1 Server Initialization and Routing Setup

This section introduces the core setup for a web server using Express.js, integrating various routes, middleware for authentication and authorization, and the initialization of a database connection. It lays the groundwork for how the server is structured, including handling CORS, parsing request bodies, and gracefully shutting down the server.

```
const express = require("express");

const { db, init } = require("./db");

const cors = require("cors");

const itemRoute = require("./routes/items");

const userRoute = require("./routes/user");

const customersRoute = require("./routes/customers");

const vendorsRoute = require("./routes/vendors");

// Auth

const authRoutes = require("./routes/auth").router;

const { verifyToken, verifyOrgAccess } = require("./routes/auth");

const app = express();

const port = 3000;

// Middleware

app.use(cors());

app.use(express.json());

app.use(express.urlencoded({ extended: true }));

init(); // Initialize the database

// Use authRoutes for authentication

app.use("/user", userRoute);

app.use("/auth", authRoutes);

// To verify every request except for login and signup

app.use(verifyToken);

app.use(verifyOrgAccess);

app.use("/", itemRoute, customersRoute, vendorsRoute);

// Listen on

app.listen(port, () => {

  console.log(`App listening at http://localhost:${port}`);

});

// Close the database connection when the app is closed

process.on("SIGINT", () => {
```

```
  db.close((err) => {
    if (err) {
      console.error(err.message);
    }
    console.log("Closed the database connection.");
    process.exit(0);
  });
});
```

2 Authentication Mechanisms and Token Management:

This section delves into the authentication flow, including user login, token generation, and validation of tokens for secure access. It details the process of verifying user credentials, issuing JWT tokens, and middleware functions to ensure that tokens are valid and users have appropriate access to resources.

```
const { router, bcrypt, jwt, getDatabaseInstance } = require("./requires");

const SECRET_KEY = "regtrgergregwe4rgrwegrwegrweg";

router.post("/login", (req, res) => {
  const db = getDatabaseInstance("main", "");
  const { email, password } = req.body;


  const query = "SELECT * FROM users WHERE email = ?";


  db.get(query, [email], (err, user) => {
    if (err) {
      return res.status(500).send("Error on the server.");
    }
    if (!user) {
      return res.status(404).send("User not found.");
    }
    const passwordIsValid = bcrypt.compareSync(password, user.password);
    if (!passwordIsValid) {
      return res.status(401).send({ auth: false, token: null });
    }
    const token = jwt.sign({ id: user.id }, SECRET_KEY, {
      expiresIn: 86400,
    });
    res.status(200).send({ auth: true, token: token });
```

```
    db.close();
  });
});
router.get("/isValidToken", (req, res) => {
  const token = req.headers["x-access-token"];
  if (!token) {
    return res.status(403).send({ auth: false, message: "No token provided." });
  }
  jwt.verify(token, SECRET_KEY, (err, decoded) => {
    if (err) {
      return res.status(440).send({ auth: false, message: "Session Expired" });
    }
  });
  res.status(200).send({ message: "Token is Valid" });
});
// Verify Token and Check Org Access Middleware
function verifyToken(req, res, next) {
  const token = req.headers["x-access-token"];
  if (!token) {
    return res.status(403).send({ auth: false, message: "No token provided." });
  }
  jwt.verify(token, SECRET_KEY, (err, decoded) => {
    if (err) {
      return res.status(440).send({ auth: false, message: "Session Expired" });
    }
    req.userId = decoded.id;
    next();
  });
}
// Verify Check Org Access Middleware
function verifyOrgAccess(req, res, next) {
  const db = getDatabaseInstance("main", "");
  const userId = req.userId;
  const query = "SELECT * FROM organization_users WHERE user_id = ?";
```

```
  db.get(query, [userId], (err, row) => {
    if (err) {
      return res.status(500).send({ message: "Error checking organization access." });
    }
    if (!row) {
      return res.status(403).send({ message: "User does not have access to any organization." });
    }
    req.body.orgId = row.organization_id;
    next();
    db.close();
  });
}
module.exports = { router, verifyToken, verifyOrgAccess };
```

