# Frontend

Next.js 15 frontend — pages, components, design system

# Pages & Routing

# Bilko Frontend Pages

**Framework:** Next.js 15 (App Router) **Current State:** Fully implemented with mock data **Future State:** Will require API integration to replace mock data

---

# Route Architecture Overview

```
graph TD
    ROOT["app/layout.tsx\n(Root HTML, Inter font, metadata)"]

    ROOT --> LANDING["/ — Landing Page\napp/page.tsx"]
    ROOT --> AUTH["(auth) group\nNo shared layout"]
    ROOT --> DASH["(dashboard) group\napp/(dashboard)/layout.tsx\nSidebar + TopBar"]

    AUTH --> LOGIN["/login\nLoginPage"]
    AUTH --> REGISTER["/register\nRegisterPage"]

    DASH --> DASHBOARD["/dashboard\nDashboard Home"]
    DASH --> INVOICES["/invoices\nInvoice List"]
    DASH --> INV_NEW["/invoices/new\nInvoice Wizard"]
    DASH --> EXPENSES["/expenses\nExpense List"]
    DASH --> EXPENSES_NEW["/expenses/new\nAdd Expense"]
    DASH --> PURCHASES["/purchases\nPurchases (alias → expenses)"]
    DASH --> BANKING["/banking\nBanking Hub"]
    DASH --> REPORTS["/reports\nReports Hub"]
    DASH --> REPORTS_PL["/reports/profit-loss\nP&L Report"]
    DASH --> REPORTS_VAT["/reports/vat\nVAT Report"]
    DASH --> SETTINGS["/settings\nSettings"]

    LANDING --> LOGIN
    LOGIN --> DASHBOARD
    REGISTER --> DASHBOARD
```

```
    classDef layout fill:#1e1e2e,color:#cdd6f4,stroke:#89b4fa
    classDef auth fill:#313244,color:#cba6f7,stroke:#cba6f7
    classDef dashboard fill:#1e1e2e,color:#a6e3a1,stroke:#a6e3a1
    classDef reports fill:#313244,color:#89dceb,stroke:#89dceb
    class ROOT,DASH layout
    class AUTH,LOGIN,REGISTER auth
    class DASHBOARD,INVOICES,INV_NEW,EXPENSES,EXPENSES_NEW,PURCHASES,BANKING,SETTINGS
dashboard
    class REPORTS,REPORTS_PL,REPORTS_VAT reports
```

---

# Layouts

## Root Layout

- **Route:** `/`
- **File:** `app/layout.tsx`
- **Purpose:** Root HTML wrapper, applies Inter font, sets metadata
- **Metadata:**
  - Title: "Bilko - Accounting Made Simple"
  - Description: "Modern accounting software for Balkan businesses"
- **Dependencies:** Inter font from Google Fonts
- **Current State:** Fully implemented

## Dashboard Layout

- **Route:** `/(dashboard)/*`
- **File:** `app/(dashboard)/layout.tsx`
- **Purpose:** Layout wrapper for all authenticated pages
- **Key Components:**
  - Sidebar (desktop persistent, mobile overlay)
  - TopBar (header with search, notifications, user menu)
- **State Management:** Local state for mobile sidebar toggle
- **Mobile Responsiveness:** Responsive sidebar with overlay
- **Current State:** Fully implemented

```
graph LR
    DL["DashboardLayout\napp/(dashboard)/layout.tsx"]
```

```
    DL --> SB["Sidebar\ncomponents/sidebar.tsx\n• Logo\n• Main nav (5 items)\n• Bottom nav (Settings)\n• Active state via usePathname"]
    DL --> TB["TopBar\ncomponents/top-bar.tsx\n• Mobile menu button\n• Search (Cmd+K)\n• Notifications bell\n• User dropdown menu"]
    DL --> SLOT["Page Slot\n{children}"]

    SB -->|"dark sidebar #111113"| SCREEN["Rendered Screen"]
    TB -->|"white top bar"| SCREEN
    SLOT -->|"light content #FAFAFA"| SCREEN
```

---

# Pages

## Dashboard (Home)

- **Route:** `/dashboard`
- **File:** `app/(dashboard)/dashboard/page.tsx`
- **Purpose:** Financial overview and quick actions
- **Key Components:**
  - Metric cards (Cash Balance, Revenue MTD, Unpaid Invoices)
  - P&L Bar Chart (6-month trend)
  - Receivables Aging (stacked bar chart)
  - Expenses by Category (donut chart)
  - Recent Transactions table
  - Quick Actions card
- **Data Requirements (Future API):**
  - `GET /api/metrics` — cashBalance, revenueMTD, unpaidInvoices, expensesMTD, profitMTD, cashFlowChange
  - `GET /api/pl/monthly` — monthly P&L data (revenue, expenses, profit)
  - `GET /api/receivables/aging` — receivables breakdown (current, 30d, 60d, 90d+)
  - `GET /api/expenses/by-category` — expenses grouped by category
  - `GET /api/transactions/recent?limit=5` — recent transactions
- **Charts:** Recharts (BarChart, PieChart) with responsive containers
- **Current State:** Mock data from `lib/mock-data.ts`
- **Mobile Responsive:** Grid adapts 1/2/3 columns based on breakpoint

```
graph TD
    DP["Dashboard Page\n/dashboard"]

    DP --> ROW1["Row 1 — Metric Cards\ngrid-cols-1 md:grid-cols-3"]
```

```
    DP --> ROW2["Row 2 — Charts\ngrid-cols-1 md:grid-cols-3"]
    DP --> ROW3["Row 3 — Tables + Actions\ngrid-cols-1 lg:grid-cols-3"]

    ROW1 --> MC1["Cash Balance\nwith trend arrow"]
    ROW1 --> MC2["Revenue MTD\ncurrent month total"]
    ROW1 --> MC3["Unpaid Invoices\nwith warning badge"]

    ROW2 --> CH1["P&L Bar Chart\nRecharts BarChart\n6-month revenue/expenses/profit"]
    ROW2 --> CH2["Receivables Aging\nRecharts StackedBarChart\ncurrent/30d/60d/90d+"]
    ROW2 --> CH3["Expenses by Category\nRecharts PieChart (donut)\nper-category breakdown"]

    ROW3 --> TBL["Recent Transactions\nRecharts Table\nlast 5 transactions"]
    ROW3 --> QA["Quick Actions\nNew Invoice\nNew Expense\nImport Bank Statement"]
```

# Invoices List

- **Route:** `/invoices`
- **File:** `app/(dashboard)/invoices/page.tsx`
- **Purpose:** Invoice management with search, filter, sort
- **Key Features:**
  - Status filter (all/draft/sent/paid/overdue)
  - Search by customer name or invoice number
  - Date range filter (this month, last month, quarter, all)
  - Sortable columns (number, customer, date, due date, amount)
  - Row actions (view, edit, send, download PDF, delete)
  - Summary row with totals by status
- **Data Requirements (Future API):**
  - `GET /api/invoices?status=X&search=Y&dateRange=Z&sort=field&dir=asc` — filtered/sorted invoice list
  - `PATCH /api/invoices/:id` — update invoice
  - `DELETE /api/invoices/:id` — delete invoice
  - `POST /api/invoices/:id/send` — send invoice via email
  - `GET /api/invoices/:id/pdf` — generate PDF
- **Current State:** Mock data, client-side filtering/sorting
- **Mobile Responsive:** Table scrolls horizontally, filters stack vertically

# Invoice Creation Wizard

- **Route:** `/invoices/new`
- **File:** `app/(dashboard)/invoices/new/page.tsx`
- **Purpose:** 6-step wizard to create and send invoices
- **Steps:**

1. **Customer Selection** — Select existing customer or add new (dialog)
2. **Invoice Details** — Invoice number, issue/due dates, net terms, currency
3. **Line Items** — Add/remove items (description, qty, unit price, VAT rate), auto-calculate totals
4. **Customization** — Optional notes and payment terms
5. **Preview** — Visual preview of generated invoice
6. **Send** — Email form (to, subject, message, copy to self) + save/download options

- **Data Requirements (Future API):**
  - `GET /api/customers` — customer list for dropdown
  - `POST /api/customers` — create new customer
  - `POST /api/invoices` — create invoice (draft)
  - `POST /api/invoices/:id/send` — send invoice via email
  - `GET /api/invoices/:id/pdf` — download PDF
- **Form Validation:**
  - Step 1: Customer required
  - Step 3: At least one line item with description required
  - Step 6: Email address required
- **Current State:** Mock data, local state, no persistence
- **Mobile Responsive:** Wizard steps adapt, form fields stack on mobile

# Expenses List

- **Route:** `/expenses`
- **File:** `app/(dashboard)/expenses/page.tsx`
- **Purpose:** Expense tracking with categorization and approval workflow
- **Key Features:**
  - Period filter (this month, last month, quarter, year)
  - Category filter (Office, Travel, Meals, Utilities, Marketing, Infrastructure, Software, Professional Services)
  - Search by description or vendor
  - Status badges (pending, approved, paid)
  - Receipt attachment indicator
  - Add Expense dialog (with upload placeholder)
  - Summary stats (total, pending, approved, paid counts)
- **Data Requirements (Future API):**
  - `GET /api/expenses?period=X&category=Y&search=Z` — filtered expense list
  - `POST /api/expenses` — create expense
  - `PATCH /api/expenses/:id` — update expense status
  - `POST /api/expenses/:id/receipt` — upload receipt (multipart)
  - `GET /api/expenses/:id/receipt` — download receipt
- **Form Fields:**
  - Amount + currency (EUR, RSD, BAM)
  - Category (dropdown)
  - Date
  - Vendor (searchable input)

    - Payment method (Cash, Card, Bank Transfer)
    - Receipt upload (placeholder UI)
    - Description (optional)
- **Current State:** Mock data, form submits to console
- **Mobile Responsive:** Filters stack, table scrolls

# Purchases (Alias)

- **Route:** `/purchases`
- **File:** `app/(dashboard)/purchases/page.tsx`
- **Purpose:** Alias to expenses page
- **Current State:** Same component as `/expenses`

# Banking

- **Route:** `/banking`
- **File:** `app/(dashboard)/banking/page.tsx`
- **Purpose:** Bank account management and reconciliation
- **Key Features:**
    - 3 tabs: Accounts, Reconcile, Transactions
    - **Accounts Tab:** List of bank accounts with balances (multiple currencies)
    - **Reconcile Tab:** Unreconciled transactions with match confidence, period selector
    - **Transactions Tab:** All bank transactions (chronological)
    - Import Transactions button (placeholder)
    - Match actions (approve, link, create new)
- **Data Requirements (Future API):**
    - `GET /api/bank-accounts` — list accounts
    - `POST /api/bank-accounts` — add account
    - `GET /api/bank-accounts/:id/transactions?reconciled=false` — unreconciled transactions
    - `POST /api/bank-accounts/:id/import` — CSV import
    - `POST /api/bank-accounts/:id/transactions/:txId/reconcile` — mark reconciled
    - `POST /api/bank-accounts/:id/transactions/:txId/link?invoiceId=X` — link to invoice/expense
- **Match Confidence Logic:** Visual indicators for 0%, <50%, 50-89%, 90%+
- **Current State:** Mock data, no reconciliation logic
- **Mobile Responsive:** Tabs stack, tables scroll

# Reports Hub

- **Route:** `/reports`
- **File:** `app/(dashboard)/reports/page.tsx`
- **Purpose:** Financial report selection and P&L preview
- **Report Cards:**

    - Profit & Loss (implemented at `/reports/profit-loss`)
    - Balance Sheet (coming soon)
    - Cash Flow Statement (coming soon)
    - VAT/PDV Report (live at `/reports/vat`)
    - Trial Balance (coming soon)
    - General Ledger (coming soon)
- **P&L Preview:**
    - Expandable Revenue/Expenses sections
    - Current month detailed breakdown
    - Export buttons (PDF, Excel) — placeholders
- **Data Requirements (Future API):**
    - `GET /api/reports/pl?month=YYYY-MM` — P&L report data
    - `GET /api/reports/balance-sheet?date=YYYY-MM-DD` — balance sheet
    - `GET /api/reports/cash-flow?period=X` — cash flow statement
- **Current State:** Only P&L and VAT reports implemented, others show "Coming Soon" badge
- **Mobile Responsive:** Report cards grid adapts

```
graph TD
    RH["Reports Hub\n/reports"]

    RH --> PL["Profit & Loss\n/reports/profit-loss\nLIVE — expandable
revenue/expenses\nfetches from API with mock fallback"]
    RH --> BS["Balance Sheet\nCOMING SOON"]
    RH --> CF["Cash Flow Statement\nCOMING SOON"]
    RH --> VAT["VAT/PDV Report\n/reports/vat\nLIVE — 3-step wizard"]
    RH --> TB["Trial Balance\nCOMING SOON"]
    RH --> GL["General Ledger\nCOMING SOON"]

    PL --> PL_R["Revenue Section\n(expandable, breakdown by category)"]
    PL --> PL_E["Expenses Section\n(expandable, breakdown by category)"]
    PL --> PL_NP["Net Profit + Margin"]
    PL --> PL_EX["Export PDF / Export Excel"]

    VAT --> VAT_S1["Step 1: Reconciliation Check\n(warn if unreconciled transactions)"]
    VAT --> VAT_S2["Step 2: VAT Audit\n(all VAT transactions table)"]
    VAT --> VAT_S3["Step 3: Return Summary\nBox 1 / Box 2 / Box 3"]

    classDef live fill:#1e3a2f,color:#a6e3a1,stroke:#a6e3a1
    classDef soon fill:#2a1e2e,color:#888,stroke:#555
    class PL,VAT live
    class BS,CF,TB,GL soon
```

