SmartBudget Canada MVP Development Roadmap

Executive Summary

This roadmap organizes 25 backlog items into 8 manageable implementation chunks, prioritizing technical dependencies, user impact, and development efficiency. Each chunk delivers cohesive value while enabling subsequent development phases.

Total Scope: 25 backlog items → 8 implementation chunks (≤3 items per chunk)
Timeline Estimate: 8-12 weeks for full MVP completion
Architecture Focus: Guest-to-authenticated user journey with robust data persistence

---

Current State Analysis

✅ Working Systems

- **Authentication**: NextAuth with credentials provider
- **Budget Creation**: Multi-step wizard with tax calculations
- **Dashboard**: Comprehensive interface with AI coaching, net worth tracking
- **Database Schema**: Well-defined Prisma models with proper relationships
- **Calculator Framework**: Base calculator component with export functionality

⚠️ Technical Debt Identified

- **Prisma Singleton Issue**: Multiple client instances created across API routes
- **Guest User Gap**: No guest functionality despite business requirement
- **Testing Infrastructure**: Zero test coverage (Jest/Playwright missing)
- **Calculator Persistence**: Only works for authenticated users
- **Migration Path**: No guest-to-user data migration system

🎯 Business Impact Priority

1. **High**: Guest user experience (significant user acquisition blocker)
2. **High**: Data persistence reliability (core product functionality)
3. **Medium**: Testing infrastructure (development velocity & quality)
4. **Medium**: Performance optimizations (user experience refinement)

Implementation Chunks

📋 CHUNK 1: Foundation & Database Reliability

Items: [H-1] Prisma singleton, [H-2] Budget totals, [H-6] Secrets cleanup
Duration: 1-2 weeks
Dependencies: None - foundational work

Objective: Establish reliable database operations and clean technical debt

Technical Tasks:
- **[H-1]** Replace direct `new PrismaClient()` instantiations with centralized singleton from `lib/db.ts`
- **[H-2]** Audit and validate budget total calculations across all API endpoints
- **[H-6]** Remove hardcoded secrets, consolidate environment variable usage

Success Criteria:
- Single Prisma client instance used across entire application
- Budget calculations verified accurate across all user scenarios
- All sensitive data properly externalized to environment variables
- No database connection pool exhaustion under load

Risk Mitigation: This chunk must complete successfully before any subsequent database-dependent work.

🔐 CHUNK 2: Guest User Infrastructure

Items: [T-1] guestId cookie, [P-1] Guest→User migration (partial), [T-3] Prisma schema (guest extensions)
Duration: 1.5-2 weeks
Dependencies: CHUNK 1 (reliable database operations)

Objective: Enable guest users to use calculators and budget tools without authentication

Technical Tasks:
- **[T-1]** Implement secure guest ID cookie system with expiration and rotation
- **[P-1]** Design guest user data models and temporary storage strategy
- **[T-3]** Extend Prisma schema to support guest user associations (budgets, calculator results)

Implementation Details:

```
// Guest ID Cookie Strategy
- HttpOnly, Secure, SameSite=Strict cookies
- 30-day expiration with sliding renewal
- Cryptographically secure random IDs
- Cookie rotation on sensitive operations

// Schema Extensions
model Budget {
  userId    String?  // Make optional for guest support
  guestId   String?  // Add guest identifier
  // ... existing fields
  @@index([guestId]) // Index for guest queries
}
```

Success Criteria:
- Guest users can create budgets and use calculators
- Guest data properly isolated and queryable
- Cookie system secure and GDPR-compliant
- Database supports both authenticated and guest workflows

🔄 CHUNK 3: Migration & Data Persistence

Items: [T-2] /api/auth/migrate, [P-1] Guest→User migration (completion), [T-4] Save calc runs
Duration: 2-2.5 weeks
Dependencies: CHUNK 2 (guest infrastructure must exist)

Objective: Seamless data migration from guest to authenticated user experience

Technical Tasks:
- **[T-2]** Build `/api/auth/migrate` endpoint with robust data transfer logic
- **[P-1]** Complete guest-to-user migration UX flow with data preservation
- **[T-4]** Enhance calculator result saving for both guest and authenticated users

Migration API Design:

```
POST /api/auth/migrate
{
  guestId: string,
  newUserId: string,
  migrationStrategy: 'merge' | 'replace'
}

Response: {
  migratedBudgets: number,
  migratedCalculations: number,
  conflicts: ConflictResolution[],
  success: boolean
}
```

UX Flow:
1. Guest creates budgets/calculations → stored with guestId
2. User signs up → migration API automatically called
3. Guest data seamlessly appears in new user's dashboard
4. Guest session cleaned up after successful migration

