Visvesvaraya Technological University, Belagavi – 590018

MINI-PROJECT REPORT
ON

B-LOCK: Blockchain based eVault application

Submitted in partial fulfillment for the award of degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE & ENGINEERING

Submitted by

Abhik L Salian	**4SO21CS004**
Ashwitha Shetty	**4SO21CS030**
H Karthik P Nayak	**4SO21CS058**
Jessica Lillian Mathew	**4SO21CS066**

Under the Guidance of

Dr. Santhosh Kumar DK
Associate Professor, Department of CSE

DEPT. OF COMPUTER SCIENCE AND ENGINEERING
ST JOSEPH ENGINEERING COLLEGE
An Autonomous Institution
(Affiliated to VTU Belagavi, Recognized by AICTE, Accredited by NBA)
Vamanjoor, Mangaluru - 575028, Karnataka
2023-24

ST JOSEPH ENGINEERING COLLEGE

An Autonomous Institution

(Affiliated to VTU Belagavi, Recognized by AICTE, Accredited by NBA)

Vamanjoor, Mangaluru - 575028, Karnataka

DEPT. OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the Mini-project work entitled **"B-LOCK: Blockchain based eVault application"** carried out by

Abhik L Salian	**4SO21CS004**
Ashwitha Shetty	**4SO21CS030**
H Karthik P Nayak	**4SO21CS058**
Jessica Lillian Mathew	**4SO21CS066**

the bonafide students of VI semester Computer Science & Engineering in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belagavi during the year 2023-2024. It is certified that all suggestions indicated during internal assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

Ms. L Hema
Mini-Project Coordinator

Dr Sridevi Saralaya
HOD-CSE

Abstract

The advent of blockchain technology has revolutionized the way data is stored and managed, security, transparency, and immutability. This report presents the development of a blockchain-based eVault, a decentralized application designed to securely store, retrieve, download, and delete files and documents. The eVault leverages blockchain's distributed ledger technology to ensure the integrity and confidentiality of users' data.

The primary objective of this project is to provide a robust and user-friendly platform for secure document management, addressing the increasing demand for data privacy and protection. The eVault application allows users to upload files directly onto the blockchain, where each document is encrypted and assigned a unique identifier for easy retrieval. Users can access their documents at any time, with the assurance that the data remains untampered due to the immutable nature of blockchain records.

Key features include secure file upload, efficient retrieval using unique identifiers, and comprehensive access control mechanisms that ensure data is accessible only to authorized users. Performance metrics indicate that the system achieves optimal transaction speeds and data retrieval times, even under high loads.

Throughout the development process, challenges such as scalability and integration with existing systems were addressed, resulting in a versatile solution that meets industry demands.

This blockchain-based eVault demonstrates significant potential across various industries, offering a secure and efficient solution for managing sensitive documents. By addressing current limitations in data storage and retrieval, this project paves the way for innovative advancements in digital security and privacy.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In today's digital age, the management and security of data have become paramount across numerous industries. With the rapid increase in data generation, there is a growing need for efficient and secure systems to store, retrieve, and manage sensitive information. Traditional storage solutions often struggle with issues of security breaches, data tampering, and unauthorized access.

Blockchain technology has emerged as a revolutionary solution to these challenges, offering a decentralized and immutable ledger that enhances data integrity, transparency, and security. By leveraging blockchain, we can ensure that data is stored in a manner that is resistant to tampering and unauthorized access, making it an ideal solution for managing sensitive documents and files.

This report introduces the development of a blockchain-based eVault, a decentralized application designed to provide a secure platform for storing, retrieving, downloading, and deleting files and documents. The eVault utilizes blockchain's unique capabilities to ensure that user data is encrypted, protected, and accessible only to authorized users, addressing the critical need for data privacy and security in various sectors.

The primary goal of this project is to create a user-friendly platform that leverages blockchain technology to offer a robust solution for document management. This platform allows users to securely upload files to the blockchain, ensuring that each document is encrypted and assigned a unique identifier for easy and secure retrieval. The immutable nature of blockchain ensures that once data is stored, it cannot be altered or deleted without proper authorization, providing a high level of trust and reliability.

1.2 Problem statement

In today's digital age, securing sensitive data is a major challenge due to the limitations of traditional storage systems, which often struggle with data breaches, unauthorized access, and tampering. There is an urgent need for a reliable solution that ensures data integrity, privacy, and security. The blockchain-based eVault addresses these issues by providing a decentralized platform for secure file storage, retrieval, download, and deletion, leveraging blockchain technology to guarantee data protection and user control.

