Wall-D: Dynamic vs. Hardcoded - Complete Multi-Tenant Analysis

Last Updated: December 2025
Purpose: Distinguish between configurable elements (company-specific) and core system components (universal)

EXECUTIVE SUMMARY

The Rule

- **DYNAMIC** = Stored in Firestore metadata → Changes instantly for that tenant (no app redeployment)
- **HARDCODED** = Built into MAUI application code → Same for ALL companies, same across all deployments

Real-World Analogy

```
HARDCODED = "How a door lock works" (same mechanism everywhere)
DYNAMIC = "Who has keys to this specific door" (changes per company)

HARDCODED = "Task status enum: PENDING, IN_PROGRESS, COMPLETED"
DYNAMIC = "Which roles can transition between these statuses"

HARDCODED = "The Task screen UI layout"
DYNAMIC = "Which fields display on that screen for each company"
```

---

1. HARDCODED (Core Application Logic)

1.1 Application Shell & Infrastructure

Component	Why Hardcoded	Details
MAUI WinUI3 Framework	Universal	Every deployment uses same .NET MAUI framework
Wallpaper replacement logic	Universal	Every Wall-D instance covers screen in same way
Non-movable, always-visible design	Universal	Core differentiator of Wall-D - can't change per company

Component	Why Hardcoded	Details
Task bar preservation	Universal	Always leave Windows taskbar untouched
Auto-start daemon service	Universal	Every installation starts on user login
WebSocket connection handling	Universal	How real-time sync works - same for all
Offline-first caching logic	Universal	SQLite caching implementation - same for all
Encrypted local storage	Universal	How tokens/data stored securely locally

1.2 Authentication & Security Core

Component	Why Hardcoded	Details
Firebase Auth integration	Universal	Every company uses Firebase Auth
JWT token validation logic	Universal	How tokens are verified - same algorithm
HTTPS requirement	Universal	All communication must be encrypted
MFA support framework	Universal	TOTP/SMS MFA implementation
Brute force protection algorithm	Universal	10 failed attempts → 24-hour lockout (same for all)
Session fingerprinting	Universal	How to detect unusual login patterns
Encryption/decryption algorithms	Universal	AES-256 encryption - standard approach
Password hashing (bcrypt)	Universal	How passwords hashed before storage

1.3 Core Data Models (Entities)

Component	Why Hardcoded	Details
User entity structure	Universal	userId, email, password_hash, tenant_id, created_at, last_login - ALWAYS these fields
Task entity structure	Universal	taskId, title, description, status, assignee_id, created_by, due_date, updated_at - CORE fields always present
Approval entity structure	Universal	approvalId, task_id, approver_id, status, reason, created_at, expires_at - ALWAYS this structure
Organization node structure	Universal	nodeId, name, type, parent_id, manager_id, level, children[] - CORE hierarchy fields
Task status enum	Universal	PENDING, ASSIGNED, IN_PROGRESS, AWAITING_REVIEW, PENDING_APPROVAL, COMPLETED, REJECTED, NEEDS_REVISION
Approval status enum	Universal	PENDING, APPROVED, REJECTED, ESCALATED

Component	Why Hardcoded	Details
User authentication status enum	Universal	ACTIVE, INACTIVE, PENDING_APPROVAL, REJECTED, SUSPENDED

1.4 Core Screen Types

Component	Why Hardcoded	Details
4 main screens exist	Universal	Developer Screen, Admin Screen, Manager Screen, Employee Screen - NO company can add/remove screens
Developer Screen purpose	Universal	System-wide administration - same role concept for all
Admin Screen purpose	Universal	Organization administration - same role concept for all
Manager Screen purpose	Universal	Team management - same role concept for all
Employee Screen purpose	Universal	Task execution - same role concept for all
Screen layout framework	Universal	Header, Navigation, Main Content, Footer - same for all
Navigation structure	Mostly Hardcoded	Sidebar navigation framework - same for all (content changes)

Component	Why Hardcoded	Details
Screen switching logic	Universal	How system determines which screen to show after login

1.5 Core Workflow Logic

Component	Why Hardcoded	Details
Task lifecycle states	Universal	PENDING → ASSIGNED → IN_PROGRESS → AWAITING_REVIEW → PENDING_APPROVAL → COMPLETED
Approval chain concept	Universal	Approvals exist in sequence - approval 1, then approval 2, then approval 3
Escalation trigger mechanism	Universal	Time-based escalation concept (after X days, escalate) - applies to all
Hierarchy traversal logic	Universal	How to find parent node, child nodes, report chain
Real-time sync mechanism	Universal	WebSocket broadcasting to affected users - same for all
Conflict resolution (last-write-wins)	Universal	Same conflict strategy for all companies
Offline queue logic	Universal	How to queue changes when offline, sync when online

1.6 UI Components & Layouts

Component	Why Hardcoded	Details
Button component	Universal	Same button styling, click handling logic
Text input component	Universal	Same validation framework, input handling
Dropdown/ComboBox component	Universal	Same control structure (though data source is dynamic)
Date picker	Universal	Same calendar control, date selection logic
Task card layout	Universal	How task displays (title, status, due date, assignee) - same structure
Modal/dialog framework	Universal	How popups work, close behavior, overlay
Grid/table component	Universal	How to render tabular data, sorting, pagination
Form renderer engine	Universal	The C# engine that generates UI from JSON schema
Notification toast component	Universal	How notifications display and disappear
Approval widget layout	Universal	How approval cards display on manager screen

1.7 Data Operations

Component	Why Hardcoded	Details
CRUD operations on tasks	Universal	Create, Read, Update, Delete logic - same for all

Component	Why Hardcoded	Details
Search functionality	Universal	How to query Firestore (though search terms vary)
Sorting logic	Universal	How to sort results by date, priority, name
Filtering logic	Universal	How to apply filters (though filter criteria changes)
Pagination	Universal	Page size = 20, next/prev logic - same for all
Data validation before submit	Universal	Required field checks, email format validation - same rules
Transaction handling	Universal	How to handle multi-step operations atomically

1.8 Communication & Integration

Component	Why Hardcoded	Details
HTTP request/response handling	Universal	How to call Firebase APIs
JSON serialization/deserialization	Universal	How to convert C# objects to JSON
Error handling & retry logic	Universal	Exponential backoff, retry count - same strategy
Network timeout handling	Universal	How long to wait before timeout (e.g., 30 seconds)
Slack API integration pattern	Universal	If integrated: how to format messages, handle tokens

