FPS (Financial Planning System) - Documentation Index

Complete Codebase Documentation Suite Version: v0.1.2.2
Last Updated: October 24, 2025

DOCUMENTS INCLUDED

This comprehensive documentation package consists of three interconnected documents:

1. CODEBASE_STRUCTURE.md (50 KB, 1,425 lines)

Purpose: Complete architecture overview and design patterns

Contents:

- Three-tier architecture diagram (Vue → Laravel → MySQL)
- Agent-based processing pattern explanation
- All 7 Agents with responsibilities and dependencies
- All 25+ API Controllers organized by module
- All 50+ Services organized by domain with detailed descriptions
- All 39 Models and their relationships
- Frontend structure (150+ Vue components, 25 views, 16 Vuex stores, 17 services)
- Module-by-module breakdown (Estate, Protection, Savings, Investment, Retirement)
- Complete data flow patterns with examples
- Cross-module integration patterns (ISA tracking, net worth, second death IHT)
- Key technologies and architectural patterns
- Test summary statistics

Best For: Understanding how the system works, learning architecture, design decisions

Key Diagrams:

- Complete request flow (Calculate IHT example)
- Cross-module asset aggregation
- Recommendation pipeline
- Vuex store flow
- Caching strategy
- Estate module complete chain
- Agent → Service → Model pattern

---

2. CODEBASE_FILE_MAP.md (34 KB, 1,063 lines)

Purpose: Quick reference for file locations and dependencies

Contents:

- Root directory structure
- Backend file locations (Agents, Controllers, Services, Models)
- Frontend file locations (Components, Views, Store, Services)
- Complete dependency relationships
- Estate module complete chain
- Cross-module asset aggregation flow
- Recommendation pipeline with code flow
- ISA allowance tracking (cross-module)
- Agent → Service → Model chain (Protection example)
- Key file responsibilities matrix
- Critical file paths for common tasks

Best For: Finding files, understanding code relationships, locating functionality

How to Use:

1. Need to add a feature? → Find the task in "Critical File Paths" section
2. Want to understand what a file does? → Check the dependency relationships
3. Need to find a controller? → Search "Backend File Locations"
4. Looking for a Vue component? → Search "Frontend File Locations"

---

3. CLAUDE.md (Project Instructions)

Purpose: Guidelines for working with the codebase

Already in repo: This file is committed and contains critical instructions:

- Application must remain fully functional
- Database backup protocol
- Available Claude AI skills
- Project overview, architecture, key details
- Coding standards (PSR-12, Vue 3, MySQL)

Key Sections:

- Module summary (Protection, Savings, Investment, Retirement, Estate)
- Asset ownership & spouse management
- Important UK rules (IHT, Pensions, ISAs)
- Dashboard & charts patterns
- Development workflow, testing, production

QUICK START GUIDE

Finding Information

Q: Where is the IHT calculation code? A:

1. → CODEBASE_FILE_MAP.md → "Estate Module Complete Chain"
2. → CODEBASE_STRUCTURE.md → "app/Services/Estate/IHTCalculator.php"

3. File location: `/Users/Chris/Desktop/fpsV2/app/Services/Estate/IHTCalculator.php`

Q: How does the protection module work? A:

1. → CODEBASE_STRUCTURE.md → "PROTECTION MODULE" section
2. See: Agent → Services → Models → Frontend flow
3. Key file: `app/Agents/ProtectionAgent.php`
4. Key service: `app/Services/Protection/CoverageGapAnalyzer.php`

Q: Where do I add a new estate asset type? A:

1. → CODEBASE_FILE_MAP.md → "Critical File Paths" section
2. Follow the step-by-step guide for "Add new data field to Estate"
3. Key files to modify: Migration, Model, Form Request, Controller, Service

Q: What's the complete data flow for calculating net worth? A:

1. → CODEBASE_STRUCTURE.md → "Cross-Module Integration" → "Net Worth Aggregation"
2. → CODEBASE_STRUCTURE.md → "DATA FLOW PATTERNS"
3. See: NetWorthService.generateNetWorth() coordinates all modules

