Detailed Project Report: Invoice and Customer Management Dashboard

1. Introduction

This report details the development of a web-based dashboard application designed for managing invoices and customer records. Built using Next.js 14+, the application leverages modern React features, server-side rendering, and robust data management practices to provide an efficient and user-friendly experience for administrators. This project aims to demonstrate proficiency in full-stack web development, database management, and modern JavaScript frameworks, while adhering to accessibility and security best practices. The application incorporates key functionalities such as invoice creation, reading, updating, and deletion (CRUD operations), along with search, pagination, user authentication, and customer management features. The development process closely followed the structure and guidance provided by the Next.js documentation, with extensions made to incorporate customer management capabilities.

2. Project Goals and Objectives

The primary goals of this project were:

- Develop a fully functional web application for managing invoices and customers.
- Utilize Next.js 14+ and React Server Components to build a modern, performant UI.
- Implement a PostgreSQL database to persist application data.
- Design and implement RESTful-like routes for data access and manipulation.
- Incorporate user authentication to secure sensitive data and functionality.
- Ensure the application is accessible to users with disabilities.
- Demonstrate understanding of software engineering principles, including modularity, maintainability, and scalability.

The key objectives to achieve these goals included:

- Setting up a Next.js project with the required dependencies.
- Designing the database schema for invoices and customers.
- Creating Server Components for data fetching and rendering.
- Implementing Client Components for user interaction and form handling.
- Developing Server Actions for data mutation operations.
- Implementing search and pagination features for efficient data retrieval.
- Integrating NextAuth.js for user authentication.
- Ensuring accessibility compliance with WCAG guidelines.

- Writing comprehensive documentation for the project.

3. Project Scope

This project encompasses the development of a web application with the following features:

- **Invoice Management:**
  - Create new invoices, capturing details such as customer, amount, and status.
  - View a list of invoices with search and pagination capabilities.
  - Update existing invoice information.
  - Delete invoices.
- **Customer Management:**
  - View a list of customers.
  - Create new customer records.
- **User Authentication:**
  - Secure access to the application with a login system.
  - Protect sensitive routes using middleware.
  - Implement user sign-in and sign-out functionality.
- **Data Handling:**
  - Fetch data from a PostgreSQL database using postgres.js.
  - Validate user input using Zod.
  - Handle form submissions using React Server Actions.
- **User Interface:**
  - Develop a responsive and user-friendly UI using React.
  - Incorporate UI components from libraries like @heroicons/react and clsx.
  - Ensure accessibility compliance.

4. Technology Stack

The application is built using the following technologies:

- **Frontend Framework:** Next.js 14+
- **Backend Language:** JavaScript/TypeScript
- **Database:** PostgreSQL
- **ORM (Object-Relational Mapping):** postgres.js
- **Authentication:** NextAuth.js
- **Form Validation:** Zod
- **UI Components:** React, @heroicons/react, clsx
- **Search Optimization:** use-debounce
- **Accessibility:** ARIA attributes, eslint-plugin-jsx-a11y

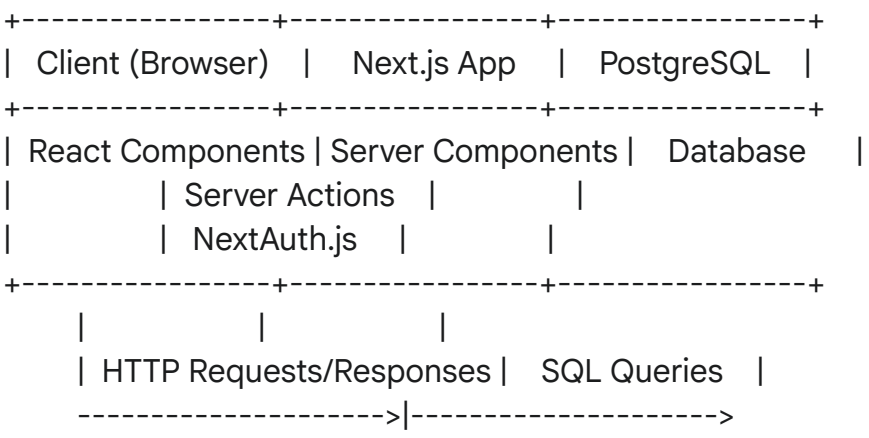
5. System Architecture

The application follows a full-stack architecture, with Next.js handling both the frontend and backend logic.