Success Criteria:
- Zero data loss during guest-to-user transition
- Migration completes within 5 seconds for typical guest datasets
- Conflict resolution for duplicate data scenarios
- Comprehensive error handling and rollback capability

🧪 CHUNK 4: Testing Infrastructure & Quality Assurance

Items: [T-6] Jest+Playwright tests, [M-5] Tests/CI, [M-1] Validation
Duration: 2-3 weeks
Dependencies: CHUNK 3 (core functionality must be stable for testing)

Objective: Establish comprehensive testing infrastructure and validation

Technical Tasks:
- **[T-6]** Set up Jest for unit/integration tests + Playwright for E2E testing
- **[M-5]** Configure CI/CD pipeline with automated test execution
- **[M-1]** Implement comprehensive input validation across all forms and APIs

Testing Strategy:

```
// Test Coverage Targets
Unit Tests (Jest):
- Tax calculation accuracy (all provinces)
- Frequency conversion logic
- Budget total computations
- Guest-to-user migration logic

Integration Tests (Jest + Supertest):
- API endpoint functionality
- Database operations
- Authentication flows

E2E Tests (Playwright):
- Complete budget creation workflow
- Guest user journey → signup → migration
- Calculator usage and result saving
- Multi-device responsive testing
```

Validation Implementation:
- **Frontend**: Zod schemas for type-safe form validation
- **Backend**: Input sanitization and business rule validation
- **Database**: Constraint enforcement and data integrity checks

Success Criteria:
- >85% code coverage across core business logic
- All critical user journeys covered by E2E tests
- CI pipeline catches regressions before deployment
- Zero unvalidated user inputs reach database

📊 CHUNK 5: Calculator & Dashboard Enhancement

Items: [P-2] Calc-result orphaning, [T-5] Build /dashboard (guest support), [T-7] Analytics alias
Duration: 1.5-2 weeks
Dependencies: CHUNK 3 (migration must work for dashboard integration)

Objective: Complete calculator persistence and dashboard functionality for all user types

Technical Tasks:
- **[P-2]** Resolve calculator result orphaning by implementing proper cleanup and migration
- **[T-5]** Extend dashboard to support guest user data with appropriate UI adaptations
- **[T-7]** Implement analytics tracking with user/guest aliasing for data continuity

Dashboard Guest Adaptations:

```
// Guest Dashboard Features
- View saved calculations (session-based)
- Limited budget history (current session only)
- Upgrade prompts with migration preview
- Anonymous usage analytics
- Data export functionality

// Analytics Strategy
- Guest users: anonymous tracking with session continuity
- Migration event: alias guest session to authenticated user
- Retention metrics: guest→user conversion funnel
```

Success Criteria:
- Guest users have functional dashboard experience
- Calculator results never orphaned during user transitions
- Analytics provide clear guest→user conversion insights
- Dashboard performance acceptable with guest data queries

🔧 CHUNK 6: UI/UX Polish & User Experience

Items: [H-3] Mortgage debts, [H-4] Job-Income Back btn, [H-5] State sync
Duration: 1-1.5 weeks
Dependencies: CHUNK 4 (testing infrastructure helps validate fixes)

Objective: Address remaining user experience issues and workflow polish

Technical Tasks:
- **[H-3]** Investigate and resolve mortgage debt calculation/display issues
- **[H-4]** Ensure job income card has proper back button functionality (verify if already fixed)
- **[H-5]** Audit and fix state synchronization issues across budget creation flow

Focus Areas:

```
// State Sync Audit Points
- Budget wizard step transitions
- Item addition/removal in guided flow
- Tax calculation updates across components
- Couple budgeting state management
- Mobile responsive state handling

// Mortgage Debt Investigation
- Integration with budget calculations
- Tax deduction handling
- Monthly payment computations
- Debt-to-income ratio calculations
```

Success Criteria:
- Budget creation flow works flawlessly on all device sizes
- No state inconsistencies during user interactions
- Mortgage debt calculations accurate and well-integrated
- User feedback confirms improved workflow experience

⚡ CHUNK 7: Performance & Optimization

Items: [M-6] Icon tree-shake, [M-2] Frequency math, [M-7] Dark-mode tokens
Duration: 1-1.5 weeks
Dependencies: CHUNK 6 (core functionality stable for optimization)

Objective: Optimize application performance and enhance visual experience

Technical Tasks:
- **[M-6]** Implement tree-shaking for Lucide React icons to reduce bundle size
- **[M-2]** Optimize frequency conversion math for better performance and accuracy
- **[M-7]** Implement comprehensive dark mode with proper design tokens