1.3 Scope

The blockchain-based eVault project aims to develop a secure and efficient platform for managing sensitive documents across various industries. This project focuses on creating a user-friendly application that allows users to store, access, and manage files with complete confidence in their security.

The eVault will feature encrypted file storage, ensuring that all documents are tamper-proof and accessible only to authorized users. It will implement unique identifiers and access control mechanisms for efficient data retrieval and user management, providing a reliable alternative to conventional storage systems.

The application is particularly suited for sectors where data privacy is critical, such as finance, healthcare, and legal services. By offering a robust solution for secure document management, the eVault project addresses the growing demand for data security and privacy, paving the way for innovative advancements in digital information management.

Chapter 2

Software Requirements Specification

2.1 Introduction

The purpose of this Software Requirements Specification (SRS) document is to outline the functional and non-functional requirements for the blockchain-based eVault project. The eVault is a decentralized application designed to provide a secure and user-friendly platform for storing, retrieving, downloading, and deleting files using blockchain technology. This document will serve as a guide for the development team and stakeholders, ensuring that all aspects of the system are well-defined and aligned with the project's goals.

This document is structured into two main sections: Functional Requirements and Non-Functional Requirements. The Functional Requirements section describes the core functionalities that the eVault must provide to meet the user's needs. The Non-Functional Requirements section details the quality attributes, performance standards, and other constraints that the system must adhere to for optimal operation.

2.2 Functional Requirements

The functional requirements define the specific behavior and functionalities of the eVault system. These requirements ensure that the application meets user expectations and provides the necessary capabilities for secure document management. The key functional requirements for the eVault include:

1. User Registration and Authentication: - The system must allow users to register and create an account using a secure registration process. - Users must be able to log in using their credentials, with multi-factor authentication for added security. - Passwords must be stored securely using encryption.

2. Secure File Upload: - Users must be able to upload files to the eVault, where each file is encrypted before storage. - The system must assign a unique identifier to each file for easy retrieval and management.

3. File Retrieval and Access Control: - Users must be able to search and retrieve files using the unique identifiers. - Access control mechanisms must be implemented to ensure that only authorized users can access specific files.

4. File Download and Deletion: - Users must be able to download files securely, with the option to decrypt them upon download. - Users should be able to delete files, ensuring that the deletion process is secure and irreversible.

5. Blockchain Integration: - The system must integrate with a blockchain platform to ensure the immutability and security of stored files. - All transactions related to file operations (upload, download, delete) must be recorded on the blockchain for auditability.

6. User Interface: - The application must provide a user-friendly interface that is intuitive and easy to navigate. - The interface should support multiple languages and accessibility features.

2.3 Non-Functional Requirements

The non-functional requirements define the quality attributes, performance standards, and other constraints of the eVault system. These requirements ensure that the application operates efficiently and meets user expectations regarding performance, security, and usability. The key non-functional requirements for the eVault include:

1. Performance: - The system must handle a high volume of transactions and file uploads/downloads without significant delays. - File retrieval times should not exceed a specified threshold, ensuring quick access to documents.

2. Scalability: - The eVault must be able to scale to accommodate a growing number of users and files. - The system should support horizontal scaling to handle increased demand.

3. Security: - All data, including user credentials and files, must be encrypted to protect against unauthorized access and breaches. - The system should comply with industry security standards and regulations (e.g., GDPR, HIPAA).

4. Reliability: - The eVault must provide high availability and uptime, minimizing downtime and disruptions. - The system should have backup and recovery mechanisms to ensure data integrity and availability.

5. Usability: - The application should be intuitive and easy to use, with clear instructions and help resources available. - User feedback mechanisms should be in place to gather input for future improvements.

6. Compatibility: - The eVault must be compatible with various operating systems and devices, including desktops, tablets, and smartphones. - The system should support major web browsers and adhere to web standards.

7. Maintainability: - The system architecture should be modular and easy to maintain, allowing for future updates and enhancements. - Code documentation and best practices should be followed to facilitate development and troubleshooting.

2.4 User Interface Requirements

This section outlines the design and functional requirements of the user interface (UI) for the eVault system. It includes considerations for usability, accessibility, and aesthetic appeal, ensuring an intuitive experience for all users.

2.4.1 Design Principles

The user interface should adhere to modern design principles to enhance usability and user satisfaction.