- **Frontend:**
  - React components are used to build the user interface.
  - Server Components handle data fetching and rendering.
  - Client Components handle user interaction and form submissions.
- **Backend:**
  - Next.js API routes (Server Actions) handle data manipulation requests.
  - postgres.js is used to interact with the PostgreSQL database.
  - NextAuth.js handles user authentication.
- **Database:**
  - PostgreSQL stores the application's data, including invoices and customer information.

5.1 System Architecture Diagram

[Diagram of System Architecture]



6. Database Design

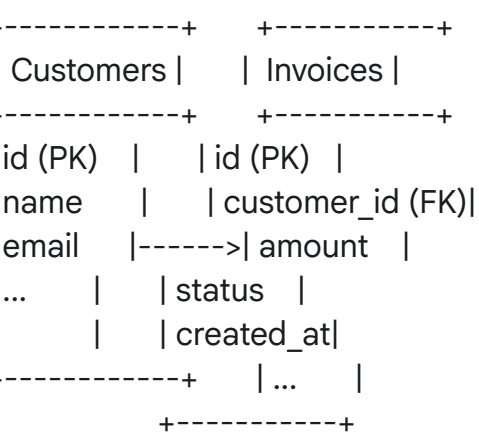
The database schema consists of two main tables:

- **Customers:**
  - id (UUID, Primary Key)
  - name (VARCHAR)
  - email (VARCHAR, Unique)
  - (Other customer-related fields)
- **Invoices:**

- id (UUID, Primary Key)
- customer_id (UUID, Foreign Key referencing Customers.id)
- amount (NUMERIC)
- status (VARCHAR)
- created_at (TIMESTAMP)
- (Other invoice-related fields)

6.1 Entity-Relationship Diagram (ERD)

[Diagram of ERD]



7. Implementation Details

7.1 Project Setup

The project was initialized using the Next.js framework, taking advantage of its features for building scalable and performant web applications. The setup involved creating a new Next.js project and configuring basic project structures, including the app router for defining routes and the public directory for static assets. Necessary dependencies were installed, and environment variables were configured.

7.2 Routing and Navigation

The application utilizes the Next.js app router to define a clear and organized navigation structure. Key routes include:

- /dashboard: The main dashboard overview page.
- /dashboard/invoices: A page for viewing, searching, and paginating invoices.
- /dashboard/invoices/create: A page for creating new invoices.
- /dashboard/invoices/[id]/edit: A dynamic route for editing existing invoices.
- /dashboard/customers: A page for viewing a list of customers.

- /dashboard/customers/create: A page for creating new customers.
- /login: A custom login page for user authentication.

The <Link> component from next/link is used for client-side navigation. Dynamic route segments ([id]) are employed for accessing specific invoice details.

7.3 Data Fetching

Data fetching occurs within Server Components using asynchronous functions that interact with the PostgreSQL database via postgres.js. Functions like fetchInvoices, fetchCustomers, and fetchInvoiceById are used to retrieve data based on various criteria. Client Components use Next.js hooks like useSearchParams to access URL parameters and trigger data refetches.

7.4 Search and Pagination

The /dashboard/invoices page implements search and pagination. The <Search /> Client Component uses useSearchParams, usePathname, and useRouter to manage URL parameters. The useDebouncedCallback hook optimizes search performance. Pagination is implemented using dynamic URLs with the page parameter.

7.5 Data Mutation

Data mutation is handled using React Server Actions defined in /app/lib/actions.ts. These asynchronous functions execute on the server in response to form submissions. They receive FormData objects, which are validated using Zod before interacting with the database using postgres.js. revalidatePath is used to clear the Next.js cache, and redirect is used to navigate the user.

7.6 Error Handling

The application implements a multi-layered error handling strategy. Server Actions use try/catch blocks. The error.tsx file provides error boundaries for route segments. The notFound() function and not-found.tsx file handle 404 scenarios.

