Books Store

Project report (CA3) submitted in fulfilment of the requirements for the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

Veera Prabhu Kumar Reddy Dontireddy

(12214170)

SUBJECT

INT252 – Web App Development with ReactJS

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

TABLE OF CONTENTS

1. **INTRODUCTION**
   - ✓ PURPOSE
   - ✓ SCOPE
   - ✓ DEFINITIONS, ACRONYMS AND ABBREVIATIONS
   - ✓ REFERENCES
   - ✓ OVERVIEW
2. **GENERAL DESCRIPTION**
   - ✓ PRODUCT PERSPECTIVE
   - ✓ PRODUCT FUNCTIONS
   - ✓ USER CHARACTERISTICS
   - ✓ GENERAL CONSTRAINTS
   - ✓ ASSUMPTIONS AND DEPENDENCIES
3. **SPECIFIC REQUIREMENTS**
   - ✓ EXTERNAL INTERFACE REQUIREMENTS
   - ✓ FUNCTIONAL REQUIREMENTS
   - ✓ NON-FUNCTIONAL REQUIREMENTS
   - ✓ DESIGN CONSTRAINTS
   - ✓ OTHER REQUIREMENTS
4. **ANALYSIS MODELS**
5. **CLIENT APPROVAL AND TRANSACTION ID PROOF**
6. **SNAPSHOTS**
7. **CONCLUSION**

1.INTRODUCTION

✓ <u>Purpose</u>

The purpose of this document is to outline the requirements, architecture, and interaction flows for a web application using React.js for the frontend, MongoDB for data storage, and Postman for API testing and validation. This document aims to provide clear specifications to guide the development and maintenance of this application.

✓ <u>Scope</u>

- React.js frontend: UI component structure, state management, and data handling.
- MongoDB database: Data modeling, schema definitions, and database operations.
- API testing using Postman: A structured approach to testing the API for various operations.

✓ <u>Definitions, Acronyms, and Abbreviations</u>

- MongoDB: A NoSQL, document-oriented database used for scalable and flexible data storage.
- Postman: A tool that allows developers to create, test, and debug API requests.
- React.js: A JavaScript library for building user interfaces, focusing on component-based architecture and reactivity.
- CRUD: Create, Read, Update, and Delete operations.
- HTTP Methods: GET, POST, PUT, and DELETE requests used to interact with the API.

✓ **References**

- React documentation
- MongoDB documentation
- Postman documentation

✓ **Overview**

This document provides detailed descriptions of the functional and non-functional requirements for an application that uses a React frontend to interact with a MongoDB database. The API, accessed by the frontend and tested via Postman, allows CRUD operations on MongoDB-stored data, focusing on real-time UI updates and error-handling practices.

2. GENERAL DESCRIPTION

✓ **Product Perspective**

The product is a full-stack web application that includes a React.js frontend, a MongoDB backend, and a RESTful API that enables communication between the two. Postman is used to verify API functionality, ensuring that the frontend can correctly retrieve, display, and update data in MongoDB.

✓ **Product Functions**

UI Rendering and Interaction (React.js):

- React components that create dynamic and interactive user experiences.
- State management to track and render user inputs and API responses.

Database Operations (MongoDB):

- MongoDB collections store application data.

- API endpoints allow data manipulation through CRUD operations.

API Testing (Postman):

- Using Postman to verify API responses for accuracy and performance.
- Testing various scenarios, including error responses and boundary conditions.

✓ **User Characteristics**

- End Users: Comfortable with basic web browsing and form interactions.
- Developers: Knowledge of REST APIs, MongoDB, and testing methodologies.

✓ **General Constraints**

- Browser Compatibility: Supports major browsers (Chrome, Firefox, Safari).
- Database Requirements: MongoDB Atlas or a locally hosted MongoDB instance.
- Testing Requirements: Postman installed to test HTTP requests and responses.

✓ **Assumptions and Dependencies**

- The application is accessible via desktop and mobile browsers.
- MongoDB and Express.js (if used) must be properly configured to handle API requests.
- The backend server should be running to ensure API connectivity with MongoDB.

3.SPECIFIC REQUIREMENTS

✓ <u>External Interface Requirements</u>

User Interfaces

- Form Components: Accept user inputs, validated and stored in MongoDB via the API.
- Table/List Components: Display fetched data dynamically, such as user entries or data logs.
- Feedback Mechanisms: Loading indicators, error messages, and modals to inform users of application status.