Component	Why Hardcoded	Details
Email integration pattern	Universal	How to call email service, format content
Log message formatting	Universal	How logs appear in output

1.9 Performance & Caching

Component	Why Hardcoded	Details
SQLite cache location	Universal	Local database path for offline data
Cache expiry logic	Universal	How long to keep cached data (e.g., 24 hours)
Cache size limit	Universal	Maximum 100MB total size
Compression algorithm	Universal	How to compress cached data
Database index strategy	Universal	Which fields are indexed for performance
Query optimization	Universal	How to structure queries for speed
Lazy loading	Universal	When to load data vs. show placeholder

1.10 Security Rules (App-Level)

Component	Why Hardcoded	Details
Token validation logic	Universal	How to verify JWT signature, check expiry
Tenant isolation check	Universal	Every query must filter by tenantId - enforced in code
Permission check pattern	Universal	How to verify user has permission before showing screen

Component	Why Hardcoded	Details
Rate limiting logic	Universal	100 requests per minute per user (same for all)
Input sanitization	Universal	How to prevent SQL injection, XSS attacks
File upload virus scan	Universal	If files uploaded: scan before storage
HTTPS certificate validation	Universal	Always validate SSL certificates

---

2. DYNAMIC (Company-Specific Configuration)

2.1 Organization Structure

Component	Why Dynamic	Storage	Example Variation
Company name & logo	Different per company	Firestore: `tenants/{tenantId}/metadata/companyInfo`	"Acme Corp" vs "TechStart Inc"
Organization hierarchy tree	Completely unique per company	Firestore: `tenants/{tenantId}/organizations/`	Company A: 3 levels, Company B: 6 levels
Department structure	Different per company	Firestore: `tenants/{tenantId}/organizations/departments/`	Company A: Sales, Engineering, HR; Company B: Sales, Engineering, HR, Operations, Finance, Legal
Office locations	Different per company	Firestore: `tenants/{tenantId}/organizations/locations/`	Company A: 1 office; Company B: 5 offices across countries
Reporting relationships	Different per company	Firestore: `organizations/{nodeId}/manager_id`	Who reports to whom
Team structures	Different per company	Firestore: `organizations/{nodeId}/team_members[]`	Team composition varies
Org node attributes	Extensible per company	Firestore: `organizations/{nodeId}/custom_attributes{}`	Company A needs "cost_center", Company B needs "project_code"

2.2 Designations & Roles

Component	Why Dynamic	Storage	Example Variation
Designation list	Different per company	Firestore: `tenants/{tenantId}/metadata/designations/{id}`	Company A: 10 designations; Company B: 25 designations
Designation hierarchy	Different per company	Firestore: `designations/{id}.reports_to[]`	CEO→VP→Manager vs. Executive→Lead→Contributor hierarchy
Custom designations	Company-specific	Firestore: `designations/{id}.tenantId`	Company A has "Chief Technology Officer"; Company B doesn't
Designation permissions	Different per company	Firestore: `designations/{id}.approval_powers[]`	Managers in Company A can approve tasks; Managers in Company B can only view
Default screen per designation	Different per company	Firestore: `designations/{id}.default_screen`	All managers see "Manager Screen" vs. some see "Analytics Screen"
Approval authority per designation	Different per company	Firestore: `designations/{id}.can_approve_tasks`	CEO in Company A can approve tasks; in Company B, team lead approves first
Multiple roles per designation	Different per company	Firestore: `designations/{id}.default_roles[]`	Manager = [manager, viewer]; Analyst = [viewer, analyst]

2.3 Forms & Fields

Component	Why Dynamic	Storage	Example Variation
Registration form fields	Different per company	Firestore: `tenants/{tenantId}/formSchemas/user_registration/`	Company A adds "Badge ID"; Company B adds "Cost Center"; Company C adds "Department Code"
Task creation form fields	Different per company	Firestore: `tenants/{tenantId}/metadata/formSchemas/task_creation/`	Company A: title, description, due_date, priority; Company B: title, description, due_date, cost_center, customer_id, priority
Field validation rules	Different per company	Firestore: `formSchemas/{formId}/fields/{fieldId}/validation`	Email required: Company A yes, Company B has optional emails

Component	Why Dynamic	Storage	Example Variation
Field labels & placeholders	Different per company (language/terminology)	Firestore: `formSchemas/{formId}/fields/{fieldId}/label`	"Task" vs. "Work Order" vs. "Activity"
Dropdown options	Different per company	Firestore: `formSchemas/{formId}/fields/{fieldId}/options[]`	Priority: (Low, Medium, High) vs. (1–5) vs. (Critical, High, Normal, Low)
Custom form types	Company creates new forms	Firestore: `tenants/{tenantId}/metadata/formSchemas/{customFormId}/`	Company A creates "Incident Report"; Company B creates "Equipment Request Form"
Dependent fields	Different per company	Firestore: `formSchemas/{formId}/fields/{fieldId}/dependsOn`	Show "Manager Approval" if Priority="High"
Field visibility rules	Different per company	Firestore: `formSchemas/{formId}/fields/{fieldId}/visibleTo[]`	Cost Center visible only to Managers; Customer ID visible to all
Required field rules	Different per company	Firestore: `formSchemas/{formId}/fields/{fieldId}/required`	Company A: all fields required; Company B: only title + assignee required
Form version control	Version per company	Firestore: `formSchemas/{formId}/versions`	Company A using v2; Company B still using v1

2.4 Workflows & Approvals

Component	Why Dynamic	Storage	Example Variation
Approval chain length	Different per company	Firestore: `tenants/{tenantId}/metadata/workflowDefinitions/{workflowId}/`	Company A: 1 approver (Manager); Company B: 3 approvers (Manager→Director→VP)
Approval authorities	Different per company	Firestore: `workflowDefinitions/{id}/approvals/{level}/approver_designati`	Level 1: Manager vs. Team Lead
Which tasks need approval	Different per company	Firestore: `workflowDefinitions/{id}/approvals/triggers/`	Company A: All tasks require approval; Company B: Only tasks with priority="High"