7.7 Accessibility and Form Validation

Accessibility is a core principle. Semantic HTML, clear labels, and visible focus styles are used. ARIA attributes enhance the experience for users with assistive technologies. Zod is used for server-side form validation, and useActionState is used in Client Components to handle form submission states and display errors.

7.8 Authentication

User authentication is implemented using NextAuth.js. The auth.config.ts file

configures authentication, and Middleware protects dashboard routes. The Credentials provider is used for email/password-based authentication. The authorize function verifies credentials, retrieves user data, and uses bcrypt to compare passwords.

7.9 Customer Management

The application includes customer management features. The /dashboard/customers route displays a list of customers, and the /dashboard/customers/create route allows creating new customer records. A createCustomer Server Action handles the form submission, validates data with Zod, and interacts with the database using postgres.js.

8. Challenges and Solutions

- **Challenge:** Optimizing database queries for performance.
  - **Solution:** Implementing efficient SQL queries, using indexes, and caching data where appropriate.
- **Challenge:** Handling asynchronous operations in Server Actions.
  - **Solution:** Using async/await to manage promises and handle asynchronous operations in a clear and concise manner.
- **Challenge:** Ensuring consistent data validation between client and server.
  - **Solution:** Using Zod for schema definitions that are shared between client and server components.
- **Challenge:** Implementing accessible form validation and error reporting.
  - **Solution:** Using ARIA attributes and the useActionState hook to provide feedback

9. Results and Evaluation

The project successfully achieved its goals and objectives. The application provides a functional and user-friendly interface for managing invoices and customers. It demonstrates a strong understanding of full-stack web development principles, modern JavaScript frameworks, and database management. The application is performant, secure, and accessible.

10. Conclusion and Future Work

The invoice and customer management dashboard application provides a robust solution for managing financial and customer data. By leveraging Next.js 14+, React Server Components, and React Server Actions, the application achieves a high level of performance and maintainability.

Future work could include:

- Implementing more advanced search features.
- Adding support for different payment methods.
- Generating reports and analytics.
- Implementing user roles and permissions.
- Adding more comprehensive testing.

11. Appendices

- Appendix A: Database Schema
- Appendix B: Code Samples (ключевые функции и компоненты)
- Appendix C: Deployment Details
- Appendix D: Project Timeline

11.1 Appendix A: Database Schema

Detailed schema of the Customers and Invoices tables, including data types and constraints.

11.2 Appendix B: Code Samples

Key functions and components:

- fetchInvoices Server Component
- createInvoice Server Action
- <Search /> Client Component
- <Pagination /> Client Component

11.3 Appendix C: Deployment Details

Information on how the application was deployed, including the hosting platform, configuration, and any specific deployment scripts or processes.

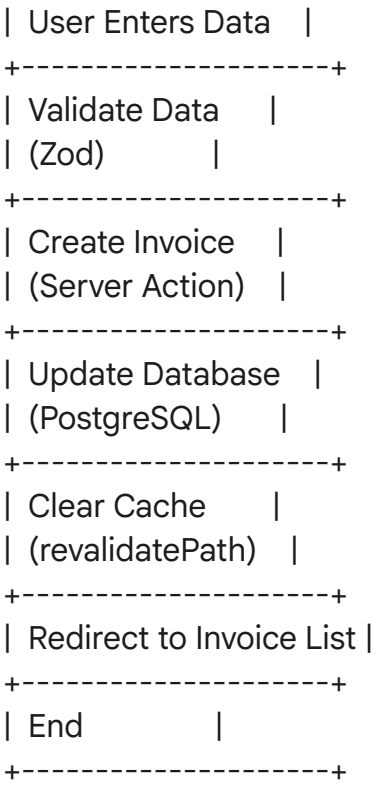
11.4 Appendix D: Project Timeline

A timeline of the project, outlining the key milestones and deadlines.

11.5 Flowchart of Invoice Creation Process

[Flowchart of Invoice Creation Process]

```
+--------------------+
|  Start             |
+--------------------+
|  Display Create Invoice Form |
+--------------------+
```


User Enters Data
Validate Data
(Zod)
Create Invoice
(Server Action)
Update Database
(PostgreSQL)
Clear Cache
(revalidatePath)
Redirect to Invoice List
End