- **Consistency:** The UI should maintain consistency in design elements across all screens, including fonts, colors, and button styles.
- **Simplicity:** The design should be clean and clutter-free, emphasizing essential functions to avoid overwhelming the user.
- **Clarity:** The interface should be organized in a way that makes it easy for users to navigate and find features. Key actions such as file upload, download, and account management should be easily accessible.

2.4.2 Accessibility

The application should be accessible to all users, including those with disabilities, by adhering to accessibility standards.

- **Screen Readers:** The UI should support screen readers to assist visually impaired users in navigating the application.
- **Keyboard Navigation:** Users should be able to navigate the application using keyboard shortcuts without relying on a mouse.
- **Color Contrast:** Ensure sufficient contrast between text and background colors to improve readability for users with visual impairments.

2.4.3 User Feedback

Feedback mechanisms should be implemented to gather user input and improve the application over time.

- **Error Messages:** Provide clear and informative error messages to guide users in resolving issues.
- **User Surveys:** Periodic surveys can be conducted to collect user feedback on the UI and identify areas for improvement.

- **Help Resources:** Offer tutorials and help sections to assist users in understanding and using the application's features.

2.5 Software Requirements

This section specifies the technical requirements of the eVault system, covering hardware, software, network, and operational aspects to ensure efficient and reliable performance.

2.5.1 Hardware Requirements

The eVault system requires specific hardware configurations to operate efficiently and support user needs.

- **Server Specifications:** Minimum server requirements include 16 GB RAM, 4-core CPU, and 500 GB SSD for optimal performance.
- **User Devices:** Users should have access to devices with at least 4 GB RAM and a modern web browser to use the application effectively.

2.5.2 Software Dependencies

The eVault relies on various software components and libraries to function correctly.

- **Operating Systems:** Compatible with Windows, macOS, and Linux for server deployments.
- **Database Systems:** Supports MySQL and MongoDB for data storage and management.
- **Programming Languages:** Developed using Python and JavaScript, leveraging frameworks such as Django and React.

2.5.3 Network Requirements

Ensure stable and secure network connectivity to support application operations and data transmission.

- **Bandwidth:** The application requires a minimum internet speed of 10 Mbps for smooth operation.
- **Firewall Settings:** Configure firewalls to allow traffic through necessary ports while blocking unauthorized access.
- **Data Encryption:** All data transmitted over the network must be encrypted using TLS to prevent interception.

Chapter 3

System Design

paragraph contents...

3.1 Architecture Design

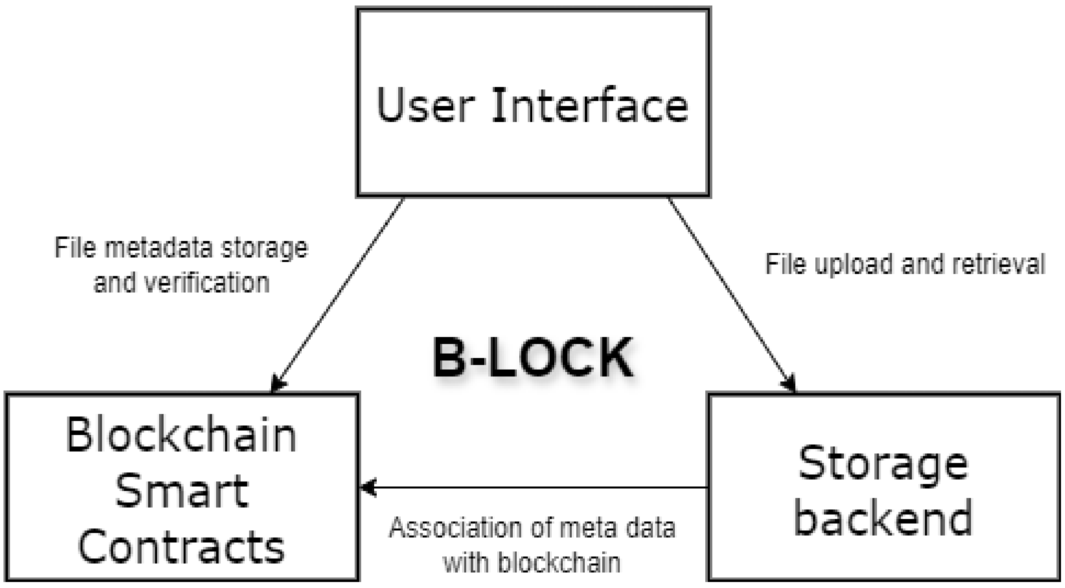
User Interface
File metadata storage and verification
File upload and retrieval
B-LOCK
Blockchain Smart Contracts
Storage backend
Association of meta data with blockchain

