GST Compliance Dashboard - Code Documentation

Table of Contents

Architecture Overview

Technology Stack

- **Framework**: Next.js 14 with App Router and TypeScript
- **UI Library**: Material-UI v5 with custom theme
- **State Management**: Redux Toolkit with async thunks
- **Charts**: Recharts with MUI theme integration
- **Styling**: MUI sx prop + Tailwind CSS utilities
- **Forms**: React Hook Form + Zod validation
- **Development**: ESLint, Prettier, TypeScript strict mode

Design Principles

- **Component-First**: Reusable, composable components
- **Type Safety**: Comprehensive TypeScript coverage
- **Responsive Design**: Mobile-first approach
- **Accessibility**: MUI components with proper ARIA support
- **Performance**: Code splitting, memoization, lazy loading
- **Maintainability**: Clear file structure and naming conventions

---

Core Concepts

1. Layout System

Location: `src/components/layouts/`

The application uses a fixed sidebar layout system:

```
// Main layout wrapper
<DashboardLayout>
  {/* Page content */}
</DashboardLayout>
```

Key Components:

- `DashboardLayout.tsx`: Main layout wrapper
- `Sidebar.tsx`: Fixed navigation sidebar
- `TopNavigation.tsx`: App bar with user menu

How it works:

- **Desktop**: Fixed sidebar (280px) with permanent visibility
- **Mobile**: Temporary drawer overlay
- **Content**: Auto-adjusts margin for sidebar width
- **Scrolling**: Sidebar fixed, content scrolls independently

2. Widget System

Location: `src/components/ui/WidgetContainer.tsx`, `src/store/slices/widgetSlice.ts`

Configurable dashboard widgets with size, visibility, and refresh controls:

```
<WidgetContainer
  id="gst-liability"
  title="GST Liability Trend"
  size="large"
  onRefresh={refreshData}
>
  <GSTLiabilityChart data={chartData} />
</WidgetContainer>
```

Features:

- Size management (small, medium, large, full)
- Show/hide toggle
- Individual refresh functionality
- Settings menu for future customization

3. Step-by-Step Wizard Pattern

Location: `src/app/filing/gstr-1/components/FilingWizard.tsx`

Multi-step form pattern used throughout the application:

```
const steps = [
  { label: 'Upload', component: UploadStep },
  { label: 'Validate', component: ValidateStep },
  // ... more steps
];

// Current step component
<StepComponent
  data={filingData}
  onUpdate={updateFilingData}
  onNext={handleNext}
  onBack={handleBack}
/>
```

File Structure & Locations

Root Structure

```
src/
├── app/                    # Next.js App Router pages
|   ├── layout.tsx         # Root layout with providers
|   ├── page.tsx           # Homepage (redirects to dashboard)
|   ├── dashboard/         # Dashboard page
|   ├── filing/            # GST filing pages
|   ├── reconciliation/    # ITC reconciliation
|   ├── invoices/          # Invoice management
|   ├── analytics/         # Analytics and reports
|   ├── notifications/     # Notifications center
|   └── settings/          # Application settings
├── components/            # Reusable components
├── lib/                   # Utilities and configurations
├── store/                 # Redux state management
└── types/                 # TypeScript type definitions
```

Component Organization

```
components/
├── charts/              # Data visualization components
|   ├── GSTLiabilityChart.tsx
|   ├── ITCUtilizationChart.tsx
|   ├── ComplianceScoreChart.tsx
|   └── FilingStatusChart.tsx
├── forms/               # Form input components
|   ├── FormInput.tsx
|   ├── FormSelect.tsx
|   ├── FormDatePicker.tsx
|   ├── FormTextarea.tsx
|   └── FormFileUpload.tsx
├── layouts/             # Layout components
|   ├── DashboardLayout.tsx
|   ├── Sidebar.tsx
|   ├── TopNavigation.tsx
|   └── AuthLayout.tsx
├── providers/           # Context providers
|   ├── ThemeProvider.tsx
|   ├── ReduxProvider.tsx
|   └── index.tsx
└── ui/                  # UI utility components
    ├── WidgetContainer.tsx
    ├── LoadingSpinner.tsx
    ├── LoadingSkeleton.tsx
    ├── ErrorBoundary.tsx
    └── Breadcrumbs.tsx
```

Component Systems

1. Chart Components

Location: `src/components/charts/`

Interactive chart components using Recharts with MUI theme integration.

GSTLiabilityChart.tsx

```tsx
interface GSTLiabilityChartProps {
  data: Array<{
    month: string;
    liability: number;
    paid: number;
  }>;
}

// Usage
<GSTLiabilityChart data={liabilityData} />
```

Features:

- Line chart showing liability vs payments
- Responsive design with custom tooltips
- MUI theme color integration
- Indian currency formatting

ComplianceScoreChart.tsx

```tsx
interface ComplianceData {
  name: string;
  value: number;
  color: string;
}

// Usage
<ComplianceScoreChart data={complianceBreakdown} />
```

Features:

- Pie chart for compliance breakdown
- Color-coded segments
- Interactive hover states
- Percentage display

2. Form Components

Location: `src/components/forms/`

Reusable form components with consistent styling and validation.

