Cost Manager Front-End Project

==========

COST MANAGER FRONT-END PROJECT Final Project

TEAM INFORMATION

TEAM MANAGER: First Name: Shai | Last Name: Dahari | ID: 066420431

TEAM MEMBERS:

1. First Name: Shai | Last Name: Dahari | ID: 066420431 | Mobile: 0544705511 | Email: shaidahari@gmail.com

2. First Name: Amit | Last Name: Yehoshafat | ID: 212359442 | Mobile: 0502227134 | Email: amityst12@gmail.com

DEPLOYMENT INFORMATION:

Deployment URL: The application is deployed on Netlify and accessible at: Cost Maneger App
Video Link: https://youtube.com/shorts/ngvLdUJyFA

COLLABORATIVE TOOLS SUMMARY:

Git and GitHub served as our primary collaboration platform, enabling effective teamwork despite our limited experience. We utilized branching to work on different features simultaneously without conflicts, and merging allowed us to integrate our changes seamlessly. Version

control helped us track modifications and revert to previous states when needed. For communication, we used Zoom and Discord for real-time meetings to discuss project requirements and resolve issues. We also leveraged AI tools like NotebookLLM to document meeting

notes efficiently. While we considered Slack initially, we found it unnecessary for our small two-person team, as direct communication through video calls proved more effective.

ADDITIONAL COMMENTS/GUIDELINES

This Cost Manager application is a React-based front-end project built with Material-UI (MUI) for the user interface. The application uses IndexedDB for local data storage and supports multi-currency expense tracking with real-time exchange rate conversion. All data is stored

locally in the browser, ensuring user privacy. The application is fully responsive and works on desktop browsers, with optimized mobile support.

Key Features:

Add expense items with amount, currency, category, and description

View detailed monthly reports with currency conversion

Visualize expenses with pie charts (by category) and bar charts (monthly trends)

Configure custom exchange rate API URLs

All data persisted locally using IndexedDB

Technical Stack:

React 19.2.0

Material-UI (MUI) v7.3.6

Recharts for data visualization

IndexedDB for local storage

Vite for build tooling

==========

CODE FILES

==========

FILE 1/12: src/App.jsx

```
/**
 * Cost Manager Application - Main application entry point
 * Provides navigation, responsive layout, and database initialization
 * Manages tab-based routing between CostForm, Dashboard, and Settings
```

```
 * @returns {JSX.Element} Complete application with mobile-responsive navigation
 */

import { useState, useEffect } from 'react';
import {
  AppBar,
  Toolbar,
  Typography,
  Container,
  Box,
  Tabs,
  Tab,
  Paper,
  useTheme,
  useMediaQuery,
  Drawer,
  IconButton,
  List,
  ListItem,
  ListItemButton,
  ListItemIcon,
  ListItemText,
```

```
  BottomNavigation,
  BottomNavigationAction,
  Divider
} from '@mui/material';
import {
  Menu as MenuIcon,
  AttachMoney,
  Assessment,
  BarChart,
  Settings as SettingsIcon,
  AddCircle,
  Close as CloseIcon
} from '@mui/icons-material';


// Application components and database utilities
import CostForm from './components/CostForm';
import Dashboard from './components/Dashboard';
import Settings from './components/Settings';
import { openCostsDB } from './utils/idb';

/**
 * Main Application component with responsive navigation and database management
 */
```

```
const App = () => {
  // Navigation state - manages active tab and mobile drawer
  const [activeTab, setActiveTab] = useState(0);
  const [mobileDrawerOpen, setMobileDrawerOpen] = useState(false);

  // Data state - triggers component refresh and tracks database status
  const [refreshTrigger, setRefreshTrigger] = useState(0);
  const [dbInitialized, setDbInitialized] = useState(false);
  const [dbError, setDbError] = useState(null);

  // Responsive design hooks - detect screen size for layout adaptation
  const theme = useTheme();
  const isMobile = useMediaQuery(theme.breakpoints.down('sm'));
  const isTablet = useMediaQuery(theme.breakpoints.down('md'));

  /**
   * Initialize IndexedDB database on application startup
   * Verifies browser support and sets up database connection
   */
  useEffect(() => {
```

```
const initializeDatabase = async () => {
  // Check if IndexedDB is supported
  if (!window.indexedDB) {
    setDbError('Your browser does not support IndexedDB. Please use a modern browser like Chrome, Firefox, or Edge.');
    return;
  }

  try {
    await openCostsDB("costsdb", 1);
    setDbInitialized(true);
  } catch (error) {
    // Handle specific IndexedDB errors
    let errorMessage = 'Failed to initialize database';

    if (error.name === 'QuotaExceededError') {
      errorMessage = 'Storage quota exceeded. Please free up some space in your browser.';
    } else if (error.name === 'InvalidStateError') {
      errorMessage = 'Database is in an invalid state. Try clearing your browser data.';
    } else {
      errorMessage = `${errorMessage}: ${error.message}`;
```

```
        }
        setDbError(errorMessage);
      }
    };
    initializeDatabase();
}, []);
/**
 * Handle tab change
 * @param {Event} event - Click event
 * @param {number} newValue - New tab index
 */
const handleTabChange = (event, newValue) => {
    setActiveTab(newValue);
    if (isMobile) {
      setMobileDrawerOpen(false);
    }
};
/**
 * Handle successful cost addition by triggering data refresh
 * Increments refresh trigger to update Dashboard with new data
```

```
 */
const handleCostAdded = () => {
   // Increment trigger to force Dashboard re-render and data reload
   setRefreshTrigger(prev => prev + 1);
};
/**
 * Toggle mobile navigation drawer open/closed state
 */
const toggleDrawer = () => {
   setMobileDrawerOpen(!mobileDrawerOpen);
};
// Navigation configuration - defines tabs with labels and icons
const navigationItems = [
   { label: 'Add Cost', icon: <AttachMoney />, index: 0 },
   { label: 'Reports & Charts', icon: <BarChart />, index: 1 },
   { label: 'Settings', icon: <SettingsIcon />, index: 2 }
];
/**
 * Render the active component based on selected tab
 */
```

```
const renderActiveComponent = () => {
  if (!dbInitialized) {
    return (
      <Box sx={{ display: 'flex', justifyContent: 'center', alignItems: 'center', minHeight: 400 }}>
        <Typography variant="h6" color="text.secondary">
          {dbError || 'Initializing database...'}
        </Typography>
      </Box>
    );
  }
  switch (activeTab) {
    case 0:
      return <CostForm onCostAdded={handleCostAdded} />;
    case 1:
      return <Dashboard refreshTrigger={refreshTrigger} />;
    case 2:
      return <Settings />;
    default:
      return <CostForm onCostAdded={handleCostAdded} />;
  }
```

```
};
/**
 * Render mobile drawer navigation
 */
const renderMobileDrawer = () => (
  <Drawer
    anchor="left"
    open={mobileDrawerOpen}
    onClose={toggleDrawer}
    PaperProps={{
      sx: { width: { xs: '80%', sm: 280 }, maxWidth: 300 }
    }}
  >
    <Box sx={{ display: 'flex', flexDirection: 'column', height: '100%' }}>
      {/* Drawer Header */}
      <Box sx={{
        display: 'flex',
        alignItems: 'center',
        justifyContent: 'space-between',
        p: 2,
```

```
    bgcolor: 'primary.main',
    color: 'white'
  }}>
    <Box sx={{ display: 'flex', alignItems: 'center' }}>
      <AttachMoney sx={{ mr: 1 }} />
      <Typography variant="h6">Cost Manager</Typography>
    </Box>
    <IconButton color="inherit" onClick={toggleDrawer} size="large">
      <CloseIcon />
    </IconButton>
  </Box>
  <Divider />
  {/* Navigation List */}
  <List sx={{ flexGrow: 1, pt: 1 }}>
    {navigationItems.map((item) => (
      <ListItem key={item.label} disablePadding>
        <ListItemButton
          selected={activeTab === item.index}
          onClick={(e) => handleTabChange(e, item.index)}
          sx={{
```

```
      py: 2,
      '&.Mui-selected': {
        bgcolor: 'primary.light',
        '&:hover': { bgcolor: 'primary.light' }
      }
    }}
  >
    <ListItemIcon sx={{
      color: activeTab === item.index ? 'primary.main' : 'inherit',
      minWidth: 48
    }}>
      {item.icon}
    </ListItemIcon>
    <ListItemText
      primary={item.label}
      primaryTypographyProps={{
        fontWeight: activeTab === item.index ? 600 : 400
      }}
    />
  </ListItemButton>
```

```
              </ListItem>
            ))}
          </List>
        </Box>
      </Drawer>
  );

  /**
   * Render mobile bottom navigation
   */
  const renderBottomNavigation = () => (
    <Paper
      sx={{
        position: 'fixed',
        bottom: 0,
        left: 0,
        right: 0,
        zIndex: 1100,
        display: { xs: 'block', sm: 'none' }
      }}
```

```
    elevation={8}
  >
    <BottomNavigation
      value={activeTab}
      onChange={(event, newValue) => setActiveTab(newValue)}
      showLabels
      sx={{
        height: 64,
        '& .MuiBottomNavigationAction-root': {
          minWidth: 'auto',
          py: 1,
          '&.Mui-selected': {
            color: 'primary.main'
          }
        },
        '& .MuiBottomNavigationAction-label': {
          fontSize: '0.7rem',
          '&.Mui-selected': {
            fontSize: '0.75rem'
          }
```

```
            }
          }}
        >
          <BottomNavigationAction label="Add" icon={<AddCircle />} />
          <BottomNavigationAction label="Reports" icon={<BarChart />} />
          <BottomNavigationAction label="Settings" icon={<SettingsIcon />} />
        </BottomNavigation>
      </Paper>
    );

    return (
      <Box sx={{ display: 'flex', flexDirection: 'column', minHeight: '100vh', bgcolor: 'background.default' }}>
        {/* AppBar Header */}
        <AppBar position="static" elevation={2}>
          <Toolbar>
            {/* Mobile menu button */}
            {isMobile && (
              <IconButton
                edge="start"
                color="inherit"
```

```
      aria-label="menu"
      onClick={toggleDrawer}
      sx={{ mr: 2 }}
    >
      <MenuIcon />
    </IconButton>
)}

{/* Application Title */}
<AttachMoney sx={{ mr: 1 }} />
<Typography variant="h6" component="h1" sx={{ flexGrow: 1 }}>
  Cost Manager
</Typography>

{/* Desktop/Tablet Navigation Tabs */}
{!isMobile && (
  <Tabs
    value={activeTab}
    onChange={handleTabChange}
    textColor="inherit"
```

```
  indicatorColor="secondary"
  sx={{
    ml: 'auto',
    '& .MuiTab-root': {
      minHeight: 64,
      px: { sm: 2, md: 3 },
      fontSize: { sm: '0.8rem', md: '0.875rem' }
    }
  }}
>
  <Tab
    label={isTablet ? "Add" : "Add Cost"}
    icon={<AttachMoney />}
    iconPosition="start"
  />
  <Tab
    label={isTablet ? "Reports" : "Reports & Charts"}
    icon={<BarChart />}
    iconPosition="start"
  />
```

```
            <Tab
              label="Settings"
              icon={<SettingsIcon />}
              iconPosition="start"
            />
          </Tabs>
        )}
      </Toolbar>
    </AppBar>
    {/* Mobile Drawer */}
    {isMobile && renderMobileDrawer()}
    {/* Main Content Area */}
    <Container
      maxWidth="lg"
      sx={{
        flexGrow: 1,
        py: { xs: 1.5, sm: 2, md: 3 },
        px: { xs: 1, sm: 2, md: 3 },
        // Add bottom padding on mobile for bottom navigation
        pb: { xs: 10, sm: 2, md: 3 }
```

```
    }}
  >
    <Paper
      elevation={0}
      sx={{
        minHeight: { xs: 'calc(100vh - 180px)', sm: 'calc(100vh - 140px)' },
        bgcolor: 'transparent'
      }}
    >
      {renderActiveComponent()}
    </Paper>
  </Container>
  {/* Mobile Bottom Navigation */}
  {renderBottomNavigation()}
  {/* Footer */}
  <Box
    component="footer"
    sx={{
      py: 2,
      px: 2,
```

```
            mt: 'auto',
            backgroundColor: theme.palette.mode === 'light' ? theme.palette.grey[200] : theme.palette.grey[800],
          }}
        >
          <Container maxWidth="lg">
            <Typography variant="body2" color="text.secondary" align="center">
              © 2026 Cost Manager Application. All data stored locally in your browser.
            </Typography>
          </Container>
        </Box>
      </Box>
    );
};

export default App;
```