Component	Why Dynamic	Storage	Example Variation
Approval conditions	Different per company	Firestore: `workflowDefinitions/{id}/conditions[]`	Company A: Approve if < $1000; Company B: Approve if < $5000 OR department="Operations"
Escalation rules	Different per company	Firestore: `tenants/{tenantId}/metadata/escalationRules/`	Company A: Escalate after 1 day; Company B: Escalate after 3 days
Escalation target	Different per company	Firestore: `escalationRules/{id}/escalate_to`	Company A escalates to CEO; Company B escalates to VP
Rejection reasons	Different per company	Firestore: `workflowDefinitions/{id}/rejectionReasons[]`	Company A: (Incomplete, Wrong Price, Needs Clarification); Company B: (Budget Exceeded, Timeline Conflict)
Notification triggers	Different per company	Firestore: `tenants/{tenantId}/metadata/notificationTriggers/`	Notify on: status change vs. approaching deadline
SLA times	Different per company	Firestore: `workflowDefinitions/{id}/sla_hours`	Company A: Task must complete in 24 hrs; Company B: 72 hrs
Custom workflow states	Company-specific	Firestore: `workflowDefinitions/{id}/customStates/`	Company A adds "ON_HOLD" state; Company B adds "ESCALATED" state

2.5 Permissions & Access Control

Component	Why Dynamic	Storage	Example Variation
Per-screen access	Different per company	Firestore: `designations/{id}/screen_access[]`	Manager → [manager_screen, analytics_screen]; Director → [manager_screen, analytics_screen, admin_screen]
Feature access	Different per company	Firestore: `designations/{id}/task_permissions[]`	Company A Manager: [create_task, assign_task, approve_task]; Company B Manager: [create_task, assign_task] only

Component	Why Dynamic	Storage	Example Variation
Data access scope	Different per company	Firestore: `designations/{id}/data_scope`	Manager sees: own team vs. Manager sees: own team + cross-functional tasks
Bulk operation permissions	Different per company	Firestore: `designations/{id}/permissions[]`	Can bulk-export data: CEO yes; Manager yes vs. Manager no; Employee no
Delete permission	Different per company	Firestore: `designations/{id}/permissions[]`	Can permanently delete tasks: Company A: yes; Company B: Manager maybe; Employee no
Custom permission rules	Company creates	Firestore: `tenants/{tenantId}/customPermissions/`	Company A: "Can assign tasks only to direct reports" vs. "Can assign to anyone"

2.6 User Preferences & Settings

Component	Why Dynamic	Storage	Example Variation
Notification preferences	Per user within company	Firestore: `tenants/{tenantId}/users/{userId}/notificationPreferences/`	User A: Get SMS alerts; User B: Gets email
Notification channels	Per user	Firestore: `users/{userId}/notificationPreferences/channels[]`	Company A team uses Slack; Company B uses email; Company C uses both
Do Not Disturb hours	Per user	Firestore: `users/{userId}/notificationPreferences/doNotDisturbHours`	User timezone dependent: 8 PM - 8 AM (India) vs. 5 PM - 9 AM (US)
Display language	Per user or company-wide	Firestore: `tenants/{tenantId}/settings/language` or `users/{userId}/preferences/language`	Company A: English; Company B: Hindi; Company C: Both options
Date/time format	Per company or user	Firestore: `tenants/{tenantId}/settings/dateFormat`	Company A: DD/MM/YYYY; Company B: MM/DD/YYYY
Currency format	Per company	Firestore: `tenants/{tenantId}/settings/currency`	Company A: INR; Company B: USD; Company C: EUR
Theme preference	Per user	Firestore: `users/{userId}/preferences/theme`	Light vs. Dark mode

Component	Why Dynamic	Storage	Example Variation
Screen auto-logout time	Per company	Firestore: `tenants/{tenantId}/settings/autoLogoutMinutes`	Company A: 30 minutes; Company B: 60 minutes; Company C: Never

2.7 Notification & Communication

Component	Why Dynamic	Storage	Example Variation
Notification templates	Different per company	Firestore: `tenants/{tenantId}/notificationTemplates/{templateId}`	"Task assigned to you by {manager_name}" vs. "New work order: {task_title}"
Email branding	Different per company	Firestore: `tenants/{tenantId}/settings/emailBranding`	Company logo, footer, colors, email header
Slack webhook URLs	Different per company	Firestore: `tenants/{tenantId}/integrations/slack/webhookUrl`	Each company has different Slack workspace
SMS provider config	Different per company	Firestore: `tenants/{tenantId}/integrations/sms/provider`	Company A uses Twilio; Company B uses AWS SNS
Email sender	Different per company	Firestore: `tenants/{tenantId}/settings/emailSender`	noreply@acme.com vs. noreply@techstartup.com
Support contact info	Different per company	Firestore: `tenants/{tenantId}/settings/supportContact`	Company A: support@acme.com; Company B: help@techstartup.com
Notification frequency	Per user	Firestore: `users/{userId}/notificationPreferences/frequency`	Immediate vs. Daily digest vs. Weekly digest

2.8 Integrations

Component	Why Dynamic	Storage	Example Variation
Enabled integrations	Different per company	Firestore: `tenants/{tenantId}/integrations`	Company A has: Slack; Company B has: Slack + Jira + Salesforce
API credentials	Different per company	Firebase Secrets: stored securely	Each company's Slack API token, Jira API key, etc.

Component	Why Dynamic	Storage	Example Variation
Integration settings	Different per company	Firestore: `tenants/{tenantId}/integrations/{serviceName}/settings/{docId}`	Slack: channel name to sync with
Webhook endpoints	Different per company	Firestore: `tenants/{tenantId}/integrations/{serviceName}/webhookUrl`	Each company provides their own webhook URL for third-party services
Custom API endpoints	Different per company	Firestore: `tenants/{tenantId}/integrations/custom/{docId}`	Company might have internal HR system with custom API