FormInput.tsx

```
interface FormInputProps {
  name: string;
  label: string;
  required?: boolean;
  error?: string;
  // ... other props
}

// Usage
<FormInput
  name="invoiceNumber"
  label="Invoice Number"
  required
  error={errors.invoiceNumber}
  register={register}
/>
```

Features:

- React Hook Form integration
- Error state handling
- Consistent MUI styling
- Required field indicators

FormFileUpload.tsx

```
interface FormFileUploadProps {
  onFilesChange: (files: File[]) => void;
  acceptedFileTypes: string[];
  maxFiles: number;
  maxSizeInMB: number;
}

// Usage
<FormFileUpload
  onFilesChange={handleFilesUpload}
  acceptedFileTypes={['.csv', '.xlsx']}
  maxFiles={1}
  maxSizeInMB={10}
/>
```

Features:

- Drag and drop functionality
- File type validation
- Size limit enforcement
- Progress indicators

State Management

Redux Store Structure

Location: `src/store/`

```
store/
├── index.ts              # Store configuration
└── slices/
    ├── dashboardSlice.ts # Dashboard data and KPIs
    └── widgetSlice.ts    # Widget configuration state
```

Dashboard Slice

Location: `src/store/slices/dashboardSlice.ts`

Manages dashboard data, KPIs, charts, and notifications.

```
interface DashboardState {
  kpis: DashboardKPIs;
  chartData: ChartData;
  notifications: Notification[];
  loading: boolean;
  error: string | null;
}

// Actions
export const fetchDashboardData = createAsyncThunk(
  'dashboard/fetchData',
  async (scenario?: MockScenario) => {
    const response = await mockApi.getDashboardData(scenario);
    return response.data;
  }
);

// Usage in components
const { kpis, loading, error } = useAppSelector(state => state.dashboard);
const dispatch = useAppDispatch();

useEffect(() => {
  dispatch(fetchDashboardData());
}, [dispatch]);
```

Widget Slice

Location: `src/store/slices/widgetSlice.ts`

Manages widget visibility, sizing, and configuration.

```
interface WidgetConfig {
  id: string;
  visible: boolean;
  size: 'small' | 'medium' | 'large' | 'full';
  refreshing: boolean;
}

// Actions
export const toggleWidgetVisibility = (id: string) => { /* ... */ };
export const updateWidgetSize = (id: string, size: WidgetSize) => { /* ... */ };
export const setWidgetRefreshing = (id: string, refreshing: boolean) => { /* ... */ };

// Usage
const widgetConfig = useAppSelector(state =>
  selectWidgetById(state, 'gst-liability')
);
```

GSTR-1 Filing System

Architecture Overview

Location: `src/app/filing/gstr-1/components/`

The GSTR-1 filing system follows a wizard pattern with 5 distinct steps:

```
FilingWizard.tsx          # Main wizard coordinator
steps/
├── UploadStep.tsx        # CSV/Excel file upload
├── ValidateStep.tsx      # Data validation and correction
├── CategorizeStep.tsx    # B2B/B2C/Export categorization
├── PreviewStep.tsx       # Summary and review
└── SubmitStep.tsx        # Submission and acknowledgment
```

Main Wizard Component

Location: `src/app/filing/gstr-1/components/FilingWizard.tsx`

Central coordinator managing wizard state and step navigation.

```tsx
interface FilingData {
  invoices: InvoiceData[];
  summary: {
    totalInvoices: number;
    totalValue: number;
    totalTax: number;
    validatedCount: number;
    errorCount: number;
  };
  currentStep: number;
  isSubmitted: boolean;
}

export const FilingWizard = ({ onComplete }: FilingWizardProps) => {
  const [currentStep, setCurrentStep] = useState(0);
  const [filingData, setFilingData] = useState<FilingData>({
    invoices: [],
    summary: { /* ... */ },
    currentStep: 0,
    isSubmitted: false
  });

  // Step components
  const steps = [
    { label: 'Upload Invoices', component: UploadStep },
    { label: 'Validate Data', component: ValidateStep },
    { label: 'Categorize Transactions', component: CategorizeStep },
    { label: 'Preview Return', component: PreviewStep },
    { label: 'Submit Return', component: SubmitStep }
  ];

  return (
    <Box>
      {/* Progress indicator */}
      <Stepper activeStep={currentStep}>
        {steps.map(step => (
          <Step key={step.label}>
            <StepLabel>{step.label}</StepLabel>
          </Step>
        ))}
      </Stepper>

      {/* Current step component */}
      <StepComponent
        data={filingData}
        onUpdate={updateFilingData}
        onNext={handleNext}
        onBack={handleBack}
      />
    </Box>
  );
};
```

Step Components

1. UploadStep.tsx

Purpose: File upload with CSV/Excel parsing and validation

Key Features:

```tsx
// File upload with drag-and-drop
<FormFileUpload
  onFilesChange={handleFileUpload}
  acceptedFileTypes={['.csv', '.xlsx']}
  maxFiles={1}
  maxSizeInMB={10}
/>

// Template download
const downloadTemplate = () => {
  const csvContent = [
    'Invoice Number,Date,Customer Name,Customer GSTIN,Taxable Value,CGST Rate,CGST Amount,SGST Rate,SGST Amount,IGST R
    'INV-001,2024-01-15,ABC Traders,27AAAAA0000A1Z5,10000,9,900,9,900,0,0,11800,1234'
  ].join('\\n');

  const blob = new Blob([csvContent], { type: 'text/csv' });
  const url = window.URL.createObjectURL(blob);
  const a = document.createElement('a');
  a.href = url;
  a.download = 'GSTR1_Invoice_Template.csv';
  a.click();
};

// CSV parsing
const parseCSV = (content: string): InvoiceData[] => {
  const lines = content.split('\\n');
  const headers = lines[0].split(',');

  return lines.slice(1)
    .filter(line => line.trim())
    .map((line, index) => {
      const values = line.split(',');
      return {
        id: `invoice-${index + 1}`,
        invoiceNumber: values[0],
        invoiceDate: values[1],
        customerName: values[2],
        customerGSTIN: values[3],
        // ... other fields
      };
    });
};
```