# VAT Report

- **Route:** `/reports/vat`
- **File:** `app/(dashboard)/reports/vat/page.tsx`
- **Purpose:** 3-step VAT return preparation
- **Steps:**
  1. **Reconciliation Check** — Warns if unreconciled bank transactions exist
  2. **VAT Audit** — Table of all VAT transactions (invoices + expenses) with net/VAT amounts
  3. **Return Summary** — VAT return boxes (Box 1: collected, Box 2: paid, Box 3: net due)
- **Data Requirements (Future API):**
  - `GET /api/bank-accounts/unreconciled-count` — reconciliation status
  - `GET /api/vat/transactions?period=YYYY-MM` — all VAT transactions
  - `GET /api/vat/return?period=YYYY-MM` — calculated VAT return data
  - `POST /api/vat/submit` — e-filing (Phase 2)
  - `GET /api/vat/export/pdf` — PDF export
  - `GET /api/vat/export/xml` — XML for e-filing
- **Calculations:** Assumes 20% standard VAT rate, converts all currencies to EUR equivalent
- **Current State:** Mock data, no submission, export placeholders
- **Mobile Responsive:** Tabs adapt, tables scroll, boxes stack

# Settings

- **Route:** `/settings`
- **File:** `app/(dashboard)/settings/page.tsx`
- **Purpose:** Multi-section settings page
- **Sections:**
  1. **Company** — Company profile (name, legal form, address, tax ID, currency, fiscal year)
  2. **Users** — User management table (name, email, role, status), invite button
  3. **Tax & Compliance** — Country, VAT registration, VAT number/rate, compliance reminders
  4. **Integrations** — Connected integrations (Intesa Bank CSV, Email SMTP), available integrations (Stripe, Fiken, Google Sheets, Slack, DocuSeal)
  5. **Notifications** — Email and in-app notification preferences
  6. **Security** — 2FA, session timeout, password policy, audit log, data export, delete company
- **Data Requirements (Future API):**
  - `GET /api/settings/company` — company data
  - `PATCH /api/settings/company` — update company
  - `GET /api/users` — user list
  - `POST /api/users/invite` — invite user
  - `GET /api/settings/tax` — tax settings

  - `PATCH /api/settings/tax` — update tax settings
  - `GET /api/integrations` — integration list
  - `POST /api/integrations/:id/connect` — connect integration
  - `GET /api/settings/notifications` — notification preferences
  - `PATCH /api/settings/notifications` — update preferences
  - `GET /api/security/audit-log` — audit log
  - `POST /api/security/data-export` — request data export
  - `DELETE /api/company` — delete company
- **Current State:** Mock data, form submits to console
- **Mobile Responsive:** Sidebar nav stacks, forms adapt

---

# Authentication Flow

```
sequenceDiagram
    participant U as User
    participant AP as AuthProvider
    participant AS as useAuthStore
    participant R as Router
    participant API as Backend API

    U->>AP: Navigate to /dashboard
    AP->>AS: checkAuth()
    AS->>API: GET /api/auth/me (Bearer token)

    alt API not configured (demo mode)
        AP-->>U: Render page directly (demoFallback=true)
    else Token valid
        API-->>AS: 200 OK — user data
        AS-->>AP: isAuthenticated=true
        AP-->>U: Render protected page
    else Token invalid / no token
        API-->>AS: 401 Unauthorized
        AS-->>AP: isAuthenticated=false
        AP->>R: router.replace('/login')
        R-->>U: Redirect to /login
        U->>U: Enter email + password
        U->>AS: login(email, password)
        AS->>API: POST /api/auth/login
        API-->>AS: 200 OK — JWT token
```

```
        AS->>R: router.push('/dashboard')
        R-->>U: Redirect to /dashboard
    end
```

---

# Screenshot Descriptions

## Dashboard

User sees:

- 3 metric cards at top (cash balance with green arrow, revenue MTD, unpaid invoices with warning badge)
- 3 charts in row below (P&L bar chart, receivables aging stacked bar, expenses donut)
- Bottom row: recent transactions table on left, quick action buttons on right

## Invoices List

User sees:

- Header with "Invoices" title and "New Invoice" button
- Filter row (status dropdown, search input, date range dropdown)
- Table with all invoices (sortable columns, status badges, action menu per row)
- Summary bar at bottom showing totals by status

## Invoice Wizard

User sees:

- Progress bar with 6 steps at top
- Current step content in card below
- Back/Next buttons at bottom (Send on final step)
- Step 3 shows line items with add/remove, running total calculation
- Step 5 shows formatted invoice preview

## Expenses

User sees:

- "Expenses" header with "Add Expense" button
- Filters (period, category, search)

- Table with date, description, category, amount, vendor, status badge, receipt icon
- Summary bar with totals (total €, pending count, approved count, paid count)
- Add dialog: form with amount+currency, category, date, vendor, payment method, receipt upload, description

# Banking

User sees:

- 3 tabs: Accounts, Reconcile, Transactions
- Accounts tab: table of bank accounts with type, currency, balance
- Reconcile tab: account selector, period, unreconciled transaction table with match confidence indicators, action buttons (✓ approve, link, create)
- Transactions tab: full transaction history with reconciled status

# VAT Report

User sees:

- 3-step tabs at top
- Step 1: Warning card if unreconciled transactions exist, "Reconcile Now" and "Continue" buttons
- Step 2: VAT transaction table (type badge, net amount, VAT rate, VAT amount), summary boxes (collected, paid, net due)
- Step 3: VAT return boxes (Box 1, Box 2, Box 3 highlighted), export buttons (PDF, XML), submit button (disabled, "Coming in Phase 2")

# Settings

User sees:

- Sidebar nav on left (6 sections)
- Content area on right showing active section
- Company section: form fields for company data
- Users section: table with invite button
- Integrations section: connected integrations cards, available integrations grid
- Security section: danger zone at bottom (delete company, red border)

---

# Notes

- **All pages use mock data** from `lib/mock-data.ts`

- **Authentication** implemented via `AuthProvider` + `useAuthStore` — demo mode active when `NEXT_PUBLIC_API_URL` not set
- **No persistence** — refreshing page loses all changes
- **Mobile-first design** — all pages tested at mobile/tablet/desktop breakpoints
- **Dark sidebar + light content area** — consistent layout across all pages

# State Management

# Bilko State Management

**Current State:** Primarily React hooks (useState, useEffect) **Installed but Minimal Use:** Zustand 4.5.0 **Future State:** Migrate to Zustand for global state

---

# Architecture Overview

```
flowchart LR
    subgraph CURRENT["Current Architecture (Phase 1)"]
        MD["lib/mock-data.ts\n(static imports)"] -->|imported directly| PG["Page
Component\n(useState + useMemo)"]
        PG -->|renders| UI["UI Components"]
        PG -->|local state only| PG
    end

    subgraph FUTURE["Future Architecture (Phase 2)"]
        API["Backend API\n(Express + PostgreSQL)"] -->|fetch| ST["Zustand Stores\nauth / org /
invoices\nexpenses / banking / ui"]
        ST -->|subscribe| PG2["Page Component\nuseInvoicesStore()"]
        PG2 -->|renders| UI2["UI Components"]
        PG2 -->|optimistic update| ST
    end

    style CURRENT fill:#1a1a2e,stroke:#4a4a6e,color:#aaa
    style FUTURE fill:#1a2e1a,stroke:#4a6e4a,color:#aaa
```

---

# Current State Patterns

## Local Component State (React useState)

**Usage:** Most components use local state for UI interactions and form data.

# Examples:

### Dashboard Page:

```
// No local state — purely presentational, uses mock data imports
```

### Invoice List Page:

```
const [statusFilter, setStatusFilter] = useState<string>("all")
const [searchQuery, setSearchQuery] = useState<string>("")
const [dateRange, setDateRange] = useState<string>("this-month")
const [sortColumn, setSortColumn] = useState<string>("date")
const [sortDirection, setSortDirection] = useState<"asc" | "desc">("desc")
```

### Invoice Wizard:

```
const [step, setStep] = useState(1)
const [customer, setCustomer] = useState<Contact | null>(null)
const [showAddCustomer, setShowAddCustomer] = useState(false)
const [invoiceDetails, setInvoiceDetails] = useState<InvoiceDetails>({...})
const [lineItems, setLineItems] = useState<LineItem[]>([...])
const [notes, setNotes] = useState("Thank you for your business!")
const [terms, setTerms] = useState("Payment due within 30 days.")
const [emailData, setEmailData] = useState({...})
```

### Expenses Page:

```
const [isDialogOpen, setIsDialogOpen] = useState(false)
const [periodFilter, setPeriodFilter] = useState("This Month")
const [categoryFilter, setCategoryFilter] = useState("All Categories")
const [searchQuery, setSearchQuery] = useState("")
const [formData, setFormData] = useState({...})
```

### Banking Page:

```
const [selectedAccount, setSelectedAccount] = useState(mockBankAccounts[0].id)
```

### Reports Page:

```
const [plExpanded, setPlExpanded] = useState({
  revenue: true,
  expenses: true
})
```

**VAT Report:**

```
const [currentStep, setCurrentStep] = useState<'reconciliation' | 'audit' |
'summary'>('reconciliation')
```

**Settings Page:**

```
const [activeSection, setActiveSection] = useState('company')
const [companyData, setCompanyData] = useState({...})
const [vatSettings, setVatSettings] = useState({...})
```

**Dashboard Layout:**

```
const [sidebarOpen, setSidebarOpen] = useState(false)
```

**Sidebar:**

```
const [expandedSections, setExpandedSections] = useState<string[]>([
  "Sales",
  "Purchases",
  "Reports"
])
```

---

# Local State Distribution by Page

```
flowchart TD
    subgraph IL["Invoice List /invoices"]
        IL1["statusFilter\nsearchQuery\ndateRange\nsortColumn\nsortDirection"]
    end

    subgraph IW["Invoice Wizard /invoices/new"]
        IW1["step (1-6)\ncustomer\nshowAddCustomer\ninvoiceDetails\nlineItems\nnotes /
terms\nemailData"]
    end
```

```
    subgraph EP["Expenses /expenses"]
        EP1["isDialogOpen\nperiodFilter\ncategoryFilter\nsearchQuery\nformData"]
    end

    subgraph BP["Banking /banking"]
        BP1["selectedAccount"]
    end

    subgraph RP["Reports /reports"]
        RP1["plExpanded\n{revenue, expenses}"]
    end

    subgraph VP["VAT /reports/vat"]
        VP1["currentStep\n'reconciliation' | 'audit' | 'summary'"]
    end

    subgraph SP["Settings /settings"]
        SP1["activeSection\ncompanyData\nvatSettings"]
    end

    subgraph LY["Dashboard Layout"]
        LY1["sidebarOpen (mobile)"]
    end

    subgraph SB["Sidebar component"]
        SB1["expandedSections: string[]"]
    end
```

---

# Computed State (React useMemo)

**Purpose:** Derived values from props/state to avoid expensive recalculations.

**Invoice List:**

```
// Filtered/sorted invoice list
const filteredInvoices = useMemo(() => {
  let filtered = [...mockInvoices]
```

```
  // Apply filters
  if (statusFilter !== "all") {
    filtered = filtered.filter((inv) => inv.status === statusFilter)
  }

  if (searchQuery) {
    const query = searchQuery.toLowerCase()
    filtered = filtered.filter(
      (inv) =>
        inv.customerName.toLowerCase().includes(query) ||
        inv.number.toLowerCase().includes(query)
    )
  }

  // Date range filter logic...

  // Sort logic...

  return filtered
}, [statusFilter, searchQuery, dateRange, sortColumn, sortDirection])

// Summary calculations
const summary = useMemo(() => {
  const total = filteredInvoices.length
  const byStatus = filteredInvoices.reduce((acc, inv) => {
    // Aggregate by status...
    return acc
  }, {})

  return { total, byStatus }
}, [filteredInvoices])
```