Hardware Interfaces

- N/A (web-based application)

Software Interfaces

- MongoDB: Provides data storage and retrieval operations.
- Express API: Acts as an intermediary for CRUD operations on MongoDB.
- Postman: Testing tool to confirm endpoint functionality.

Communications Interfaces

- HTTP/HTTPS: API communication between React frontend and MongoDB via REST endpoints.

✓ <u>Functional Requirements</u>

Frontend Component Rendering

- Render based on the application's current state, ensuring that changes in data trigger UI updates.
- Separate logic into reusable, modular components (e.g., Form, Table, List, Error Boundary).

State Management

- Use State: Manage individual component states, like form inputs.

- Use Effect: Fetch data on initial load and synchronize with the backend.
- Context API or Redux: Used if global state management is needed, e.g., to share user session data.

API Integration for CRUD Operations

- Create: Accepts JSON data via POST requests to create a new entry in MongoDB.
- Read: Uses GET requests to retrieve data from MongoDB and render it in the UI.
- Update: Uses PUT requests to modify existing MongoDB documents.
- Delete: Allows deletion of specific records via DELETE requests.

MongoDB Schema Design

- Data Model: Define schema with Mongoose to structure data (e.g., user data, product details).
- Document Validation: MongoDB schema should validate data before saving it to ensure consistency.

Postman Testing Workflow

- Request Setup: Create test collections with HTTP requests for all CRUD operations.
- Environment Variables: Use variables for dynamic testing across different environments.
- Automated Tests: Write test scripts to verify API response status codes and data structure.

✓ Non-Functional Requirements

Performance

- Optimize component loading by using lazy loading where possible.
- Minimize MongoDB query times and ensure indexes for frequent queries.

Reliability

- MongoDB must be available and consistent.

- Error boundaries in React handle frontend errors without affecting the entire app.

Availability

- The application should operate continuously, ensuring MongoDB and backend uptime.

Security

- Input validation in React to avoid sending malicious data.
- Enforce role-based access control if there are different user permissions.
- Use HTTPS for secure communication.

Maintainability

- Modular structure and comments for React components.
- Consistent code organization for MongoDB schema and API endpoints.

Portability

- The React and MongoDB setup should be compatible across major OS platforms.

✓ <u>**Design Constraints**</u>

- The MongoDB schema must follow the defined structure, with fields validated to ensure compatibility with the frontend.
- API response structures must be consistent for reliable UI handling.

✓ <u>**Other Requirements**</u>

- Comprehensive error handling in MongoDB operations.
- Clear documentation of API endpoints, including request and response examples.

4. ANALYSIS MODELS

✓ **<u>Data Flow Diagrams (DFD):</u>**

User Input and API Calls:

- The user interacts with the React interface, triggering events that update the state and make API calls.