2. ValidateStep.tsx

Purpose: Comprehensive data validation with auto-fix capabilities

Validation Rules:

```
const validateInvoice = (invoice: InvoiceData): ValidationError[] => {
  const errors: ValidationError[] = [];

  // Invoice number validation
  if (!invoice.invoiceNumber) {
    errors.push({
      field: 'invoiceNumber',
      type: 'required',
      message: 'Invoice number is required',
      severity: 'error'
    });
  }

  // GSTIN validation
  if (invoice.customerGSTIN && !validateGSTIN(invoice.customerGSTIN)) {
    errors.push({
      field: 'customerGSTIN',
      type: 'format',
      message: 'Invalid GSTIN format',
      severity: 'error',
      autoFix: true
    });
  }

  // Amount validation
  const calculatedTotal = invoice.taxableValue + invoice.cgstAmount +
                          invoice.sgstAmount + invoice.igstAmount;
  if (Math.abs(calculatedTotal - invoice.invoiceValue) > 0.01) {
    errors.push({
      field: 'invoiceValue',
      type: 'calculation',
      message: 'Total amount mismatch with tax calculations',
      severity: 'warning',
      autoFix: true
    });
  }

  return errors;
};

// Auto-fix functionality
const autoFixErrors = (invoices: InvoiceData[]): InvoiceData[] => {
  return invoices.map(invoice => {
    const errors = validateInvoice(invoice);
    let fixedInvoice = { ...invoice };

    errors.forEach(error => {
      if (!error.autoFix) return;

      switch (error.type) {
        case 'format':
          if (error.field === 'customerGSTIN') {
            fixedInvoice.customerGSTIN = formatGSTIN(invoice.customerGSTIN);
          }
          break;
        case 'calculation':
          const recalculatedTotal = fixedInvoice.taxableValue +
                                    fixedInvoice.cgstAmount +
                                    fixedInvoice.sgstAmount +
                                    fixedInvoice.igstAmount;
          fixedInvoice.invoiceValue = recalculatedTotal;
          break;
      }
    });

    return fixedInvoice;
  });
};
```

3. CategorizeStep.tsx

Purpose: Categorize invoices as B2B, B2C, Export, or Nil-rated

Auto-categorization Logic:

```
  const autoCategorize = (invoices: InvoiceData[]): InvoiceData[] => {
    return invoices.map(invoice => {
      if (invoice.category) return invoice; // Skip if already categorized

      let suggestedCategory: InvoiceCategory = 'B2B';

      // If customer has GSTIN, it's B2B
      if (invoice.customerGSTIN && invoice.customerGSTIN.trim() !== '') {
        suggestedCategory = 'B2B';
      }
      // If invoice value is high (>50000), likely B2B
      else if (invoice.invoiceValue > 50000) {
        suggestedCategory = 'B2B';
      }
      // If no GST charged, likely Nil rated
      else if (invoice.cgstAmount === 0 && invoice.sgstAmount === 0 && invoice.igstAmount === 0) {
        suggestedCategory = 'NilRated';
      }
      // Otherwise B2C
      else {
        suggestedCategory = 'B2C';
      }

      return { ...invoice, category: suggestedCategory };
    });
  };

  // HSN code suggestions with auto-GST calculation
  const handleHSNSuggestion = (invoiceId: string, hsnCode: string) => {
    const hsnDetails = getHSNCodeDetails(hsnCode);
    const suggestedGSTRate = getSuggestedGSTRate(hsnCode);

    const updatedInvoices = invoices.map(inv => {
      if (inv.id === invoiceId) {
        const taxableValue = inv.taxableValue;
        const gstRate = suggestedGSTRate / 100;
        const totalGst = taxableValue * gstRate;
        const cgst = totalGst / 2;
        const sgst = totalGst / 2;

        return {
          ...inv,
          hsnCode,
          description: hsnDetails?.description || inv.description,
          cgstAmount: cgst,
          sgstAmount: sgst,
          igstAmount: 0, // Assuming intrastate
          invoiceValue: taxableValue + totalGst
        };
      }
      return inv;
    });
  };
```

4. PreviewStep.tsx

Purpose: Final review with comprehensive summaries

Summary Calculations:

```
  const summary = useMemo(() => {
    const categorySummary = {
      B2B: { count: 0, taxableValue: 0, cgst: 0, sgst: 0, igst: 0, total: 0 },
      B2C: { count: 0, taxableValue: 0, cgst: 0, sgst: 0, igst: 0, total: 0 },
      Export: { count: 0, taxableValue: 0, cgst: 0, sgst: 0, igst: 0, total: 0 },
      NilRated: { count: 0, taxableValue: 0, cgst: 0, sgst: 0, igst: 0, total: 0 }
    };

    data.invoices.forEach(invoice => {
      const category = invoice.category || 'B2B';
      if (category in categorySummary) {
        const cat = categorySummary[category];
        cat.count++;
        cat.taxableValue += invoice.taxableValue;
        cat.cgst += invoice.cgstAmount;
        cat.sgst += invoice.sgstAmount;
        cat.igst += invoice.igstAmount;
        cat.total += invoice.invoiceValue;
      }
    });

    // HSN-wise summary
    const hsnSummary = data.invoices.reduce((acc, invoice) => {
      const key = invoice.hsnCode;
      if (!acc[key]) {
        acc[key] = {
          description: invoice.description,
          quantity: 0,
          taxableValue: 0,
          taxAmount: 0
        };
      }
      acc[key].quantity += invoice.quantity;
      acc[key].taxableValue += invoice.taxableValue;
      acc[key].taxAmount += invoice.cgstAmount + invoice.sgstAmount + invoice.igstAmount;
      return acc;
    }, {});

    return { categorySummary, hsnSummary };
  }, [data.invoices]);
```