3 User and Organization Management API:

Focused on the creation and management of users and organizations, this part introduces APIs for registering users, hashing passwords, and creating isolated databases for organizations. It exemplifies transactional operations and the dynamic creation of organization-specific data structures.

```
const { router, path, fs, sqlite3, getDatabaseInstance, bcrypt } = require("../requires");
async function hashPassword(plainTextPassword) {
  const saltRounds = 10;
  try {
    const hash = await bcrypt.hash(plainTextPassword, saltRounds);
    return hash;
  } catch (error) {
    console.error("Error hashing password:", error);
  }
}
const createOrganizationDatabase = (organizationId) => {
  const orgDbPath = path.resolve(__dirname, `../../db/org/${organizationId}.sqlite`);
  const exists = fs.existsSync(orgDbPath);
  if (!exists) {
    const orgDb = new sqlite3.Database(orgDbPath, sqlite3.OPEN_READWRITE | sqlite3.OPEN_CREATE,
(err) => {
      if (err) {
        console.error(`Error opening database for organization ID: ${organizationId}`, err.message);
```

```
  return;
}
// Define SQL for creating necessary tables
const tableCreationSqlStatements = [
  `CREATE TABLE IF NOT EXISTS items (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    description TEXT,
    price REAL,
    sku TEXT,
    stock INTEGER,
    last_modified DATETIME DEFAULT CURRENT_TIMESTAMP
  );`,
  `CREATE TABLE IF NOT EXISTS vendors (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    contact_name TEXT,
    contact_email TEXT,
    address TEXT,
    phone TEXT,
    previous_orders INTEGER,
    notes TEXT
  );`,
  `CREATE TABLE IF NOT EXISTS customers (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    contact_name TEXT,
    contact_email TEXT,
    address TEXT,
    phone TEXT,
    previous_orders INTEGER,
    notes TEXT
  );`,
  // Add more tables as needed
```

```
      ];
      // Execute each SQL statement to create the tables
      tableCreationSqlStatements.forEach((sql) => {
        orgDb.run(sql, [], (err) => {
          if (err) {
            console.error("Error creating table", err.message);
          }
        });
      });
      // Closing the database
      orgDb.close();
    });
  }
};
// Route to create user and organization
router.post("/createUserAndOrganization", async (req, res) => {
  const { userName, userEmail, userPassword, organizationName } = req.body;
  let userId;
  let organizationId;
  const hashedUserPassword = await hashPassword(userPassword);
  db = getDatabaseInstance("", "main");

  // Start a serialized transaction
  db.serialize(() => {
    db.run(`BEGIN TRANSACTION;`);
    // Insert user
    db.run(`INSERT INTO users (name, email, password) VALUES (?, ?, ?)`, [userName, userEmail,
hashedUserPassword], function (err) {
      if (err) {
        console.error(err.message);
        db.run(`ROLLBACK;`);
        return res.status(500).send("Error creating user");
      }
      userId = this.lastID;
      // Insert organization
```

```
    db.run(`INSERT INTO organizations (name) VALUES (?)`, [organizationName], function (err) {
      if (err) {
        console.error(err.message);
        db.run(`ROLLBACK;`);
        return res.status(500).send("Error creating organization");
      }
      organizationId = this.lastID;
      createOrganizationDatabase(organizationId); // Create org-specific DB
      // Link user to organization
      db.run(`INSERT INTO organization_users (organization_id, user_id) VALUES (?, ?)`, [organizationId,
userId], function (err) {
        if (err) {
          console.error(err.message);
          db.run(`ROLLBACK;`);
          return res.status(500).send("Error linking user to organization");
        }
        db.run(`COMMIT;`);
        res.send("User and organization created successfully");
        db.close();
      });
    });
  });
});
module.exports = router;
```