2.9 Metadata & Configuration

Component	Why Dynamic	Storage	Example Variation
Task priority levels	Different per company	Firestore: `tenants/{tenantId}/metadata/taskPriorities/{docId}`	Company A: [Low, Medium, High]; Company B: [1, 2, 3, 4, 5]; Company C: [Critical, High, Normal, Low]
Task categories/types	Different per company	Firestore: `tenants/{tenantId}/metadata/taskCategories/{docId}`	Company A: [Feature, Bug, Documentation]; Company B: [Development, Testing, DevOps]
Task status labels	Different per company	Firestore: `tenants/{tenantId}/metadata/taskStatuses/{docId}`	Most use default; Company B renames to: [Backlog, Ready, In Dev, QA, Deployed]
Custom field definitions	Different per company	Firestore: `tenants/{tenantId}/metadata/customFields/{docId}`	Company A adds “Cost Center” field; Company B adds “Customer ID” + “Project Code”
Department list	Different per company	Firestore: `tenants/{tenantId}/metadata/departments/{docId}`	Company A: 5 departments; Company B: 12 departments
Cost centers	Different per company	Firestore: `tenants/{tenantId}/metadata/costCenters/{docId}`	Company A: CC01-CC10; Company B: CC001-CC050
Project list	Different per company	Firestore: `tenants/{tenantId}/metadata/projects/{docId}`	Company A: 3 projects; Company B: 20+ projects
Company policies	Different per company	Firestore: `tenants/{tenantId}/metadata/policies/{docId}`	Expense limit: Company A: $1000, Company B: $5000

3. DETAILED EXAMPLES: Dynamic vs. Hardcoded in Action

Example 1: Task Creation Form

HARDCODED (Application Code)

```
// MAUI Application Code (C#)
public class DynamicFormRenderer
{
    // This logic is THE SAME for every company
    public async Task<StackPanel> RenderFormAsync(string formId, string tenantId)
    {
        // 1. Fetch form schema from Firestore (generic code)
        var formSchema = await _firestore.GetDocument($"tenants/{tenantId}/metadata/formSche

        // 2. Create container (same layout for all)
        var container = new StackPanel();

        // 3. For each field in schema, create appropriate control
        foreach (var field in formSchema.Fields)
        {
            // This logic is universal - handles ANY field type
            var control = CreateControl(field.Type, field.Properties);

            // Apply validation (same pattern for all)
            if (field.Required)
            {
                AddRequiredValidator(control);
            }

            container.Children.Add(control);
        }

        // 4. Return rendered form (same structure for all companies)
        return container;
    }

    private Control CreateControl(string fieldType, Dictionary<string, object> properties)
    {
        // UNIVERSAL field-type handling
        return fieldType switch
        {
            "text" => new TextBox { Placeholder = properties["placeholder"]?.ToString() },
```

```
            "email" => new TextBox { InputScope = InputScopeNameValue.EmailSmtpAddress },
            "date" => new DatePicker(),
            "dropdown" => new ComboBox(),
            "checkbox" => new CheckBox(),
            _ => null
        };
    }
}

// The rendering engine is identical for ALL companies
```

DYNAMIC (Firestore Configuration) Company A's Task Creation Form:

```
{
  "formId": "task_creation",
  "name": "Create New Task",
  "fields": [
    {
      "id": "title",
      "type": "text",
      "label": "Task Title",
      "required": true,
      "placeholder": "Enter task title"
    },
    {
      "id": "description",
      "type": "textarea",
      "label": "Description",
      "required": true
    },
    {
      "id": "due_date",
      "type": "date",
      "label": "Due Date",
      "required": true
    },
    {
      "id": "priority",
      "type": "dropdown",
      "label": "Priority",
      "required": true,
      "options": ["Low", "Medium", "High"]
    }
  ]
}
```

Company B's Task Creation Form (SAME CODE, DIFFERENT CONFIG):

```
{
  "formId": "task_creation",
  "name": "Create New Task",
  "fields": [
    {
      "id": "title",
      "type": "text",
      "label": "Task Title",
      "required": true,
      "placeholder": "Enter task title"
    },
    {
      "id": "description",
      "type": "textarea",
      "label": "Description",
      "required": true
    },
    {
      "id": "due_date",
      "type": "date",
      "label": "Due Date",
      "required": true
    },
    {
      "id": "priority",
      "type": "dropdown",
      "label": "Priority",
      "required": true,
      "options": ["1", "2", "3", "4", "5"]
    },
    {
      "id": "cost_center",
      "type": "dropdown",
      "label": "Cost Center",
      "required": true,
      "dataSource": "firestore",
      "collection": "costCenters",
      "displayField": "name"
    },
    {
      "id": "customer_id",
      "type": "autocomplete",
      "label": "Customer",
```

```
      "required": true,
      "dataSource": "firestore",
      "collection": "customers"
    },
    {
      "id": "project_code",
      "type": "text",
      "label": "Project Code",
      "required": false,
      "validation": "^[A-Z]{2}[0-9]{4}$"
    }
  ]
}
```

Impact: - MAUI code unchanged - same form renderer - Company A gets simple form (4 fields) - Company B gets extended form (7 fields) - Change form instantly → edit JSON in Firebase Console - Deploy to Company A at 2 PM, Company B still uses old form - Zero code recompilation needed

---

Example 2: Approval Workflow

HARDCODED (Application Logic)

```
// MAUI Application Code (C#)
public class ApprovalEngine
{
    // This logic is THE SAME for every company
    public async Task ProcessApprovalChainAsync(string taskId, string tenantId)
    {
        var task = await _firestore.GetTask(taskId, tenantId);

        // 1. ALWAYS check if approval needed
        if (!task.RequiresApproval) return;

        // 2. ALWAYS get approval chain from metadata
        var approvalChain = await GetApprovalChainAsync(task, tenantId);

        // 3. ALWAYS create sequential approval records
        foreach (int level = 0; level < approvalChain.Count; level++)
        {
            var approval = new Approval
            {
                TaskId = taskId,
                ApproverId = approvalChain[level].UserId,
                Level = level,
```

```
                Status = ApprovalStatus.Pending,
                CreatedAt = DateTime.Now,
                ExpiresAt = DateTime.Now.AddDays(2)
            };

            await _firestore.CreateApproval(approval, tenantId);

            // 4. ALWAYS send notification
            await _notifications.SendApprovalNotificationAsync(approval, tenantId);
        }

        // 5. ALWAYS update task status
        task.Status = TaskStatus.PendingApproval;
        await _firestore.UpdateTask(task, tenantId);
    }

    // Get approval chain based on company's workflow definition
    private async Task<List<ApprovingUser>> GetApprovalChainAsync(Task task, string tenantId
    {
        // Query company's workflow definition (DYNAMIC)
        var workflowDef = await _firestore
            .GetDocument($"tenants/{tenantId}/metadata/workflowDefinitions/task_approval");

        var approvalChain = new List<ApprovingUser>();

        // UNIVERSAL logic: iterate through approval levels
        for (int level = 0; level < workflowDef.ApprovalLevels.Count; level++)
        {
            var approverDesignation = workflowDef.ApprovalLevels[level].ApproverDesignation;

            // Find user with that designation in hierarchy
            var approver = await FindApproverInHierarchyAsync(task.AssigneeId, approverDesig
            approvalChain.Add(approver);
        }

        return approvalChain;
    }
}

// The approval engine code is identical for ALL companies
```