==========

FILE 2/12: src/index.css

```
:root {
  font-family: system-ui, Avenir, Helvetica, Arial, sans-serif;
  line-height: 1.5;
```

```
  font-weight: 400;
  color-scheme: light dark;
  color: rgba(255, 255, 255, 0.87);
  background-color: #242424;

  font-synthesis: none;
  text-rendering: optimizeLegibility;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
a {
  font-weight: 500;
  color: #646cff;
  text-decoration: inherit;
}
a:hover {
  color: #535bf2;
}
body {
  margin: 0;
```

```
  display: flex;
  place-items: center;
  min-width: 320px;
  min-height: 100vh;
}
h1 {
  font-size: 3.2em;
  line-height: 1.1;
}
button {
  border-radius: 8px;
  border: 1px solid transparent;
  padding: 0.6em 1.2em;
  font-size: 1em;
  font-weight: 500;
  font-family: inherit;
  background-color: #1a1a1a;
  cursor: pointer;
  transition: border-color 0.25s;
}
```

```
button:hover {
  border-color: #646cff;
}
button:focus,
button:focus-visible {
  outline: 4px auto -webkit-focus-ring-color;
}
@media (prefers-color-scheme: light) {
  :root {
    color: #213547;
    background-color: #ffffff;
  }
  a:hover {
    color: #747bff;
  }
  button {
    background-color: #f9f9f9;
  }
}================================================
FILE 3/12: src/components/CostForm.jsx
```

```
/**
 * CostForm Component - Expense entry form with validation
 * Provides input fields for amount, currency, category, and description
 * Handles form validation, database persistence, and user feedback
 * @param {Object} props.onCostAdded - Callback function triggered after successful cost addition
 * @returns {JSX.Element} Form component for adding new expenses
 */

import { useState } from 'react';
import {
  Box,
  TextField,
  Button,
  MenuItem,
  Paper,
  Typography,
  Alert,
  CircularProgress,
  Grid
} from '@mui/material';
```

```
import { AddCircleOutline } from '@mui/icons-material';
import { openCostsDB } from '../utils/idb';
import { CURRENCIES, CATEGORIES } from '../utils/constants';

/**
 * Main CostForm component for expense entry
 * @param {Object} props - Component props
 * @param {Function} props.onCostAdded - Callback executed after successful cost addition
 */
const CostForm = ({ onCostAdded }) => {
  // Form data state - manages user input values
  const [formData, setFormData] = useState({
    sum: '',
    currency: 'USD',
    category: '',
    description: ''
  });

  // UI state - controls loading indicators and user feedback
  const [loading, setLoading] = useState(false);
```

```
const [error, setError] = useState(null);
const [success, setSuccess] = useState(false);
const [validationErrors, setValidationErrors] = useState({});

/**
 * Handle input field changes and clear validation errors
 * Updates form data state as user types in any field
 * @param {Event} event - Input change event
 */
const handleChange = (event) => {
  // Extract field name and value from input event
  const { name, value } = event.target;

  // Update form state with new field value
  setFormData(prev => ({
    ...prev,
    [name]: value
  }));

  // Clear validation error for this field when user starts typing
```

```
    if (validationErrors[name]) {
      setValidationErrors(prev => ({
        ...prev,
        [name]: null
      }));
    }
};

/**
 * Validate all form fields before submission
 * Checks amount, category, and description for valid input
 * @returns {boolean} True if all validations pass, false otherwise
 */
const validateForm = () => {
    const errors = {};

    // Validate amount field - must be positive number within limits
    const sumValue = parseFloat(formData.sum);
    if (!formData.sum || isNaN(sumValue) || sumValue <= 0) {
      errors.sum = 'Please enter a valid positive amount';
```

```
} else if (sumValue > 999999999) {
   errors.sum = 'Amount is too large (maximum: 999,999,999)';
}

// Validate category selection - required field
if (!formData.category) {
   errors.category = 'Please select a category';
}

// Validate description - required field with length limit
const trimmedDescription = formData.description.trim();
if (!trimmedDescription) {
   errors.description = 'Please enter a description';
} else if (trimmedDescription.length > 500) {
   errors.description = 'Description is too long (maximum: 500 characters)';
}

// Store validation errors and return whether form is valid
setValidationErrors(errors);
return Object.keys(errors).length === 0;
```

```
};

/**
 * Handle form submission with validation and database persistence
 * Validates input, saves to IndexedDB, and provides user feedback
 * @param {Event} event - Form submit event
 */
const handleSubmit = async (event) => {
  // Prevent default form submission behavior
  event.preventDefault();

  // Reset UI feedback states
  setError(null);
  setSuccess(false);

  // Validate all form fields before proceeding
  if (!validateForm()) {
    return; // Stop submission if validation fails
  }
```

```
// Show loading indicator during save operation
setLoading(true);

try {
  // Prepare cost object with validated and formatted data
  const cost = {
    sum: parseFloat(formData.sum),
    currency: formData.currency,
    category: formData.category,
    description: formData.description.trim(),
    date: new Date().toISOString()
  };

  // Connect to database and save the new cost entry
  const db = await openCostsDB("costsdb", 1);
  await db.addCost(cost);

  // Display success feedback to user
  setSuccess(true);
```

```
    // Reset form to initial state for next entry
    setFormData({
      sum: '',
      currency: 'USD',
      category: '',
      description: ''
    });

    // Notify parent component
    if (onCostAdded) {
      onCostAdded();
    }

    // Clear success message after 3 seconds
    setTimeout(() => {
      setSuccess(false);
    }, 3000);

  } catch (err) {
    setError(`Failed to add cost: ${err.message}`);
```

```
    } finally {
      setLoading(false);
    }
  };

  return (
    <Paper
      elevation={3}
      sx={{
        p: { xs: 2, sm: 3, md: 4 },
        maxWidth: 600,
        mx: 'auto',
        mt: { xs: 1, sm: 2, md: 3 },
        borderRadius: { xs: 2, sm: 3 }
      }}
    >
      <Box sx={{ display: 'flex', alignItems: 'center', mb: { xs: 2, sm: 3 } }}>
        <AddCircleOutline sx={{ fontSize: { xs: 28, sm: 32 }, mr: 1, color: 'primary.main' }} />
        <Typography variant="h5" component="h2" sx={{ fontSize: { xs: '1.25rem', sm: '1.5rem' } }}>
          Add New Expense
```

```
    </Typography>
  </Box>

  {/* Error Alert */}
  {error && (
    <Alert severity="error" sx={{ mb: 2 }} onClose={() => setError(null)}>
      {error}
    </Alert>
  )}

  {/* Success Alert */}
  {success && (
    <Alert severity="success" sx={{ mb: 2 }}>
      Cost added successfully!
    </Alert>
  )}

  <form onSubmit={handleSubmit}>
    <Grid container spacing={{ xs: 2, sm: 3 }}>
      {/* Amount Field */}
```

```
<Grid item xs={12} sm={6}>
  <TextField
    fullWidth
    label="Amount"
    name="sum"
    type="number"
    value={formData.sum}
    onChange={handleChange}
    inputProps={{
      step: '0.01',
      min: '0',
      style: { fontSize: '1.1rem' }
    }}
    slotProps={{
      inputLabel: {
        shrink: true,
        sx: { fontSize: '1rem', fontWeight: 500 }
      }
    }}
    error={!!validationErrors.sum}
```

```
      helperText={validationErrors.sum || 'Enter the expense amount'}
      disabled={loading}
      required
      placeholder="0.00"
    />
</Grid>

{/* Currency Field */}
<Grid item xs={12} sm={6}>
   <TextField
      fullWidth
      select
      label="Currency"
      name="currency"
      value={formData.currency}
      onChange={handleChange}
      disabled={loading}
      required
      slotProps={{
         inputLabel: {
```

```
        shrink: true,
        sx: { fontSize: '1rem', fontWeight: 500 }
      }
    }}
    SelectProps={{
      sx: { fontSize: '1.1rem' }
    }}
    helperText="Select currency type"
  >
    {CURRENCIES.map((currency) => (
      <MenuItem
        key={currency}
        value={currency}
        sx={{ fontSize: '1rem', py: 1.5 }}
      >
        {currency === 'USD' && '$ USD - US Dollar'}
        {currency === 'EURO' && '€ EUR - Euro'}
        {currency === 'GBP' && '£ GBP - British Pound'}
        {currency === 'ILS' && ' ₪ILS - Israeli Shekel'}
      </MenuItem>
```

```
      ))}
    </TextField>
</Grid>

{/* Category Field */}
<Grid item xs={12}>
    <TextField
      fullWidth
      select
      label="Category"
      name="category"
      value={formData.category}
      onChange={handleChange}
      error={!!validationErrors.category}
      helperText={validationErrors.category || 'Select expense category'}
      disabled={loading}
      required
      slotProps={{
        inputLabel: {
          shrink: true,
```

```
        sx: { fontSize: '1rem', fontWeight: 500 }
      }
    }}
    SelectProps={{
      sx: { fontSize: '1rem' }
    }}
  >
    {CATEGORIES.map((category) => (
      <MenuItem
        key={category}
        value={category}
        sx={{
          py: 1.5,
          fontSize: '1rem',
          '&:hover': { bgcolor: 'action.hover' }
        }}
      >
        {category}
      </MenuItem>
    ))}
```

```
    </TextField>
  </Grid>

  {/* Description Field */}
  <Grid item xs={12}>
    <TextField
      fullWidth
      label="Description"
      name="description"
      value={formData.description}
      onChange={handleChange}
      multiline
      rows={3}
      error={!!validationErrors.description}
      helperText={validationErrors.description || 'Provide details about this expense'}
      disabled={loading}
      required
      slotProps={{
        inputLabel: {
          shrink: true,
```

```
            sx: { fontSize: '1rem', fontWeight: 500 }
          }
        }}
        inputProps={{
          style: { fontSize: '1rem' }
        }}
        placeholder="What was this expense for?"
      />
    </Grid>

    {/* Submit Button */}