**Invoice Wizard:**

```
// Customer list
const customers = useMemo(
  () => mockContacts.filter((c) => c.type === "customer"),
  []
)
```

```
// Calculate totals
const totals = useMemo(() => {
  const subtotal = lineItems.reduce(
    (sum, item) => sum + item.quantity * item.unitPrice,
    0
  )
  const vatTotal = lineItems.reduce(
    (sum, item) =>
      sum + item.quantity * item.unitPrice * (item.vatRate / 100),
    0
  )
  const total = subtotal + vatTotal
  return { subtotal, vatTotal, total }
}, [lineItems])
```

**Expenses Page:**

```
// Filtered expenses
const filteredExpenses = useMemo(() => {
  return mockExpenses.filter(expense => {
    const matchesCategory = categoryFilter === "All Categories" || expense.category ===
categoryFilter
    const matchesSearch = searchQuery === "" ||
      expense.description.toLowerCase().includes(searchQuery.toLowerCase()) ||
      expense.vendor.toLowerCase().includes(searchQuery.toLowerCase())
    return matchesCategory && matchesSearch
  })
}, [categoryFilter, searchQuery])

// Stats
const stats = useMemo(() => {
  const total = filteredExpenses.reduce((sum, exp) => {
    // Convert to EUR and sum...
  }, 0)
  const pending = filteredExpenses.filter(e => e.status === 'pending').length
  const approved = filteredExpenses.filter(e => e.status === 'approved').length
  const paid = filteredExpenses.filter(e => e.status === 'paid').length

  return { total, pending, approved, paid }
}, [filteredExpenses])
```

**Banking Page:**

```
// Total balance in EUR
const totalBalanceEUR = useMemo(() => {
  return mockBankAccounts.reduce((sum, acc) => {
    const eurAmount = acc.currency === 'EUR' ? acc.balance :
                      acc.currency === 'RSD' ? acc.balance / 117 :
                      acc.balance / 2 // BAM to EUR
    return sum + eurAmount
  }, 0)
}, [])

// Unreconciled transactions
const unreconciledTransactions = useMemo(() => {
  return mockBankTransactions.filter(tx => !tx.reconciled && tx.bankAccountId ===
selectedAccount)
}, [selectedAccount])
```

---

# Navigation State (Next.js usePathname)

**Sidebar:**

```
const pathname = usePathname()

const isActive = (href: string | undefined) => {
  if (!href) return false
  return pathname === href
}
```

**Usage:** Highlights active navigation item based on current route.

---

# Router State (Next.js useRouter)

**Invoice Wizard:**

```
const router = useRouter()

const handleNext = () => {
```

```
  if (step === 6) {
    alert("Invoice sent!")
    router.push("/invoices")
  } else {
    setStep(step + 1)
  }
}

const handleCancel = () => {
  if (confirm("Are you sure you want to cancel? All changes will be lost.")) {
    router.push("/invoices")
  }
}
```

**Usage:** Programmatic navigation after form submission or cancel.

---

# Data Flow (Current)

```
flowchart LR
    MD["lib/mock-
data.ts\nmockInvoices\nmockExpenses\nmockBankAccounts\nmockContacts\nmockBankTransactions"]

    MD -->|"import { mockInvoices }"| IL["Invoice List\nfilter → sort → display"]
    MD -->|"import { mockExpenses }"| EP["Expenses Page\nfilter → stats → display"]
    MD -->|"import { mockBankAccounts }"| BP["Banking Page\ncurrency conversion → display"]
    MD -->|"import { mockContacts }"| IW["Invoice Wizard\nfilter customers → wizard"]
    MD -->|"import metrics"| DP["Dashboard\nmetrics → charts"]

    IL -->|"useMemo(filteredInvoices)"| IT["Invoice Table"]
    IL -->|"useMemo(summary)"| IS["Summary Bar"]

    EP -->|"useMemo(filteredExpenses)"| ET["Expense Table"]
    EP -->|"useMemo(stats)"| ES["Summary Stats"]

    BP -->|"useMemo(totalBalanceEUR)"| BA["Balance Display"]
    BP -->|"useMemo(unreconciledTx)"| REC["Reconcile Tab"]
```

# Mock Data Import Pattern

All pages import mock data directly:

```
import { mockInvoices, mockExpenses, mockBankAccounts } from "@/lib/mock-data"
```

**Issues:**

- No centralized state
- Data changes lost on page refresh
- No persistence
- Each page re-imports same data

# Data Transformation

Components transform mock data for display:

```
// Dashboard: Calculate metrics from raw data
const dashboardMetrics = {
  cashBalance: 2478170,
  revenueMTD: 485700,
  unpaidInvoices: 218200,
  // ...
}

// Invoice list: Filter/sort/search
const filteredInvoices = mockInvoices.filter(...)

// Banking: Currency conversion
const totalBalanceEUR = mockBankAccounts.reduce((sum, acc) => {
  const eurAmount = convertToEUR(acc.balance, acc.currency)
  return sum + eurAmount
}, 0)
```

---

# Zustand (Installed but Not Used)

**Package:** `zustand: ^4.5.0` (installed in package.json)

**Current Usage:** None

**Planned Usage:** Global state stores for:

- User authentication state
- Organization/company data
- Cached invoices/expenses/contacts
- UI preferences (theme, sidebar expanded)

---

# Future State Architecture (Phase 2)

## Planned Zustand Store Structure

```
classDiagram
    class AuthStore {
        +user: User | null
        +isAuthenticated: boolean
        +isLoading: boolean
        +token: string | null
        +error: string | null
        +login(email, password) Promise~void~
        +logout() void
        +refreshToken() Promise~void~
        +checkAuth() Promise~boolean~
    }

    class OrgStore {
        +organization: Organization | null
        +settings: OrgSettings
        +updateSettings(settings) Promise~void~
    }

    class InvoicesStore {
        +invoices: Invoice[]
        +isLoading: boolean
        +error: string | null
        +fetchInvoices(filters) Promise~void~
        +createInvoice(data) Promise~Invoice~
        +updateInvoice(id, data) Promise~void~
        +deleteInvoice(id) Promise~void~
```

```
    +sendInvoice(id, emailData) Promise~void~
}

class ExpensesStore {
    +expenses: Expense[]
    +isLoading: boolean
    +fetchExpenses(filters) Promise~void~
    +createExpense(data) Promise~Expense~
    +updateExpense(id, data) Promise~void~
    +deleteExpense(id) Promise~void~
}

class BankingStore {
    +accounts: BankAccount[]
    +transactions: BankTransaction[]
    +isLoading: boolean
    +fetchAccounts() Promise~void~
    +fetchTransactions(accountId) Promise~void~
    +importTransactions(accountId, file) Promise~void~
    +reconcileTransaction(txId) Promise~void~
    +linkTransaction(txId, type, id) Promise~void~
}

class UIStore {
    +sidebarOpen: boolean
    +sidebarExpandedSections: string[]
    +theme: string
    +toggleSidebar() void
    +toggleSection(section) void
    +setTheme(theme) void
}

AuthStore --> OrgStore : loads org on login
AuthStore --> InvoicesStore : scoped by orgId
AuthStore --> ExpensesStore : scoped by orgId
AuthStore --> BankingStore : scoped by orgId
```

# Planned Zustand Stores

## Auth Store

```
// Planned: stores/auth.ts
interface AuthState {
  user: User | null
  isAuthenticated: boolean
  token: string | null
  login: (email: string, password: string) => Promise<void>
  logout: () => void
  refreshToken: () => Promise<void>
}
```

## Organization Store

```
// Planned: stores/organization.ts
interface OrgState {
  organization: Organization | null
  settings: OrgSettings
  updateSettings: (settings: Partial<OrgSettings>) => Promise<void>
}
```

## Invoices Store

```
// Planned: stores/invoices.ts
interface InvoicesState {
  invoices: Invoice[]
  isLoading: boolean
  error: string | null
  fetchInvoices: (filters: InvoiceFilters) => Promise<void>
  createInvoice: (data: InvoiceCreateData) => Promise<Invoice>
  updateInvoice: (id: string, data: Partial<Invoice>) => Promise<void>
  deleteInvoice: (id: string) => Promise<void>
  sendInvoice: (id: string, emailData: EmailData) => Promise<void>
}
```

## Expenses Store

```
// Planned: stores/expenses.ts
interface ExpensesState {
  expenses: Expense[]
  isLoading: boolean
  fetchExpenses: (filters: ExpenseFilters) => Promise<void>
```

```
  createExpense: (data: ExpenseCreateData) => Promise<Expense>
  updateExpense: (id: string, data: Partial<Expense>) => Promise<void>
  deleteExpense: (id: string) => Promise<void>
}
```

## Banking Store

```
// Planned: stores/banking.ts
interface BankingState {
  accounts: BankAccount[]
  transactions: BankTransaction[]
  isLoading: boolean
  fetchAccounts: () => Promise<void>
  fetchTransactions: (accountId: string) => Promise<void>
  importTransactions: (accountId: string, file: File) => Promise<void>
  reconcileTransaction: (txId: string) => Promise<void>
  linkTransaction: (txId: string, linkType: string, linkId: string) => Promise<void>
}
```

## UI Store

```
// Planned: stores/ui.ts
interface UIState {
  sidebarOpen: boolean
  sidebarExpandedSections: string[]
  theme: 'light' | 'dark'
  toggleSidebar: () => void
  toggleSection: (section: string) => void
  setTheme: (theme: 'light' | 'dark') => void
}
```

# Migration Strategy

1. **Phase 2a:** Create stores with API integration
2. **Phase 2b:** Replace useState with store hooks in components
3. **Phase 2c:** Add optimistic updates and caching
4. **Phase 2d:** Implement persistence (localStorage for UI prefs)

**Example Migration:**

**Before (current):**

```
// Invoice list page
const [invoices, setInvoices] = useState<Invoice[]>(mockInvoices)
```

**After (Phase 2):**

```
// Invoice list page
import { useInvoicesStore } from '@/stores/invoices'

const { invoices, fetchInvoices, isLoading } = useInvoicesStore()

useEffect(() => {
  fetchInvoices({ status: statusFilter, dateRange })
}, [statusFilter, dateRange])
```

---

# API Integration Pattern (Future)

## API Client (lib/api.ts)

```
// Planned: lib/api.ts
const API_BASE = process.env.NEXT_PUBLIC_API_URL

export const api = {
  get: async (endpoint: string) => {
    const res = await fetch(`${API_BASE}${endpoint}`, {
      headers: { Authorization: `Bearer ${getToken()}` }
    })
    if (!res.ok) throw new Error(await res.text())
    return res.json()
  },

  post: async (endpoint: string, data: any) => {
    const res = await fetch(`${API_BASE}${endpoint}`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${getToken()}`
      },
```

```
      body: JSON.stringify(data)
    })
    if (!res.ok) throw new Error(await res.text())
    return res.json()
  },

  // PATCH, DELETE...
}
```

# Store Integration

```
// Planned: stores/invoices.ts
export const useInvoicesStore = create<InvoicesState>((set, get) => ({
  invoices: [],
  isLoading: false,
  error: null,

  fetchInvoices: async (filters) => {
    set({ isLoading: true, error: null })
    try {
      const data = await api.get(`/invoices?${buildQueryString(filters)}`)
      set({ invoices: data, isLoading: false })
    } catch (error) {
      set({ error: error.message, isLoading: false })
    }
  },

  createInvoice: async (data) => {
    const invoice = await api.post('/invoices', data)
    set({ invoices: [...get().invoices, invoice] })
    return invoice
  },

  // Other CRUD methods...
}))
```

---

# Future Data Flow

```
flowchart LR
    subgraph BROWSER["Browser"]
        subgraph STORES["Zustand Stores"]
            AS["AuthStore\ntoken, user"]
            IS["InvoicesStore\ninvoices, isLoading"]
            ES["ExpensesStore\nexpenses, isLoading"]
            BS["BankingStore\naccounts, transactions"]
            US["UIStore\nsidebarOpen, theme"]
        end

        subgraph PAGES["Pages"]
            IP["Invoice List"]
            IWP["Invoice Wizard"]
            EP["Expenses"]
            BP["Banking"]
        end

        subgraph PERSIST["Persistence"]
            LS["localStorage\nUI prefs (UIStore)"]
            CK["httpOnly Cookie\nJWT token"]
        end
    end

    BE["Backend API\n(Express + PostgreSQL)"]

    AS -->|"Authorization: Bearer"| BE
    BE -->|"invoices[]"| IS
    BE -->|"expenses[]"| ES
    BE -->|"accounts[], txs[]"| BS

    IS --> IP
    IS --> IWP
    ES --> EP
    BS --> BP

    US <-->|persist| LS
    AS <-->|token| CK
```

# Loading States (Future)

**Current:** No loading states (instant mock data)

**Future:** Loading skeletons and states

```
// Component usage (future)
const { invoices, isLoading } = useInvoicesStore()

if (isLoading) {
  return <InvoiceListSkeleton />
}
```

---

# Error Handling (Future)

**Current:** No error handling (mock data never fails)

**Future:** Error boundaries and toast notifications

```
// Store error state (future)
const { error } = useInvoicesStore()

if (error) {
  toast.error(`Failed to load invoices: ${error}`)
}
```

---

# Optimistic Updates (Future)

**Concept:** Update UI immediately, rollback on API failure

```
// Example: Delete invoice (future)
deleteInvoice: async (id) => {
  // Optimistic update
  const prevInvoices = get().invoices
  set({ invoices: prevInvoices.filter(inv => inv.id !== id) })
```

```
  try {
    await api.delete(`/invoices/${id}`)
  } catch (error) {
    // Rollback on failure
    set({ invoices: prevInvoices, error: error.message })
    toast.error('Failed to delete invoice')
  }
}
```

---

# State Persistence (Future)

**UI Preferences:** localStorage **Auth Token:** httpOnly cookie (secure) **Data Cache:** sessionStorage (optional, for performance)

```
// Example: Persist sidebar state (future)
export const useUIStore = create(
  persist<UIState>(
    (set) => ({
      sidebarOpen: false,
      toggleSidebar: () => set((state) => ({ sidebarOpen: !state.sidebarOpen }))
    }),
    { name: 'bilko-ui-preferences' }
  )
)
```

---

# Summary

**Current State:**

- React hooks (useState, useMemo, useEffect)
- Mock data imports
- Local component state
- No persistence
- No global state
- Zustand installed but unused

**Future State (Phase 2):**

- Zustand stores for global state
- API integration layer
- Loading/error states
- Optimistic updates
- State persistence (UI prefs)
- JWT token management

# Frontend — Status & Architecture

# Bilko Web — Next.js Frontend

## BookStack — Provjeri PRVO

Prije traženja bilo čega — provjeri BookStack (http://localhost:6875). Centralna baza znanja za tools, skills, hooks, agents, rules, projekte, klijente, dokumentaciju. Ako odgovor postoji tamo — NE TRAŽI dalje.