DYNAMIC (Workflow Configuration) Company A's Workflow (Simple - 1 Approver):

```
{
  "workflowId": "task_approval",
```

```
  "name": "Task Approval Workflow",
  "approvalLevels": [
    {
      "level": 1,
      "approverDesignation": "manager",
      "approverTitle": "Assigned Team Manager",
      "requiresComment": false,
      "canReject": true,
      "mustCompleteWithin": 24
    }
  ],
  "escalationRules": [
    {
      "trigger": "notApprovedAfterHours",
      "afterHours": 24,
      "escalateTo": "vp"
    }
  ]
}
```

Company B's Workflow (Complex - 3 Approvers):

```
{
  "workflowId": "task_approval",
  "name": "Task Approval Workflow",
  "approvalLevels": [
    {
      "level": 1,
      "approverDesignation": "team_lead",
      "approverTitle": "Team Lead",
      "requiresComment": true,
      "canReject": true,
      "mustCompleteWithin": 24
    },
    {
      "level": 2,
      "approverDesignation": "manager",
      "approverTitle": "Department Manager",
      "requiresComment": true,
      "canReject": true,
      "mustCompleteWithin": 48
    },
    {
      "level": 3,
      "approverDesignation": "director",
      "approverTitle": "Director",
      "requiresComment": false,
```

```
      "canReject": true,
      "mustCompleteWithin": 72
    }
  ],
  "escalationRules": [
    {
      "trigger": "notApprovedAfterHours",
      "afterHours": 24,
      "escalateTo": "vp"
    },
    {
      "trigger": "notApprovedAfterHours",
      "afterHours": 72,
      "escalateTo": "ceo"
    }
  ]
}
```

Impact: - Approval engine code unchanged - Company A: all tasks go to manager (1-level approval) - Company B: all tasks go to team lead → manager → director (3-level approval) - Add new approval level → Edit JSON in Firebase - Change escalation after 24 hrs to 48 hrs → Edit JSON instantly - Deploy same app binary to both companies

---

Example 3: Designation & Permissions

HARDCODED (Core Concept)

```
// MAUI Application Code (C#)
public class AuthorizationEngine
{
    // ALWAYS follow this permission check pattern
    public async Task<bool> CanUserDoActionAsync(string userId, string action, string tenant
    {
        // 1. Get user
        var user = await _firestore.GetUser(userId, tenantId);

        // 2. Get user's designation
        var designation = await _firestore.GetDesignation(user.DesignationId, tenantId);

        // 3. Check if action is in designation's permissions
        bool hasPermission = designation.Permissions.Contains(action);

        return hasPermission;
    }
```

```
    // ALWAYS check screen access this way
    public async Task<Screen> DetermineUserScreenAsync(string userId, string tenantId)
    {
        // 1. Get user
        var user = await _firestore.GetUser(userId, tenantId);

        // 2. Get user's designation
        var designation = await _firestore.GetDesignation(user.DesignationId, tenantId);

        // 3. Check which screens user can access
        var accessibleScreens = designation.ScreenAccess;

        if (accessibleScreens.Count == 1)
            return LoadScreen(accessibleScreens[0]); // Auto-show single screen
        else
            return ShowScreenSelector(accessibleScreens); // Let user choose
    }
}

// Authorization logic is THE SAME for all companies
```

DYNAMIC (Permission Configuration) Company A Designations:

```
{
  "tenantId": "company_a",
  "designations": {
    "ceo": {
      "name": "Chief Executive Officer",
      "hierarchy_level": 1,
      "reports_to": [],
      "permissions": [
        "create_task", "assign_task", "approve_task", "complete_task",
        "view_analytics", "export_data", "manage_users", "manage_forms",
        "manage_workflows", "delete_tasks", "view_all_tasks"
      ],
      "screen_access": ["developer", "admin", "manager"],
      "can_delegate_to": ["vp"]
    },
    "manager": {
      "name": "Department Manager",
      "hierarchy_level": 3,
      "reports_to": ["ceo", "vp"],
      "permissions": [
        "create_task", "assign_task", "approve_task",
        "view_team_tasks", "view_analytics"
```

```
      ],
      "screen_access": ["manager"],
      "can_delegate_to": ["team_lead"]
    },
    "employee": {
      "name": "Software Developer",
      "hierarchy_level": 5,
      "reports_to": ["manager", "team_lead"],
      "permissions": [
        "view_assigned_tasks", "complete_task", "view_own_analytics"
      ],
      "screen_access": ["employee"],
      "can_delegate_to": []
    }
  }
}
```

Company B Designations (MUCH MORE COMPLEX):

```
{
  "tenantId": "company_b",
  "designations": {
    "ceo": {
      "name": "Chief Executive Officer",
      "hierarchy_level": 1,
      "reports_to": [],
      "permissions": ["all"],
      "screen_access": ["developer", "admin", "manager", "analytics"],
      "can_delegate_to": ["cfo", "vp_engineering"]
    },
    "cfo": {
      "name": "Chief Financial Officer",
      "hierarchy_level": 1,
      "reports_to": ["ceo"],
      "permissions": [
        "create_task", "assign_task", "approve_task", "view_all_tasks",
        "view_financial_analytics", "approve_expenses", "export_data"
      ],
      "screen_access": ["admin", "manager", "finance"],
      "can_delegate_to": ["controller"]
    },
    "vp_engineering": {
      "name": "VP Engineering",
      "hierarchy_level": 2,
      "reports_to": ["ceo"],
      "permissions": [
        "create_task", "assign_task", "approve_task", "view_team_tasks",
```

```
      "view_engineering_analytics", "manage_tech_stack"
    ],
    "screen_access": ["manager", "engineering"],
    "can_delegate_to": ["engineering_lead"]
  },
  "engineering_lead": {
    "name": "Engineering Team Lead",
    "hierarchy_level": 3,
    "reports_to": ["vp_engineering", "manager"],
    "permissions": [
      "create_task", "assign_task", "approve_task",
      "view_team_tasks", "view_engineering_analytics"
    ],
    "screen_access": ["manager"],
    "can_delegate_to": []
  },
  "senior_engineer": {
    "name": "Senior Software Engineer",
    "hierarchy_level": 4,
    "reports_to": ["engineering_lead", "manager"],
    "permissions": [
      "view_assigned_tasks", "complete_task", "mentor_junior",
      "approve_code_review", "view_own_analytics"
    ],
    "screen_access": ["employee", "engineer"],
    "can_delegate_to": ["junior_engineer"]
  },
  "junior_engineer": {
    "name": "Junior Software Engineer",
    "hierarchy_level": 5,
    "reports_to": ["senior_engineer", "engineering_lead"],
    "permissions": [
      "view_assigned_tasks", "complete_task", "view_own_analytics"
    ],
    "screen_access": ["employee"],
    "can_delegate_to": []
  },
  "finance_analyst": {
    "name": "Finance Analyst",
    "hierarchy_level": 4,
    "reports_to": ["cfo", "controller"],
    "permissions": [
      "create_expense_report", "view_expense_reports",
      "approve_expenses_below_limit", "view_financial_analytics"
    ],
    "screen_access": ["employee", "finance"],
```

```
      "can_delegate_to": []
    }
    // ... more designations
  }
}
```