    <Grid item xs={12}>
      <Button
        type="submit"
        variant="contained"
        fullWidth
        size="large"
        disabled={loading}
        startIcon={loading ? <CircularProgress size={20} /> : <AddCircleOutline />}
        sx={{
```

```
                    py: { xs: 1.5, sm: 1.75 },
                    fontSize: { xs: '0.95rem', sm: '1rem' },
                    fontWeight: 600,
                    borderRadius: 2
                  }}
                >
                  {loading ? 'Adding...' : 'Add Expense'}
                </Button>
              </Grid>
            </Grid>
          </form>
        </Paper>
    );
};
export default CostForm;
==================================================

FILE 4/12: src/components/Dashboard.jsx

/**
 * Dashboard Component - Main data visualization and reporting interface
 * Displays monthly expense reports, category pie charts, and yearly bar charts
```

```
 * Handles currency conversion and responsive design for mobile/desktop
 */

import { useState, useEffect } from 'react';
import { Box, Paper, CircularProgress, Alert, Tabs, Tab } from '@mui/material';
import { openCostsDB } from '../utils/idb';
import { fetchAndConvertWithUrl } from '../utils/helperFunctions';
import DashboardFilters from './DashboardFilters';
import MonthlyCostTable from './MonthlyCostTable';
import CategoryPieChart from "./CategoryPieChart.jsx";
import YearlyBarChart from "./YearlyBarChart.jsx";

/**
 * Main Dashboard component with expense tracking and visualization
 * @param {Object} props - Component props
 * @param {any} props.refreshTrigger - Triggers data reload when changed
 * @returns {JSX.Element} Dashboard component with tabbed interface
 */
const Dashboard = ({ refreshTrigger }) => {
   // Filter state - user-selected time period and currency
```

```
const [selectedMonth, setSelectedMonth] = useState(new Date().getMonth());
const [selectedYear, setSelectedYear] = useState(new Date().getFullYear());
const [displayCurrency, setDisplayCurrency] = useState('USD');
const [activeTab, setActiveTab] = useState(0);

// Data state - holds processed expense information
const [monthlyCosts, setMonthlyCosts] = useState([]);
const [categoryData, setCategoryData] = useState([]);
const [monthlyData, setMonthlyData] = useState([]);

// UI state - manages loading and error display
const [loading, setLoading] = useState(true);
const [error, setError] = useState(null);
const [monthlyTotal, setMonthlyTotal] = useState(0);

/**
 * Effect hook to reload data when filters or refresh trigger changes
 * Monitors: selectedMonth, selectedYear, displayCurrency, refreshTrigger
 */
useEffect(() => {
```

```
    loadData();
}, [selectedMonth, selectedYear, displayCurrency, refreshTrigger]);


/**
 * Effect hook to recalculate monthly total when costs or currency changes
 * Updates total whenever monthlyCosts or displayCurrency changes
 */
useEffect(() => {
    const updateTotal = async () => {
      // Calculate total only if we have cost data
      if (monthlyCosts.length > 0) {
        const total = await calculateTotal();
        setMonthlyTotal(total);
      } else {
        setMonthlyTotal(0); // Reset to zero for empty months
      }
    };
    updateTotal();
}, [monthlyCosts, displayCurrency]);
```

```
/**
 * Fetch and process cost data from IndexedDB
 * Loads monthly costs, chart data with currency conversion
 */
const loadData = async () => {
  // Initialize loading state and clear previous errors
  setLoading(true);
  setError(null);

  // Open database connection
  const db = await openCostsDB("costsdb", 1);
  //await db.setSetting('exchangeRateUrl', DEFAULT_EXCHANGE_URL);

  try {
    // Fetch monthly costs from database
    const monthCostsData = await db.getCostsByMonth(selectedMonth, selectedYear);

    // Convert monthly costs to display currency with current exchange rates
    const costsWithConverted = await fetchAndConvertWithUrl(db, monthCostsData, displayCurrency);
    setMonthlyCosts(costsWithConverted);
```

```
      // Generate category breakdown data for pie chart visualization
      const categoryData = await db.getPieChartData(selectedYear, selectedMonth + 1, displayCurrency);
      setCategoryData(categoryData);

      // Generate yearly overview data for bar chart visualization
      const monthlyData = await db.getBarChartData(selectedYear, displayCurrency);
      setMonthlyData(monthlyData);

    } catch (err) {
      // Handle and display any data loading errors
      setError(`Failed to load data: ${err.message}`);
    } finally {
      // Ensure loading state is cleared regardless of success/failure
      setLoading(false);
    }
};

/**
 * Calculate total amount for monthly costs in display currency
```

```
 * @returns {Promise<number>} Total amount in selected display currency
 */
const calculateTotal = async () => {
  try {
    // Open database connection for exchange rate access
    const db = await openCostsDB("costsdb", 1);

    // Convert all monthly costs to display currency
    const costsWithConverted = await fetchAndConvertWithUrl(db, monthlyCosts, displayCurrency);

    // Sum all converted amounts for monthly total
    return costsWithConverted.reduce((total, cost) => total + cost.convertedAmount, 0);
  } catch (error) {
    // Return zero as safe fallback on error
    return 0;
  }
};

/**
 * Format currency amount for display with proper symbols and locale
```

```
 * Handles invalid amounts and unsupported currency codes gracefully
 * @param {number} amount - Amount to format
 * @returns {string} Formatted currency string
 */
const formatCurrency = (amount) => {
  // Sanitize input amount to prevent display errors
  if (amount === undefined || amount === null || isNaN(amount)) {
    amount = 0;
  }

  try {
    // Use browser's Intl API for proper currency formatting
    return new Intl.NumberFormat('en-US', {
      style: 'currency',
      currency: displayCurrency
    }).format(amount);
  } catch (error) {
    // Fallback formatting for unsupported currency codes (e.g., EURO)
    const symbol = displayCurrency === 'EURO' ? '€' : displayCurrency;
    return `${symbol}${amount.toFixed(2)}`;
```

```
  }
};

/**
 * Format date string for user-friendly display
 * @param {string} dateString - ISO date string or date-compatible string
 * @returns {string} Formatted date in 'MMM DD, YYYY' format
 */
const formatDate = (dateString) => {
  // Convert to Date object and format for display
  return new Date(dateString).toLocaleDateString('en-US', {
    year: 'numeric',
    month: 'short',
    day: 'numeric'
  });
};

// Generate year options for dropdown (current year and 5 previous years)
const yearOptions = Array.from({ length: 6 }, (_, i) => new Date().getFullYear() - i);
```

```
// Display loading spinner while data is being fetched
if (loading) {
  return (
    <Box sx={{ display: 'flex', justifyContent: 'center', alignItems: 'center', minHeight: 400 }}>
      <CircularProgress />
    </Box>
  );
}

return (
  <Box sx={{ p: { xs: 1, sm: 2, md: 3 } }}>
    {/* Filter controls for month, year, and currency selection */}
    <DashboardFilters
      selectedMonth={selectedMonth}
      setSelectedMonth={setSelectedMonth}
      selectedYear={selectedYear}
      setSelectedYear={setSelectedYear}
      displayCurrency={displayCurrency}
      setDisplayCurrency={setDisplayCurrency}
      yearOptions={yearOptions}
```

```
/>

{/* Display error message if data loading fails */}
{error && (
  <Alert severity="error" sx={{ mb: 2 }} onClose={() => setError(null)}>
    {error}
  </Alert>
)}

{/* Tab navigation for switching between different dashboard views */}
<Paper elevation={2} sx={{ mb: { xs: 2, sm: 3 }, borderRadius: 2 }}>
  <Tabs
    value={activeTab}
    onChange={(e, newValue) => setActiveTab(newValue)}
    variant="fullWidth"
    sx={{
      '& .MuiTab-root': {
        fontSize: { xs: '0.7rem', sm: '0.875rem' },
        minHeight: { xs: 48, sm: 56 },
        py: { xs: 1, sm: 1.5 },
```

```
            px: { xs: 0.5, sm: 2 }
          }
        }}
      >
        {/* Tab 0: Monthly Report - shows full expense table */}
        <Tab label={<Box sx={{ display: { xs: 'none', sm: 'block' } }}>Monthly Report</Box>}
          icon={<Box sx={{ display: { xs: 'block', sm: 'none' }, fontSize: '0.75rem' }}>Report</Box>} />
        {/* Tab 1: Category Chart - shows pie chart breakdown */}
        <Tab label={<Box sx={{ display: { xs: 'none', sm: 'block' } }}>Category Chart</Box>}
          icon={<Box sx={{ display: { xs: 'block', sm: 'none' }, fontSize: '0.75rem' }}>Categories</Box>} />
        {/* Tab 2: Yearly Overview - shows bar chart of monthly trends */}
        <Tab label={<Box sx={{ display: { xs: 'none', sm: 'block' } }}>Yearly Overview</Box>}
          icon={<Box sx={{ display: { xs: 'block', sm: 'none' }, fontSize: '0.75rem' }}>Yearly</Box>} />
      </Tabs>
    </Paper>

    {/* Tab 0: Monthly Report - detailed expense table with summary */}
    {activeTab === 0 && (
      <MonthlyCostTable
        monthlyCosts={monthlyCosts}
```

```
      monthlyTotal={monthlyTotal}
      selectedMonth={selectedMonth}
      selectedYear={selectedYear}
      displayCurrency={displayCurrency}
      formatCurrency={formatCurrency}
      formatDate={formatDate}
    />
)}

{/* Tab 1: Category Pie Chart - visual breakdown by expense category */}
{activeTab === 1 && (
    <CategoryPieChart
      categoryData={categoryData}
      selectedMonth={selectedMonth}
      selectedYear={selectedYear}
      formatCurrency={formatCurrency}
    />
)}

{/* Tab 2: Yearly Bar Chart - monthly spending trends for selected year */}
```

```
        {activeTab === 2 && (
          <YearlyBarChart
            monthlyData={monthlyData}
            selectedYear={selectedYear}
            displayCurrency={displayCurrency}
            formatCurrency={formatCurrency}
          />
        )}
      </Box>
    );
};
export default Dashboard;
