1. Project Overview — ARDA

1.1 Introduction

ARDA (Employee Central Hub) is an enterprise-grade internal organizational management platform designed to centralize workflow automation, task execution, hierarchical supervision, and structured communication across an organization.

The system enforces role-based operational control while maintaining departmental autonomy. It integrates workflow-driven ticketing, structured project execution, cross-department coordination, performance monitoring, and contextual communication into a unified system.

ARDA is built around traceability, accountability, and hierarchical governance.

---

2. Role-Based Hierarchical Structure

ARDA operates on a strict role-based permission architecture.

There are **six user types**:

1. Employee
2. Department Head
3. HR Employee
4. HR Head
5. Executive
6. Admin

2.1 Employees

Employees are standard operational users.

Responsibilities:

- Work on assigned tickets
- Execute project deliverables
- Complete sprint actions
- Participate in contextual chats
- Comment on department announcements
- Schedule and accept appointments
- Manage personal calendar entries

Employees cannot:

- Create workflows
- Assign departmental tasks
- Access employee monitoring dashboards

2.2 Department Heads

Department Heads are functional leaders of a single department.

Permissions:

- Create department-level workflows
- Assign employees within workflows
- Create and manage projects
- Create and manage sprints
- Assign deliverables and actions
- Create department announcements (with comments enabled)
- Monitor employee logs within their department

Workflow Limitation:
Department Heads can only create workflows involving employees from their own department.

2.3 HR Employee

HR Employees function similarly to standard employees but with extended communication privileges.

Additional Permission:

- Create organizational announcements (view-only, no comments)

HR Employees do not create workflows unless assigned HR Head privileges.

2.4 HR Head

HR Head is the functional leader of the HR department.

Permissions:

- All HR Employee permissions
- Create HR department workflows
- Assign HR tasks and projects
- Monitor HR employee logs
- Create organization-wide or department-specific announcements (view-only)

Workflow Limitation:
Can only create workflows within the HR department.

2.5 Executive

Executives are multi-department functional heads.

Permissions:

- Create workflows for multiple departments under their authority
- Assign projects and sprints (within scope)
- Monitor employee logs across supervised departments
- View workload distribution and health metrics

Workflow Limitation:
Executives can only create workflows among employees in departments under their control.

Executives cannot create unrestricted organization-wide workflows.

2.6 Admin

Admins operate at the system governance level.

2.6.1 User Management (Full CRUD)

Admins can:

- Create users
- Update user information
- Change roles and titles
- Assign or modify departments
- Approve users
- Reset/change passwords
- Delete users
- Modify role assignments

Admins maintain full control over user lifecycle management.

2.6.2 Super Workflow System

Admins can create **Super Workflows**.

Super Workflows:

- Can include employees from any department
- Allow cross-department ticket routing
- Override normal departmental workflow restrictions
- Enable organization-wide process flows

This allows structured inter-department collaboration.

---

3. Workflow Architecture

ARDA supports two workflow types:

1. Departmental Workflows
2. Super Workflows

3.1 Departmental Workflows

Created by:

- Department Heads
- HR Head
- Executives (within scope)

Characteristics:

- Restricted to employees within authorized departments
- Define sequential employee routing
- Reusable across multiple tickets

3.2 Super Workflows

Created only by:

- Admin

Characteristics:

- Cross-department routing
- Organization-wide employee selection
- Enables multi-department collaboration
- Used for high-level or inter-department processes
- Allows for viewing to be set so only selective people can make tickets on super workflows

---

3.3 Custom Workflow Forms

Each workflow (including Super Workflows) supports **customizable form configuration**.

When creating a workflow, the creator can define:

- Custom input fields
- Field types (text, select, date, etc.)
- Required vs optional fields
- Structured submission format

When a ticket is created:

- The user must fill the workflow-specific custom form
- Ticket data structure depends on its assigned workflow
- Ensures structured and process-specific ticket information

This allows dynamic ticket schemas depending on process type.

4. Core Functional Modules

4.1 Ticketing System

The ticketing system is workflow-driven.

Flow:

1. User selects workflow
2. Custom workflow form is presented
3. User submits ticket
4. Ticket enters defined workflow sequence
5. Moves step-by-step across assigned employees

Tickets strictly follow the assigned workflow order.