4 Management API This will similar for all the modules

Describes the implementation of CRUD (Create, Read, Update, Delete) operations for item management within an organization. It outlines how to interact with a database to manage items, including listing, adding, editing, and deleting items, along with handling organizational contexts.

```
const { router, getDatabaseInstance } = require("../requires");
router.get("/items", (req, res) => {
  const { orgId } = req.body;
  const db = getDatabaseInstance(orgId);
  const query = "SELECT * FROM items";
  db.all(query, [], (err, rows) => {
```

```
    if (err) {
      res.status(500).json({ error: err.message });
      return;
    }
    res.json({
      items: rows,
    });
  });
  db.close();
});
router.get("/items/details/:itemId", (req, res) => {
  const { itemId } = req.params;
  const { orgId } = req.body;
  const db = getDatabaseInstance(orgId);
  const query = "SELECT * FROM items WHERE id = ?";
  db.get(query, [itemId], (err, row) => {
    if (err) {
      res.status(500).json({ error: err.message });
      return;
    }
    if (row) {
      res.json(row);
    } else {
      res.status(404).json({ message: "Item not found" });
    }
  });
  db.close();
});
router.post("/items/create", (req, res) => {
  const { orgId, name, description, price, sku, stock, last_modified } = req.body;
  const db = getDatabaseInstance(orgId);
  const query = "INSERT INTO items (name, description, price, sku, stock, last_modified) VALUES (?, ?, ?, ?,
?, ?)";
  db.run(query, [name, description, price, sku, stock, last_modified], function (err) {
    if (err) {
```

```
      res.status(500).json({ error: err.message });
      return;
    }
    res.json({
      message: "Item created successfully",
      id: this.lastID,
    });
  });
  db.close();
});
router.put("/items/edit/:itemId", (req, res) => {
  const { itemId } = req.params;
  const { orgId, name, description, price, sku, stock, last_modified } = req.body;
  const db = getDatabaseInstance(orgId);
  const query = `UPDATE items SET name = ?, description = ?, price = ?, sku = ?, stock = ?, last_modified = ?
WHERE id = ?`;
  db.run(query, [name, description, price, sku, stock, last_modified, itemId], function (err) {
    if (err) {
      res.status(500).json({ error: err.message });
      return;
    }
    if (this.changes > 0) {
      res.json({ message: "Item updated successfully" });
    } else {
      res.status(404).json({ message: "Item not found" });
    }
  });
  db.close();
});
router.delete("/items/delete/:itemId", (req, res) => {
  const { itemId } = req.params;
  const { orgId } = req.body;
  const db = getDatabaseInstance(orgId);
  const query = "DELETE FROM items WHERE id = ?";
  db.run(query, [itemId], function (err) {
```

```
    if (err) {
      res.status(500).json({ error: err.message });
      return;
    }
    if (this.changes > 0) {
      res.json({ message: "Item deleted successfully" });
    } else {
      res.status(404).json({ message: "Item not found" });
    }
  });
  db.close();
});
module.exports = router;
```