5. **SubmitStep.tsx**

Purpose: Multi-stage submission with progress tracking

Submission Process:

```
const submissionSteps = [
  { label: 'Data Validation', description: 'Validating invoice data and calculations' },
  { label: 'Format Conversion', description: 'Converting to GST portal format' },
  { label: 'Digital Signature', description: 'Applying digital signature' },
  { label: 'Portal Submission', description: 'Submitting to GST portal' },
  { label: 'Acknowledgment', description: 'Receiving filing acknowledgment' }
];

const handleSubmit = async () => {
  setLoading(true);
  setSubmissionError('');

  try {
    // Step 1: Data Validation
    setCurrentStep(0);
    await new Promise(resolve => setTimeout(resolve, 1500));

    // Step 2: Format Conversion
    setCurrentStep(1);
    await new Promise(resolve => setTimeout(resolve, 2000));

    // Step 3: Digital Signature
    setCurrentStep(2);
    await new Promise(resolve => setTimeout(resolve, 1000));

    // Step 4: Portal Submission
    setCurrentStep(3);
    await new Promise(resolve => setTimeout(resolve, 3000));

    // Simulate random failure for demo
    if (Math.random() < 0.1) {
      throw new Error('Portal connection timeout. Please try again.');
    }

    // Step 5: Acknowledgment
    setCurrentStep(4);
    await new Promise(resolve => setTimeout(resolve, 1000));

    const ackNumber = `GST-${Date.now().toString().slice(-8)}`;
    setAcknowledgmentNumber(ackNumber);
    setIsSubmitted(true);
    setShowSuccessDialog(true);

  } catch (error) {
    setSubmissionError(error.message);
    setCurrentStep(-1);
  } finally {
    setLoading(false);
  }
};

// Acknowledgment download
const downloadAcknowledgment = () => {
  const acknowledgment = `
GST RETURN FILING ACKNOWLEDGMENT
================================

GSTIN: 27AAAAA0000A1Z5
Return Period: ${new Date().toLocaleString('en-IN', { month: 'long', year: 'numeric' })}
Return Type: GSTR-1
Filing Date: ${new Date().toLocaleDateString('en-IN')}
Acknowledgment Number: ${acknowledgmentNumber}

SUMMARY:
--------
Total Invoices: ${data.summary.totalInvoices}
Total Taxable Value: ${formatCurrency(data.summary.totalValue)}
Total Tax: ${formatCurrency(data.summary.totalTax)}

Status: Successfully Filed
  `.trim();

  const blob = new Blob([acknowledgment], { type: 'text/plain' });
  const url = window.URL.createObjectURL(blob);
  const a = document.createElement('a');
  a.href = url;
  a.download = `GSTR1_Acknowledgment_${acknowledgmentNumber}.txt`;
  a.click();
  window.URL.revokeObjectURL(url);
};
```

HSN Code Database

Structure

Location: `src/lib/data/hsn-codes.ts`

Comprehensive HSN (Harmonized System of Nomenclature) code database with 60+ codes.

```typescript
export interface HSNCode {
  code: string;           // HSN code (e.g., "1001")
  description: string;    // Product description
  category: 'B2B' | 'B2C' | 'Export' | 'NilRated';
  gstRate: number;       // GST rate percentage
  chapter: string;       // HSN chapter name
  subheading?: string;   // Optional subheading
}

export const hsnCodes: HSNCode[] = [
  // Chapter 10: Cereals
  {
    code: '1001',
    description: 'Wheat and meslin',
    category: 'B2B',
    gstRate: 0,
    chapter: 'Cereals'
  },
  // Chapter 84: Machinery
  {
    code: '8471',
    description: 'Automatic data processing machines and units',
    category: 'B2B',
    gstRate: 18,
    chapter: 'Machinery'
  },
  // Services
  {
    code: '9961',
    description: 'Information technology software services',
    category: 'B2B',
    gstRate: 18,
    chapter: 'Services'
  }
];
```

Utility Functions

```typescript
// Search HSN codes by query
export const searchHSNCodes = (query: string): HSNCode[] => {
  const lowercaseQuery = query.toLowerCase();
  return hsnCodes.filter(hsn =>
    hsn.code.includes(query) ||
    hsn.description.toLowerCase().includes(lowercaseQuery) ||
    hsn.chapter.toLowerCase().includes(lowercaseQuery)
  );
};

// Get specific HSN code details
export const getHSNCodeDetails = (code: string): HSNCode | undefined => {
  return hsnCodes.find(hsn => hsn.code === code);
};

// Get suggested GST rate for HSN code
export const getSuggestedGSTRate = (hsnCode: string): number => {
  const hsn = getHSNCodeDetails(hsnCode);
  return hsn?.gstRate || 18; // Default to 18% if not found
};

// Get all available chapters
export const getChapters = (): string[] => {
  return Array.from(new Set(hsnCodes.map(hsn => hsn.chapter))).sort();
};

// Get HSN codes by category
export const getHSNCodesByCategory = (category: string): HSNCode[] => {
  return hsnCodes.filter(hsn => hsn.category === category);
};
```

Usage in Components

```
// In CategorizeStep.tsx
const CategoryizeStep = () => {
  return (
    <Autocomplete
      options={hsnCodes.map(hsn => hsn.code)}
      filterOptions={(options, { inputValue }) => {
        if (!inputValue) return options.slice(0, 20);
        const filtered = searchHSNCodes(inputValue);
        return filtered.map(hsn => hsn.code).slice(0, 10);
      }}
      renderOption={(props, option) => {
        const hsn = getHSNCodeDetails(option);
        return (
          <li {...props}>
            <Box>
              <Typography variant="body2">{option}</Typography>
              <Typography variant="caption" color="text.secondary">
                {hsn?.description}
              </Typography>
              <Chip label={`${hsn?.gstRate}%`} size="small" />
            </Box>
          </li>
        );
      }}
      onChange={(e, value) => handleHSNSuggestion(invoiceId, value)}
    />
  );
};
```

Data Validation Engine

Validation Types

Location: Various step components and utility functions

The validation engine supports multiple validation types with auto-fix capabilities.