Impact: - Authorization code unchanged - Company A: 3 designations, simple permissions - Company B: 7+ designations, complex permission matrix - Add new designation → Add JSON object in Firebase - Change "manager can delete tasks" permission → Edit JSON boolean - Deploy same MAUI binary everywhere

---

4. THE CRITICAL BOUNDARY: Where Dynamic Meets Hardcoded

4.1 Database Query Pattern (THE BOUNDARY)

```
// HARDCODED: Query Pattern (same for all companies)
public async Task<List<Task>> GetMyTasksAsync(string userId, string tenantId)
{
    // Step 1: HARDCODED - Always filter by tenantId first
    var query = _firestore
        .Collection($"tenants/{tenantId}/tasks");

    // Step 2: HARDCODED - Always check authorization
    var permissions = await GetUserPermissionsAsync(userId, tenantId);
    if (!permissions.Contains("view_tasks")) return null;

    // Step 3: DYNAMIC - Apply role-specific filters
    var dataScope = await GetUserDataScopeAsync(userId, tenantId);

    if (dataScope == "my_team") // DYNAMIC from designation config
    {
        // Get user's team members
        var teamMembers = await GetTeamMembersAsync(userId, tenantId);
        query = query.Where("assignee_id", "in", teamMembers);
    }
    else if (dataScope == "all") // DYNAMIC from designation config
    {
        // No filter - see all
    }

    // Step 4: HARDCODED - Always apply basic filters
    query = query
        .Where("status", "!=", "COMPLETED")
```

```
        .OrderBy("due_date");

    return await query.GetAsync();
}

// PATTERN:
// 1. Hardcoded tenantId filter (security boundary)
// 2. Hardcoded permission check (security boundary)
// 3. Dynamic scope from company configuration
// 4. Hardcoded ordering logic
```

4.2 Screen Display Logic (THE BOUNDARY)

```
// HARDCODED: Screen switching logic
public async Task LoadUserScreenAsync(string userId, string tenantId)
{
    // Step 1: HARDCODED - These 4 screens always exist
    var availableScreens = new[] { "Developer", "Admin", "Manager", "Employee" };

    // Step 2: Get user's designation
    var user = await _firestore.GetUser(userId, tenantId);
    var designation = await _firestore.GetDesignation(user.DesignationId, tenantId);

    // Step 3: DYNAMIC - Which screens can this designation access
    var accessibleScreens = designation.ScreenAccess; // ["manager", "employee"]

    // Step 4: HARDCODED - Determine which screen to show
    if (accessibleScreens.Contains("manager") &&
        accessibleScreens.Contains("employee"))
    {
        // Show screen selector
        var choice = await ShowScreenSelectorAsync(accessibleScreens);
        LoadScreen(choice);
    }
    else
    {
        // Show single accessible screen
        LoadScreen(accessibleScreens[0]);
    }
}

// PATTERN:
// 1. Hardcoded: These 4 screen types exist
// 2. Hardcoded: This is how we determine access
// 3. Dynamic: Which screens the company grants per designation
// 4. Hardcoded: This is how we show/load screens
```

4.3 Form Validation (THE BOUNDARY)

```
// HARDCODED: Validation framework
public bool ValidateFormSubmission(FormSubmission submission, FormSchema schema)
{
    var errors = new List<string>();

    // HARDCODED: Loop through all fields
    foreach (var field in schema.Fields)
    {
        var value = submission.GetFieldValue(field.Id);

        // HARDCODED: Required field check
        if (field.Required && string.IsNullOrEmpty(value))
        {
            errors.Add($"{field.Label} is required");
            continue;
        }

        // HARDCODED: Standard validations (same for all)
        if (field.Type == "email" && !IsValidEmail(value))
        {
            errors.Add($"{field.Label} must be valid email");
            continue;
        }

        if (field.Type == "phone" && !IsValidPhone(value))
        {
            errors.Add($"{field.Label} must be valid phone");
            continue;
        }

        // DYNAMIC: Custom regex validation (company-specific pattern)
        if (!string.IsNullOrEmpty(field.ValidationRegex))
        {
            if (!Regex.IsMatch(value, field.ValidationRegex))
            {
                errors.Add($"{field.Label} format is invalid");
            }
        }
    }

    return errors.Count == 0;
}

// PATTERN:
```

```
// 1. Hardcoded: Loop through fields
// 2. Hardcoded: Required check always the same
// 3. Hardcoded: Standard type validations (email, phone, date)
// 4. Dynamic: Custom regex pattern per company per field
```

---

5. DEPLOYMENT IMPLICATIONS

5.1 Same Binary, Different Behavior

```
SCENARIO: Deploy Wall-D to 100 companies

Binary: WallD.exe (single file)
Version: 1.0.0
File size: 50 MB

Install to Company A:
  Extract WallD.exe
  Create shortcut on desktop
  User logs in
  Fetches Company A's metadata from Firestore
     Company A has 3 designations
     Company A task form has 4 fields
     Company A approval: 1 level
     Company A sees Manager Screen
  Company A gets THEIR customized experience

Install to Company B (SAME BINARY):
  Extract WallD.exe (identical)
  Create shortcut on desktop
  User logs in
  Fetches Company B's metadata from Firestore
     Company B has 7 designations
     Company B task form has 7 fields
     Company B approval: 3 levels
     Company B sees Developer Screen
  Company B gets THEIR customized experience

KEY INSIGHT:
- ONE binary file serves 100+ companies
- Each sees different UI/workflow based on Firestore metadata
- Update metadata = instant change for that company
- Update app code = requires rebuild but only for shared features
```