## Tech Stack

- **Framework:** Next.js 15 (App Router)
- **React:** 19.0.0
- **TypeScript:** 5.3.0
- **Styling:** Tailwind CSS 4 + shadcn/ui
- **State:** Zustand 4.5.0 (installed but mostly React hooks)
- **Charts:** Recharts 2.15.0
- **Icons:** Lucide React

## Pages (App Router)

All pages under `app/(dashboard)/`:

- `dashboard/page.tsx` — Revenue, expenses, charts
- `invoices/page.tsx` — Invoice list
- `invoices/new/page.tsx` — 6-step invoice wizard
- `expenses/page.tsx` — Expense list
- `purchases/page.tsx` — Alias to expenses
- `banking/page.tsx` — Placeholder
- `reports/page.tsx` — Reports hub
- `reports/vat/page.tsx` — VAT report
- `settings/page.tsx` — User settings

# Components

**UI (shadcn/ui):** 17 components in `components/ui/`

- avatar, button, card, dialog, dropdown-menu, input, label, select, separator, sheet, skeleton, table, tabs

**Layout:**

- `components/sidebar.tsx` — Dark navigation sidebar
- `components/top-bar.tsx` — Header with user menu

# Design System

**Embedded in `tailwind.config.ts`:** 73 tokens

- **Colors:** primary (#00E5A0), sidebar dark (#111113), chart colors
- **Typography:** Inter font, 8 sizes (xs to 4xl)
- **Spacing:** 8px grid (xs, sm, md, lg, xl, 2xl, 3xl)
- **Radius:** 4 values (sm: 6px, md: 8px, lg: 12px, full: 9999px)
- **Shadows:** card, modal, dropdown
- **Breakpoints:** sm (640px), md (768px), lg (1024px), xl (1280px)

# Mock Data

**CRITICAL:** All data from `lib/mock-data.ts`

- Revenue, expenses, invoices, bank accounts, contacts
- **MUST be replaced** with real API calls when backend ready
- Flag all mock data usage with comments: `// TODO: Replace with API call`

# State Management

- **Zustand installed** but not yet used
- Currently: React hooks (useState, useEffect)
- **Future:** Migrate to Zustand stores for global state (user, org, auth)

# Development Rules

1. **No production mock data** — Always flag mock data usage
2. **Design system tokens** — Use tokens from tailwind.config.ts, NEVER hardcode colors
3. **Responsive** — Mobile-first, test at all breakpoints
4. **Accessibility** — Use shadcn/ui primitives (Radix UI), semantic HTML
5. **TypeScript strict** — No `any` types without explicit justification

# API Integration (Future)

When backend ready:

- Create `lib/api.ts` with fetch wrappers
- Replace mock-data imports with API calls
- Add loading states, error handling
- Implement auth token management (JWT)

# Component Inventory

# Bilko Component Inventory

**Last Updated:** 2026-02-20 **Source of Truth:** Filesystem scan of `apps/web/components/`

---

# Component Hierarchy Overview

```
graph TD
    APP["app/layout.tsx\nRoot Layout"]

    APP --> LAND["Landing Page\napp/page.tsx"]
    APP --> AUTH_GRP["Auth Group\napp/(auth)"]
    APP --> DASH_GRP["Dashboard Group\napp/(dashboard)/layout.tsx"]

    LAND --> LN["Landing Components\ncomponents/landing/"]
    LN --> NAVBAR["Navbar"]
    LN --> HERO["Hero"]
    LN --> FEATURES["Features"]
    LN --> PRICING["Pricing"]
    LN --> TESTIMONIALS["Testimonials"]
    LN --> FOOTER["Footer"]

    AUTH_GRP --> LOGIN_PG["LoginPage\ncomponents/ui/input\ncomponents/ui/button"]
    AUTH_GRP --> REG_PG["RegisterPage\ncomponents/ui/input\ncomponents/ui/button"]

    DASH_GRP --> SIDEBAR["Sidebar\ncomponents/sidebar.tsx"]
    DASH_GRP --> TOPBAR["TopBar\ncomponents/top-bar.tsx"]
    DASH_GRP --> PAGES["Dashboard Pages"]

    PAGES --> DP["Dashboard\n/dashboard"]
    PAGES --> IP["Invoices\n/invoices"]
    PAGES --> IWP["Invoice Wizard\n/invoices/new"]
    PAGES --> EP["Expenses\n/expenses"]
```

```
    PAGES --> BP["Banking\n/banking"]
    PAGES --> RP["Reports\n/reports"]
    PAGES --> PLP["Profit & Loss\n/reports/profit-loss"]
    PAGES --> VP["VAT Report\n/reports/vat"]
    PAGES --> SP["Settings\n/settings"]

    classDef layout fill:#1e2235,color:#89b4fa,stroke:#89b4fa
    classDef landing fill:#2a1e35,color:#cba6f7,stroke:#cba6f7
    classDef auth fill:#2a1e1e,color:#f38ba8,stroke:#f38ba8
    classDef page fill:#1e2e1e,color:#a6e3a1,stroke:#a6e3a1
    class APP,DASH_GRP layout
    class LAND,LN,NAVBAR,HERO,FEATURES,PRICING,TESTIMONIALS,FOOTER landing
    class AUTH_GRP,LOGIN_PG,REG_PG auth
    class SIDEBAR,TOPBAR,PAGES,DP,IP,IWP,EP,BP,RP,PLP,VP,SP page
```

---

# Layout Components

## Sidebar

- **File:** `components/sidebar.tsx`
- **Purpose:** Dark left navigation sidebar with hierarchical menu
- **Props:** None (uses pathname from Next.js navigation)
- **Features:**
  - Active link highlighting (primary green border + background)
  - Dark theme (#111113 background)
  - Logo at top ("bilko" with SVG icon)
  - Arrow indicators on Sales and Purchases (hasSubmenu)
- **Navigation Items:**
  - Dashboard (direct link, LayoutDashboard icon)
  - Sales → `/invoices` (DollarSign icon, hasSubmenu)
  - Purchases → `/purchases` (CreditCard icon, hasSubmenu)
  - Banking (Landmark icon)
  - Reports (BarChart3 icon)
  - Settings (bottom nav, Settings icon)
- **State:** None — uses `usePathname()` for active detection
- **Dependencies:** Lucide icons (LayoutDashboard, DollarSign, CreditCard, Landmark, BarChart3, Settings, ChevronRight)

## TopBar

- **File:** `components/top-bar.tsx`
- **Purpose:** Header bar with search, notifications, user menu
- **Props:**
  - `onMenuClick?: () => void` — callback for mobile menu toggle
- **Features:**
  - Mobile menu button (hidden on desktop)
  - Mobile logo (hidden on desktop)
  - Search input (placeholder: "Search... (Cmd+K)")
  - Notification bell icon (no badge count yet)
  - User dropdown menu (Profile, Settings, Logout)
- **Dependencies:** Lucide icons (Search, Bell, Menu, User), shadcn/ui dropdown-menu
- **Mobile Responsive:** Shows menu button + logo on mobile, hides on desktop

```
graph LR
    SB["Sidebar\ncomponents/sidebar.tsx"]

    SB --> LOGO["Logo Section\nImage svg + 'bilko' text"]
    SB --> MAINNAV["Main Navigation\nnav > ul > li"]
    SB --> BOTTOMNAV["Bottom Navigation\n(Settings)"]

    MAINNAV --> DASH_LI["Dashboard\nLayoutDashboard icon"]
    MAINNAV --> SALES_LI["Sales (→ /invoices)\nDollarSign icon + ChevronRight"]
    MAINNAV --> PURCH_LI["Purchases (→ /purchases)\nCreditCard icon + ChevronRight"]
    MAINNAV --> BANK_LI["Banking\nLandmark icon"]
    MAINNAV --> REP_LI["Reports\nBarChart3 icon"]

    BOTTOMNAV --> SET_LI["Settings\nSettings icon"]

    SB -->|"isActive(href)"| ACTIVE["Active State\nbg-primary/10 + text-primary"]
    SB -->|"inactive"| INACTIVE["Hover State\nhover:bg-sidebar-hover"]
```

---

# UI Components (shadcn/ui)

All components in `components/ui/` are from shadcn/ui library (Radix UI primitives + Tailwind).

```
graph TD
    SHADCN["shadcn/ui Components\ncomponents/ui/"]

    SHADCN --> FORM_GRP["Form Components"]
```

```
    SHADCN --> DISPLAY_GRP["Display Components"]
    SHADCN --> OVERLAY_GRP["Overlay Components"]
    SHADCN --> FEEDBACK_GRP["Feedback Components"]

    FORM_GRP --> INPUT["Input\ntext, email, number, date, search"]
    FORM_GRP --> SELECT["Select\nRadix Select primitive\nSelectTrigger + SelectContent + SelectItem"]
    FORM_GRP --> TEXTAREA["Textarea\nmulti-line input, 3-6 rows"]
    FORM_GRP --> LABEL["Label\nRadix Label, accessible"]

    DISPLAY_GRP --> CARD["Card\nCard + CardHeader + CardTitle\n+ CardDescription + CardContent + CardFooter"]
    DISPLAY_GRP --> TABLE["Table\nTable + TableHeader + TableBody\n+ TableRow + TableHead + TableCell"]
    DISPLAY_GRP --> BADGE["Badge\ndefault / secondary / success\n/ warning / destructive"]
    DISPLAY_GRP --> AVATAR["Avatar\nRadix Avatar (not yet used)"]
    DISPLAY_GRP --> SEPARATOR["Separator\nhorizontal/vertical"]
    DISPLAY_GRP --> SKELETON["Skeleton\npulse animation (not yet used)"]

    OVERLAY_GRP --> DIALOG["Dialog\nRadix Dialog\nDialogContent + DialogHeader\n+ DialogFooter"]
    OVERLAY_GRP --> DROPDOWN["DropdownMenu\nRadix DropdownMenu\nTrigger + Content + Item"]
    OVERLAY_GRP --> TABS["Tabs\nRadix Tabs\nTabsList + TabsTrigger + TabsContent"]
    OVERLAY_GRP --> SHEET["Sheet\nRadix Dialog styled\n(not yet used)"]

    FEEDBACK_GRP --> BUTTON["Button\ndefault / destructive / outline\n/ secondary / ghost / link\nsm / default / lg / icon sizes"]
```

# Avatar

- **File:** `components/ui/avatar.tsx`
- **Purpose:** User avatar with fallback
- **Radix Primitive:** `@radix-ui/react-avatar`
- **Usage:** Not yet used in current pages (ready for user profile)

# Badge

- **File:** `components/ui/badge.tsx`
- **Purpose:** Status indicators (draft, sent, paid, overdue, success, warning, etc.)
- **Variants:** default, secondary, success, warning, destructive

- **Usage:**
  - Invoice status badges
  - Expense status badges
  - "Coming Soon" tags on report cards
  - User status in settings
- **Props:** `variant?: "default" | "secondary" | "destructive" | "outline" | "success" | "warning"`