================================================

FILE 5/12: src/components/DashboardFilters.jsx

/**
 * DashboardFilters Component - Filter controls for dashboard
 * Manages month, year, and currency selection for expense data filtering
 * Provides responsive UI with mobile-optimized display options
 */
```

```
import {
  Box,
  Paper,
  TextField,
  MenuItem,
  Grid
} from '@mui/material';
import { MONTHS, CURRENCIES } from '../utils/constants';

/**
 * DashboardFilters function for selecting time period and currency filters
 * @param {Object} props - Component props
 * @param {number} props.selectedMonth - Currently selected month index (0-11)
 * @param {Function} props.setSelectedMonth - Callback to update selected month
 * @param {number} props.selectedYear - Currently selected year
 * @param {Function} props.setSelectedYear - Callback to update selected year
 * @param {string} props.displayCurrency - Currently selected currency code
 * @param {Function} props.setDisplayCurrency - Callback to update display currency
 * @param {Array} props.yearOptions - Array of available year options for dropdown
 * @returns {JSX.Element} Filter controls component with month, year, and currency selectors
```

```
 */
const DashboardFilters = ({
                    selectedMonth,
                    setSelectedMonth,
                    selectedYear,
                    setSelectedYear,
                    displayCurrency,
                    setDisplayCurrency,
                    yearOptions
                  }) => {
  return (
    <Paper elevation={2} sx={{ p: { xs: 1.5, sm: 2, md: 3 }, mb: { xs: 2, sm: 3 }, borderRadius: 2 }}>
      {/* Grid layout for filter controls - responsive column sizing */}
      <Grid container spacing={{ xs: 1.5, sm: 2 }} alignItems="center">
        {/* Month selection dropdown - responsive display (full name on desktop, abbreviation on mobile) */}
        <Grid item xs={4} sm={4}>
          <TextField
            fullWidth
            select
            label="Month"
```

```
      value={selectedMonth}
      onChange={(e) => setSelectedMonth(e.target.value)}
      size="small"
      slotProps={{ inputLabel: { sx: { fontSize: { xs: '0.8rem', sm: '1rem' } } } }}
    >
      {/* Render month options with responsive text display */}
      {MONTHS.map((month, index) => (
        <MenuItem key={month} value={index}>
          {/* Full month name on larger screens */}
          <Box sx={{ display: { xs: 'none', sm: 'block' } }}>{month}</Box>
          {/* Abbreviated month name (first 3 letters) on mobile */}
          <Box sx={{ display: { xs: 'block', sm: 'none' } }}>{month.substring(0, 3)}</Box>
        </MenuItem>
      ))}
    </TextField>
  </Grid>

  {/* Year selection dropdown - displays available year options */}
  <Grid item xs={4} sm={4}>
    <TextField
```

```
      fullWidth
      select
      label="Year"
      value={selectedYear}
      onChange={(e) => setSelectedYear(e.target.value)}
      size="small"
      slotProps={{ inputLabel: { sx: { fontSize: { xs: '0.8rem', sm: '1rem' } } } }}
    >
      {/* Render year options from provided yearOptions array */}
      {yearOptions.map((year) => (
        <MenuItem key={year} value={year}>
          {year}
        </MenuItem>
      ))}
    </TextField>
</Grid>

{/* Currency selection dropdown - displays supported currencies */}
<Grid item xs={4} sm={4}>
    <TextField
```

```
                fullWidth
                select
                label="Currency"
                value={displayCurrency}
                onChange={(e) => setDisplayCurrency(e.target.value)}
                size="small"
                slotProps={{ inputLabel: { sx: { fontSize: { xs: '0.8rem', sm: '1rem' } } } }}
              >
                {/* Render currency options from constants */}
                {CURRENCIES.map((currency) => (
                  <MenuItem key={currency} value={currency}>
                    {currency}
                  </MenuItem>
                ))}
              </TextField>
            </Grid>
          </Grid>
        </Paper>
    );
};
```

```
export default DashboardFilters;
==================================================

FILE 6/12: src/components/MonthlyCostTable.jsx

/**
 * MonthlyCostTable Component - Monthly expense report with summary and table
 * Displays total expenses card and detailed cost table with currency conversion
 * Shows original currency amounts and converted amounts in display currency
 */

import {
  Box,
  Card,
  CardContent,
  Typography,
  Alert,
  Table,
  TableBody,
  TableCell,
  TableContainer,
  TableHead,
```

```
  TableRow,
  Chip,
  Paper
} from '@mui/material';
import { MONTHS } from '../utils/constants';

/**
 * MonthlyCostTable function for displaying monthly expense summary and details
 * @param {Object} props - Component props
 * @param {Array} props.monthlyCosts - Array of cost objects for selected month
 * @param {number} props.monthlyTotal - Total expenses for the month in display currency
 * @param {number} props.selectedMonth - Selected month index (0-11)
 * @param {number} props.selectedYear - Selected year
 * @param {string} props.displayCurrency - Currency code for converted amounts
 * @param {Function} props.formatCurrency - Function to format amount as currency string
 * @param {Function} props.formatDate - Function to format date string for display
 * @returns {JSX.Element} Monthly expense summary card and detailed cost table
 */
const MonthlyCostTable = ({
                        monthlyCosts,
```

```
                monthlyTotal,
                selectedMonth,
                selectedYear,
                displayCurrency,
                formatCurrency,
                formatDate
              }) => {
return (
  <Box>
    {/* Summary card displaying total expenses for selected month */}
    <Card sx={{ mb: { xs: 2, sm: 3 }, borderRadius: 2 }}>
      <CardContent sx={{ p: { xs: 2, sm: 3 }, '&:last-child': { pb: { xs: 2, sm: 3 } } }}>
        {/* Month and year header */}
        <Typography variant="h6" gutterBottom sx={{ fontSize: { xs: '1rem', sm: '1.25rem' } }}>
          {MONTHS[selectedMonth]} {selectedYear} Summary
        </Typography>
        {/* Total amount in display currency - prominently displayed */}
        <Typography variant="h4" color="primary" sx={{ fontSize: { xs: '1.5rem', sm: '2rem', md: '2.125rem' } }}>
          {formatCurrency(monthlyTotal)}
        </Typography>
```

```
      {/* Transaction count summary */}
      <Typography variant="body2" color="text.secondary">
        Total expenses ({monthlyCosts.length} transactions)
      </Typography>
    </CardContent>
  </Card>

  {/* Display empty state message if no expenses for selected month */}
  {monthlyCosts.length === 0 ? (
    <Alert severity="info">
      No expenses recorded for {MONTHS[selectedMonth]} {selectedYear}
    </Alert>
  ) : (
    /* Detailed expense table with scrollable container for mobile */
    <TableContainer component={Paper} sx={{ borderRadius: 2, maxHeight: { xs: 400, sm: 'none' } }}>
      <Table size="small" stickyHeader>
        {/* Table header with column labels */}
        <TableHead>
          <TableRow>
            <TableCell sx={{ fontSize: { xs: '0.75rem', sm: '0.875rem' }, py: { xs: 1, sm: 2 } }}>Date</TableCell>
```

```
                {/* Category column hidden on mobile for space optimization */}
                <TableCell sx={{ fontSize: { xs: '0.75rem', sm: '0.875rem' }, py: { xs: 1, sm: 2 }, display: { xs: 'none', md: 'table-cell' }
}}>Category</TableCell>
                <TableCell sx={{ fontSize: { xs: '0.75rem', sm: '0.875rem' }, py: { xs: 1, sm: 2 } }}>Description</TableCell>
                {/* Original amount in original currency */}
                <TableCell align="right" sx={{ fontSize: { xs: '0.75rem', sm: '0.875rem' }, py: { xs: 1, sm: 2 } }}>Amount</TableCell>
                {/* Converted amount in display currency */}
                <TableCell align="right" sx={{ fontSize: { xs: '0.75rem', sm: '0.875rem' }, py: { xs: 1, sm: 2 } }}>{displayCurrency}</TableCell>
              </TableRow>
            </TableHead>
            {/* Table body with expense rows */}
            <TableBody>
              {monthlyCosts.map((cost) => {
                // Extract converted amount for display
                const convertedAmount = cost.convertedAmount;
                return (
                  <TableRow key={cost.id} hover>
                    {/* Formatted date display */}
                    <TableCell sx={{ fontSize: { xs: '0.7rem', sm: '0.875rem' }, py: { xs: 0.75, sm: 1.5 } }}>
                      {formatDate(cost.date)}
```

```
</TableCell>
{/* Category chip - hidden on mobile screens */}
<TableCell sx={{ display: { xs: 'none', md: 'table-cell' }, py: { xs: 0.75, sm: 1.5 } }}>
  <Chip label={cost.category} size="small" color="primary" variant="outlined" />
</TableCell>
{/* Description with text truncation for long entries */}
<TableCell sx={{
  fontSize: { xs: '0.7rem', sm: '0.875rem' },
  py: { xs: 0.75, sm: 1.5 },
  maxWidth: { xs: 100, sm: 200 },
  overflow: 'hidden',
  textOverflow: 'ellipsis',
  whiteSpace: 'nowrap'
}}>
  {cost.description}
</TableCell>
{/* Original amount with currency code */}
<TableCell align="right" sx={{ fontSize: { xs: '0.7rem', sm: '0.875rem' }, py: { xs: 0.75, sm: 1.5 } }}>
  {cost.sum.toFixed(2)} {cost.currency}
</TableCell>
```

```
                        {/* Converted amount in display currency - emphasized with bold font */}
                        <TableCell align="right" sx={{ fontSize: { xs: '0.7rem', sm: '0.875rem' }, py: { xs: 0.75, sm: 1.5 }, fontWeight: 600 }}>
                          {formatCurrency(convertedAmount)}
                        </TableCell>
                      </TableRow>
                    );
                  })}
                </TableBody>
              </Table>
            </TableContainer>
          )}
        </Box>
    );
};
export default MonthlyCostTable;
```