Validation Error Structure

```
interface ValidationError {
  field: string;           // Field name with error
  type: 'required' | 'format' | 'calculation' | 'range' | 'duplicate';
  message: string;         // Human-readable error message
  severity: 'error' | 'warning' | 'info';
  autoFix?: boolean;       // Whether error can be auto-fixed
  suggestedValue?: any;    // Suggested correction value
}
```

Core Validation Rules

```typescript
// 1. Required Field Validation
const validateRequired = (value: any, fieldName: string): ValidationError[] => {
  if (!value || (typeof value === 'string' && value.trim() === '')) {
    return [{
      field: fieldName,
      type: 'required',
      message: `${fieldName} is required`,
      severity: 'error'
    }];
  }
  return [];
};

// 2. GSTIN Format Validation
const validateGSTIN = (gstin: string): ValidationError[] => {
  const gstinRegex = /^[0-9]{2}[A-Z]{5}[0-9]{4}[A-Z]{1}[1-9A-Z]{1}Z[0-9A-Z]{1}$/;

  if (gstin && !gstinRegex.test(gstin)) {
    return [{
      field: 'customerGSTIN',
      type: 'format',
      message: 'Invalid GSTIN format',
      severity: 'error',
      autoFix: true,
      suggestedValue: formatGSTIN(gstin)
    }];
  }
  return [];
};

// 3. Amount Calculation Validation
const validateAmounts = (invoice: InvoiceData): ValidationError[] => {
  const errors: ValidationError[] = [];

  const calculatedTotal = invoice.taxableValue + invoice.cgstAmount +
                          invoice.sgstAmount + invoice.igstAmount;

  if (Math.abs(calculatedTotal - invoice.invoiceValue) > 0.01) {
    errors.push({
      field: 'invoiceValue',
      type: 'calculation',
      message: `Total amount mismatch. Expected: ${calculatedTotal}`,
      severity: 'warning',
      autoFix: true,
      suggestedValue: calculatedTotal
    });
  }

  // GST rate validation
  const gstRate = ((invoice.cgstAmount + invoice.sgstAmount + invoice.igstAmount) /
                   invoice.taxableValue) * 100;

  const validGSTRates = [0, 5, 12, 18, 28];
  const closestRate = validGSTRates.reduce((prev, curr) =>
    Math.abs(curr - gstRate) < Math.abs(prev - gstRate) ? curr : prev
  );

  if (Math.abs(gstRate - closestRate) > 1) {
    errors.push({
      field: 'gstRate',
      type: 'calculation',
      message: `GST rate appears incorrect. Current: ${gstRate.toFixed(2)}%, Suggested: ${closestRate}%`,
      severity: 'warning',
      autoFix: true
    });
  }

  return errors;
};

// 4. Date Validation
const validateDate = (dateStr: string, fieldName: string): ValidationError[] => {
  const errors: ValidationError[] = [];

  if (!dateStr) return validateRequired(dateStr, fieldName);

  const date = new Date(dateStr);
  const now = new Date();

  if (isNaN(date.getTime())) {
    errors.push({
      field: fieldName,
      type: 'format',
      message: 'Invalid date format'
```

```
        message: 'Invalid date format',
        severity: 'error'
      });
    } else if (date > now) {
      errors.push({
        field: fieldName,
        type: 'range',
        message: 'Date cannot be in the future',
        severity: 'error'
      });
    }

    return errors;
  };

  // 5. Duplicate Detection
  const validateDuplicates = (invoices: InvoiceData[]): ValidationError[] => {
    const errors: ValidationError[] = [];
    const invoiceNumbers = new Set<string>();

    invoices.forEach((invoice, index) => {
      if (invoiceNumbers.has(invoice.invoiceNumber)) {
        errors.push({
          field: 'invoiceNumber',
          type: 'duplicate',
          message: `Duplicate invoice number: ${invoice.invoiceNumber}`,
          severity: 'error'
        });
      }
      invoiceNumbers.add(invoice.invoiceNumber);
    });

    return errors;
  };
```

Auto-Fix Functions

```typescript
// Format GSTIN with proper structure
const formatGSTIN = (gstin: string): string => {
  if (!gstin) return gstin;

  const cleaned = gstin.replace(/[^A-Z0-9]/g, '').toUpperCase();
  if (cleaned.length === 15) {
    return cleaned.substring(0, 2) + cleaned.substring(2, 7) +
           cleaned.substring(7, 11) + cleaned.substring(11, 12) +
           cleaned.substring(12, 13) + 'Z' + cleaned.substring(13, 15);
  }
  return cleaned;
};

// Fix amount calculations
const fixAmountCalculations = (invoice: InvoiceData): InvoiceData => {
  const totalTax = invoice.cgstAmount + invoice.sgstAmount + invoice.igstAmount;
  const correctedTotal = invoice.taxableValue + totalTax;

  return {
    ...invoice,
    invoiceValue: correctedTotal
  };
};

// Auto-fix all fixable errors
const autoFixInvoice = (invoice: InvoiceData): InvoiceData => {
  let fixedInvoice = { ...invoice };
  const errors = validateInvoice(invoice);

  errors.forEach(error => {
    if (!error.autoFix) return;

    switch (error.type) {
      case 'format':
        if (error.field === 'customerGSTIN') {
          fixedInvoice.customerGSTIN = formatGSTIN(invoice.customerGSTIN);
        }
        break;
      case 'calculation':
        if (error.field === 'invoiceValue') {
          fixedInvoice = fixAmountCalculations(fixedInvoice);
        }
        break;
    }
  });

  return fixedInvoice;
};
```