5.2 Configuration Changes Without Redeployment

```
MONDAY 9 AM: Company A's COO says "Add Cost Center field to tasks"
  You: Log into Firebase Console
  You: Navigate to tenants/company_a/metadata/formSchemas/task_creation
  You: Add new field to JSON
  You: Click Save
  BANG! 2 seconds later:
     All Company A employees see new Cost Center field
     Zero application restart needed
     Already deployed, already running
     Just fetched new form schema

MONDAY 10 AM: Company B's VP Engineering says "Approval now needs 3 levels"
  You: Log into Firebase Console
  You: Navigate to tenants/company_b/metadata/workflowDefinitions
  You: Update approvalLevels array (add level 2 and 3)
  You: Click Save
  BANG! All Company B tasks now go through 3-level approval
     All managers see 3 approval steps
     Existing tasks pick up new workflow automatically
     Zero code compilation

KEY BENEFIT:
- Configuration changes are instant
- No recompilation
- No redeployment
- No restart
- Change visible within seconds
```

5.3 Backwards Compatibility

```
SCENARIO: You need to add new field type "signature_pad"

Current code (Hardcoded - ALL VERSIONS):
public Control CreateControl(string fieldType, Dictionary<string, object> props)
{
    return fieldType switch
    {
        "text" => new TextBox(),
        "email" => new TextBox(),
        "date" => new DatePicker(),
        "dropdown" => new ComboBox(),
        _ => null  // Unknown types = ignored
    };
}
```

```
Add signature_pad support:
  Update CreateControl() to handle "signature_pad"
  Rebuild MAUI app
  Redeploy to all customers
     This ONLY happens when you need new UI control
     NOT for configuration changes
     NOT for permission changes
     NOT for form fields (unless new control type needed)

Meanwhile:
  Existing customers keep running old version
  Company A creates form with new signature_pad field (in JSON)
  Company A WAITS until they upgrade to new app version
  Company B still uses old version, no impact
  Phased rollout possible

KEY INSIGHT:
- Core logic updates (hardcoded) = everyone must upgrade
- Configuration updates (dynamic) = can be deployed selectively
- Version compatibility = metadata must be backwards compatible
```

---

6. SPECIAL CASES: Seems Hardcoded But Actually Dynamic

6.1 Organization Hierarchy

Seems Hardcoded: "All companies have managers and employees"

Actually Dynamic:

```
Company A Hierarchy:
  CEO (1 person)
  VP Sales (1 person)
     Sales Managers (3 people)
        Sales Reps (12 people)
  VP Engineering (1 person)
     Engineering Leads (2 people)
        Engineers (8 people)
(2-3 levels, 25 people total)

Company B Hierarchy:
  Executive Team (board structure)
  Department Heads (6 people)
     Team Leads (18 people)
```

```
       Contributors (80 people)
       Interns (12 people)
    Specialists (various)
  Administrative Support
(4-5 levels, 200+ people total)

Company C Hierarchy (Flat):
  CEO (1 person)
  All employees report to CEO (15 people)
(1 level, 15 people total)

// Same hierarchy traversal code works for all
// Different tree structures completely dynamic
```

6.2 Task Fields

Seems Hardcoded: "All companies have tasks with title, description"

Actually Dynamic:

```
Company A Task: title, description, due_date, priority
(4 fields)

Company B Task: title, description, due_date, priority,
                cost_center, customer_id, project_code,
                estimated_hours, resource_pool
(9 fields)

Company C Task: title, description, assigned_project,
               timeframe, approval_required_level
(5 fields, different fields)

// Same task entity in code
// Fields populated from form schema
// Different companies → different fields in UI
```

6.3 Approval Processes

Seems Hardcoded: "Tasks need approval"

Actually Dynamic:

```
Company A: All tasks need 1-level approval (Manager)

Company B: Tasks need 3-level approval IF:
           - Priority = High
           - Cost > $5000
           - Department = Operations
```

```
            Otherwise 1-level approval

Company C: No approval needed for any tasks
            (approval feature disabled in metadata)

// Same approval engine code handles all
// Approval rules completely dynamic in metadata
```

---

7. CHANGE MANAGEMENT MATRIX

Change Type	Category	How to Change	Recompile?	Redeploy?	Restart App?	Immediate?	Who Makes Change?
Add designation	Dynamic	Edit Firestore JSON	No	No	No	Yes	Admin console
Change manager permissions	Dynamic	Edit Firestore JSON	No	No	No	Yes	Admin console
Add form field	Dynamic	Edit Firestore JSON	No	No	No	Yes (form refresh)	Admin console
Change approval chain	Dynamic	Edit Firestore JSON	No	No	No	Yes	Admin console
Add new screen type	Hardcoded	C# code	Yes	Yes	Yes	(after deploy)	Developer
Change form rendering logic	Hardcoded	C# code	Yes	Yes	Yes	(after deploy)	Developer

Change Type	Category	How to Change	Recompile?	Redeploy?	Restart App?	Immediate?	Who Makes Change
Add new field type (signature pad)	Hardcoded	C# code + XAML	Yes	Yes	Yes	(after deploy)	Developer
Fix authentication bug	Hardcoded	C# code	Yes	Yes	Yes	(after deploy)	Developer
Change UI layout	Hardcoded	XAML markup	Yes	Yes	Yes	(after deploy)	Developer
Update Slack integration	Hardcoded	C# code	Yes	Yes	Yes	(after deploy)	Developer
Customize company logo	Dynamic	Upload to Firebase	No	No	No	Yes	Company admin
Change notification email template	Dynamic	Edit Firestore JSON	No	No	No	Yes	Company admin
Update company name	Dynamic	Edit Firestore document	No	No	No	Yes	Company admin

8. RED FLAGS: Common Mistakes in Multi-Tenant Design

MISTAKE 1: Hardcoding Company-Specific Logic

WRONG:

```
public bool CanApproveTask(User user)
{
    if (user.CompanyName == "Acme Corp" && user.Designation == "Manager")
        return true;

    if (user.CompanyName == "TechCorp" && user.Designation == "Lead")
        return true;

    return false;
}
// Problem: Adding new company = code change + recompile
```