==========

FILE 7/12: src/components/CategoryPieChart.jsx

```
/**
 * CategoryPieChart Component - Pie chart visualization of expenses by category
 * Displays category breakdown and legend for selected month
```

```
 * Used in Dashboard Tab 1 (Category Chart)
 */

import { Box, Paper, Typography, Alert, Grid } from '@mui/material';
import { PieChart, Pie, Cell, ResponsiveContainer, Tooltip } from 'recharts';
import { MONTHS, COLORS } from '../utils/constants';

/**
 * CategoryPieChart function for displaying expense breakdown by category
 * @param {Object} props - Component props
 * @param {Array} props.categoryData - Array of category objects with category name and amount
 * @param {number} props.selectedMonth - Selected month index (0-11)
 * @param {number} props.selectedYear - Selected year (e.g., 2025)
 * @param {Function} props.formatCurrency - Function to format amount as currency string
 * @returns {JSX.Element} Pie chart with category legend
 */
const CategoryPieChart = ({
                        categoryData,
                        selectedMonth,
                        selectedYear,
```

```
                formatCurrency
              }) => {
  return (
    <Paper elevation={2} sx={{ p: { xs: 1.5, sm: 2, md: 3 }, borderRadius: 2 }}>
      {/* Chart title with selected month and year */}
      <Typography variant="h6" gutterBottom sx={{ fontSize: { xs: '1rem', sm: '1.25rem' } }}>
        Expenses by Category - {MONTHS[selectedMonth]} {selectedYear}
      </Typography>

      {/* Display empty state message if no category data available */}
      {categoryData.length === 0 ? (
        <Alert severity="info">
          No data available for the selected month
        </Alert>
      ) : (
        <>
          {/* Responsive container for pie chart visualization */}
          <Box sx={{ width: '100%', height: { xs: 280, sm: 350, md: 400 } }}>
            <ResponsiveContainer width="100%" height="100%">
              <PieChart>
```

```
<Pie
  // Transform category data to Recharts format (requires 'name' and 'value' keys)
  data={categoryData.map(item => ({
    ...item,
    name: item.category || item.name || 'Unknown',
    value: item.amount || item.value || 0
  }))}
  cx="50%"
  cy="50%"
  labelLine={false}
  // Custom label function to show category name and percentage
  label={({ name, percent }) => {
    // Truncate long category names for better readability
    const shortName = name.length > 10 ? name.substring(0, 10) + '...' : name;
    return `${shortName} (${(percent * 100).toFixed(0)}%)`;
  }}
  outerRadius="70%"
  fill="#8884d8"
  dataKey="value"
>
```

```
            {/* Assign colors to pie chart segments using color palette */}
            {categoryData.map((entry, index) => (
              <Cell key={`cell-${index}`} fill={COLORS[index % COLORS.length]} />
            ))}
          </Pie>
          {/* Format tooltip values as currency when hovering over segments */}
          <Tooltip formatter={(value) => formatCurrency(value)} />
        </PieChart>
      </ResponsiveContainer>
    </Box>

    {/* Category Legend - displays color-coded category list with amounts */}
    <Grid container spacing={{ xs: 1, sm: 2 }} sx={{ mt: { xs: 1, sm: 2 } }}>
      {categoryData.map((item, index) => (
        <Grid item xs={6} sm={4} md={3} key={item.category}>
          <Box sx={{ display: 'flex', alignItems: 'center' }}>
            {/* Color indicator box matching pie chart segment color */}
            <Box
              sx={{
                width: { xs: 12, sm: 16 },
```

```
                height: { xs: 12, sm: 16 },
                bgcolor: COLORS[index % COLORS.length],
                mr: 1,
                borderRadius: 0.5,
                flexShrink: 0
              }}
            />
            {/* Category name and formatted amount display */}
            <Typography variant="body2" sx={{
              fontSize: { xs: '0.7rem', sm: '0.875rem' },
              overflow: 'hidden',
              textOverflow: 'ellipsis',
              whiteSpace: 'nowrap'
            }}>
              {item.category}: {formatCurrency(item.amount)}
            </Typography>
          </Box>
        </Grid>
      ))}
    </Grid>
```

```
            </>
          )}
      </Paper>
    );
};
export default CategoryPieChart;
==================================================
FILE 8/12: src/components/YearlyBarChart.jsx
/**
 * YearlyBarChart Component - Bar chart visualization of monthly expenses
 * Displays spending trends across all months of selected year
 * Used in Dashboard Tab 2 (Yearly Overview)
 */

import {
  Box,
  Paper,
  Typography,
  Alert
} from '@mui/material';
```

```
import {
  BarChart,
  Bar,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
  Legend,
  ResponsiveContainer
} from 'recharts';
import { MONTHS } from '../utils/constants';

/**
 * YearlyBarChart function for visualizing monthly expenses across a year
 * @param {Object} props - Component props
 * @param {Array} props.monthlyData - Array of monthly expense objects with amount and month
 * @param {number} props.selectedYear - Selected year for display
 * @param {string} props.displayCurrency - Currency code for amount display
 * @param {Function} props.formatCurrency - Function to format amount as currency string
 * @returns {JSX.Element} Bar chart component showing monthly spending trends
```

```
 */
const YearlyBarChart = ({
                    monthlyData,
                    selectedYear,
                    displayCurrency,
                    formatCurrency
                  }) => {
  return (
    <Paper elevation={2} sx={{ p: { xs: 1.5, sm: 2, md: 3 }, borderRadius: 2 }}>
      {/* Chart title with selected year */}
      <Typography variant="h6" gutterBottom sx={{ fontSize: { xs: '1rem', sm: '1.25rem' } }}>
        Monthly Expenses - {selectedYear}
      </Typography>

      {/* Display empty state if no data or all amounts are zero */}
      {!monthlyData || monthlyData.length === 0 || monthlyData.every(item => item.amount === 0) ? (
        <Alert severity="info">
          No data available for {selectedYear}
        </Alert>
      ) : (
```

```
/* Responsive container for bar chart visualization */
<Box sx={{ width: '100%', height: { xs: 280, sm: 350, md: 400 } }}>
  <ResponsiveContainer width="100%" height="100%">
    <BarChart
      // Transform data to include abbreviated month names for X-axis labels
      data={monthlyData.map(item => ({
        ...item,
        // Convert month number (1-12) to abbreviated month name (Jan, Feb, etc.)
        monthName: MONTHS[item.month - 1] ? MONTHS[item.month - 1].substring(0, 3) : `M${item.month}`
      }))}
      margin={{ top: 5, right: 5, left: 0, bottom: 5 }}
    >
      {/* Grid lines for easier value reading */}
      <CartesianGrid strokeDasharray="3 3" />
      {/* X-axis with month abbreviations - show all months without skipping */}
      <XAxis
        dataKey="monthName"
        tick={{ fontSize: 10 }}
        interval={0}
      />
```

```
                {/* Y-axis for amount values with compact width */}
                <YAxis tick={{ fontSize: 10 }} width={50} />
                {/* Tooltip showing formatted currency values on hover */}
                <Tooltip formatter={(value) => formatCurrency(value)} />
                {/* Legend showing currency code */}
                <Legend wrapperStyle={{ fontSize: '12px' }} />
                {/* Bar chart with amount data and currency label */}
                <Bar dataKey="amount" fill="#0088FE" name={`Amount (${displayCurrency})`} />
              </BarChart>
            </ResponsiveContainer>
          </Box>
        )}
      </Paper>
    );
};
export default YearlyBarChart;
==================================================
FILE 9/12: src/components/Settings.jsx
/**
 * Settings Component - Application configuration interface
```

```
 * Manages custom exchange rate API URLs and displays application information
 * Provides URL validation, connection testing, and fallback handling
 * @returns {JSX.Element} Settings page with URL configuration and app info
 */

import { useState, useEffect } from 'react';
import {
  Box,
  Paper,
  Typography,
  TextField,
  Button,
  Alert,
  CircularProgress,
  Divider
} from '@mui/material';
import { Settings as SettingsIcon, Save, Refresh } from '@mui/icons-material';
import { openCostsDB } from '../utils/idb';

/**
```

```
 * Main Settings component for application configuration
 */
const Settings = () => {
  // Form state - manages user input for exchange rate URL
  const [exchangeRateUrl, setExchangeRateUrl] = useState('');

  // UI state - controls loading indicators and user feedback
  const [loading, setLoading] = useState(false);
  const [initialLoading, setInitialLoading] = useState(true);
  const [error, setError] = useState(null);
  const [success, setSuccess] = useState(false);
  const [testingConnection, setTestingConnection] = useState(false);

  /**
   * Effect hook to load saved settings when component mounts
   */
  useEffect(() => {
    loadSettings();
  }, []);
```

```
/**
 * Load saved exchange rate URL from IndexedDB on component initialization
 */
const loadSettings = async () => {
  // Show initial loading indicator
  setInitialLoading(true);

  try {
    // Connect to database and retrieve saved URL setting
    const db = await openCostsDB("costsdb", 1);
    const savedUrl = await db.getSetting('exchangeRateUrl');

    // Populate form field if URL was previously saved
    if (savedUrl) {
      setExchangeRateUrl(savedUrl);
    }
  } catch (err) {