UI/UX Components

Loading & Error States

Location: src/components/ui/

LoadingSpinner.tsx

```typescript
interface LoadingSpinnerProps {
  size?: 'small' | 'medium' | 'large';
  message?: string;
}

export const LoadingSpinner = ({ size = 'medium', message }: LoadingSpinnerProps) => {
  const sizeMap = { small: 20, medium: 40, large: 60 };

  return (
    <Box display="flex" flexDirection="column" alignItems="center" gap={2}>
      <CircularProgress size={sizeMap[size]} />
      {message && (
        <Typography variant="body2" color="text.secondary">
          {message}
        </Typography>
      )}
    </Box>
  );
};
```

LoadingSkeleton.tsx

```tsx
interface LoadingSkeletonProps {
  variant: 'dashboard' | 'list' | 'form' | 'table';
  count?: number;
}

export const LoadingSkeleton = ({ variant, count = 1 }: LoadingSkeletonProps) => {
  const renderSkeleton = () => {
    switch (variant) {
      case 'dashboard':
        return (
          <Grid container spacing={3}>
            {Array.from({ length: 4 }).map((_, i) => (
              <Grid item xs={12} sm={6} md={3} key={i}>
                <Card>
                  <CardContent>
                    <Skeleton variant="text" width="60%" />
                    <Skeleton variant="text" width="40%" />
                    <Skeleton variant="rectangular" height={60} />
                  </CardContent>
                </Card>
              </Grid>
            ))}
          </Grid>
        );

      case 'table':
        return (
          <TableContainer>
            <Table>
              <TableHead>
                <TableRow>
                  {Array.from({ length: 5 }).map((_, i) => (
                    <TableCell key={i}>
                      <Skeleton variant="text" width="80%" />
                    </TableCell>
                  ))}
                </TableRow>
              </TableHead>
              <TableBody>
                {Array.from({ length: count }).map((_, i) => (
                  <TableRow key={i}>
                    {Array.from({ length: 5 }).map((_, j) => (
                      <TableCell key={j}>
                        <Skeleton variant="text" width="90%" />
                      </TableCell>
                    ))}
                  </TableRow>
                ))}
              </TableBody>
            </Table>
          </TableContainer>
        );

      default:
        return <Skeleton variant="text" />;
    }
  };

  return <>{renderSkeleton()}</>;
};
```

ErrorBoundary.tsx

```tsx
interface ErrorBoundaryState {
  hasError: boolean;
  error?: Error;
}

export class ErrorBoundary extends Component<
  PropsWithChildren<{}>,
  ErrorBoundaryState
> {
  constructor(props: PropsWithChildren<{}>) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error: Error): ErrorBoundaryState {
    return { hasError: true, error };
  }

  componentDidCatch(error: Error, errorInfo: ErrorInfo) {
    console.error('Error caught by boundary:', error, errorInfo);
  }

  render() {
    if (this.state.hasError) {
      return (
        <Container maxWidth="sm" sx={{ py: 8 }}>
          <Card>
            <CardContent sx={{ textAlign: 'center', p: 4 }}>
              <ErrorOutlineIcon sx={{ fontSize: 64, color: 'error.main', mb: 2 }} />
              <Typography variant="h5" gutterBottom>
                Something went wrong
              </Typography>
              <Typography variant="body1" color="text.secondary" paragraph>
                We're sorry, but something unexpected happened. Please try refreshing the page.
              </Typography>
              <Button
                variant="contained"
                onClick={() => window.location.reload()}
                startIcon={<RefreshIcon />}
              >
                Refresh Page
              </Button>
            </CardContent>
          </Card>
        </Container>
      );
    }

    return this.props.children;
  }
}
```

Navigation Components

Breadcrumbs.tsx

```tsx
interface BreadcrumbsProps {
  items?: Array<{
    label: string;
    href?: string;
  }>;
}

export const Breadcrumbs = ({ items }: BreadcrumbsProps) => {
  const pathname = usePathname();

  const breadcrumbItems = items || generateBreadcrumbs(pathname);

  return (
    <MuiBreadcrumbs aria-label="breadcrumb" sx={{ mb: 2 }}>
      <Link
        color="inherit"
        href="/dashboard"
        sx={{ display: 'flex', alignItems: 'center' }}
      >
        <HomeIcon sx={{ mr: 0.5 }} fontSize="inherit" />
        Dashboard
      </Link>

      {breadcrumbItems.map((item, index) => {
        const isLast = index === breadcrumbItems.length - 1;

        return isLast ? (
          <Typography key={item.label} color="text.primary">
            {item.label}
          </Typography>
        ) : (
          <Link key={item.label} color="inherit" href={item.href}>
            {item.label}
          </Link>
        );
      })}
    </MuiBreadcrumbs>
  );
};

const generateBreadcrumbs = (pathname: string) => {
  const pathSegments = pathname.split('/').filter(Boolean);

  const breadcrumbs = pathSegments.map((segment, index) => {
    const href = '/' + pathSegments.slice(0, index + 1).join('/');
    const label = segment.charAt(0).toUpperCase() +
                  segment.slice(1).replace(/-/g, ' ');

    return { label, href };
  });

  return breadcrumbs;
};
```

API & Data Layer

Mock API System

Location: `src/lib/api/`

Comprehensive mock API system for development and testing.