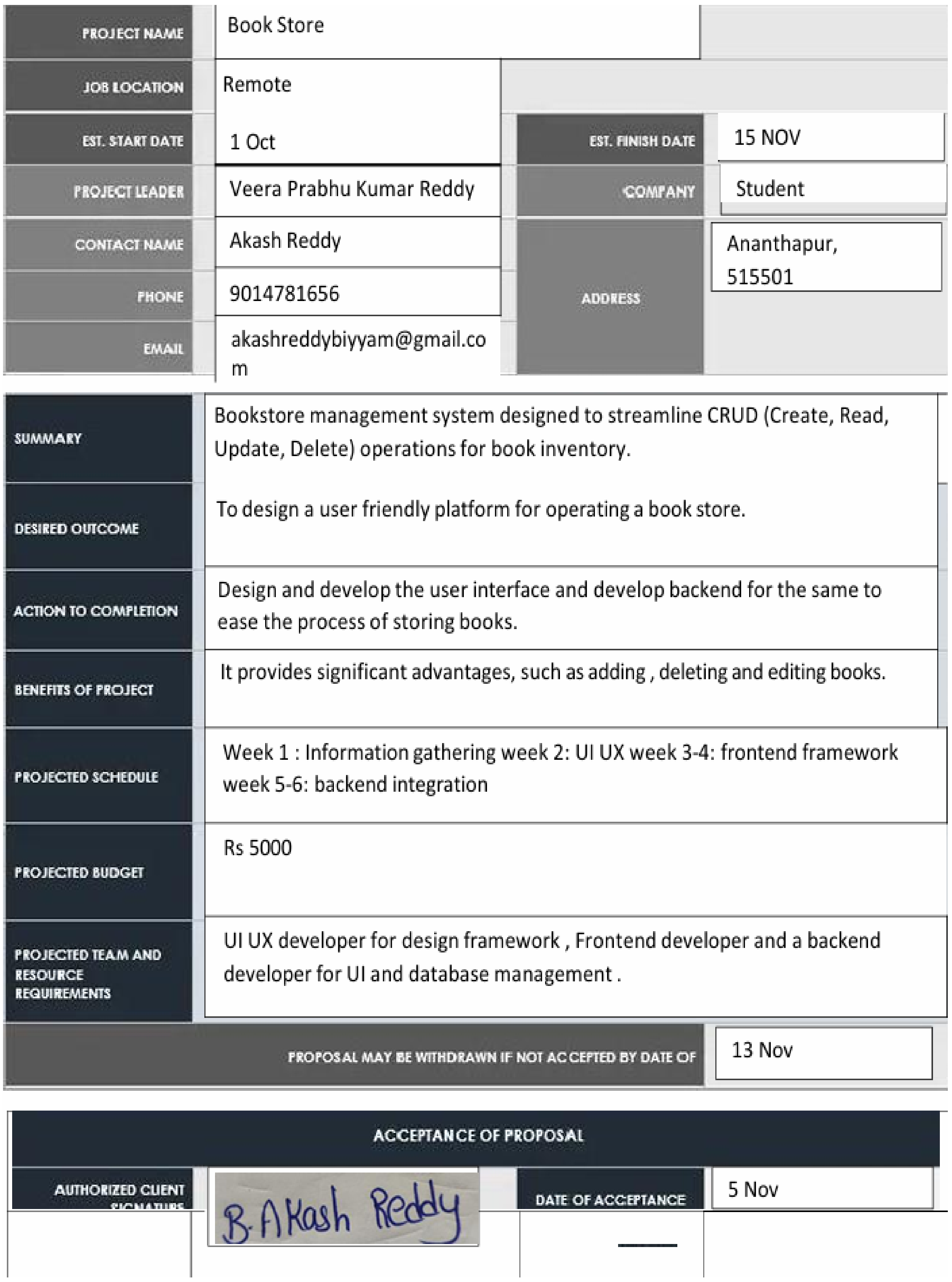
API Endpoint Requests:

- API requests are sent from the frontend to the Express API, which connects to MongoDB to store or retrieve data.

MongoDB Data Operations:

- Create: API sends data from the frontend to MongoDB.
- Read: MongoDB returns requested data to the frontend.
- Update: Updates existing data in MongoDB.
- Delete: Deletes data in MongoDB based on request.
- Postman Testing:
- Testing each API endpoint with sample data, ensuring CRUD functionality.
- Validating that responses match expected formats and handling of errors or invalid inputs.
- Frontend State Update:
- React updates the UI based on API responses, reflecting new data or error messages as necessary.