Figure 3.1: System Architecture Diagram

Figure 3.1 illustrates the high-level architecture of the B-LOCK system. This architecture comprises several key components: the user interface, blockchain smart contracts, and the storage backend. The user interface allows users to interact with the application, facilitating file uploads and retrieval. The blockchain smart contracts, deployed on Ethereum, handle file metadata storage and verification. The storage backend, implemented using Firebase, manages the actual file storage and retrieval. This design ensures a secure and user-friendly application where file integrity and ownership are maintained through blockchain technology.

3.2 Decomposition Description

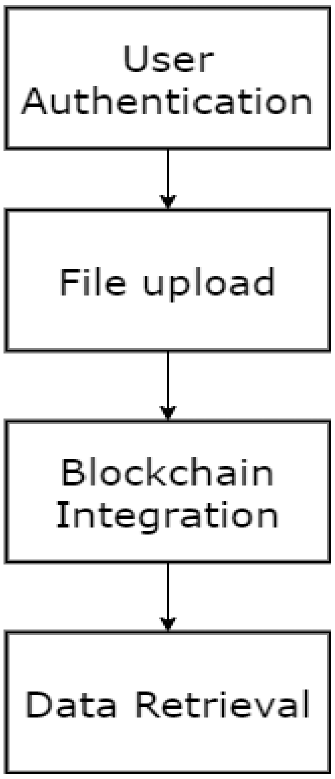

Figure 3.2: Flow chart

Figure 3.2 presents a decomposition diagram (or a flow chart) of the B-LOCK system. This diagram breaks down the system into its core modules: user authentication, file upload, blockchain integration, and data retrieval. Each module is responsible for specific functionalities:

- **User Authentication:** Manages user sign-in and sign-up, including email verification and session management.
- **File Upload:** Handles file selection, uploading to Firebase Storage, and associating file metadata with the blockchain.
- **Blockchain Integration:** Interacts with Ethereum smart contracts to store and retrieve file hashes and metadata.
- **Data Retrieval:** Provides mechanisms for users to access their uploaded files and associated data.

3.3 Data Flow Design

Figure 3.3 depicts the data flow within the B-LOCK application. The diagram illustrates the process from file upload to storage and retrieval. When a user uploads a file, the

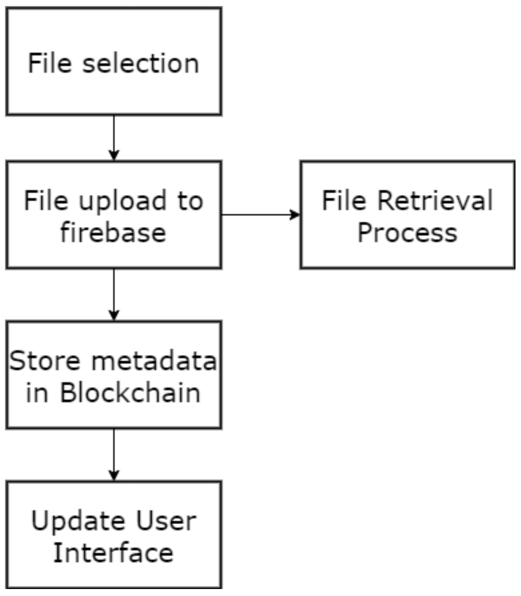

Figure 3.3: Dataflow Diagram

file is first hashed and then uploaded to Firebase Storage. The file's metadata, including its name and hash, is sent to the Ethereum smart contract for secure storage on the blockchain. Upon successful upload, the application updates the user interface to reflect the new file and its status. During file retrieval, the application queries both Firebase and the blockchain to provide users with the necessary information and access to their files.

Chapter 4

Implementation

4.1 System Overview

The system is a blockchain-based e-vault application designed to securely store and manage files with metadata. The core components include:

- **Frontend:** A React application for user interaction.
- **Backend:** Firebase for file storage and Firestore for metadata management.
- **Blockchain:** Ethereum for file hash storage and verification.