mockData.ts

```ts
export interface APIResponse<T> {
  data: T;
  success: boolean;
  message?: string;
  timestamp: string;
}

export interface DashboardAPIData {
  kpis: {
    currentLiability: number;
    availableITC: number;
    complianceScore: number;
    pendingReturns: number;
    liabilityTrend: number;
    itcTrend: number;
  };
  chartData: {
    gstLiability: Array<{ month: string; liability: number; paid: number }>;
    itcUtilization: Array<{ month: string; available: number; utilized: number }>;
    complianceBreakdown: Array<{ name: string; value: number; color: string }>;
    filingStatus: Array<{ month: string; gstr1: number; gstr3b: number; gstr9: number }>;
  };
  notifications: Array<{
    id: string;
    type: 'warning' | 'info' | 'success';
    title: string;
    subtitle: string;
    timestamp: string;
  }>;
}

export type MockScenario = 'default' | 'highLiability' | 'perfectCompliance';

export const mockScenarios: Record<MockScenario, DashboardAPIData> = {
  default: {
    kpis: {
      currentLiability: 125000,
      availableITC: 45000,
      complianceScore: 92,
      pendingReturns: 2,
      liabilityTrend: -12,
      itcTrend: 8
    },
    // ... more data
  },

  highLiability: {
    kpis: {
      currentLiability: 850000,
      availableITC: 125000,
      complianceScore: 68,
      pendingReturns: 5,
      liabilityTrend: 45,
      itcTrend: -15
    },
    // ... more data
  }
};
```

mockApi.ts

```
// Network delay simulation
const simulateNetworkDelay = (min = 300, max = 800) => {
  const delay = Math.random() * (max - min) + min;
  return new Promise(resolve => setTimeout(resolve, delay));
};

// Random failure simulation
const simulateRandomFailure = (failureRate = 0.05) => {
  return Math.random() < failureRate;
};

let currentScenario: MockScenario = 'default';

export const mockApi = {
  // Get dashboard data
  async getDashboardData(scenario?: MockScenario): Promise<APIResponse<DashboardAPIData>> {
    await simulateNetworkDelay(500, 1200);

    if (simulateRandomFailure()) {
      throw new Error('Network error: Unable to fetch dashboard data');
    }

    const selectedScenario = scenario || currentScenario;

    return {
      data: mockScenarios[selectedScenario],
      success: true,
      message: 'Dashboard data retrieved successfully',
      timestamp: new Date().toISOString(),
    };
  },

  // Update KPIs (for testing)
  async updateKPIs(updates: Partial<DashboardAPIData['kpis']>): Promise<APIResponse<DashboardAPIData['kpis']>> {
    await simulateNetworkDelay(200, 500);

    if (simulateRandomFailure(0.02)) {
      throw new Error('Failed to update KPIs');
    }

    const currentData = mockScenarios[currentScenario];
    const updatedKPIs = { ...currentData.kpis, ...updates };

    mockScenarios[currentScenario].kpis = updatedKPIs;

    return {
      data: updatedKPIs,
      success: true,
      message: 'KPIs updated successfully',
      timestamp: new Date().toISOString(),
    };
  },

  // Add notification
  async addNotification(notification: {
    title: string;
    subtitle: string;
    type: 'info' | 'warning' | 'success'
  }): Promise<APIResponse<DashboardAPIData['notifications'][0]>> {
    await simulateNetworkDelay(100, 300);

    const newNotification: DashboardAPIData['notifications'][0] = {
      id: `${Date.now()}-${Math.random().toString(36).substr(2, 9)}`,
      type: notification.type,
      title: notification.title,
      subtitle: notification.subtitle,
      timestamp: new Date().toISOString(),
    };

    (mockScenarios[currentScenario].notifications as any[]).unshift(newNotification);

    // Keep only the latest 20 notifications
    if (mockScenarios[currentScenario].notifications.length > 20) {
      mockScenarios[currentScenario].notifications =
        mockScenarios[currentScenario].notifications.slice(0, 20);
    }

    return {
      data: newNotification,
      success: true,
      message: 'Notification added successfully',
      timestamp: new Date().toISOString(),
    };
  }
```

```
  },

  // Switch scenario (for testing different business states)
  setScenario(scenario: MockScenario) {
    currentScenario = scenario;
  },

  getCurrentScenario() {
    return currentScenario;
  }
};
```

API Usage in Components

```
// In Redux slice
export const fetchDashboardData = createAsyncThunk(
  'dashboard/fetchData',
  async (scenario?: MockScenario) => {
    const response = await mockApi.getDashboardData(scenario);
    return response.data;
  }
);

// In component
const Dashboard = () => {
  const { data, loading, error } = useAppSelector(state => state.dashboard);
  const dispatch = useAppDispatch();

  useEffect(() => {
    dispatch(fetchDashboardData());
  }, [dispatch]);

  const handleRefresh = useCallback(() => {
    dispatch(fetchDashboardData());
  }, [dispatch]);

  if (loading) return <LoadingSkeleton variant="dashboard" />;
  if (error) return <ErrorBoundary />;

  return (
    <Grid container spacing={3}>
      {/* Dashboard content */}
    </Grid>
  );
};
```

Development Patterns

Component Pattern Guidelines

1. Component Structure

```tsx
// Standard component template
interface ComponentProps {
  // Props interface first
}

export const Component = ({ prop1, prop2 }: ComponentProps) => {
  // Hooks at the top
  const [state, setState] = useState();
  const dispatch = useAppDispatch();

  // Computed values
  const computedValue = useMemo(() => {
    return expensiveCalculation(prop1);
  }, [prop1]);

  // Event handlers
  const handleClick = useCallback(() => {
    // Handler logic
  }, [dependencies]);

  // Effects
  useEffect(() => {
    // Side effects
  }, [dependencies]);

  // Render
  return (
    <Box>
      {/* Component JSX */}
    </Box>
  );
};
```

2. Error Handling Pattern

```tsx
const ComponentWithErrorHandling = () => {
  const [error, setError] = useState<string | null>(null);
  const [loading, setLoading] = useState(false);

  const handleAsyncOperation = async () => {
    try {
      setLoading(true);
      setError(null);

      const result = await someAsyncOperation();

      // Handle success
    } catch (err) {
      setError(err instanceof Error ? err.message : 'An error occurred');
    } finally {
      setLoading(false);
    }
  };

  if (error) {
    return (
      <Alert severity="error">
        {error}
        <Button onClick={() => setError(null)}>Retry</Button>
      </Alert>
    );
  }

  return (
    <Box>
      {loading && <LoadingSpinner />}
      {/* Component content */}
    </Box>
  );
};
```

