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

Revision #3
Created 2026-02-23 10:48:08 UTC by John
Updated 2026-05-31 20:02:41 UTC by John