4.2 Pseudocode and Algorithms

4.2.1 File Upload Algorithm

```
function uploadFile(file):
  hash = generateFileHash(file.name, currentTimestamp)
  storageRef = createStorageReference(user.uid, hash, file.name)
  uploadTask = uploadFileToStorage(storageRef, file)

  onUploadProgress(uploadTask):
      progress = calculateUploadProgress(uploadTask)
      displayUploadProgress(progress)

  onUploadComplete(uploadTask):
      downloadURL = getDownloadURL(uploadTask)
      storeFileHashInBlockchain(fileName, fileHash)
      saveFileMetadataInFirestore(fileName, downloadURL, user.uid)
      displaySuccessMessage("File uploaded successfully")

  onUploadError(uploadTask):
        displayErrorMessage("Error uploading file")
```

Listing 4.1: File Upload Algorithm

- **Generate File Hash:** Create a unique hash using the file name and current timestamp.
- **Create Storage Reference:** Create a reference in Firebase Storage using the hash and file name.
- **Upload File:** Perform the file upload asynchronously.
- **Progress Monitoring:** Update progress based on the upload state.
- **Completion Handling:** On success, store the file hash in the blockchain and metadata in Firestore.
- **Error Handling:** Display appropriate error messages if the upload fails.

4.2.2 User Authentication Algorithm

```
    function signIn(email, password):
     try:
         user = authenticateUserWithEmail(email, password)
         if user.isEmailVerified:
             redirectToDashboard()
         else:
             displayErrorMessage("Email not verified")
     except AuthenticationError as e:
         displayErrorMessage(e.message)
```

Listing 4.2: User Authentication Algorithm

- **Authenticate User:** Check user credentials against Firebase Authentication.
- **Email Verification Check:** Ensure the user's email is verified before granting access.
- **Error Handling:** Handle and display errors during authentication.

4.3 Key Components Implementation

4.3.1 Frontend (React)

The frontend is developed using React.js and consists of the following key components:

- **SignIn Component:** Handles user login.
- **SignUp Component:** Manages user registration and email verification.
- **Upload Component:** Manages file upload and progress display.
- **Retrieve Component:** Allows users to view and manage uploaded files.

```
import React, { useState } from 'react';
import { signInWithEmailAndPassword } from 'firebase/auth';
import { auth } from './firebase';

const SignIn = ({ onSignIn, switchToSignUp }) => {
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [error, setError] = useState('');

    const handleSubmit = async (e) => {
        e.preventDefault();
        try {
            const userCredential = await signInWithEmailAndPassword(auth
            const user = userCredential.user;

            if (user.emailVerified) {
                onSignIn(email, password);
                setError(''); // Clear previous errors
            } else {
                setError('Please verify your email before signing in.');
                alert('Email not verified. Please check your inbox for t
            }
        } catch (err) {
            console.error("Sign-in error:", err);
            setError(err.message);
        }
    };

    return (
        <div className="auth-form">
            <h2>Sign In</h2>
            <form onSubmit={handleSubmit}>
                <input
                    type="email"
                    placeholder="Email"
                    value={email}
```

```
                        onChange={(e) => setEmail(e.target.value)}
                        required
                    />
                    <input
                        type="password"
                        placeholder="Password"
                        value={password}
                        onChange={(e) => setPassword(e.target.value)}
                        required
                    />
                    <button type="submit">Sign In</button>
                </form>
                {error && <p className="error">{error}</p>}
                <p>Don't have an account? <button onClick={switchToSignUp}>S
            </div>
        );
    };

    export default SignIn;
```

Listing 4.3: Frontend Code Snippet

4.3.2 Backend (Firebase and Blockchain)

- **Firebase Storage:** Stores the uploaded files.
- **Firestore Database:** Saves metadata about the files.
- **Ethereum Blockchain:** Stores the file hashes for verification.

```
   import { ref, uploadBytesResumable, getDownloadURL } from "firebase/stor
import { collection, addDoc } from "firebase/firestore";

const uploadFile = async (file, user, storage, db, evault, account) => {
    const hash = CryptoJS.SHA256(file.name + new Date().toISOString()).toSt
    const storageRef = ref(storage, `uploads/${user.uid}/${hash}_${file.nam
    const uploadTask = uploadBytesResumable(storageRef, file);

    uploadTask.on(
        "state_changed",
        (snapshot) => {
            const progress = (snapshot.bytesTransferred / snapshot.totalByt
            console.log(`Upload is ${progress.toFixed(2)}% done`);
        },
        (error) => {
            console.error("Upload error:", error);
        },
        async () => {
            const downloadURL = await getDownloadURL(uploadTask.snapshot.re
            try {
                await evault.methods.storeFile(file.name, hash).send({ from
                await addDoc(collection(db, "files"), {
                    name: file.name,
                    owner: user.uid,
                    downloadURL,
                    timestamp: new Date(),
                });
                console.log("File uploaded successfully");
            } catch (error) {
                console.error("Error saving file data", error);
            }
        }
    );
};
```