# Button

- **File:** `components/ui/button.tsx`
- **Purpose:** Primary UI button
- **Variants:** default, destructive, outline, secondary, ghost, link
- **Sizes:** default, sm, lg, icon
- **Usage:** All action buttons throughout the app
- **Radix Primitive:** `@radix-ui/react-slot` (asChild support)

# Card

- **File:** `components/ui/card.tsx`
- **Purpose:** Content container with header/content sections
- **Sub-components:**
  - `Card` — outer wrapper
  - `CardHeader` — header section
  - `CardTitle` — title text
  - `CardDescription` — subtitle/description text
  - `CardContent` — body content
  - `CardFooter` — footer section (not used yet)
- **Usage:**
  - Dashboard metric cards
  - Report cards
  - Settings content wrapper
  - Banking account/transaction tables
- **Shadow:** `shadow-card` (0 2px 8px rgba(0, 0, 0, 0.08))

# Dialog

- **File:** `components/ui/dialog.tsx`
- **Purpose:** Modal dialogs
- **Radix Primitive:** `@radix-ui/react-dialog`
- **Sub-components:**
  - `Dialog` — wrapper
  - `DialogTrigger` — trigger button

    - `DialogContent` — modal content
    - `DialogHeader` — header section
    - `DialogTitle` — modal title
    - `DialogDescription` — modal description
    - `DialogFooter` — action buttons
- **Usage:**
    - Add Customer dialog (invoice wizard)
    - Add Expense dialog
- **Overlay:** Black 50% opacity, click-to-close

# Dropdown Menu

- **File:** `components/ui/dropdown-menu.tsx`
- **Purpose:** Context menus and dropdowns
- **Radix Primitive:** `@radix-ui/react-dropdown-menu`
- **Sub-components:**
    - `DropdownMenu` — wrapper
    - `DropdownMenuTrigger` — trigger button
    - `DropdownMenuContent` — menu content
    - `DropdownMenuItem` — menu item
    - `DropdownMenuLabel` — label text
    - `DropdownMenuSeparator` — divider
- **Usage:**
    - Invoice row actions (view, edit, send, download, delete)
    - User menu in top bar
- **Shadow:** `shadow-dropdown` (0 4px 16px rgba(0, 0, 0, 0.10))

# Input

- **File:** `components/ui/input.tsx`
- **Purpose:** Text input field
- **Types Supported:** text, email, number, date, search
- **Usage:**
    - All form inputs (invoice wizard, expense form, settings)
    - Search inputs (invoice list, expense list)
    - Filter inputs
- **Styling:** Border, padding, focus ring (primary color)

# Label

- **File:** `components/ui/label.tsx`
- **Purpose:** Form field labels
- **Radix Primitive:** `@radix-ui/react-label`

- **Usage:** All form field labels
- **Accessibility:** Proper label-input association

# Select

- **File:** `components/ui/select.tsx`
- **Purpose:** Dropdown select input
- **Radix Primitive:** `@radix-ui/react-select`
- **Sub-components:**
  - `Select` — wrapper
  - `SelectTrigger` — trigger button
  - `SelectValue` — selected value display
  - `SelectContent` — dropdown content
  - `SelectItem` — option item
- **Usage:**
  - Status filters (invoices, expenses)
  - Date range filters
  - Currency selectors
  - Category selectors
  - Settings dropdowns
- **Styling:** Chevron icon, border, focus ring

# Separator

- **File:** `components/ui/separator.tsx`
- **Purpose:** Visual divider line
- **Radix Primitive:** `@radix-ui/react-separator`
- **Orientations:** horizontal, vertical
- **Usage:** Not heavily used yet (potential in settings/forms)

# Sheet

- **File:** `components/ui/sheet.tsx`
- **Purpose:** Slide-out panel (mobile sidebar alternative)
- **Radix Primitive:** `@radix-ui/react-dialog` (styled as sheet)
- **Usage:** Not yet used (potential for mobile sidebar instead of overlay)
- **Direction:** Can slide from left/right/top/bottom

# Skeleton

- **File:** `components/ui/skeleton.tsx`
- **Purpose:** Loading placeholder
- **Usage:** Used in `AuthProvider` loading state and `reports/profit-loss` page

- **Animation:** Pulse animation
- **Future Use:** Data fetching loading states

# Table

- **File:** `components/ui/table.tsx`
- **Purpose:** Data table
- **Sub-components:**
  - `Table` — wrapper
  - `TableHeader` — header section
  - `TableBody` — body section
  - `TableRow` — row
  - `TableHead` — header cell
  - `TableCell` — data cell
  - `TableCaption` — caption text (not used)
  - `TableFooter` — footer section (not used)
- **Usage:**
  - Invoice list
  - Expense list
  - Bank transactions
  - VAT transactions
  - Recent transactions (dashboard)
  - Settings (users, integrations)
- **Styling:** Border, alternating row hover

# Tabs

- **File:** `components/ui/tabs.tsx`
- **Purpose:** Tab navigation
- **Radix Primitive:** `@radix-ui/react-tabs`
- **Sub-components:**
  - `Tabs` — wrapper
  - `TabsList` — tab button container
  - `TabsTrigger` — tab button
  - `TabsContent` — tab panel
- **Usage:**
  - Banking page (Accounts, Reconcile, Transactions)
  - VAT Report (Reconciliation, Audit, Summary)
- **Styling:** Primary underline for active tab

# Textarea

- **File:** `components/ui/textarea.tsx`
- **Purpose:** Multi-line text input

- **Usage:**
  - Invoice notes (customization step)
  - Invoice terms (customization step)
  - Email message (send step)
  - Expense description (optional)
- **Rows:** Configurable (default: 3-6 rows)

---

# Landing Components

```
graph TD
    LP["Landing Page\napp/page.tsx"]

    LP --> LNAV["Navbar\ncomponents/landing/navbar.tsx\nlogo + nav links + CTA button"]
    LP --> LHERO["Hero\ncomponents/landing/hero.tsx\nheadline + subtext + CTAs"]
    LP --> LFEAT["Features\ncomponents/landing/features.tsx\nfeature grid cards"]
    LP --> LPRICE["Pricing\ncomponents/landing/pricing.tsx\npricing tiers"]
    LP --> LTESTI["Testimonials\ncomponents/landing/testimonials.tsx\ncustomer quotes"]
    LP --> LFOOT["Footer\ncomponents/landing/footer.tsx\nlinks + copyright"]
```

---

# Chatbot Components

```
graph TD
    CW["ChatWidget\ncomponents/chatbot/ChatWidget.tsx\nfloating chat button + panel"]

    CW --> CM["ChatMessage\ncomponents/chatbot/ChatMessage.tsx\nindividual message bubble"]
    CW --> CI["ChatInput\ncomponents/chatbot/ChatInput.tsx\ntext input + send button"]
```

---

# Chart Components

Charts are built with **Recharts 2.15.0** (not custom components). Key chart types used:

- **BarChart** — P&L trend, receivables aging
- **PieChart** — Expenses by category (donut)
- **ResponsiveContainer** — Wrapper for all charts (100% width, fixed height)
- **XAxis, YAxis, CartesianGrid, Tooltip, Legend** — Chart primitives

**Chart Colors (from tailwind.config.ts):**

- Revenue: `#22C55E` (chart-revenue)
- Expense: `#EF4444` (chart-expense)
- Profit: `#3B82F6` (chart-profit)
- Neutral: `#6B7280` (chart-neutral)

---

# Utility Components

## cn (lib/utils.ts)

- **Purpose:** Utility function for conditional class names
- **Usage:** `cn("base-class", condition && "conditional-class")`
- **Dependencies:** `clsx` + `tailwind-merge`

## AuthProvider (lib/auth-provider.tsx)

- **Purpose:** Route guard — redirects unauthenticated users to `/login`
- **Demo Mode:** Activates automatically when `NEXT_PUBLIC_API_URL` is not set, bypassing auth check
- **Public Paths:** `/login`, `/register`, `/forgot-password`, `/`
- **Loading State:** Uses `Skeleton` component while checking auth

---

# Page-Specific Components

**Currently none standalone.** All components are reusable shadcn/ui components + layout components. No page-specific extracted components yet.

Potential future page-specific components:

- `InvoicePreview` (extracted from wizard step 5)
- `MetricCard` (extracted from dashboard)
- `TransactionRow` (extracted from dashboard/banking)
- `ExpenseFormDialog` (extracted from expenses page)

---

# Component Usage Map

| Component | Used In | Count |
|---|---|---|
| Button | All pages | 50+ |
| Card | Dashboard, Reports, Banking, Settings, Expenses | 20+ |
| Table | Invoices, Expenses, Banking, Reports, Dashboard, Settings | 10+ |
| Input | Invoice wizard, Expenses, Settings, TopBar | 30+ |
| Select | Invoices, Expenses, Banking, Settings, Invoice wizard | 20+ |
| Badge | Invoices, Expenses, Banking, Reports, Settings | 15+ |
| Dialog | Invoice wizard, Expenses | 2 |
| Dropdown Menu | Invoices, TopBar | 2 |
| Tabs | Banking, VAT Report | 2 |
| Textarea | Invoice wizard | 3 |
| Sidebar | All pages | 1 |
| TopBar | All pages | 1 |
| Label | All forms | 40+ |
| Skeleton | AuthProvider, P&L Report | 2+ |
| Separator | Minimal | 1-2 |
| Avatar | None yet | 0 |
| Sheet | None yet | 0 |

---

# Component Relationships by Page

```
graph LR
    subgraph INVOICE_LIST["Invoice List /invoices"]
        IL_CARD["Card\n(filter container)"]
        IL_SEL["Select\n(status filter)"]
        IL_INP["Input\n(search)"]
        IL_TBL["Table\n(invoice rows)"]
        IL_BADGE["Badge\n(status)"]
        IL_DD["DropdownMenu\n(row actions)"]
        IL_BTN["Button\n(New Invoice)"]
```

```
    end

    subgraph INVOICE_WIZ["Invoice Wizard /invoices/new"]
        IW_DLG["Dialog\n(Add Customer)"]
        IW_SEL["Select\n(customer, currency, VAT)"]
        IW_INP["Input\n(fields per step)"]
        IW_TA["Textarea\n(notes, terms, email)"]
        IW_BTN["Button\n(Back, Next, Send)"]
    end

    subgraph BANKING["Banking /banking"]
        BK_TABS["Tabs\n(Accounts, Reconcile, Txs)"]
        BK_TBL["Table\n(transactions)"]
        BK_BADGE["Badge\n(match confidence)"]
        BK_BTN["Button\n(approve, link, create)"]
    end

    subgraph REPORTS["Reports /reports/vat"]
        R_TABS["Tabs\n(3-step VAT wizard)"]
        R_TBL["Table\n(VAT transactions)"]
        R_CARD["Card\n(return boxes)"]
        R_BADGE["Badge\n(invoice/expense type)"]
    end
```

---

# Notes

- **All UI components** are from shadcn/ui (no custom variants yet)
- **No phantom components** — this list is exhaustive based on filesystem scan
- **Radix UI primitives** provide accessibility out of the box
- **Tailwind CSS 4** for styling (all tokens in tailwind.config.ts)
- **Lucide React** for all icons (consistent icon library)
- **Landing components** exist in `components/landing/` — not covered in original docs
- **Chatbot components** exist in `components/chatbot/` — ChatWidget, ChatMessage, ChatInput

# Design System

# Bilko Design System

**Source:** Extracted from `apps/web/tailwind.config.ts` + brand identity spec **Design Language:** Modern Balkan accounting SaaS — clean, professional, accessible

---

# Token Architecture Overview

```
graph TD
    TC["tailwind.config.ts\n(73 design tokens)"]

    TC --> COLORS["Color Tokens"]
    TC --> TYPE["Typography Tokens"]
    TC --> SPACE["Spacing Tokens"]
    TC --> RADIUS["Border Radius Tokens"]
    TC --> SHADOW["Shadow Tokens"]
    TC --> BP["Breakpoint Tokens"]

    COLORS --> PRIMARY["Primary Brand\n#00E5A0 / #00B380 / #33EBB3"]
    COLORS --> STATUS["Status Colors\nsuccess / warning / error / info"]
    COLORS --> TEXT["Text Scale\nprimary / secondary / muted"]
    COLORS --> BG["Backgrounds\nlight #FAFAFA / surface #FFF"]
    COLORS --> SIDEBAR["Sidebar (Dark Theme)\nbg / text / text-muted / active / hover"]
    COLORS --> CHART["Chart Colors\nrevenue / expense / profit / neutral"]

    TYPE --> FONT["Inter (Google Fonts)\nxs 12px → 4xl 40px"]
    TYPE --> WEIGHT["Weights\n400 / 500 / 600 / 700"]

    SPACE --> GRID["8px Grid\nxs 4px → 3xl 64px"]

    RADIUS --> SIZES["6px / 8px / 12px / full"]

    SHADOW --> ELEV["card / modal / dropdown"]
```

```
    BP --> SCREEN["640px / 768px / 1024px / 1280px"]
```