Ensures:

- Accountability
- Structured approvals
- Traceable processing

4.2 Projects System

Projects represent structured long-term initiatives.

Characteristics:

- Created by Functional Heads
- Divided into Deliverables

- Deliverables contain execution-level tasks
- Dedicated contextual chat
- Synced to stakeholder calendars
- Includes health monitoring

4.3 Sprints System

Sprints represent execution cycles.

Characteristics:

- Divided into Actions
- Automatically start with a default action
- Additional actions can be added
- Dedicated contextual chat
- Synced to participant calendars
- Includes health monitoring

Sprints treat the sprint itself as an action, ensuring consistent execution modeling.

---

5. Health Monitoring System

Projects and Sprints include a Health Indicator mechanism.

Health tracking evaluates:

- Completion percentage
- Delay risk
- Workload pressure

- Execution efficiency

Functional Heads and Executives can monitor:

- Number of active tickets per employee
- Active projects
- Sprint assignments
- Health of assigned work units

This enables early detection of overload or operational bottlenecks.

---

6. Announcement System

6.1 Department Announcements

- Created by Department Heads or Executives
- Visible within department
- Comments enabled

6.2 Organizational Announcements

- Created by HR Employee or HR Head
- Can target entire organization or selected departments
- View-only (no comments)

This separation ensures structured communication governance.

---

7. Appointment System

Employees can request appointments with one or multiple employees.

Characteristics:

- Requires acceptance from invitees
- Supports multi-stakeholder scheduling
- Automatically synced to calendars

---

8. Calendar System

Each employee has a unified calendar.

Manual Entries:

- Personal tasks
- Personal notes
- Day Canavs

Automatically Synced:

- Projects
- Sprints
- Accepted Appointments

Provides a consolidated operational view.

---

9. Contextual Chat System

Projects and Sprints include isolated chat environments.

Purpose:

- Preserve discussion context
- Maintain execution-level communication

- Avoid cross-module communication clutter

Chats are strictly tied to their parent entity.

---

10. System Design Principles

ARDA is built on:

1. Hierarchical governance
2. Departmental autonomy
3. Workflow-enforced accountability
4. Cross-department coordination via Super Workflows
5. Contextual communication isolation
6. Calendar-synchronized operational visibility
7. Health-based performance monitoring

11. Codebase Overview

11.1 Architectural Pattern

ARDA is built using **Next.js App Router architecture** with a strongly structured, role-based view separation model.

The application follows these architectural principles:

- Single entry page per role
- Role-based dashboard segregation
- Heavy component composition inside limited route files
- Context-driven UI consistency
- Modular feature grouping inside `/app/components`

The system intentionally minimizes the number of route files and centralizes rendering logic within structured component hierarchies.

11.2 Page Structure Strategy

ARDA contains only **7 `page.tsx` files**:

1. Auth (Login / Forgot Password)
2. Employee
3. Department Head
4. HR Employee
5. HR Head
6. Executive

7. Admin

All business logic and UI rendering are composed inside these pages through nested components.

This ensures:

- Controlled entry points
- Predictable routing
- Centralized layout injection
- Easier debugging of view-level issues

11.3 App Router Structure

```
app/
│
├── (auth)/
│   └── login/
│       ├── components/
│       ├── page.tsx
│       └── layout.tsx
│
├── (Dashboard)/
│   ├── admin/
│   ├── dept-head/
│   ├── employee/
│   ├── executive/
│   ├── hr-employee/
│   ├── hr-head/
│   └── layout.tsx
│
└── api/
```

(auth)

Handles:

- Login
- Forgot password
- Authentication logic
- Layout isolation for auth views

(Dashboard)

Handles:

- Role-based dashboards
- Layout wrapping
- Sidebar + Header injection
- View rendering per user type

Each role folder represents a controlled view environment.