Q: How does second death IHT planning work? A:

1. → CODEBASE_STRUCTURE.md → "SecondDeathIHTCalculator" in services
2. → CODEBASE_STRUCTURE.md → "CROSS-MODULE INTEGRATION" → "Second Death IHT Planning"
3. Key flow: SurvivingSpouseIHTPlanning.vue → estateService → SecondDeathIHTCalculator

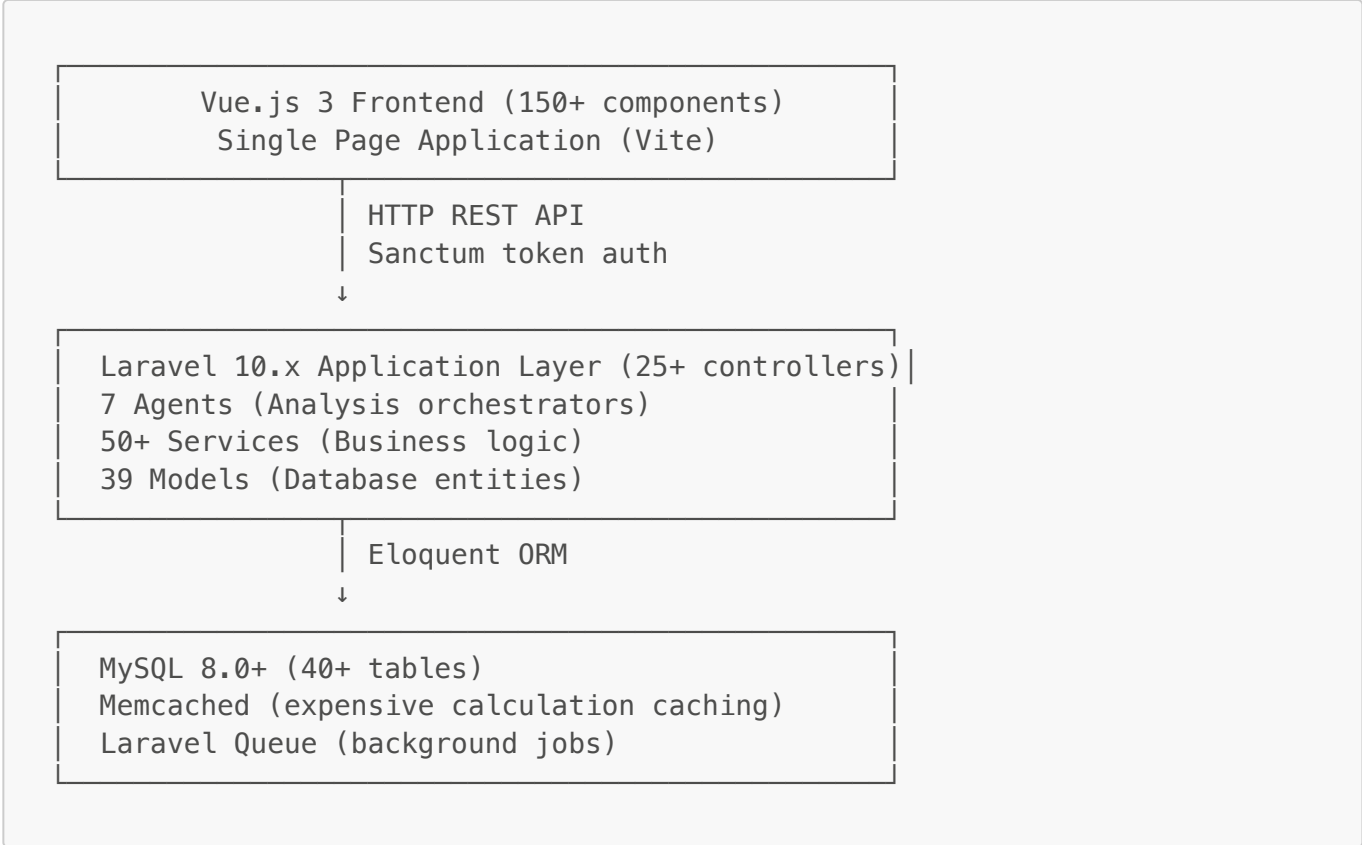
---

CODEBASE STATISTICS

Metric	Count
Total PHP Files	~400
Total Vue Components	150+
Total JavaScript Services	17
Total Vuex Stores	16
Database Models	39
Agent Classes	7 (1 base + 6 module-specific)
Service Classes	50+
API Controllers	25+
Form Request Classes	30+
Database Tables	40+
Backend Code	4,800+ lines
Frontend Code	15,000+ lines