RIGHT:

```
public async Task<bool> CanApproveTaskAsync(User user, string tenantId)
{
    var designation = await _firestore.GetDesignation(user.DesignationId, tenantId);
    return designation.Permissions.Contains("approve_task");
}
// Solution: Logic reads from company's designation config
```

MISTAKE 2: Mixing Company Data with System Data

WRONG:

```
// Company data AND system data in same collection
db.Collection("users")
   .Where("companyId", "==", "company_a")
   .Where("role", "==", "admin");
// Problem: If query forgets companyId filter, data leakage
```

RIGHT:

```
// Company data isolated in tenant directory
db.Collection("tenants/company_a/users")
   .Where("designation", "==", "admin");
// Solution: tenantId is part of path, impossible to forget
```

MISTAKE 3: Fetching Wrong Tenant's Metadata

WRONG:

```
public async Task<Form> GetFormAsync(string formId)
{
    return await _firestore
        .Collection("forms")
        .Document(formId)
        .GetAsync();
    // Problem: If Company A and Company B both have "user_registration" form,
    // might return wrong one
}
```

RIGHT:

```
public async Task<Form> GetFormAsync(string formId, string tenantId)
{
    return await _firestore
        .Collection($"tenants/{tenantId}/forms")
        .Document(formId)
        .GetAsync();
    // Solution: TenantId is explicit in path
}
```

MISTAKE 4: Assuming Same Structure for All Companies

WRONG:

```
var taskFields = new[] { "title", "description", "due_date", "priority" };
foreach (var field in taskFields)
{
    task[field] = form.GetValue(field);
}
// Problem: Company B has 7 fields, Company C has 3 different fields
// This hardcoded list breaks
```

RIGHT:

```
var formSchema = await GetFormSchema("task_creation", tenantId);
foreach (var field in formSchema.Fields)
{
    var value = form.GetValue(field.Id);
    task[field.Id] = value;
}
// Solution: Reads fields from company's form schema
```

MISTAKE 5: Static Configuration at App Startup

WRONG:

```
// On app startup
var companyDesignations = await _firestore.GetAllDesignations();
```

```
AppState.Designations = companyDesignations;

// Later: Company changes designation permissions
// App doesn't see it until restart
```

RIGHT:

```
// Real-time listener
var subscription = _firestore
    .Collection($"tenants/{tenantId}/metadata/designations")
    .OnSnapshot(snapshot =>
    {
        AppState.Designations = snapshot.ToList();
        RefreshUI(); // Update immediately
    });

// Company changes permissions → UI updates in real-time
```

---

9. QUICK REFERENCE: Which Category?

Use this decision tree:

```
Question: "If Company A customizes this, does Company B need to recompile?"

  YES → HARDCODED (it's core logic)
   Examples: "Login flow", "Form rendering engine", "Permission checking algorithm"

  NO → DYNAMIC (it's configuration)
   Examples: "Manager permissions", "Approval chain", "Form fields", "Task statuses"

---

Question: "Can the admin change this in Firebase Console?"

  YES → DYNAMIC (it's metadata)
   Examples: "Designations", "Forms", "Workflows", "Notification templates"

  NO → HARDCODED (it's code)
   Examples: "Session management", "Encryption", "Screen types", "Task lifecycle states"

---

Question: "Does this change require rebuilding the application?"

  YES → HARDCODED
  NO → DYNAMIC
```

---

10. CONCRETE WALL-D EXAMPLE: Task Screen

What's Hardcoded?

```
// MyTasks.xaml.cs - MAUI code
public partial class MyTasksScreen : Page
{
    // HARDCODED: Screen type exists
    public MyTasksScreen() { }

    // HARDCODED: Screen loads assigned tasks from Firestore
    private async Task LoadTasksAsync()
    {
        var userId = _authService.CurrentUser.Id;
        var tasks = await _firestore.GetTasksAssignedToAsync(userId);
        TasksList.ItemsSource = tasks;
    }

    // HARDCODED: Task card layout (title, due date, status, assignee)
    // HARDCODED: Status indicators (color coding)
    // HARDCODED: Click handler to open task details
    // HARDCODED: Real-time listener for updates
}
```

What's Dynamic?

```
{
  "tenantId": "company_a",
  "screens": {
    "employee": {
      "name": "Employee Screen",
      "widgets": [
        {
          "type": "task_list",
          "title": "My Tasks",
          "showFields": ["title", "due_date", "priority", "status"],
          "sortBy": "due_date",
          "filterBy": ["assignee_id", "status"]
        }
      ]
    }
  }
}

// Company B version:
```

```
{
  "tenantId": "company_b",
  "screens": {
    "employee": {
      "name": "Employee Screen",
      "widgets": [
        {
          "type": "task_list",
          "title": "My Work Items",
          "showFields": ["title", "due_date", "priority", "status", "project_code", "cost_ce
          "sortBy": "priority",
          "filterBy": ["assignee_id", "status", "project_code"]
        }
      ]
    }
  }
}
```

Result: - Same task screen code for both companies - Company A sees 4 columns (title, due_date, priority, status) - Company B sees 6 columns (includes project_code, cost_center) - Different sort orders (Company A: due_date; Company B: priority) - Different filters available - Zero code changes needed

---

FINAL SUMMARY TABLE

Aspect	Hardcoded	Dynamic
Definition	Built into MAUI binary	Stored in Firestore metadata
Change Impact	Requires rebuild & redeploy	Instant update, no restart
Who Changes It	Developers (C#)	Company admins (JSON in Firebase)
Scope	Same for ALL companies	Different per company
Examples	Authentication, form rendering, task lifecycle	Designations, permissions, form fields, workflows
Backwards Compatibility	All companies must upgrade	Can be backwards compatible
Deployment	One binary → all get same update	One binary → all see different configs
Time to Change	Hours (code + test + build)	Seconds (JSON edit)
Testing	Automated + manual QA	Configuration validation only

Aspect	Hardcoded	Dynamic
Risk	High (affects all companies)	Low (affects one company)
Frequency	Monthly patches	Weekly or more

END OF DOCUMENT

This document serves as your multi-tenant architecture blueprint. When designing Wall-D features, always ask: "Is this hardcoded (core) or dynamic (config)?" Your answer determines deployment strategy, testing approach, and maintenance burden.