    // Display error if settings cannot be loaded
    setError(` Failed to load settings: ${err.message}` );
  } finally {
```

```
      // Hide loading indicator regardless of outcome
      setInitialLoading(false);
    }
};

/**
 * Handle saving exchange rate URL to database
 * Validates URL format and provides user feedback
 */
const handleSave = async () => {
    // Initialize saving state and clear previous messages
    setLoading(true);
    setError(null);
    setSuccess(false);

    try {
      // Validate URL format before saving (if URL provided)
      if (exchangeRateUrl && !isValidUrl(exchangeRateUrl)) {
        throw new Error('Please enter a valid URL');
      }
```

```
    // Connect to database and persist the URL setting
    const db = await openCostsDB("costsdb", 1);
    await db.setSetting('exchangeRateUrl', exchangeRateUrl);

    // Show success feedback to user
    setSuccess(true);

    // Auto-hide success message after 3 seconds
    setTimeout(() => {
      setSuccess(false);
    }, 3000);

  } catch (err) {
    // Display error message if save operation fails
    setError(`Failed to save settings: ${err.message}`);
  } finally {
    // Clear loading state regardless of success/failure
    setLoading(false);
  }
```

```
};

/**
 * Test connection to the exchange rate API URL
 * Validates URL accessibility and response format
 */
const handleTestConnection = async () => {
  // Validate that URL is provided before testing
  if (!exchangeRateUrl) {
    setError('Please enter a URL to test');
    return;
  }

  // Initialize testing state and clear previous messages
  setTestingConnection(true);
  setError(null);

  try {
    // Attempt to fetch data from the provided URL
    const response = await fetch(exchangeRateUrl);
```

```
// Check if HTTP request was successful
if (!response.ok) {
  throw new Error(`HTTP error! status: ${response.status}`);
}

// Parse JSON response from the API
const data = await response.json();

// Validate that response contains expected data structure
if (!data || typeof data !== 'object') {
  throw new Error('Invalid response format');
}

// Check for required currencies (accept both EUR and EURO formats)
const hasEuro = data['EUR'] !== undefined || data['EURO'] !== undefined;
const requiredCurrencies = ['USD', 'GBP', 'ILS'];
const missingCurrencies = requiredCurrencies.filter(curr =>
  data[curr] === undefined && data[curr.toUpperCase()] === undefined
);
```

```
// Add EUR/EURO to missing currencies if neither is found
if (!hasEuro) {
  missingCurrencies.push('EUR/EURO');
}

// Note: Missing currencies detected but don't fail test
if (missingCurrencies.length > 0) {
  // Missing currencies logged but test continues
}

// Show success message for successful connection
setSuccess(true);
setError(null);

// Auto-hide success message after 3 seconds
setTimeout(() => {
  setSuccess(false);
}, 3000);
```

```
    } catch (err) {
      // Display detailed error message for failed connection
      setError(` Connection test failed: ${err.message}. Using default exchange rates. `);
    } finally {
      // Clear testing state regardless of outcome
      setTestingConnection(false);
    }
};

/**
 * Reset exchange rate URL to default (empty) setting
 * Clears custom URL and reverts to application default
 */
const handleReset = async () => {
    // Clear form field and UI messages
    //setExchangeRateUrl(DEFAULT_EXCHANGE_URL);
    setExchangeRateUrl('');
    setError(null);
    setSuccess(false);
```

```
    try {
      // Connect to database and clear the custom URL setting
      const db = await openCostsDB("costsdb", 1);
      await db.setSetting('exchangeRateUrl', '');

      // Show confirmation that reset was successful
      setSuccess(true);

      // Auto-hide success message after 3 seconds
      setTimeout(() => {
        setSuccess(false);
      }, 3000);
    } catch (err) {
      // Display error if reset operation fails
      setError(`Failed to reset settings: ${err.message}`);
    }
  };

  /**
   * Validate URL format using browser's URL constructor
```

```
 * @param {string} string - URL string to validate
 * @returns {boolean} True if URL is valid, false otherwise
 */
const isValidUrl = (string) => {
  try {
    // Use URL constructor for comprehensive validation
    new URL(string);
    return true;
  } catch (_) {
    // Return false for any invalid URL format
    return false;
  }
};

if (initialLoading) {
  return (
    <Box sx={{ display: 'flex', justifyContent: 'center', alignItems: 'center', minHeight: 400 }}>
      <CircularProgress />
    </Box>
  );
```

```
}

return (
  <Box sx={{ p: { xs: 1, sm: 2, md: 3 }, maxWidth: 800, mx: 'auto' }}>
    <Paper elevation={3} sx={{ p: { xs: 2, sm: 3, md: 4 }, borderRadius: 2 }}>
      <Box sx={{ display: 'flex', alignItems: 'center', mb: { xs: 2, sm: 3 } }}>
        <SettingsIcon sx={{ fontSize: { xs: 28, sm: 32 }, mr: 1, color: 'primary.main' }} />
        <Typography variant="h5" component="h2" sx={{ fontSize: { xs: '1.25rem', sm: '1.5rem' } }}>
          Application Settings
        </Typography>
      </Box>

      {/* Error Alert */}
      {error && (
        <Alert severity="error" sx={{ mb: 2 }} onClose={() => setError(null)}>
          {error}
        </Alert>
      )}

      {/* Success Alert */}
```

```
{success && (
  <Alert severity="success" sx={{ mb: 2 }}>
    Settings saved successfully!
  </Alert>
)}

{/* Exchange Rate URL Section */}
<Box sx={{ mb: { xs: 3, sm: 4 } }}>
  <Typography variant="h6" gutterBottom sx={{ fontSize: { xs: '1rem', sm: '1.25rem' } }}>
    Exchange Rate API
  </Typography>
  <Typography variant="body2" color="text.secondary" sx={{ mb: 2, fontSize: { xs: '0.8rem', sm: '0.875rem' } }}>
    Configure a custom URL for fetching real-time exchange rates. If not set or if the URL fails,
    the app uses a default API endpoint. Note: Exchange rates may not reflect real-time values
    depending on the data source.
  </Typography>

  <TextField
    fullWidth
    label="Exchange Rate API URL"
```

```
    value={exchangeRateUrl}
    onChange={(e) => setExchangeRateUrl(e.target.value)}
    placeholder="https://api.example.com/rates"
    disabled={loading || testingConnection}
    sx={{ mb: 2 }}
    helperText="Example: https://api.exchangerate-api.com/v4/latest/USD"
    size="medium"
    slotProps={{
      inputLabel: {
        sx: { fontSize: { xs: '0.9rem', sm: '1rem' } }
      }
    }}
/>

<Box sx={{
    display: 'flex',
    gap: { xs: 1, sm: 2 },
    flexWrap: 'wrap',
    flexDirection: { xs: 'column', sm: 'row' }
}}>
```

```
<Button
  variant="contained"
  startIcon={loading ? <CircularProgress size={20} /> : <Save />}
  onClick={handleSave}
  disabled={loading || testingConnection}
  fullWidth={false}
  sx={{
    minWidth: { xs: '100%', sm: 'auto' },
    py: { xs: 1.25, sm: 1 }
  }}
>
  {loading ? 'Saving...' : 'Save Settings'}
</Button>

<Button
  variant="outlined"
  startIcon={testingConnection ? <CircularProgress size={20} /> : <Refresh />}
  onClick={handleTestConnection}
  disabled={loading || testingConnection || !exchangeRateUrl}
  sx={{
```

```
            minWidth: { xs: '100%', sm: 'auto' },
            py: { xs: 1.25, sm: 1 }
          }}
        >
          {testingConnection ? 'Testing...' : 'Test Connection'}
        </Button>

        <Button
          variant="text"
          onClick={handleReset}
          disabled={loading || testingConnection}
          sx={{
            minWidth: { xs: '100%', sm: 'auto' },
            py: { xs: 1.25, sm: 1 }
          }}
        >
          Reset to Default
        </Button>
      </Box>
    </Box>
```

```
<Divider sx={{ my: { xs: 2, sm: 3 } }} />

{/* Default Exchange Rates Info */}
<Box sx={{ mb: 3 }}>
  <Typography variant="h6" gutterBottom>
    Default Exchange Rates
  </Typography>
  <Typography variant="body2" color="text.secondary" sx={{ mb: 2 }}>
    When no custom URL is configured, exchange rates are fetched from the default API endpoint
    hosted on GitHub Pages. This ensures currency conversion is always available.
  </Typography>
</Box>

<Divider sx={{ my: 3 }} />

{/* Application Info */}
<Box>
  <Typography variant="h6" gutterBottom>
    About
```

```
</Typography>
<Typography variant="body2" color="text.secondary" paragraph>
  <strong>Cost Manager Application</strong>
</Typography>
<Typography variant="body2" color="text.secondary" paragraph>
  Version: 1.0.0
</Typography>
<Typography variant="body2" color="text.secondary" paragraph>
  This application helps you track and manage your expenses across multiple currencies.
  All data is stored locally in your browser using IndexedDB.
</Typography>
<Typography variant="body2" color="text.secondary">
  <strong>Features:</strong>
</Typography>
<ul style={{ marginTop: 8 }}>
  <li>
    <Typography variant="body2" color="text.secondary">
      Track expenses in multiple currencies (USD, EUR, GBP, ILS)
    </Typography>
  </li>
```

```
<li>
  <Typography variant="body2" color="text.secondary">
    Categorize expenses for better organization
  </Typography>
</li>
<li>
  <Typography variant="body2" color="text.secondary">
    View monthly reports with automatic currency conversion
  </Typography>
</li>
<li>
  <Typography variant="body2" color="text.secondary">
    Visualize spending patterns with interactive charts
  </Typography>
</li>
<li>
  <Typography variant="body2" color="text.secondary">
    All data stored securely in your browser
  </Typography>
</li>
```

```
            </ul>
          </Box>
        </Paper>
      </Box>
    );
};
export default Settings;
```