Metric	Count
Documentation Files	3 (this + 2 detailed)

ARCHITECTURE AT A GLANCE



KEY ARCHITECTURAL PATTERNS

1. Agent Pattern

- Each module has an Agent (ProtectionAgent, SavingsAgent, etc.)
- Standardized interface: analyze() → generateRecommendations() → buildScenarios()
- Agents orchestrate multiple Services
- Results cached at agent level for performance

2. Service Layer Pattern

- Business logic encapsulated in Services
- Organized by domain (Estate, Protection, Savings, etc.)
- Services call Models, NOT controllers
- Testable and reusable
- Support dependency injection

3. Model Layer Pattern

- Eloquent relationships define data structure
- Type casting for type safety

- No business logic in models
- Foreign key relationships enforce data integrity

4. Controller Pattern

- Thin controllers (validation + service coordination)
- Form Requests validate input with custom rules
- Services/Agents do the heavy lifting
- Standardized JSON responses

5. Vuex Store Pattern

- State: Single source of truth per module
- Getters: Computed selectors (memoized)
- Actions: Async operations (API calls via services)
- Mutations: Synchronous state updates
- Clear naming: `module/ACTION_NAME`

6. Vue Component Pattern

- Smart components: Coordinate logic, call store/services
- Dumb components: Pure presentation, props + events
- Computed properties for reactive data
- Tabs for module dashboards

7. Service Pattern (JavaScript)

- Wrapper around axios API calls
- One service per module
- Consistent error handling via interceptors
- Bearer token automatically added to requests

MODULE BREAKDOWN QUICK REFERENCE

Module	Responsibility	Agent	Key Service	Primary Model
Protection	Life/CI/IP coverage analysis	ProtectionAgent	CoverageGapAnalyzer	ProtectionProfile, *Policy
Savings	Emergency fund, ISA tracking	SavingsAgent	ISATracker, EmergencyFundCalculator	SavingsAccount, SavingsGoal
Investment	Portfolio analysis, Monte Carlo	InvestmentAgent	PortfolioAnalyzer, MonteCarloSimulator	InvestmentAccount, Holding

Module	Responsibility	Agent	Key Service	Primary Model
Retirement	Pension planning, readiness	RetirementAgent	PensionProjector, AnnualAllowanceChecker	DCPension, DBPension, StatePension
Estate	IHT, gifting, net worth	EstateAgent	IHTCalculator, NetWorthAnalyzer	Asset, Liability, Gift, Trust
Holistic	Cross-module planning	CoordinatingAgent	HolisticPlanner, RecommendationsAggregator	All modules

CRITICAL CONFIGURATION

Tax Rules Configuration

Location: `config/uk_tax_config.php` (or database table `tax_configurations`)

Contains all 2025/26 UK tax rules:

- Income tax bands & allowances
- National Insurance thresholds
- ISA limits (£20,000 per year)
- Pension annual allowance (£60,000)
- IHT: NRB £325k, RNRB £175k, 40% rate
- PET/CLT gifting rules

Environment Configuration

Location: `.env` file

Critical settings:

- `VITE_API_BASE_URL` - Backend API URL
- Database credentials (MySQL)
- Cache driver (Memcached)
- Queue driver (database)
- Mail configuration

DEVELOPMENT WORKFLOW

Setup

```
composer install && npm install
cp .env.example .env && php artisan key:generate
php artisan migrate && php artisan db:seed --class=TaxConfigurationSeeder
```