CLIENT APPROVAL FORM



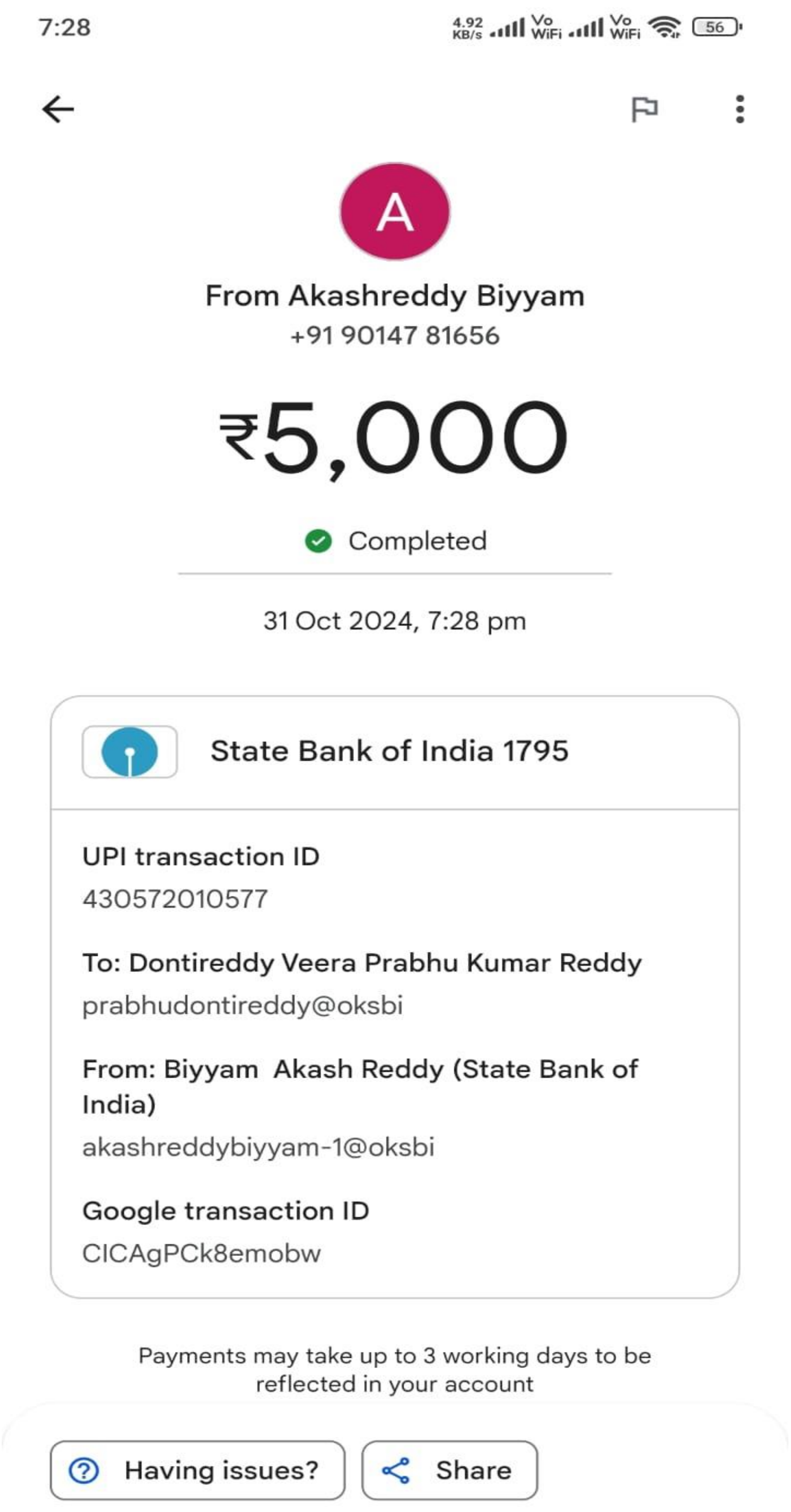
PROJECT NAME	Book Store		
JOB LOCATION	Remote		
EST. START DATE	1 Oct	EST. FINISH DATE	15 NOV
PROJECT LEADER	Veera Prabhu Kumar Reddy	COMPANY	Student
CONTACT NAME	Akash Reddy	ADDRESS	Ananthapur, 515501
PHONE	9014781656		
EMAIL	akashreddybiyyam@gmail.com		

SUMMARY	Bookstore management system designed to streamline CRUD (Create, Read, Update, Delete) operations for book inventory.
DESIRED OUTCOME	To design a user friendly platform for operating a book store.
ACTION TO COMPLETION	Design and develop the user interface and develop backend for the same to ease the process of storing books.
BENEFITS OF PROJECT	It provides significant advantages, such as adding , deleting and editing books.
PROJECTED SCHEDULE	Week 1 : Information gathering week 2: UI UX week 3-4: frontend framework week 5-6: backend integration
PROJECTED BUDGET	Rs 5000
PROJECTED TEAM AND RESOURCE REQUIREMENTS	UI UX developer for design framework , Frontend developer and a backend developer for UI and database management .
PROPOSAL MAY BE WITHDRAWN IF NOT ACCEPTED BY DATE OF	13 Nov

ACCEPTANCE OF PROPOSAL

AUTHORIZED CLIENT SIGNATURE	B.Akash Reddy	DATE OF ACCEPTANCE	5 Nov

TRANSACTION ID PROOF



7:28

From Akashreddy Biyyam
+91 90147 81656

₹5,000

Completed

31 Oct 2024, 7:28 pm

State Bank of India 1795

UPI transaction ID
430572010577

To: Dontireddy Veera Prabhu Kumar Reddy
prabhudontireddy@oksbi

From: Biyyam Akash Reddy (State Bank of India)
akashreddybiyyam-1@oksbi

Google transaction ID
CICAgPCk8emobw

Payments may take up to 3 working days to be reflected in your account