==========

FILE 10/12: src/utils/constants.js

```
/**
 * Application-wide constants
 * Centralized definitions for currencies, categories, months, colors, and API endpoints
 */
// Supported currencies for expense entry (EURO maps to EUR via normalizeRates)
export const CURRENCIES = ['USD', 'EURO', 'GBP', 'ILS'];
// Predefined expense categories for user selection
export const CATEGORIES = [
  'Food & Dining',
  'Transportation',
  'Housing',
```

```
  'Utilities',
  'Entertainment',
  'Healthcare',
  'Sport',
  'Shopping',
  'Education',
  'Travel',
  'Other'
];

// Month names for user-friendly date display
export const MONTHS = [
  'January', 'February', 'March', 'April', 'May', 'June',
  'July', 'August', 'September', 'October', 'November', 'December'
];

// Color palette for chart visualization - 10 distinct colors
export const COLORS = [
  '#0088FE', '#00C49F', '#FFBB28', '#FF8042', '#8884D8',
  '#82CA9D', '#FFC658', '#FF6B9D', '#C780E8', '#4ECDC4'
```

```
];
// Export default exchange rate API endpoint for use across application modules
export const DEFAULT_EXCHANGE_URL = 'https://shaidahari.github.io/exchaneRates_json/exchange-rates.json';
================================================
FILE 11/12: src/utils/helperFunctions.js
/**
 * Helper Functions - Centralized utilities for the Cost Manager application
 * Contains currency conversion, exchange rate fetching, and other shared utilities
 */

/**
 * Normalize rates object to handle EUR/EURO equivalence
 * The GitHub JSON uses "EURO" while the app uses "EUR"
 * @param {Object} rates - Exchange rates object from API
 * @returns {Object} Normalized rates with both EUR and EURO mappings
 */
export const normalizeRates = function (rates) {
   // Create copy to avoid mutating original rates object
   const normalized = { ...rates };
```

```
  // Ensure EUR/EURO equivalence for API compatibility
  if (normalized.EURO !== undefined && normalized.EUR === undefined) {
    normalized.EUR = normalized.EURO;
  }
  if (normalized.EUR !== undefined && normalized.EURO === undefined) {
    normalized.EURO = normalized.EUR;
  }
  return normalized;
};

/**
 * Convert amount from one currency to another using provided exchange rates
 * @param {number} amount - Amount to convert
 * @param {string} fromCurrency - Source currency code
 * @param {string} toCurrency - Target currency code
 * @param {Object} rates - Exchange rates object from API
 * @returns {number} Converted amount in target currency
 */
export const convertCurrency = function (amount, fromCurrency, toCurrency, rates) {
  // Skip conversion if currencies are the same
```

```
if (fromCurrency === toCurrency) {
  return amount;
}

// Validate amount is a valid number
if (typeof amount !== 'number' || isNaN(amount)) {
  return 0;
}

// Apply EUR/EURO normalization for API compatibility
const normalizedRates = normalizeRates(rates);

// Extract exchange rates for source and target currencies
const fromRate = normalizedRates[fromCurrency];
const toRate = normalizedRates[toCurrency];

// Handle missing exchange rates gracefully
if (fromRate === undefined || toRate === undefined) {
  return amount; // Return original amount as fallback
}
```

```
  // Convert via USD (base currency) then to target currency
  const usdAmount = amount / fromRate;
  return usdAmount * toRate;
};

/**
 * Fetch exchange rates from database settings and convert cost amounts to target currency
 * Handles the complete flow: get custom URL from settings -> fetch rates -> convert costs
 * @param {Object} db - Database wrapper instance with getExchangeRatesUrl method
 * @param {Array} costs - Array of cost objects to convert
 * @param {string} currency - Target currency code (USD, EUR, GBP, ILS, EURO)
 * @returns {Array} Costs array with added convertedAmount property for each cost
 */
export const fetchAndConvertWithUrl = async function (db, costs, currency) {
  // Early return for empty or invalid cost arrays
  if (!costs || costs.length === 0) {
    return [];
  }
```

```
// Retrieve exchange rate API URL from database settings
const exchangeUrl = await db.getExchangeRatesUrl();

// Fetch current exchange rates from configured API endpoint
const response = await fetch(exchangeUrl);

// Handle HTTP errors from exchange rate API
if (!response.ok) {
   throw new Error(` Failed to fetch exchange rates: HTTP ${response.status}` );
}

// Parse JSON response from exchange rate API
const rates = await response.json();

// Validate response format before processing
if (!rates || typeof rates !== 'object') {
   throw new Error('Invalid exchange rates format received from server');
}
// Apply currency conversion to each cost and add convertedAmount
return costs.map(cost => ({
```

```
      ...cost,
      convertedAmount: convertCurrency(cost.sum, cost.currency, currency, rates)
    }));
};==================================================
FILE 12/12: src/utils/idb.js
/**
 * IndexedDB Database Wrapper for Cost Manager Application
 * Provides Promise-based interface for storing costs and settings in browser's IndexedDB
 * Supports currency conversion with dynamic exchange rates
 */

import { fetchAndConvertWithUrl } from './helperFunctions';
import { DEFAULT_EXCHANGE_URL } from './constants';

/**
 * Opens and initializes the costs database with object stores
 * @param {string} databaseName - Name of the IndexedDB database
 * @param {number} databaseVersion - Version number for database schema
 * @returns {Promise<Object>} Database wrapper object with CRUD operations
 */
```

```
export const openCostsDB = async function (databaseName, databaseVersion) {
    return new Promise((resolve, reject) => {
        const request = indexedDB.open(databaseName, databaseVersion);

        request.onsuccess = function (event) {
            const db = event.target.result;

            const dbWrapper = {
                /** Add a new cost entry to the database with current timestamp */
                addCost: async function (cost) {
                    return new Promise((resolve, reject) => {
                        // Create transaction for writing to costs store
                        const transaction = db.transaction(['costs'], 'readwrite');
                        const store = transaction.objectStore('costs');

                        // Add current timestamp to cost object
                        const costWithDate = {
                            sum: cost.sum,
                            currency: cost.currency,
                            category: cost.category,
```

```
        description: cost.description,
        date: new Date()
      };

      // Add cost to IndexedDB store
      const request = store.add(costWithDate);

      request.onsuccess = function (event) {
        resolve(costWithDate);
      };

      request.onerror = function (event) {
        reject(event.target.error);
      };
    });
  },
  /** Retrieve all costs with auto-generated IndexedDB keys as id property */
  getAllCosts: async function () {
    return new Promise((resolve, reject) => {
      // Create read-only transaction
```

```
const transaction = db.transaction(['costs'], 'readonly');
const store = transaction.objectStore('costs');
const request = store.openCursor(); // Use cursor to get both data and keys
const costs = [];

request.onsuccess = function (event) {
  const cursor = event.target.result;
  if (cursor) {
    // Extract cost data and add IndexedDB key as id
    const cost = cursor.value; // Get the cost data
    cost.id = cursor.key; // Add IndexedDB auto-generated key as id property
    costs.push(cost);
    cursor.continue(); // Move to next record
  } else {
    resolve(costs); // No more records, return all costs with IDs
  }
};

request.onerror = function (event) {
  reject(new Error('Failed to get costs'));
```

```
      };
    });
  },
  /** Filter costs by specific month and year */
  getCostsByMonth: async function (month, year) {
    try {
      // Get all costs from database