Optimization Strategy:

```
// Icon Tree-Shaking
- Replace star imports: import * as Icons from 'lucide-react'
- Use selective imports: import { Calculator, Save } from 'lucide-react'
- Bundle analyzer to verify size reduction

// Frequency Math Optimization
- Memoize expensive calculations
- Cache conversion results
- Optimize decimal precision handling
- Benchmark before/after performance

// Dark Mode Implementation
- CSS custom properties for color tokens
- System preference detection
- Persistent user preference storage
- Smooth theme transitions
```

Success Criteria:
- >20% reduction in JavaScript bundle size
- Frequency calculations complete in <10ms for complex scenarios
- Dark mode provides excellent visual experience across all components
- Lighthouse performance score >90

🚀 CHUNK 8: Final Polish & Production Readiness

Items: [M-3] Tax-data path, [M-4] Accessibility, [M-8] Currency i18n, [M-9] Error sanitation
Duration: 1.5-2 weeks
Dependencies: All previous chunks (final integration and polish)

Objective: Complete production readiness with accessibility, internationalization, and error handling

Technical Tasks:
- **[M-3]** Optimize tax data loading and caching strategy for better performance
- **[M-4]** Implement comprehensive accessibility features (WCAG 2.1 compliance)
- **[M-8]** Add currency internationalization support (CAD focus with USD/EUR future)
- **[M-9]** Implement comprehensive error sanitization and user-friendly error messages

Production Readiness Checklist:

```
// Accessibility Implementation
- ARIA labels and roles for all interactive elements
- Keyboard navigation for entire application
- Screen reader compatibility testing
- Color contrast compliance (4.5:1 minimum)
- Focus management and visual indicators

// Currency Internationalization
- Intl.NumberFormat for proper currency formatting
- Provincial tax calculation currency handling
- Multi-currency calculation support infrastructure
- Locale-specific number input handling

// Error Handling Strategy
- Global error boundary implementation
- API error response standardization
- User-friendly error messages (no technical details)
- Error tracking and monitoring integration
- Graceful degradation for network issues
```

Success Criteria:
- Application passes WCAG 2.1 AA accessibility audit
- Currency handling accurate across all Canadian locales
- Error messages provide clear user guidance without exposing system details
- Production monitoring catches and tracks all error scenarios
- Application ready for public launch

Risk Mitigation & Contingency Plans

High-Risk Dependencies

1. **CHUNK 1→2**: Database reliability must be established before guest system
2. **CHUNK 2→3**: Guest infrastructure must work before migration can be built
3. **CHUNK 3→4**: Core functionality must be stable before comprehensive testing

Fallback Strategies

- **Guest System Complexity**: If guest-to-user migration proves overly complex, implement guest session export/import as interim solution
- **Testing Infrastructure**: If full E2E testing setup is delayed, prioritize unit tests and manual testing protocols
- **Performance Issues**: If optimization work uncovers fundamental architectural issues, prioritize stability over performance gains

Timeline Buffers

- Each chunk includes 20% time buffer for unexpected complexity
- Critical path items (CHUNKS 1-3) prioritized for early completion
- Polish items (CHUNKS 6-8) can be descoped if timeline pressures emerge

---

Success Metrics & Acceptance Criteria

Technical Metrics

- **Reliability**: Zero database connection issues under normal load
- **Performance**: <3 second page load times, <500ms API response times
- **Test Coverage**: >85% for critical business logic, 100% for financial calculations
- **Accessibility**: WCAG 2.1 AA compliance across all user-facing components

Business Metrics

- **Guest Conversion**: >30% of guest users convert to authenticated accounts
- **Data Integrity**: Zero data loss during guest-to-user migration
- **User Experience**: <2% drop-off rate in budget creation workflow
- **Error Rate**: <1% API error rate in production

User Acceptance Criteria

- Guest users can create budgets and use calculators without friction
- Account creation process preserves all guest data seamlessly
- Dashboard provides comprehensive financial management for all user types
- Application works excellently on mobile devices (primary user base)

Implementation Guidelines

Development Standards

- All database operations use centralized Prisma singleton
- Input validation implemented at both frontend and backend layers
- Error handling provides user-friendly messages without technical details
- Mobile-first responsive design for all new components
- TypeScript strict mode for all new code

Testing Requirements

- Unit tests for all financial calculation logic
- Integration tests for API endpoints
- E2E tests for critical user journeys
- Accessibility testing for all UI components
- Performance testing under simulated load

Code Review Process

- All chunks require architectural review before implementation
- Financial calculation changes require independent verification
- Security-sensitive code (auth, data migration) requires additional review
- Performance impact assessment for all optimization work

This roadmap provides a clear path to MVP completion while maintaining high quality standards and user experience excellence. Each chunk builds upon previous work while delivering independent value to users.