Development Servers

```
php artisan serve              # Laravel (port 8000)
npm run dev                    # Vite (hot module reload)
php artisan queue:work         # Queue worker (if needed)
```

Testing

```
./vendor/bin/pest              # All tests
./vendor/bin/pest --testsuite=Unit    # Unit tests only
npm run test                   # Frontend tests
```

Database Operations

```
php artisan migrate            # Forward migrations (SAFE)
php artisan migrate:rollback   # Rollback last migration
# AVOID: migrate:fresh, migrate:refresh without backup
```

Production

```
composer install --optimize-autoloader --no-dev
npm run build
php artisan config:cache && php artisan route:cache
```

COMMON TASKS & FILE LOCATIONS

Add a New Feature to a Module

1. Define Database Schema

```
Location: database/migrations/YYYY_MM_DD_*.php
Example: Create table for new feature with proper types
```

2. Create Model

```
Location: app/Models/ or app/Models/Estate/
Include: $fillable array, relationships, type casting
```

3. Create Service

```
Location: app/Services/[Module]/[Feature]Service.php
Include: Business logic, dependency injection, caching
```

4. Update Agent

```
Location: app/Agents/[Module]Agent.php
Include: Call new service in analyze() method
```

5. Create Controller Endpoint

```
Location: app/Http/Controllers/Api/[Module]Controller.php
Include: Request validation, service call, JSON response
```

6. Create Form Request

```
Location: app/Http/Requests/[Module]/[Action]Request.php
Include: Validation rules, authorization
```

7. Add Route

```
Location: routes/api.php
Include: Proper verb (POST/PUT/DELETE), middleware
```

8. Create JS Service

```
Location: resources/js/services/[module]Service.js
Include: API call wrapper, error handling
```

9. Update Vuex Store

```
Location: resources/js/store/modules/[module].js
Include: State, getters, actions, mutations
```

10. Create Vue Components

```
Location: resources/js/components/[Module]/
Include: Form/list components, integrate with store
```

11. Update Dashboard

```
Location: resources/js/views/[Module]/[Module]Dashboard.vue
Include: Import components, add tab or section
```

KEY CODE EXAMPLES

Service Method Pattern

```php
// In app/Services/Estate/IHTCalculator.php
public function calculateIHTLiability(
    Collection $assets,
    IHTProfile $profile,
    ?Collection $gifts = null,
    ?Collection $trusts = null,
    float $liabilities = 0,
    ?Will $will = null,
    ?User $user = null
): array {
    // Calculate gross estate
    $grossEstateValue = $assets->sum('current_value');

    // Apply spouse exemption
    $spouseExemption = $this->calculateSpouseExemption($user, $will);

    // Calculate taxable estate
    $taxableEstate = max(0, $grossEstateValue - $liabilities -
$spouseExemption);

    // Calculate IHT at 40%
    $ihtLiability = $taxableEstate > $profile->available_nrb
        ? ($taxableEstate - $profile->available_nrb) * 0.40
        : 0;

    return [
        'gross_estate' => $grossEstateValue,
        'liabilities' => $liabilities,
        'spouse_exemption' => $spouseExemption,
        'taxable_estate' => $taxableEstate,
        'iht_liability' => $ihtLiability
    ];
}
```

Vue Component Pattern

```
<template>
  <div class="component">
```

```
    <h2>{{ title }}</h2>
    <form @submit.prevent="submitForm">
      <input v-model="formData.name" />
      <button type="submit">Save</button>
    </form>
  </div>
</template>

<script>
import estateService from '@/services/estateService';

export default {
  name: 'AssetForm',
  props: {
    asset: {
      type: Object,
      default: null
    }
  },
  data() {
    return {
      formData: this.asset || { name: '' }
    };
  },
  computed: {
    title() {
      return this.asset ? 'Edit Asset' : 'Add Asset';
    }
  },
  methods: {
    async submitForm() {
      try {
        const response = this.asset
          ? await estateService.updateAsset(this.asset.id, this.formData)
          : await estateService.storeAsset(this.formData);

        this.$emit('save', response.data);
        this.$emit('close');
      } catch (error) {
        console.error('Error saving asset:', error);
      }
    }
  }
};
</script>
```