5 Main template DataTable and Layout components

React component for displaying and interacting with tabular data, including features like sorting, searching, and actions (edit, delete). It demonstrates how to dynamically handle user input and data manipulation, integrating Material UI components for a rich interactive experience.

```
import { useState, useEffect, forwardRef, useImperativeHandle } from "react";
import {
  Table,
  TableBody,
  TableCell,
  TableContainer,
  Menu,
  MenuItem,
  TableHead,
  TableRow,
  Paper,
  IconButton,
  Typography,
  TextField,
} from "@mui/material";
import MoreVertIcon from "@mui/icons-material/MoreVert";
import PropTypes from "prop-types";
import ArrowUpwardIcon from "@mui/icons-material/ArrowUpward";
```

```
import ArrowDownwardIcon from "@mui/icons-material/ArrowDownward";
import DeleteIcon from "@mui/icons-material/Delete";
import EditIcon from "@mui/icons-material/Edit";
import LoadingComponent from "./LoadingComponent";
const DataTable = forwardRef(({ columns, data, onActionClick, loading, loadingMessage, searchParam }, ref)
=> {
  const [sortConfig, setSortConfig] = useState({ key: null, direction: "ascending" });
  const [searchText, setSearchText] = useState("");
  const [filteredItems, setFilteredItems] = useState(data);
  const [anchorEl, setAnchorEl] = useState(null);
  const [currentItemId, setCurrentItemId] = useState(null);
  useImperativeHandle(ref, () => ({
    closeFunction() {
      handleClose();
    },
  }));
  useEffect(() => {
    const filteredData = data.filter((item) =>
item[searchParam].toString().toLowerCase().includes(searchText.toLowerCase()));
    setFilteredItems(filteredData);
  }, [searchText, data, searchParam]);
  const handleClick = (event, id) => {
    setAnchorEl(event.currentTarget);
    setCurrentItemId(id);
  };
  const handleClose = () => {
    setAnchorEl(null);
    setCurrentItemId(null);
  };

  const sortData = (key) => {
    if (sortConfig.key === key && sortConfig.direction === "descending") {
      setSortConfig({ key, direction: "ascending" });
    } else if (sortConfig.key === key && sortConfig.direction === "ascending") {
      setSortConfig({ key: null, direction: null });
```

```
  } else {
    setSortConfig({ key, direction: "descending" });
  }
};
useEffect(() => {
  if (sortConfig.key !== null && sortConfig.direction !== null) {
    const sortedItems = [...filteredItems].sort((a, b) => {
      if (a[sortConfig.key] < b[sortConfig.key]) {
        return sortConfig.direction === "ascending" ? -1 : 1;
      }
      if (a[sortConfig.key] > b[sortConfig.key]) {
        return sortConfig.direction === "ascending" ? 1 : -1;
      }
      return 0;
    });
    setFilteredItems(sortedItems);
  } else {
    setFilteredItems(filteredItems);
  }
}, [sortConfig, filteredItems]);
return (
  <>
    {loading === true && <LoadingComponent message={loadingMessage} />}
    {loading === false && (
      <>
        <div style={{ width: "40%" }}>
          <TextField
            label="Search"
            variant="outlined"
            fullWidth
            margin="normal"
            value={searchText}
            onChange={(e) => setSearchText(e.target.value)}
          />
```

```
</div>
<TableContainer component={Paper}>
  <Table aria-label="data table">
    <TableHead>
      <TableRow>
        {columns.map((column) => (
          <TableCell
            key={column.id}
            onClick={() => column.sortable && sortData(column.id)}
            style={{ cursor: column.sortable ? "pointer" : "default" }}
          >
            <div
              style={{
                display: "flex",
                alignItems: "center",
                gap: "4px",
              }}
            >
              <b>{column.label}</b>
              {column.sortable &&
                sortConfig.key === column.id &&
                (sortConfig.direction === "ascending" ? (
                  <ArrowUpwardIcon sx={{ fontSize: "small" }} />
                ) : sortConfig.direction === "descending" ? (
                  <ArrowDownwardIcon sx={{ fontSize: "small" }} />
                ) : null)}
            </div>
          </TableCell>
        ))}
        <TableCell>
          <b>Actions</b>
        </TableCell>
      </TableRow>
    </TableHead>
```

```
            <TableBody>
              {filteredItems.map((item, index) => (
                <TableRow key={index}>
                  {columns.map((column) => (
                    <TableCell key={column.id}>{item[column.id]}</TableCell>
                  ))}
                  <TableCell>
                    <IconButton onClick={(event) => handleClick(event, item.id)}>
                      <MoreVertIcon />
                    </IconButton>
                    <Menu id="action-menu" anchorEl={anchorEl} keepMounted open={Boolean(anchorEl)} onClose={handleClose}>
                      <MenuItem onClick={() => onActionClick("edit", currentItemId)}>
                        <EditIcon sx={{ fontSize: "medium" }} />
                      </MenuItem>
                      <MenuItem onClick={() => onActionClick("delete", currentItemId)}>
                        <DeleteIcon sx={{ fontSize: "medium" }} />
                      </MenuItem>
                    </Menu>
                  </TableCell>
                </TableRow>
              ))}
            </TableBody>
          </Table>
        </TableContainer>
      </>
    )}
    {loading === false && filteredItems.length === 0 && (
      <Typography
        sx={{
          display: "flex",
          justifyContent: "center",
          alignItems: "center",
          fontSize: "2rem",
          color: "primary.main",
```

```
            textTransform: "uppercase",
            padding: "1em 0",
            letterSpacing: "0.1em",
          }}
        >
          Nothing Found
        </Typography>
      )}
    </>
  );
});
DataTable.displayName = "DataTable";
DataTable.propTypes = {
  columns: PropTypes.array.isRequired,
  data: PropTypes.array.isRequired,
  onActionClick: PropTypes.func.isRequired,
  loading: PropTypes.bool,
  loadingMessage: PropTypes.string,
  searchParam: PropTypes.string.isRequired,
};
export default DataTable;
```