---

# Color Palette

## Primary Brand Colors

```
Primary:        #00E5A0  (Vibrant teal-green — primary CTA, links, active states)
Primary Dark:   #00B380  (Darker variant for hover states)
Primary Light:  #33EBB3  (Lighter variant for backgrounds)
```

## Status Colors

```
Success:        #22C55E  (Green — success states, paid invoices, positive metrics)
Warning:        #F59E0B  (Amber — warnings, pending items, aging invoices)
Error:          #EF4444  (Red — errors, overdue invoices, negative actions)
Info:           #3B82F6  (Blue — informational messages, neutral data)
```

## Text Colors

```
Text Primary:   #111113  (Near-black — body text, headings)
Text Secondary: #6B7280  (Gray-600 — secondary text, labels)
Text Muted:     #888888  (Gray-500 — muted text, placeholders)
```

## Background Colors

```
Background Light:   #FAFAFA  (Off-white — main content area background)
Background Surface: #FFFFFF  (White — card backgrounds, modals)
```

## Border Color

```
Border:         #E5E7EB  (Gray-200 — borders, dividers)
```

## Chart Colors

```
Chart Revenue:  #22C55E  (Green — revenue bars/lines)
Chart Expense:  #EF4444  (Red — expense bars/lines)
Chart Profit:   #3B82F6  (Blue — profit bars/lines)
Chart Neutral:  #6B7280  (Gray — neutral data points)
```

# Sidebar Colors (Dark Theme)

```
Sidebar BG:         #111113  (Near-black — sidebar background)
Sidebar Text:       #FAFAFA  (Off-white — sidebar text)
Sidebar Text Muted: #888888  (Gray — inactive menu items)
Sidebar Active:     #00E5A0  (Primary green — active menu item)
Sidebar Hover:      #1F1F23  (Slightly lighter black — hover state)
```

**Usage:**

- Sidebar = dark (#111113) with light text
- Content area = light (#FAFAFA) with dark text
- Cards = white (#FFFFFF) surface on light background

---

# Color Relationship Map

```
graph LR
    subgraph BRAND["Brand Identity"]
        P["Primary\n#00E5A0"]
        PD["Primary Dark\n#00B380\n(hover)"]
        PL["Primary Light\n#33EBB3\n(bg tint)"]
        P --> PD
        P --> PL
    end

    subgraph LAYOUT["Layout Zones"]
        SBG["Sidebar BG\n#111113"]
        SBG --> STEXT["Sidebar Text\n#FAFAFA"]
        SBG --> SMUTED["Sidebar Muted\n#888888"]
        SBG --> SACTIVE["Active Item\n#00E5A0"]

        CBG["Content BG\n#FAFAFA"]
        CBG --> SURF["Card Surface\n#FFFFFF"]
```

```
        CBG --> BORDER["Border\n#E5E7EB"]
    end

    subgraph SEMANTIC["Semantic Status"]
        SUCCESS["Success\n#22C55E\n(paid, positive)"]
        WARN["Warning\n#F59E0B\n(pending, aging)"]
        ERR["Error\n#EF4444\n(overdue, delete)"]
        INFO["Info\n#3B82F6\n(neutral data)"]
    end

    subgraph TEXT_SCALE["Text Scale"]
        TP["Text Primary\n#111113"]
        TS["Text Secondary\n#6B7280"]
        TM["Text Muted\n#888888"]
    end

    P -->|"active states"| SACTIVE
    SUCCESS -->|"chart-revenue"| CR["Chart Revenue"]
    ERR -->|"chart-expense"| CE["Chart Expense"]
    INFO -->|"chart-profit"| CP["Chart Profit"]
```

---

# Typography

## Font Family

```
Sans Serif: Inter, ui-sans-serif, system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI",
sans-serif
```

**Source:** Google Fonts Inter (variable font) **Fallback:** System UI fonts for performance

## Font Sizes

```
xs:   12px  (Small badges, captions)
sm:   14px  (Table cells, secondary text, form labels)
base: 16px  (Body text, default)
lg:   18px  (Subheadings, emphasized text)
xl:   20px  (Section titles)
```

```
2xl:  24px  (Card titles, small headings)
3xl:  32px  (Page headings)
4xl:  40px  (Hero text, large numbers)
```

# Font Weights

```
normal:    400  (Body text)
medium:    500  (Emphasis, table headers)
semibold:  600  (Subheadings, button text)
bold:      700  (Headings, metric numbers)
```

# Type Scale Usage

- **Headings:**
  - Page title: `text-3xl font-bold` (32px, 700)
  - Section title: `text-2xl font-bold` (24px, 700)
  - Card title: `text-base font-semibold` (16px, 600)
- **Body:**
  - Default: `text-base font-normal` (16px, 400)
  - Table cells: `text-sm font-medium` (14px, 500)
  - Muted text: `text-sm text-text-muted` (14px, #888888)
- **Numbers:**
  - Metrics: `text-3xl font-bold` (32px, 700)
  - Totals: `text-2xl font-bold` (24px, 700)
  - Amounts: `text-base font-medium` (16px, 500)

---

# Spacing System (8px Grid)

```
xs:  4px   (Tight spacing, icon gaps)
sm:  8px   (Input padding, small gaps)
md:  16px  (Default spacing, card padding)
lg:  24px  (Section spacing)
xl:  32px  (Large section spacing)
2xl: 48px  (Major section breaks)
3xl: 64px  (Hero spacing)
```

**Usage:**

- Card padding: `p-6` (24px)

- Form field gap: `space-y-4` (16px)
- Section spacing: `space-y-6` (24px)
- Grid gap: `gap-6` (24px)

**Consistency:** All spacing uses 8px increments (4px, 8px, 16px, 24px, 32px, 48px, 64px)

---

# Border Radius

```
sm:   6px   (Small elements, badges)
md:   8px   (Buttons, inputs, cards)
lg:   12px  (Modals, large cards)
full: 9999px (Circular elements, pills)
```

**Usage:**

- Cards: `rounded-md` (8px)
- Buttons: `rounded-md` (8px)
- Inputs: `rounded-md` (8px)
- Badges: `rounded-sm` (6px)
- User avatar: `rounded-full` (circular)

---

# Shadows

```
Card Shadow:     0 2px 8px rgba(0, 0, 0, 0.08)   (Subtle card elevation)
Modal Shadow:    0 8px 24px rgba(0, 0, 0, 0.12)  (Dialog/modal elevation)
Dropdown Shadow: 0 4px 16px rgba(0, 0, 0, 0.10)  (Dropdown menu elevation)
```

**Usage:**

- Cards: `shadow-card`
- Modals/dialogs: `shadow-modal`
- Dropdown menus: `shadow-dropdown`

**Philosophy:** Subtle shadows for depth, avoid heavy shadows (material design style)

---

# Design Token Relationships

```
classDiagram
    class ButtonTokens {
        +height_default: 40px
        +height_sm: 32px
        +height_lg: 48px
        +padding_x: 16px
        +border_radius: 8px (md)
        +font_size: 14px medium
        +variant_default: primary-bg white-text
        +variant_outline: border-only transparent
        +variant_ghost: no-border no-bg
        +variant_destructive: error-red white-text
    }

    class InputTokens {
        +height: 40px
        +padding_x: 12px
        +border: 1px solid #E5E7EB
        +border_radius: 8px (md)
        +font_size: 14px normal
        +focus_ring: 2px primary-color
    }

    class CardTokens {
        +background: #FFFFFF
        +border: 1px solid #E5E7EB
        +border_radius: 8px (md)
        +shadow: 0 2px 8px rgba-8pct
        +padding: 24px
    }

    class BadgeTokens {
        +padding: 4px 8px
        +border_radius: 6px (sm)
        +font_size: 12px medium
        +variant_success: green-bg
        +variant_warning: amber-bg
        +variant_destructive: red-bg
        +variant_secondary: light-gray-bg
    }
```

```
    class TableTokens {
        +row_height: 48px
        +cell_padding_x: 12px
        +cell_padding_y: 16px
        +border: 1px solid #E5E7EB
        +hover: light-gray #FAFAFA
        +header_weight: medium
        +header_color: text-secondary
    }

    class ColorTokens {
        +primary: #00E5A0
        +primary_dark: #00B380
        +success: #22C55E
        +warning: #F59E0B
        +error: #EF4444
        +info: #3B82F6
        +text_primary: #111113
        +text_secondary: #6B7280
        +border: #E5E7EB
        +surface: #FFFFFF
        +bg_light: #FAFAFA
    }

    ButtonTokens --> ColorTokens : uses primary, error
    InputTokens --> ColorTokens : uses border, primary (focus)
    CardTokens --> ColorTokens : uses surface, border
    BadgeTokens --> ColorTokens : uses success, warning, error
    TableTokens --> ColorTokens : uses border, bg-light (hover)
```

---

# Breakpoints

```
sm:  640px   (Small tablets, large phones)
md:  768px   (Tablets)
lg:  1024px  (Small desktops, large tablets)
xl:  1280px  (Desktops)
```

**Mobile-First Strategy:**

- Base styles = mobile (< 640px)
- `sm:` = small tablet (640px+)
- `md:` = tablet/desktop toggle (768px+)
- `lg:` = desktop layout (1024px+)
- `xl:` = wide desktop (1280px+)

**Responsive Patterns:**

- Grid: `grid-cols-1 md:grid-cols-2 lg:grid-cols-3`
- Sidebar: `hidden md:block` (hide on mobile)
- Filters: `flex-col sm:flex-row` (stack on mobile, row on tablet+)

---

# Responsive Layout Behavior

```
graph TD
    subgraph MOBILE["Mobile (< 640px)"]
        M1["Sidebar: hidden\n(overlay when toggled)"]
        M2["TopBar: shows hamburger + logo"]
        M3["Grid: single column"]
        M4["Filters: stacked vertically"]
        M5["Tables: horizontal scroll"]
    end

    subgraph TABLET["Tablet (768px+)"]
        T1["Sidebar: visible, persistent"]
        T2["TopBar: shows search + user menu"]
        T3["Grid: 2 columns"]
        T4["Filters: row layout"]
        T5["Tables: full width"]
    end

    subgraph DESKTOP["Desktop (1024px+)"]
        D1["Sidebar: 256px fixed width"]
        D2["TopBar: full bar"]
        D3["Grid: 3 columns"]
        D4["Settings: sidebar + content split"]
        D5["Banking: full tab content"]
    end
```

# Component Tokens

## Button

- **Height:** 40px (default), 32px (sm), 48px (lg), 40px (icon)
- **Padding:** 16px horizontal (default), 12px (sm), 20px (lg)
- **Border Radius:** 8px (md)
- **Font:** 14px medium (default), 12px (sm), 16px (lg)
- **Variants:**
  - Default: Primary green background, white text
  - Outline: Border only, transparent background
  - Ghost: No border, no background, hover shows background
  - Destructive: Error red background, white text

## Input

- **Height:** 40px
- **Padding:** 12px horizontal
- **Border:** 1px solid #E5E7EB (border color)
- **Border Radius:** 8px (md)
- **Font:** 14px normal
- **Focus:** Primary color ring (2px)

## Card

- **Background:** #FFFFFF (surface)
- **Border:** 1px solid #E5E7EB
- **Border Radius:** 8px (md)
- **Shadow:** 0 2px 8px rgba(0, 0, 0, 0.08)
- **Padding:** 24px (default content padding)

## Badge

- **Padding:** 4px 8px
- **Border Radius:** 6px (sm)
- **Font:** 12px medium
- **Variants:**
  - Default: Gray background
  - Success: Green background
  - Warning: Amber background

- Destructive: Red background
- Secondary: Light gray

## Table

- **Row Height:** 48px (default)
- **Cell Padding:** 12px horizontal, 16px vertical
- **Border:** 1px solid #E5E7EB (between rows)
- **Hover:** Light gray background (#FAFAFA)
- **Header:** Medium font weight, secondary text color

---

# Icon System

**Library:** Lucide React (v0.469.0) **Size:** Consistent 16px (`w-4 h-4`) or 20px (`w-5 h-5`) **Usage:**

- Buttons: 16px icon + 8px gap from text
- Menu items: 20px icon + 12px gap from text
- Standalone: 24px or larger

**Common Icons:**

- Plus (add actions)
- Search (search inputs)
- Menu (mobile sidebar toggle)
- User (user menu)
- Bell (notifications)
- ChevronDown/Right (expandable sections)
- Check (success, reconciliation)
- X (close, delete)
- Download (export actions)
- Send (send email)

---

# Chart Design Tokens

## Chart Colors (Recharts)

```
Revenue:  #22C55E  (Green bars)
Expense:  #EF4444  (Red bars)
```

```
Profit:   #3B82F6  (Blue bars)
Neutral:  #6B7280  (Gray — when no semantic meaning)
```

## Chart Typography

- **Axis Labels:** 12px normal
- **Tooltip:** 14px medium
- **Legend:** 12px normal

