domain.
* Queue workers and cache optimization recommended for production use.

****

* Backend uses Laravel logging (Monolog) configured via .env LOG\_CHANNEL.
* composer dev script includes pail for interactive logs during development.
* Consider centralized logging and alerting for production.

****

* Violations Data Model
* The backend Violation model supports only `internet\_user\_id` and `comment` as fillable. There is no `violation\_type\_id`, `date`, or `count` per record stored natively. The ViolationsType model relates via hasMany but is not referenced by Violation with a FK field.
* Frontend must:
* Submit violations with only these fields unless backend is extended.
* Display violation “type” by joining inferred relationships or by enhancing backend to include a type FK and related fields.
* If richer reporting is needed (types, timestamps per violation, severity), extend the backend schema and serialization accordingly.
* Permissions Granularity
* Ensure the check.access middleware’s permission names match the UI checks to prevent UX mismatches.

****

* Enhance Violation schema to include:
  + violation\_type\_id (FK), occurred\_at (timestamp), severity, and optionally created\_by (auditing).
* Add pagination and filtering for all list endpoints by default.
* Introduce OpenAPI/Swagger documentation generated from routes/controllers.
* Add E2E tests covering critical flows (login, create user, activate account, add violation, generate report).
* Internationalization (i18n) for frontend forms and labels.

****

* First-Time Setup (Local)
  + Backend: composer install → .env → key:generate → migrate → artisan serve
  + Frontend: npm install → set API base in config.ts → npm run dev
  + Login via POST /api/login and obtain Sanctum token, or use the UI LoginForm.tsx.
* Common Maintenance
  + Rotate Sanctum tokens for compromised sessions.
  + Back up DB before schema changes.
  + Monitor logs for 401/403 spikes indicating permission misconfiguration.

**Support and Ownership**

* Primary Owners: MOPH Software Development Team
* Code Repositories:
  + Frontend: https://github.com/fiafghan/IUMS-Frontend.git
  + Backend: https://github.com/Nisarahmad0011/InternetUser.git
* Issue Tracking
  + Track defects and features aligned with the endpoints and UI screens named in this document.
  + Include endpoint and component file references for reproducibility.

**Appendix**: Notable Frontend Screens and Components (by path)

* Site
  + IUMS-Frontend/src/site/Home.tsx, About.tsx, Contact.tsx, SiteNavbar.tsx, SiteFooter.tsx, SiteHero.tsx
* Internet Users
* IUMS-Frontend/src/internetUsers/AllUsers.tsx
* IUMS-Frontend/src/internetUsers/editModal.tsx
* IUMS-Frontend/src/internetUsers/AddInternetUsers.tsx
* IUMS-Frontend/src/internetUsers/steps/step1.tsx–step4.tsx
* IUMS-Frontend/src/internetUsers/reActivation.tsx, AllReactivations.tsx
* IUMS-Frontend/src/internetUsers/reports.tsx, dashboard.tsx
* IUMS-Frontend/src/internetUsers/addViolationOnaUser.tsx, AllViolationsFromUsers.tsx, employeeViolationForm.tsx
* IUMS-Frontend/src/internetUsers/addviolationType.tsx, allviolationtypes.tsx
* Shared Components
  + IUMS-Frontend/src/components/Sidebar.tsx, Header.tsx, UserFilters.tsx, userRow.tsx
  + IUMS-Frontend/src/components/APISearchableComboBox.tsx, ApiDropDown.tsx
  + IUMS-Frontend/src/components/deputyMinistrySummaryChart.tsx, groupTypePieChart.tsx
  + IUMS-Frontend/src/components/AdminRoute.tsx, PrivateRoute.tsx, RoleChecker.tsx
  + IUMS-Frontend/src/components/Spinner.tsx, ProgressBar.tsx, scrollToTop.tsx

**Appendix**: Backend Controllers and Models (by path)

* Controllers
  + InternetUser/app/Http/Controllers/AuthController.php
  + InternetUser/app/Http/Controllers/api/app/InternetUser/InternetUserController.php
  + InternetUser/app/Http/Controllers/api/app/Violation/ViolationController.php
  + InternetUser/app/Http/Controllers/api/app/Violation/ViolationTypeController.php
  + InternetUser/app/Http/Controllers/api/app/Directorate/DirectorateController.php
  + InternetUser/app/Http/Controllers/api/app/EmploymentType/EmploymentTypeController.php
  + InternetUser/app/Http/Controllers/api/app/Device\_type/DeviceTypeController.php
  + InternetUser/app/Http/Controllers/api/Group/GroupController.php
  + InternetUser/app/Http/Controllers/api/app/Account/AccountActivationController.php
* Models
  + InternetUser/app/Models/InternetUser.php
  + InternetUser/app/Models/Person.php
  + InternetUser/app/Models/Violation.php
  + InternetUser/app/Models/ViolationsType.php
  + InternetUser/app/Models/DeviceType.php
  + InternetUser/app/Models/Directorate.php, DirectorateType.php
  + InternetUser/app/Models/EmploymentType.php
  + InternetUser/app/Models/AccountActivation.php
  + InternetUser/app/Models/Group.php, Role.php, permission.php, RolePermission.php

**Credits & Acknowledgements**

The Internet Users Management System (IUMS) was developed primarily by **Fardin Ibrahimi** and **Nesar Ahmad Awwab**, software developers of the **Ministry of Public Health's Information Technology Directorate,** who generated this documentation. **Copyright 2025**. All rights granted.   
We are grateful for the assistance and direction provided by our stakeholders and team members, who indirectly helped to establish this system.