3. Form Pattern

```tsx
interface FormData {
  field1: string;
  field2: number;
}

const FormComponent = ({ onSubmit }: { onSubmit: (data: FormData) => void }) => {
  const {
    register,
    handleSubmit,
    formState: { errors, isSubmitting },
    reset
  } = useForm<FormData>();

  const submitHandler = async (data: FormData) => {
    try {
      await onSubmit(data);
      reset();
    } catch (error) {
      // Handle error
    }
  };

  return (
    <form onSubmit={handleSubmit(submitHandler)}>
      <FormInput
        name="field1"
        label="Field 1"
        required
        register={register}
        error={errors.field1?.message}
      />

      <Button
        type="submit"
        disabled={isSubmitting}
        loading={isSubmitting}
      >
        Submit
      </Button>
    </form>
  );
};
```

TypeScript Patterns

1. Component Props

```tsx
// Base props interface
interface BaseComponentProps {
  className?: string;
  children?: React.ReactNode;
}

// Specific component props
interface SpecificComponentProps extends BaseComponentProps {
  title: string;
  onAction: (value: string) => void;
  variant?: 'primary' | 'secondary';
}

// Generic component props
interface GenericComponentProps<T> {
  items: T[];
  renderItem: (item: T) => React.ReactNode;
  keyExtractor: (item: T) => string;
}
```

2. Event Handlers

```tsx
// Specific event handlers
type ClickHandler = (event: React.MouseEvent<HTMLButtonElement>) => void;
type ChangeHandler = (event: React.ChangeEvent<HTMLInputElement>) => void;
type SubmitHandler = (event: React.FormEvent<HTMLFormElement>) => void;

// Generic event handlers
type EventHandler<T extends React.SyntheticEvent> = (event: T) => void;
```

3. Redux Patterns

```
// Slice state interface
interface SliceState {
  data: DataType[];
  loading: boolean;
  error: string | null;
  filters: FilterType;
}

// Async thunk with proper typing
export const fetchData = createAsyncThunk<
  DataType[],           // Return type
  FilterType,           // Argument type
  {                     // Thunk API config
    rejectValue: string;
  }
>('slice/fetchData', async (filters, { rejectWithValue }) => {
  try {
    const response = await api.getData(filters);
    return response.data;
  } catch (error) {
    return rejectWithValue(error.message);
  }
});
```

Quick Reference

Key File Locations

Component	Location	Purpose
Main Layout	`src/components/layouts/DashboardLayout.tsx`	App layout wrapper
Sidebar	`src/components/layouts/Sidebar.tsx`	Navigation sidebar
GSTR-1 Wizard	`src/app/filing/gstr-1/components/FilingWizard.tsx`	Filing wizard coordinator
HSN Database	`src/lib/data/hsn-codes.ts`	HSN code database
Redux Store	`src/store/index.ts`	State management setup
Dashboard Slice	`src/store/slices/dashboardSlice.ts`	Dashboard state
Mock API	`src/lib/api/mockApi.ts`	API simulation
Theme Config	`src/lib/theme/index.ts`	MUI theme setup
Type Definitions	`src/types/index.ts`	TypeScript types
Validation Utils	`src/lib/utils/index.ts`	Validation functions

Common Commands

```
# Development
npm run dev          # Start development server
npm run build        # Build for production
npm run start        # Start production server
npm run lint         # Run ESLint
npm run type-check   # Run TypeScript check

# File operations
npm run clean        # Clean build artifacts
npm run format       # Format code with Prettier
```

Import Patterns

```tsx
// React and hooks
import { useState, useEffect, useCallback, useMemo } from 'react';

// MUI components
import {
  Box, Typography, Button, Card, CardContent
} from '@mui/material';

// MUI icons
import {
  Dashboard as DashboardIcon,
  Settings as SettingsIcon
} from '@mui/icons-material';

// Internal components
import { DashboardLayout } from '@/components/layouts';
import { LoadingSpinner } from '@/components/ui';

// Redux
import { useAppSelector, useAppDispatch } from '@/store';
import { fetchDashboardData } from '@/store/slices/dashboardSlice';

// Utilities
import { formatCurrency } from '@/lib/utils';
import { mockApi } from '@/lib/api';

// Types
import type { DashboardKPIs, InvoiceData } from '@/types';
```

Styling Patterns

```tsx
// MUI sx prop for component styling
<Box sx={{
  display: 'flex',
  flexDirection: { xs: 'column', md: 'row' },
  gap: 2,
  p: 3,
  bgcolor: 'background.paper',
  borderRadius: 1,
  boxShadow: 1
}}>

// Theme access in components
const theme = useTheme();
<Box sx={{
  color: theme.palette.primary.main,
  [theme.breakpoints.up('md')]: {
    display: 'flex'
  }
}}>

// Responsive design
<Grid container spacing={{ xs: 2, md: 3 }}>
  <Grid item xs={12} md={6} lg={4}>
    {/* Content */}
  </Grid>
</Grid>
```

Testing Patterns (Future)

```tsx
// Component testing template
describe('ComponentName', () => {
  const defaultProps = {
    prop1: 'value1',
    prop2: jest.fn()
  };

  beforeEach(() => {
    jest.clearAllMocks();
  });

  it('renders correctly', () => {
    render(<ComponentName {...defaultProps} />);
    expect(screen.getByText('Expected Text')).toBeInTheDocument();
  });

  it('handles user interaction', async () => {
    render(<ComponentName {...defaultProps} />);

    await user.click(screen.getByRole('button', { name: 'Click me' }));

    expect(defaultProps.prop2).toHaveBeenCalledWith('expected value');
  });
});
```

Last Updated: September 13, 2025

This documentation covers the complete codebase structure and implementation patterns for the GST Compliance Dashboard project.