      const allCosts = await this.getAllCosts();
      // Filter by matching month and year
      return allCosts.filter(cost => {
        const costDate = new Date(cost.date);
        return costDate.getMonth() === month && costDate.getFullYear() === year;
      });
    } catch (error) {
      throw new Error(` Error getting costs by month: ${error.message}` );
    }
  },
  /** Generate monthly report with currency-converted costs and total */
  getReport: async function (year, month, currency) {
    return new Promise(async (resolve, reject) => {
```

```
try {
  // Create read-only transaction
  const transaction = db.transaction(['costs'], 'readonly');
  const store = transaction.objectStore('costs');
  const request = store.getAll();

  request.onsuccess = async function (event) {
    try {
      const allCosts = event.target.result;

      // Filter costs by specified year and month
      const filteredCosts = allCosts.filter(cost => {
        const costDate = new Date(cost.date);
        return costDate.getFullYear() === year &&
          costDate.getMonth() + 1 === month;
      });

      // Format costs for report output
      const reportCosts = filteredCosts.map(cost => ({
        sum: cost.sum,
```

```
    currency: cost.currency,
    category: cost.category,
    description: cost.description,
    date: { day: new Date(cost.date).getDate() }
}));

// Convert costs to target currency
const costsWithConverted = await fetchAndConvertWithUrl(dbWrapper, filteredCosts, currency);

// Calculate total amount in target currency
const total = costsWithConverted.reduce((sum, cost) => {
    return sum + cost.convertedAmount;
}, 0);

// Build final report object
const report = {
    year,
    month,
    costs: reportCosts,
    total: { currency, total: Math.round(total * 100) / 100 }
```

```
            };
            resolve(report);
          }

          catch (error) {
            reject(error);
          }
        };

        request.onerror = function (event) {
          reject(event.target.error);
        };
      } catch (error) {
        reject(error);
      }
    });
  },
  /** Get category-based data for pie chart visualization with currency conversion */
  getPieChartData: async function (year, month, currency) {
    return new Promise(async (resolve, reject) => {
```

```
try {
  // Create read-only transaction
  const transaction = db.transaction(['costs'], 'readonly');
  const store = transaction.objectStore('costs');
  const request = store.getAll();

  request.onsuccess = async function (event) {
    try {
      const allCosts = event.target.result;

      // Filter costs by specified year and month
      const filteredCosts = allCosts.filter(cost => {
        const costDate = new Date(cost.date);
        return costDate.getFullYear() === year &&
          costDate.getMonth() + 1 === month;
      });

      // Convert costs to target currency
      const costsWithConverted = await fetchAndConvertWithUrl(dbWrapper, filteredCosts, currency);
```

```
    // Group costs by category and sum amounts
    const categoryTotals = {};

    costsWithConverted.forEach(cost => {
      if (!categoryTotals[cost.category]) {
        categoryTotals[cost.category] = 0;
      }
      categoryTotals[cost.category] += cost.convertedAmount;
    });

    // Format data for pie chart visualization
    const pieData = Object.keys(categoryTotals).map(category => ({
      category,
      amount: Math.round(categoryTotals[category] * 100) / 100,
      currency
    }));

    resolve(pieData);
  } catch (error) {
    reject(error);
```

```
            }
          };

          request.onerror = function (event) {
            reject(event.target.error);
          };
        } catch (error) {
          reject(error);
        }
      });
    },
    /** Get monthly spending data for bar chart visualization with currency conversion */
    getBarChartData: async function (year, currency) {
      return new Promise(async (resolve, reject) => {
        try {
          // Create read-only transaction
          const transaction = db.transaction(['costs'], 'readonly');
          const store = transaction.objectStore('costs');
          const request = store.getAll();
```

```
request.onsuccess = async function (event) {
  try {
    const allCosts = event.target.result;

    // Filter costs by specified year
    const filteredCosts = allCosts.filter(cost => {
      const costDate = new Date(cost.date);
      return costDate.getFullYear() === year;
    });

    // Convert costs to target currency
    const costsWithConverted = await fetchAndConvertWithUrl(dbWrapper, filteredCosts, currency);

    // Initialize monthly totals for all 12 months
    const monthlyTotals = {};

    for (let month = 1; month <= 12; month++) {
      monthlyTotals[month] = 0;
    }
```

```
      // Sum costs by month
      costsWithConverted.forEach(cost => {
        const costMonth = new Date(cost.date).getMonth() + 1;
        monthlyTotals[costMonth] += cost.convertedAmount;
      });

      // Format data for bar chart visualization
      const barData = Object.keys(monthlyTotals).map(month => ({
        month: parseInt(month),
        amount: Math.round(monthlyTotals[month] * 100) / 100,
        currency
      }));

      resolve(barData);
    } catch (error) {
      reject(error);
    }
};

request.onerror = function (event) {
```

```
            reject(event.target.error);
          };
        } catch (error) {
          reject(error);
        }
      });
  },

  /** Store application settings in database */
  setSetting: async function (key, value) {
    return new Promise((resolve, reject) => {
      // Create write transaction for settings store
      const transaction = db.transaction(['settings'], 'readwrite');
      const store = transaction.objectStore('settings');
      const request = store.put({ key, value });

      request.onsuccess = function (event) {
        resolve();
      };
```

```
      request.onerror = function (event) {
        reject(event.target.error);
      };
    });
},

/** Retrieve application setting value by key */
getSetting: async function (key) {
  return new Promise((resolve, reject) => {
    // Create read-only transaction for settings store
    const transaction = db.transaction(['settings'], 'readonly');
    const store = transaction.objectStore('settings');
    const request = store.get(key);

    request.onsuccess = function (event) {
      const result = event.target.result;
      resolve(result ? result.value : null);
    };

    request.onerror = function (event) {
```

```
            reject(event.target.error);
          };
        });
      },
      /** Get exchange rate API URL from settings or return default */
      getExchangeRatesUrl: async function () {
        try {
          // Try to get custom URL from settings
          const customUrl = await this.getSetting('exchangeRateUrl');
          return customUrl || DEFAULT_EXCHANGE_URL;
        } catch (error) {
          // Fall back to default URL if error occurs
          return DEFAULT_EXCHANGE_URL;
        }
      }
    };

    resolve(dbWrapper);
};
```

```
    request.onerror = function (event) {
      reject(event.target.error);
    };
    // Initialize database schema on first creation or version upgrade
    request.onupgradeneeded = function (event) {
      const db = event.target.result;
      // Create costs object store with auto-incrementing keys
      if (!db.objectStoreNames.contains('costs')) {
        const costsStore = db.createObjectStore('costs', { autoIncrement: true });
        costsStore.createIndex('date', 'date', { unique: false });
        costsStore.createIndex('category', 'category', { unique: false });
      }
      // Create settings object store for app configuration
      if (!db.objectStoreNames.contains('settings')) {
        db.createObjectStore('settings', { keyPath: 'key' });
      }
    };
  });
};
==================================================
```