Listing 4.4: Backend Code Snippet

4.4 Integration

The frontend components interact with the Firebase backend and Ethereum blockchain to provide a seamless user experience. Authentication is handled via Firebase Authentication, ensuring that users are securely signed in. File uploads are managed through Firebase Storage, allowing users to upload and access their files easily. Metadata related to the files, such as names and download URLs, is stored in Firestore, providing quick and reliable access to file information. The integration with the Ethereum blockchain guarantees that file hashes are securely stored and verifiable, leveraging smart contracts to maintain the integrity and ownership of each file. This combination of Firebase and blockchain technologies ensures that the application is both secure and scalable, providing a robust solution for digital file management.

Chapter 5

Results and Discussion

this chapter contains the paragraphs as shown below

5.1 Face detection

Figure 5.1: Face detection

Above figure 8.1 shows initial face detection process using opencv and dlib. It convert the image to grayscale, apply the model using cv2.detectMultiScale(), and draw bounding boxes around the detected faces using cv2.rectangle(). Display or save the result using cv2.imshow() or cv2.imwrite().

5.2 Speaker recognition

Figure 5.2: Speaker recognition 1,person 1

Figure 5.3: Speaker recognition 1,person 2

Figure 5.4: Speaker recognition 2,person 1

Figure 5.5: Speaker recognition 2,person 2

Figure 5.6: Speaker recognition 3,person 1

Figure 5.7: Speaker recognition 3,person 2

Above figures from 8.2 to 8.7 shows speaker recognition process using opencv and dlib.Speaker detection using cv2 and dlib involves utilizing dlib's pre-trained models along with cv2 functions to detect and locate human faces. By combining face detection with additional techniques such as audio analysis or lip movement tracking, speaker detection can be achieved in various applications like video conferencing or surveillance.

Chapter 6

Conclusion and Future Work

[1] [5] The Project will help in narrowing the imprecise communication problem in real-time data using speech separation and speaker identification technique by Deep Learning and Image Processing algorithms. This will impact the communication and security sectors in a greater extent. Overall, this project aims to develop an application or method that can help to separate the audio-visual speech and enhance it based on speaker identification.
This project can be further developed as:

- By incorporating more real-world testing and gathering feedback from individual units.
- The system can be connected with communication devices or services to enable the users to communicate with others with ease.

[7] This project has a great potential to make a positive impact on communication and security situations. Its continuous improvement will be important to make this impact even greater example for citing and bibtex for journal paper [2], conference paper [6], citing website [4], citing book [3]

References

[1] Muhammad Farrukh Bashir et al. “Subjective answers evaluation using machine learning and natural language processing”. In: *IEEE Access* 9 (2021), pp. 158972–158983.

[2] Florin-Gabriel Croitoru, Marius Leordeanu, and Radu Tudor Ionescu. “Diffusion Models for Image Super-Resolution: A Survey”. In: *Journal of Imaging* 9.1 (2023), p. 16.

[3] Paul Adrien Maurice Dirac. “The Principles of Quantum Mechanics”. In: International series of monographs on physics. Clarendon Press, 1981. ISBN: 9780198520115.

[4] Donald Knuth. *Knuth: Computers and Typesetting.* URL: `http://www-cs-faculty.stanford.edu/~uno/abcde.html`. accessed on 01.09.2016.

[5] Himani Mittal and M. Syamala Devi. “Computerized Evaluation of Subjective Answers Using Hybrid Technique”. In: *Innovations in Computer Science and Engineering.* Ed. by H. S. Saini, Rishi Sayal, and Sandeep Singh Rawat. Singapore: Springer Singapore, 2016, pp. 295–303. ISBN: 978-981-10-0419-3.

[6] Omar Abdulwahabe Mohamad. “Smart Home Security based on optimal wireless sensor network routing protocols”. In: *2015 7th International Conference on Electronics, Computers and Artificial Intelligence (ECAI).* IEEE. 2015, SSS–17.

[7] Lee Smith. “City of Dreams”. In: *Tablet* 18 March (2008).