11.4 Role-Based View Composition

Inside each role folder (e.g., `admin`, `dept-head`, `employee`):

- `page.tsx` acts as the entry renderer.
- A `components` folder holds:
    - Sidebar
    - Header
    - HomeContent (widgets etc.)
    - TeamContent (for heads/executives/admin)
    - AppointmentView wrapper

    - CalendarView wrapper
    - Other role-specific content

Important Pattern

Example:

```
HomeContent/
   ├── SectionA.tsx
   ├── SectionB.tsx
   └── index.tsx (compiled aggregator)
```

The `HomeContent` folder contains multiple subcomponents which are composed inside a main `HomeContent` file, which is then imported and rendered inside `page.tsx`.

This pattern is repeated for:

- TeamContent
- AppointmentView
- CalendarView
- HRManageUsersContent
- Etc.

This approach keeps `page.tsx` clean and minimal.

11.5 Employee Logs & Team Views

For:

- Department Heads
- HR Head
- Executive

- Admin

A **TeamContent** folder exists.

This handles:

- Employee logs
- Active tickets count
- Projects assigned
- Sprint involvement
- Health monitoring visibility

Only supervisory roles have access to these views.

11.6 Components Directory (Global Logic Layer)

`app/components/`

This folder contains feature-level components grouped by functionality:

Examples:

- appointments
- calendars
- DeptTickets
- ProjectManagement
- OrgAnnouncements
- ticketing
- WorkflowComponents

- ManageUsersContent
- HRDownloadLogs
- universal/
- super/
- layout/

Each folder name reflects its domain responsibility.

This is the primary business logic layer of the frontend.

11.6.1 Project Management Refactor

Legacy folders:

- `depthead-project`
- `employee-project`

These are deprecated.

All project-related logic is now consolidated under:

```
ProjectManagement/
  ├── depthead/
  ├── employee/
  ├── shared/
```

This centralizes:

- Project logic
- Sprint logic
- Deliverables
- Actions

- Health calculations

Developers should not use legacy project folders.

11.7 Workflow Components

Located under:

`universal/WorkflowComponents`

These handle:

- Workflow creation UI
- Form customization logic
- Ticket form rendering
- Dynamic field configuration

These components support:

- Departmental Workflows

The logic for super workflows is handled in the super folder.

Form configuration is dynamic and workflow-specific.

11.8 Context Architecture

`app/context/ThemeContext.tsx`

The `ThemeContext` is critical to the UI architecture.

It ensures:

- Consistent theming across all components
- Dark/light mode synchronization
- Centralized style logic
- Uniform UI state propagation

All UI components rely on this context for styling consistency.

`ThemeProvider.tsx` exists but is considered legacy.

Developers should use `ThemeContext` moving forward.

11.9 Layout Injection Pattern

Each major route has:

- A `layout.tsx`
- Role-specific headers
- Role-specific sidebars

This ensures:

- Structural consistency
- Minimal duplication
- Role isolation

The Dashboard layout wraps all role pages.

11.10 API Layer

`app/api/`

Handles:

- Authentication endpoints
- Ticket operations
- Workflow creation
- Project/sprint CRUD
- Appointment handling
- Announcement operations
- User management

API routes correspond to backend operations and enforce permission checks based on role.

11.11 Design Conventions

1. Page files are thin controllers.
2. Business logic lives inside components.
3. UI grouping follows feature-domain naming.
4. Role segregation is directory-based.
5. Context is used for global UI consistency.
6. Legacy folders are preserved but not used.
7. Component folders reflect domain logic clearly.

11.12 Debugging Strategy for Developers

When debugging:

Authentication issues

→ Check `(auth)` folder and API auth routes.

Role rendering issues

→ Check specific role folder inside `(Dashboard)`.

Project/Sprint issues

→ Check `ProjectManagement/`.

Workflow or ticket issues

→ Check `ticketing/` and `WorkflowComponents/`.

Theme/UI inconsistencies

→ Check `ThemeContext.tsx`.

Employee log issues

→ Check `TeamContent` inside relevant role.

11.13 Architectural Summary

ARDA's frontend architecture is:

- Role-segregated
- Component-composed
- Context-driven
- Domain-grouped
- Minimal page-entry based

The system intentionally limits route files and pushes complexity into structured, reusable component modules.

12. Database Architecture Overview

12.1 Architecture Philosophy

ARDA uses a **schema-defined MongoDB structure**, where all active collection schemas are defined inside the `/models` directory.