## Chart Layout

- **Responsive Container:** 100% width, fixed height (250px default)
- **Cartesian Grid:** Dashed, #E5E7EB stroke
- **Tooltip Background:** White with border
- **Border Radius:** 8px (md)

---

# Accessibility

## Color Contrast

- All text colors meet WCAG AA standards
- Primary text (#111113) on white = 16.17:1 (AAA)
- Secondary text (#6B7280) on white = 4.69:1 (AA)
- Primary green (#00E5A0) on white = 2.92:1 (fails — used for accents only, not body text)

## Focus Indicators

- All interactive elements have visible focus ring
- Focus ring color: Primary green (#00E5A0)
- Focus ring width: 2px

## Semantic HTML

- Proper heading hierarchy (h1 → h2 → h3)
- Form labels properly associated with inputs
- ARIA labels on icon-only buttons
- Table headers properly scoped

---

# Design Principles

1. **Clarity over Decoration** — Data-first, minimal ornamentation
2. **Consistent Spacing** — 8px grid, predictable rhythm
3. **Accessible by Default** — WCAG AA minimum, Radix UI primitives
4. **Mobile-First** — Responsive from 375px+ (iPhone SE)
5. **Dark Sidebar + Light Content** — Clear visual separation
6. **Primary Color as Accent** — Green (#00E5A0) for actions, not backgrounds
7. **Subtle Shadows** — Elevation without heaviness
8. **Data-Dense UI** — Tables, charts, metrics — optimized for information density

---

# Brand Identity Alignment

From `~/system/specs/bilko-brand-identity.md`:

- **Primary Color:** #00E5A0 (implemented)
- **Typography:** Inter (implemented)
- **Tone:** Modern, professional, Balkan-focused (implemented)
- **Dark Sidebar:** #111113 (implemented)
- **Logo Placement:** Top left sidebar (implemented)

---

# Future Tokens (Phase 2)

When implementing API integration:

- **Loading States:** Skeleton component colors
- **Error States:** Error message backgrounds (#FEF2F2, light red)
- **Success States:** Success message backgrounds (#F0FDF4, light green)
- **Toast Notifications:** Background, text, border colors
- **Dark Mode:** Full dark theme variant (optional)

# Forms & Validation

# Bilko Forms Documentation

**Current State:** Native HTML forms with React state **Validation:** Client-side JavaScript validation (no schema validation yet) **Future State:** Zod schemas for validation, react-hook-form for form management

---

# Forms Overview

```
graph TD
    FORMS["Bilko Forms"]

    FORMS --> IW["Invoice Wizard\n6-step multi-page form\n/invoices/new"]
    FORMS --> EF["Expense Form\nDialog (modal)\n/expenses"]
    FORMS --> SF["Settings Forms\n6 sections\n/settings"]
    FORMS --> AUTH["Auth Forms\nLogin + Register\n/login /register"]

    IW --> IW1["Step 1: Customer"]
    IW --> IW2["Step 2: Details"]
    IW --> IW3["Step 3: Line Items"]
    IW --> IW4["Step 4: Customization"]
    IW --> IW5["Step 5: Preview"]
    IW --> IW6["Step 6: Send"]

    EF --> EF1["Amount + Currency"]
    EF --> EF2["Category + Date"]
    EF --> EF3["Vendor + Payment Method"]
    EF --> EF4["Receipt Upload (placeholder)"]
    EF --> EF5["Description (optional)"]

    SF --> SF1["Company Profile"]
    SF --> SF2["Tax & Compliance"]
    SF --> SF3["Notification Preferences"]
```

```
    SF --> SF4["Security Settings"]

    AUTH --> AUTH1["Login\nemail + password"]
    AUTH --> AUTH2["Register\nname, company, email, password, country"]
```

---

# Invoice Creation Wizard (6-Step Multi-Page Form)

**Route:** `/invoices/new` **File:** `app/(dashboard)/invoices/new/page.tsx`

## Wizard State Machine

```
stateDiagram-v2
    [*] --> Step1_Customer : navigate to /invoices/new

    Step1_Customer : Step 1 — Customer Selection
    Step1_Customer : Select existing customer (dropdown)
    Step1_Customer : OR open Add Customer dialog
    Step1_Customer : Validation: customer required

    Step2_Details : Step 2 — Invoice Details
    Step2_Details : invoiceNumber (auto-generated)
    Step2_Details : issueDate, dueDate
    Step2_Details : netTerms (Net 15/30/60 → auto-updates dueDate)
    Step2_Details : currency (EUR/RSD/BAM)

    Step3_LineItems : Step 3 — Line Items
    Step3_LineItems : description (required), qty, unitPrice
    Step3_LineItems : vatRate (0/10/17/20/25%)
    Step3_LineItems : total auto-calculated (read-only)
    Step3_LineItems : Validation: min 1 item with description

    Step4_Customize : Step 4 — Customization
    Step4_Customize : notes (optional textarea)
    Step4_Customize : terms (optional textarea)
    Step4_Customize : No validation required
```

```
    Step5_Preview : Step 5 — Preview
    Step5_Preview : Read-only invoice render
    Step5_Preview : All data from previous steps
    Step5_Preview : No validation

    Step6_Send : Step 6 — Send / Save
    Step6_Send : to (email, pre-filled from customer)
    Step6_Send : subject, message (pre-filled template)
    Step6_Send : sendCopy (checkbox)
    Step6_Send : Save as Draft / Download PDF / Send Invoice

    Step1_Customer --> Step2_Details : Next (customer selected)
    Step1_Customer --> Step1_Customer : Next (no customer) — alert shown

    Step2_Details --> Step3_LineItems : Next
    Step2_Details --> Step1_Customer : Back

    Step3_LineItems --> Step4_Customize : Next (has valid item)
    Step3_LineItems --> Step3_LineItems : Next (no items) — alert shown
    Step3_LineItems --> Step2_Details : Back

    Step4_Customize --> Step5_Preview : Next
    Step4_Customize --> Step3_LineItems : Back

    Step5_Preview --> Step6_Send : Next
    Step5_Preview --> Step4_Customize : Back

    Step6_Send --> [*] : Send Invoice → alert + redirect /invoices
    Step6_Send --> Step5_Preview : Back
    Step6_Send --> [*] : Cancel → confirm dialog → /invoices
```

# Form Structure

## Step 1: Customer Selection

**Fields:**

- **Customer** (required)
  - Type: Select (dropdown)

- Options: All customers from contacts (type='customer')
- Can add new customer via dialog
- Validation: Required before proceeding to step 2

**Add Customer Dialog:**

- **Name** (required)
  - Type: Text
  - Validation: Required
- **Email** (required)
  - Type: Email
  - Validation: Required, valid email format
- **Phone** (optional)
  - Type: Tel
  - Validation: None
- **Tax ID** (optional)
  - Type: Text
  - Validation: None

**Validation:**

- Alert shown if user tries to proceed without selecting customer
- Form submission triggers inline alert (no schema validation)

---

## Step 2: Invoice Details

**Fields:**

- **Invoice Number**
  - Type: Text
  - Default: Auto-generated (e.g., "INV-2026-009")
  - Validation: None (can be edited)
- **Issue Date**
  - Type: Date
  - Default: Today's date
  - Validation: None
- **Due Date**
  - Type: Date
  - Default: 30 days from issue date
  - Validation: None
- **Net Terms** (shortcut selector)
  - Type: Select
  - Options: Net 15, Net 30, Net 60
  - Behavior: Auto-calculates due date when selected
  - Validation: None
- **Currency**

- Type: Select
- Options: EUR, RSD, BAM
- Default: EUR
- Validation: None

**Behavior:**

- Net terms selector auto-updates due date field
- All fields can be manually overridden

---

## Step 3: Line Items

**Repeating Fields (Line Items):**

Each line item contains:

- **Description** (required)
  - Type: Text
  - Placeholder: "Service or product description"
  - Validation: At least one item must have description
- **Quantity**
  - Type: Number
  - Default: 1
  - Min: 1
  - Validation: Positive number
- **Unit Price**
  - Type: Number
  - Default: 0
  - Min: 0
  - Step: 0.01
  - Validation: Non-negative
- **VAT Rate**
  - Type: Select
  - Options: 0%, 10%, 17%, 20%, 25%
  - Default: 20%
  - Validation: None
- **Total** (calculated, read-only)
  - Type: Text (disabled input)
  - Calculation: `quantity * unitPrice * (1 + vatRate/100)`
  - Display: Formatted currency

**Actions:**

- **Add Item** button: Appends new empty line item
- **Remove Item** button (X): Removes line item (disabled if only 1 item)

**Totals Display (read-only):**

- Subtotal (before VAT)
- VAT Total
- Grand Total (with VAT)

**Validation:**

- Alert shown if user tries to proceed with all empty descriptions
- Form submission requires at least one item with description

---

## Step 4: Customization

**Fields:**

- **Notes** (optional)
  - Type: Textarea
  - Default: "Thank you for your business!"
  - Placeholder: "Add a note for your customer..."
  - Validation: None
- **Terms** (optional)
  - Type: Textarea
  - Default: "Payment due within 30 days."
  - Placeholder: "Payment terms and conditions..."
  - Validation: None

**Behavior:**

- Both fields are optional
- Default values pre-populated but can be cleared

---

## Step 5: Preview (Read-Only)

**No form fields.** Displays formatted invoice preview with all data from previous steps.

**Preview Elements:**

- Invoice title ("INVOICE")
- From/To addresses
- Invoice number, date, due date
- Line items table
- Subtotal, VAT, Total
- Notes (if provided)
- Terms (if provided)

**No validation.** Step is purely visual review.

---

## Step 6: Send/Save

**Email Form:**

- **To** (required)
  - Type: Email
  - Default: Pre-filled with customer email
  - Validation: Valid email format (no schema yet)
- **Subject** (required)
  - Type: Text
  - Default: "Invoice {invoiceNumber}"
  - Validation: Required
- **Message** (required)
  - Type: Textarea
  - Default: Pre-filled template
  - Rows: 6
  - Validation: Required
- **Send Me a Copy** (optional)
  - Type: Checkbox
  - Default: Unchecked
  - Validation: None

**Action Buttons:**

- **Save as Draft** — Alert placeholder (no API)
- **Download PDF** — Alert placeholder (no API)
- **Send Invoice** — Alert + redirect to `/invoices` (no API)

**Validation:**

- No schema validation
- Form submission triggers alert "Invoice sent!"

---

# Line Item Data Flow

```
flowchart LR
    DESC["description\n(text input)"]
    QTY["quantity\n(number input)"]
    PRICE["unitPrice\n(number input)"]
    VAT["vatRate\n(Select: 0/10/17/20/25%)"]
```

```
    QTY --> CALC["useMemo(totals)\nsubtotal = qty * price\nvatTotal = subtotal * rate\ntotal = subtotal + vatTotal"]
    PRICE --> CALC
    VAT --> CALC

    CALC --> ROW_TOTAL["Row Total\n(read-only display)"]
    CALC --> SUBTOTAL["Subtotal\n(sum of all rows)"]
    CALC --> VAT_TOTAL["VAT Total\n(sum of all VAT)"]
    CALC --> GRAND_TOTAL["Grand Total\n(subtotal + vatTotal)"]

    DESC -->|"required"| VALID["Validation\nAt least 1 item\nwith description"]
```

---

# Form State Management

**Local State:**

```
const [step, setStep] = useState(1)
const [customer, setCustomer] = useState<Contact | null>(null)
const [showAddCustomer, setShowAddCustomer] = useState(false)
const [invoiceDetails, setInvoiceDetails] = useState<InvoiceDetails>({
  number: "INV-2026-009",
  issueDate: "2026-02-20",
  dueDate: "2026-03-22",
  currency: "EUR"
})
const [lineItems, setLineItems] = useState<LineItem[]>([
  { description: "", quantity: 1, unitPrice: 0, vatRate: 20, total: 0 }
])
const [notes, setNotes] = useState("Thank you for your business!")
const [terms, setTerms] = useState("Payment due within 30 days.")
const [emailData, setEmailData] = useState({
  to: "",
  subject: "",
  message: "",
  sendCopy: false
})
```

**No persistence:** All state lost on page refresh or navigation away.

**No Zod schemas:** Validation is inline JavaScript (alert boxes).

---

# Expense Form (Dialog)

**Route:** `/expenses` **Component:** Dialog triggered by "Add Expense" button

# Expense Form Flow

```
flowchart TD
    BTN["'Add Expense' Button\nExpenses Page"]
    BTN --> DLG["Dialog Opens\nisDialogOpen=true"]

    DLG --> AMOUNT["Amount (number, required)"]
    DLG --> CURRENCY["Currency (Select: EUR/RSD/BAM)"]
    DLG --> CATEGORY["Category (Select,
required)\nOffice/Travel/Meals/Utilities\nMarketing/Infrastructure\nSoftware/Professional
Services"]
    DLG --> DATE["Date (date input, required)"]
    DLG --> VENDOR["Vendor (text, optional)"]
    DLG --> PAYMENT["Payment Method (Select, optional)\nCash/Card/Bank Transfer"]
    DLG --> RECEIPT["Receipt Upload\n(placeholder UI — no real upload)"]
    DLG --> DESC["Description (text, optional)"]

    AMOUNT --> SUBMIT["Save Expense button"]
    CATEGORY --> SUBMIT

    SUBMIT -->|"current"| CONSOLE["console.log(formData)\nDialog closes\nForm resets"]
    SUBMIT -->|"future"| API["POST /api/expenses\nRefetch expense list"]

    DLG --> CANCEL["Cancel button\nDialog closes\nForm resets"]
```