Controller Pattern

```
// In app/Http/Controllers/Api/EstateController.php
public function storeAsset(StoreAssetRequest $request): JsonResponse
{
    $user = $request->user();
```

```
        $validated = $request->validated();

        try {
            $validated['user_id'] = $user->id;
            $asset = Asset::create($validated);

            // Invalidate cache after write
            Cache::forget("estate_analysis_{$user->id}");

            return response()->json([
                'success' => true,
                'message' => 'Asset created successfully',
                'data' => $asset,
            ], 201);
        } catch (\Exception $e) {
            return response()->json([
                'success' => false,
                'message' => 'Failed to create asset: '.$e->getMessage(),
            ], 500);
        }
    }
```

DEBUGGING & TROUBLESHOOTING

Check Database Integrity

```
# Verify all migrations applied
php artisan migrate:status

# Check user data
php artisan tinker
>>> User::with('protectionProfile', 'lifeInsurancePolicies')->first();
```

Test API Endpoints

```
# Login and get token
curl -X POST http://localhost:8000/api/auth/login \
  -d "email=user@example.com&password=password"

# Call endpoint with token
curl -X GET http://localhost:8000/api/estate \
  -H "Authorization: Bearer {token}"
```

Clear Cache

```
```

```
php artisan cache:clear
php artisan config:clear
php artisan route:clear
```

Check Logs

```
# Laravel logs
tail -f storage/logs/laravel.log

# Frontend errors (browser console)
# Check Network tab for 401/422/500 responses
```

IMPORTANT NOTES

Data Privacy

- All queries filtered by `user_id`
- Spouse permissions via `SpousePermission` model
- No cross-user data leakage

Performance

- Expensive calculations cached (1 hour default)
- ISA tracking uses efficient aggregation
- Monte Carlo runs as background job
- Indexes on `user_id` and foreign keys

UK-Specific Rules

- Tax year: April 6 - April 5
- ISA allowance: £20,000 per tax year
- Pension annual allowance: £60,000
- IHT nil rate band: £325,000 (£650,000 if spouse transfer)
- IHT rate: 40% (after using NRB)

Database Backup Protocol

CRITICAL: Before any destructive operation:

```
# Via admin panel: Create backup
# Verify file: storage/app/backups/*.sql

# Or via command: php artisan backup:create
```

NEXT STEPS

For New Developers

1. Read: CLAUDE.md (project overview)
2. Read: CODEBASE_STRUCTURE.md (architecture understanding)
3. Explore: Key files from CODEBASE_FILE_MAP.md
4. Run: `./vendor/bin/pest` to verify tests pass
5. Start: Pick a small task from the codebase

For New Features

1. Check: CODEBASE_FILE_MAP.md → "Critical File Paths"
2. Read: Relevant section in CODEBASE_STRUCTURE.md
3. Find: Similar existing feature as reference
4. Implement: Follow the step-by-step guide
5. Test: Write Pest tests for business logic
6. Document: Add comments to complex calculations

For Understanding Modules

1. Find: Module section in CODEBASE_STRUCTURE.md
2. Review: Agent class (app/Agents/[Module]Agent.php)
3. Review: Key Service (app/Services/[Module]/)
4. Review: Models (app/Models/)
5. Review: Vue Dashboard (resources/js/views/[Module]/)

SUPPORT

Documentation Files:

- `/Users/Chris/Desktop/fpsV2/CODEBASE_STRUCTURE.md` (50 KB, full architecture)
- `/Users/Chris/Desktop/fpsV2/CODEBASE_FILE_MAP.md` (34 KB, file locations & dependencies)
- `/Users/Chris/Desktop/fpsV2/CLAUDE.md` (project instructions, already in repo)

Questions to Answer:

- "How does [module] work?" → CODEBASE_STRUCTURE.md
- "Where is [functionality]?" → CODEBASE_FILE_MAP.md
- "What are the rules?" → CLAUDE.md
- "How do I add/modify [feature]?" → CODEBASE_FILE_MAP.md → Critical File Paths