The models folder acts as the authoritative source for:

- Collection structure
- Field definitions
- References
- Business constraints
- Role-based relationships

MongoDB itself does not enforce rigid schemas beyond what is defined in these models.

12.2 Model Directory Overview

```
models/
|
├── ProjectManagement/
|   ├── Project.ts
|   ├── Sprint.ts
|
├── Announcement.js
├── Appointment.js
├── ExecutiveDepartments.ts
├── FormData.ts
├── Functionality.ts
├── OrgAnnouncement.js
├── SuperFunctionality.ts
├── Task.js (legacy)
├── Ticket.ts
├── TimeIntents.js
├── User.ts
├── UserPreference.js
├── Project.js (legacy)
├── Sprint.js (legacy)
├── CalendarEvent.js (legacy)
```

12.3 Active Core Collections

12.3.1 User (`User.ts`)

Represents authentication identity and role mapping.

Responsibilities:

- Authentication credentials
- Role type
- Department reference
- Approval state

This model governs access control logic.

12.3.2 FormData (`FormData.ts`)

⚠ Important: Despite its name, this stores employee profile data.

Contains:

- Employee details
- Role metadata
- Department mapping
- Possibly workflow assignment data

This acts as the operational employee information store.

12.3.3 ExecutiveDepartments (`ExecutiveDepartments.ts`)

Defines supervisory mapping:

Executive → Multiple Departments

Used to:

- Determine cross-department authority
- Filter accessible employee logs
- Restrict workflow creation scope

This model is critical for permission enforcement.

12.3.4 Ticket (`Ticket.ts`)

Represents workflow-driven request entities.

Contains:

- Workflow reference
- Current stage index
- Assigned employee
- Creator reference
- Status
- Submitted form data

Ticket structure is partially dynamic because form fields depend on workflow configuration.

12.3.5 Workflow Models

Functionality.ts

Represents Departmental Workflows.

Contains:

- Department reference
- Ordered employee sequence
- Custom form structure
- Workflow metadata

Restricted to department-level scope.

SuperFunctionality.ts

Represents Super Workflows.

Created by:

- Admin only

Contains:

- Cross-department employee sequence
- Organization-wide routing
- Custom form configuration

Enables inter-department ticket processing.

12.4 Project Management System

Active project logic exists inside:

`ProjectManagement/`

12.4.1 Project (`ProjectManagement/Project.ts`)

Represents long-term initiatives.

Contains:

- Deliverables
- Assigned employees
- Status
- Deadlines
- Health indicator fields
- Chat references

12.4.2 Sprint (`ProjectManagement/Sprint.ts`)

Represents execution cycles.

Contains:

- Actions
- Default initial action
- Assigned employees
- Status
- Health indicator fields
- Deadlines
- Chat references

Sprints are structured as action containers.

12.5 TimeIntents (`TimeIntents.js`)

This is the active calendar storage collection.

Stores:

- Manual tasks
- Personal notes
- Auto-synced project entries
- Auto-synced sprint entries
- Appointment entries

Each record includes:

- User reference
- Event type
- Time boundaries
- Linked entity reference (if applicable)

This replaces the legacy `CalendarEvent` model.

12.6 Appointment (`Appointment.js`)

Represents internal meeting scheduling.

Contains:

- Creator reference
- Invited employees

- Acceptance state
- Scheduled time
- Status

Synced into TimeIntents for each participant upon acceptance.

12.7 Announcement Models

Announcement.js

Department-level announcements
 (Comment-enabled)

OrgAnnouncement.js

HR-level announcements
 (View-only, no comments)

12.8 UserPreference (`UserPreference.js`)

Stores user-specific UI preferences.

Currently used for:

- Theme mode (light / dark)

This integrates with `ThemeContext` in the frontend to maintain UI consistency.

12.9 Legacy Models

The following models are deprecated and should not be used for new logic:

- Project.js
- Sprint.js
- CalendarEvent.js
- Task.js (if fully replaced by structured actions)
- Frontend `depthead-project` and `employee-project` folders

All active project logic resides inside `ProjectManagement/`.