# Form Fields

- **Amount** (required)
  - Type: Number
  - Placeholder: "0.00"
  - Validation: Required (no schema)
- **Currency** (required)

    - Type: Select
    - Options: EUR, RSD, BAM
    - Default: EUR
    - Width: 24px (narrow select next to amount)
    - Validation: None
- **Category** (required)
    - Type: Select
    - Options: Office, Travel, Meals, Utilities, Marketing, Infrastructure, Software, Professional Services
    - Placeholder: "Select category"
    - Validation: Required
- **Date** (required)
    - Type: Date
    - Default: Today's date
    - Validation: None
- **Vendor** (optional)
    - Type: Text
    - Placeholder: "Search vendor..."
    - Validation: None
    - Note: Not a searchable autocomplete yet — plain text input
- **Payment Method** (optional)
    - Type: Select
    - Options: Cash, Card, Bank Transfer
    - Placeholder: "Select method"
    - Validation: None
- **Receipt** (optional)
    - Type: File upload (placeholder UI only)
    - Display: Dashed border div with "Upload or Drag" text
    - Behavior: No actual upload implemented
    - Validation: None
- **Description** (optional)
    - Type: Text
    - Placeholder: "Additional notes..."
    - Validation: None

# Form Actions

- **Cancel** — Closes dialog, resets form
- **Save Expense** — Logs form data to console, closes dialog, resets form

**No API submission.** Form data not persisted.

**No Zod schemas.** Validation is JavaScript logic in form submit handler.

---

# Settings Forms

**Route:** `/settings` **File:** `app/(dashboard)/settings/page.tsx`

# Settings Navigation Flow

```
stateDiagram-v2
    [*] --> Company : default section

    Company : Company Profile
    Company : name, legalForm, address, city\npostalCode, country, taxId\nbaseCurrency, fiscalYearStart

    Users : Users & Roles
    Users : User management table\n(name, email, role, status)\nInvite User button

    Tax : Tax & Compliance
    Tax : country (Serbia/BiH/Croatia)\nvatRegistered (checkbox)\nvatNumber (conditional)\nvatRate (conditional)\ncompliance reminder checkboxes

    Integrations : Integrations
    Integrations : Connected: Intesa Bank CSV, Email SMTP\nAvailable: Stripe, Fiken\nGoogle Sheets, Slack, DocuSeal

    Notifications : Notification Preferences
    Notifications : Email: paid, overdue, approved, synced\nIn-App: invoice updates, expense updates\nreconciliation matches

    Security : Security Settings
    Security : Enable 2FA button\nSession Timeout Select\nPassword Policy checkboxes\nAudit Log / Data Export / Delete Company

    Company --> Users : sidebar nav click
    Company --> Tax : sidebar nav click
    Company --> Integrations : sidebar nav click
    Company --> Notifications : sidebar nav click
    Company --> Security : sidebar nav click
```

```
    Users --> Company : sidebar nav click
    Tax --> Company : sidebar nav click
    Integrations --> Company : sidebar nav click
    Notifications --> Company : sidebar nav click
    Security --> Company : sidebar nav click
```

# Company Profile Form

**Fields:**

- **Company Name** (required)
  - Type: Text
  - Default: "SnowIT d.o.o."
- **Legal Form**
  - Type: Select
  - Options: d.o.o., a.d., Preduzetnik
  - Default: "d.o.o."
- **Address**
  - Type: Text
  - Default: "Zmaja od Bosne"
- **City**
  - Type: Text
  - Default: "Sarajevo"
- **Postal Code**
  - Type: Text
  - Default: "71000"
- **Country**
  - Type: Text
  - Default: "BiH"
- **Tax ID / PIB / JIB**
  - Type: Text
  - Default: "4200000000"
- **Base Currency**
  - Type: Select
  - Options: EUR, RSD, BAM
  - Default: "EUR"
- **Fiscal Year Start**
  - Type: Select
  - Options: Jan 1, Apr 1, Jul 1, Oct 1
  - Default: "Jan 1"

**Action:**

- **Save Changes** — Alert placeholder (no API)

**Validation:** None (no required fields enforced)

---

# Tax & Compliance Form

**Fields:**

- **Country**
  - Type: Select
  - Options: Serbia, BiH, Croatia
  - Default: "Serbia"
- **VAT Registered**
  - Type: Checkbox
  - Default: Checked
- **VAT Number** (conditional, shown only if VAT registered)
  - Type: Text
  - Default: "RS123456789"
  - Placeholder: "Enter VAT number"
- **VAT Rate** (conditional, shown only if VAT registered)
  - Type: Select
  - Options: 17% (BiH), 20% (Serbia), 25% (Croatia)
  - Default: "20"

**Compliance Reminders:**

- **VAT filing deadlines** — Checkbox (default: checked)
- **Annual tax returns** — Checkbox (default: checked)
- **Payroll tax deadlines** — Checkbox (default: unchecked)

**Action:**

- **Save Settings** — Alert placeholder (no API)

**Validation:** None

---

# Notification Preferences

**Email Notifications:**

- Invoice paid — Checkbox (default: checked)
- Invoice overdue — Checkbox (default: checked)
- Expense approved — Checkbox (default: unchecked)
- Bank account synced — Checkbox (default: checked)

**In-App Notifications:**

- Invoice updates — Checkbox (default: checked)
- Expense updates — Checkbox (default: checked)
- Reconciliation matches — Checkbox (default: unchecked)

**Action:**

- **Save Preferences** — Alert placeholder (no API)

**Validation:** None

---

# Security Settings

**Two-Factor Authentication:**

- **Enable 2FA** button — No functionality yet

**Session Timeout:**

- Type: Select
- Options: 15 minutes, 30 minutes, 1 hour, 4 hours
- Default: 30 minutes

**Password Policy:**

- Minimum 12 characters — Checkbox (default: checked)
- Require special characters — Checkbox (default: checked)
- Expire passwords after 90 days — Checkbox (default: unchecked)

**Actions:**

- **View Audit Log** — No functionality yet
- **Request Data Export** — No functionality yet
- **Delete Company** (Danger Zone) — No functionality yet

**Validation:** None

---

# Auth Forms

## Login Form (`/login`)

```
flowchart TD
    LP["LoginPage\n/login"]
    LP --> EMAIL["Email input\ntype=email, required"]
    LP --> PASS["Password input\ntype=password, show/hide toggle"]
    LP --> SUBMIT["Submit Button\n'Prijavite se'"]

    SUBMIT --> STORE["useAuthStore.login(email, password)"]
    STORE -->|"success"| DASH["router.push('/dashboard')"]
    STORE -->|"error"| ERR["Error banner\n(red bg, border, message from store)"]

    LP --> FORGOT["Forgot password link\n(no functionality yet)"]
    LP --> REG["Link to /register"]
```

# Register Form (`/register`)

**Fields:**

- **First Name** (required)
- **Last Name** (required)
- **Company** (required)
- **Email** (required)
- **Password** (required, show/hide toggle)
- **Country** (required, Select)

**Submit:** Calls `api.auth.register()` → auto-login via `useAuthStore.login()` → redirect to `/dashboard`

---

# Validation Architecture

```
flowchart LR
    subgraph CURRENT["Current (Phase 1)"]
        INLINE["Inline JS Validation\nalert() on submit\nno schema, no real-time feedback"]
        INLINE --> INVOICE_V["Invoice Wizard\n• Step 1: if(!customer) alert()\n• Step 3:
if(!descriptions) alert()"]
        INLINE --> EXPENSE_V["Expense Form\n• if(!amount || !category) return"]
        INLINE --> AUTH_V["Auth Forms\n• HTML required attribute\n• type=email browser
validation"]
    end
```

```
    subgraph FUTURE["Future (Phase 2)"]
        ZOD["Zod Schemas\ncentralized, type-safe, reusable"]
        RHF["react-hook-form\nuseForm + zodResolver"]
        ZOD --> RHF
        RHF --> RT["Real-time Validation\nfield-level, on-change or on-blur"]
        RHF --> EM["Error Messages\ninline under each field"]
        RHF --> ES["Error State Styling\nred border + error text"]
    end
```

---

# Future Form Enhancements (Phase 2)

## Zod Schema Validation

**Planned:** Replace inline validation with Zod schemas

**Example (Invoice Wizard Step 1):**

```
import { z } from 'zod'

const customerSchema = z.object({
  id: z.string(),
  name: z.string().min(1, "Customer name required"),
  email: z.string().email("Valid email required"),
  phone: z.string().optional(),
  taxId: z.string().optional()
})
```

**Benefits:**

- Type-safe validation
- Reusable schemas for API/DB
- Better error messages
- Centralized validation logic

---

## react-hook-form Integration

**Planned:** Replace useState with react-hook-form

**Example (Expense Form):**

```
import { useForm } from 'react-hook-form'
import { zodResolver } from '@hookform/resolvers/zod'

const expenseSchema = z.object({
  amount: z.number().positive("Amount must be positive"),
  currency: z.enum(['EUR', 'RSD', 'BAM']),
  category: z.string().min(1, "Category required"),
  date: z.string(),
  vendor: z.string().optional(),
  paymentMethod: z.string().optional(),
  description: z.string().optional()
})

const { register, handleSubmit, formState: { errors } } = useForm({
  resolver: zodResolver(expenseSchema)
})
```

**Benefits:**

- Automatic error handling
- Less boilerplate
- Better performance (no re-renders on every keystroke)
- Built-in dirty/touched state

---

# Field-Level Validation

**Planned:** Real-time validation as user types

**Example (Email Field):**

```
<Input
  {...register("email")}
  type="email"
  error={errors.email?.message}
/>
{errors.email && (
  <span className="text-error text-sm">{errors.email.message}</span>
)}
```

**Current State:** No real-time validation, only on form submit.

---

# Form Persistence

**Planned:** Save draft forms to localStorage

**Use Cases:**

- Invoice wizard state saved between page refreshes
- Expense form data saved if user closes dialog accidentally

**Implementation:**

```
// Save to localStorage on every state change
useEffect(() => {
  localStorage.setItem('invoice-draft', JSON.stringify(invoiceState))
}, [invoiceState])

// Load from localStorage on mount
useEffect(() => {
  const draft = localStorage.getItem('invoice-draft')
  if (draft) setInvoiceState(JSON.parse(draft))
}, [])
```

---

# File Upload (Receipt Attachment)

**Current State:** Placeholder UI only (dashed border div)

**Future Implementation:**

- Drag-and-drop file upload
- File size validation (max 5MB)
- File type validation (PDF, JPG, PNG)
- Preview uploaded file
- Remove uploaded file
- Upload to backend API

**API Endpoint (planned):**

```
POST /api/expenses/:id/receipt
Content-Type: multipart/form-data
```

# Autocomplete/Search Fields

**Current State:** Plain text inputs

**Future Enhancement (Vendor Field):**

- Searchable dropdown
- Autocomplete from existing vendors
- Create new vendor inline
- Match by partial name

**Library:** Radix UI Combobox or react-select

---

# Multi-Currency Conversion

**Current State:** User manually selects currency

**Future Enhancement:**

- Fetch live exchange rates
- Auto-convert amounts for display
- Store both original currency and base currency
- Show converted amounts in tooltips

---

# Summary

**Current Forms:**

1. Invoice Wizard (6-step) — Customer, Details, Line Items, Customization, Preview, Send
2. Expense Form (dialog) — Amount, Category, Date, Vendor, Receipt, etc.
3. Company Profile — All company settings
4. Tax & Compliance — VAT settings
5. Notification Preferences — Email/in-app notification toggles
6. Security Settings — 2FA, session timeout, password policy
7. Login Form — Email + password, auth via Zustand store
8. Register Form — Name, company, email, password, country

**Validation:**

- Inline JavaScript (alert boxes) for wizard and expense form
- HTML `required` attribute + browser validation for auth forms

- No schema validation
- No real-time validation
- No error state styling

**State Management:**

- React useState for all forms
- No persistence (lost on refresh)
- No form libraries (native HTML forms)
- Auth forms use `useAuthStore` from Zustand (login/register)

**Future (Phase 2):**

- Zod schemas for validation
- react-hook-form for form management
- Field-level validation
- Form persistence (localStorage)
- File upload functionality
- Autocomplete/search fields
- API integration for submission