12.10 High-Level Relationship Overview

User → Department
User → Role

Executive → Departments (via ExecutiveDepartments)

Ticket → Workflow (Functionality / SuperFunctionality)
Workflow → Ordered Employees

Project → Deliverables → Employees
Sprint → Actions → Employees

Appointment → Multiple Users
TimeIntent → User → Linked Entity

UserPreference → User

12.11 Dynamic Workflow Form System

Each workflow stores:

- Custom field definitions
- Field types
- Required flags

When creating a ticket:

1. Workflow is selected.
2. Custom form schema is fetched.
3. User submits data.
4. Data is stored inside Ticket document.

This allows process-specific ticket data structures without rigid schema enforcement.

12.12 Developer Notes

1. TimeIntents is the active calendar storage model.
2. ProjectManagement folder contains the only valid Project/Sprint schemas.
3. ExecutiveDepartments controls multi-department permissions.
4. FormData stores employee profile information.
5. UserPreference manages theme persistence.
6. Dynamic forms make Ticket documents partially variable.

13. Code Overview

13.1 (auth)

Contains all the logic for the login page and forgot password.

13.2 (Dashboard)

Contains secluded folders for each type of user.

```
app/
|
├── (Dashboard)/
|   ├── admin/
|   ├── dept-head/
|   ├── employee/
|   ├── executive/
|   ├── hr-employee/
|   ├── hr-head/
|   └── layout.tsx
|
```

The folders inside each view contain components and dashboard (page.tsx). The components folder further contains folders for Home Content and Team Content (optional, based on role). The Home Content folder contains code for all the widgets on the home screen except for the announcements widgets. The components folder inside the view also contains the code for that view's header, sidebar etc.

Note: The page.tsx contained in the dashboard folder of a view calls all the components that are rendered.

13.3 app/components

This folder contains all the UI components that are rendered on the page.tsx.

13.3.1 appointments

These files contain all the logic for the appointments which are compiled in the AppointmentList.tsx which is then called into a wrapper in the relative views' content folder.

13.3.2 calendars

All logic relevant to the calendars is stored in the calendars folder with the file PersonalCalendar.tsx compiling them and they are also called in the wrapper in the relative views' content folder.

13.3.3 Department Announcements

The department announcements logic for the all employee view (deptheads can perform CRUD operations on the announcements) is handled in this folder with the AnnouncementsBoardWidget.tsx handling the home page widget and the file AnnouncementsPage.tsx handling the dedicated announcements page.

13.3.4 Organization Announcements

Similar to the department announcements, the logic for the org announcements widget on home page is handled in the universal/OrgAnnouncementsComponents folder with the dedicated page in universal/OrgAnnouncementsPage.tsx. The logic for the new announcement is handled in the OrgAnnouncements folder.

13.3.5 DeptTickets

This page handles the logic for the view of all tickets to the dephead in my department page. There is logic for all projects & sprints in the department but that isn't implemented.

13.3.6 EmployeeTicketLogs

This logic is for when the depthead clicks on an employee in my department page and the logs for tickets and projects and sprints appear.

13.3.7 ManageUsersContent

This is for the admin (super user) to perform CRUD operations on all users.

13.3.8 ProjectManagement

This folder contains the entire project management logic for both employee and depthead with both relevant logics segregated into their specific folders.

13.3.9 Workflows

The logic for workflow creation is managed in the universal/workflowComponents folder and compiled in the universal/workflowsContent.tsx. The entire logic for the super workflows is handled in the super/workflows folder.

13.3.10 Ticketing

The logic for creating a ticket, viewing created tickets, assigned tickets and actions upon assigned tickets is managed in the ticketing folder with everything appropriately named.

13.3.11 Other files

The files and logic mentioned is used directly along with relevant api routes and utils and schemas. There are other folders in the components folder which aren't mentioned in the description above. They are considered legacy and aren't actively used in the code. However much of the working code calls them or in some way relies on them so it is good to have them there. The org info and policies pages are intentionally hidden in the sidebar and quick actions because the material needed for them hasn't been provided. The settings page has a lot of components which are managed entirely in the universal folder. About the theme context, as mentioned above, it is the file that designated colors and appearance to all UI components so it is a very necessary file.