Mansoura University

Faculty of computer and information
information science -AI department

Anti-AI

Graduation Project (2)

by

Mohamed Abd-elaziz

Ranim Ahmed Eldefrawy

Nourhan Mahmoud Rashad

Rawan Adel Madian

Ayat Mohammed Othman

Tasneem Ahmed Alhelaly

Ghada Hafez Abdelhamid

Mohamed Mahmoud Elmahdy

NourAldin Abdalkader Hassan

Mohanned Ahmed Saleh

A project submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Information Technology

Supervised by

Prof. Sara Elsayed Elmetwaly

Eng. Amr Eledkawy

2nd Semester 2023-2024

Abstract

The objective of this proposal is to combat the prevalent cruel and deceptive use of AI by raising awareness among users about unethical AI behavior. Our aim is to provide users with the necessary tools and knowledge to identify and avoid fake AI applications while promoting the ethical use of AI technology to foster trust and integrity in AI ecosystems.

Artificial intelligence has experienced unprecedented growth and integration into various aspects of our lives, offering numerous benefits and opportunities. However, this rapid advancement has also given rise to ethical concerns as malicious actors exploit AI for deceptive purposes, such as generating manipulated images or spreading misleading news. Immediate action is imperative to safeguard against these unethical practices and ensure responsible AI usage.

User engagement and empowerment are vital components of our approach. We will develop user-friendly platforms and channels where individuals can report suspicious AI applications and provide feedback on their experiences. By actively involving users in the detection process, we can gather valuable insights and ensure a proactive approach to identifying and addressing fake AI applications.

Continuous improvement and accountability will be prioritized through the implementation of robust monitoring systems and regular audits. We will evaluate the effectiveness of our detection mechanisms and stay updated on the latest developments in AI ethics to continually enhance our strategies and adapt to emerging challenges.

By implementing these initiatives, we aim to equip users with the necessary knowledge and resources to identify and avoid fake AI applications, promoting ethical AI usage. Our ultimate goal is to foster trust and integrity in AI ecosystems, ensuring that this transformative technology is harnessed for the betterment of society.

Keywords Ai-usage; fake-ai; ethical; detection; Transformative technology; trust

Acknowledgement

First and foremost, We are immensely grateful to our project supervisor, Prof. Sara Elmetwally, and Eng. Amr Eledkawy, for their invaluable guidance, mentorship, and expertise throughout this journey. Their insightful feedback, encouragement, and unwavering support have been instrumental in shaping the direction and quality of our project.

We would like to extend our gratitude to the team members for their knowledge, expertise, and dedication to education. Their teachings and insights have laid a strong foundation in the understanding of artificial intelligence and its applications.

We would also like to thank our classmates and friends who have provided a supportive and collaborative environment. Their discussions, feedback, and encouragement have been invaluable in refining our ideas and pushing us to achieve the best.

We are indebted to the numerous researchers and scholars in the field of artificial intelligence whose groundbreaking work has served as a source of inspiration and knowledge. Their contributions have paved the way for advancements in AI and have enriched our understanding of the subject.

Last but not least, We would like to convey our heartfelt appreciation to our families and loved ones for their unwavering support, encouragement, and understanding throughout this demanding journey. Their belief in us and constant encouragement have been our driving force.

To all those mentioned above and to anyone else who has directly or indirectly contributed to the successful completion of our graduation project, We offer our deepest thanks. Your support, guidance, and contributions have been vital in shaping our growth as students and researchers in the field.

Thank you all for being an integral part of this remarkable journey.

Table of Content

Abstract **2**
Acknowledgement **3**
Table of Content **4**
List of Figures **6**
List of Tables **13**
Chapter 1 **2**
Chapter 1: Introduction **3**
1.1. Introduction 3
1.2. Problem Definition 3
1.3. Project Objectives 3
1.4. Project Scope 4
1.5. Contributions of This Study 7
1.6. Project Timeline 9
1.7. Document Organization 9
Chapter 2 **11**
Chapter 2: Literature Review **12**
2.1 Introduction 12
2.2 Background 13
2.3 Review of Relevant Work 15
2.4 Relationship between the Relevant Work and Our Work 18
2.5 Summary 20
Chapter 3 **22**
Chapter 3: System Analysis **23**
3.1 Introduction 23
3.2 Analysis of Existing Systems 23
3.3 System Requirements 25
3.3.1 Functional Requirements 25
3.3.2 Non-Functional Requirements 26
3.3.3 Design Objectives 27
3.4 System Architecture 30
3.5 Development Methodology 31
3.5.1 Use Case Diagrams 31
3.5.2 Use Case Description (Detailed Use Cases) 33
3.5.3 Sequence Diagram 34
3.6 Tools and Languages 36

3.7 Summary 38
Chapter 4 **39**
Chapter 4: System Design **40**
4.1 Introduction 40
4.2 ERD Diagram 40
4.3 Class Diagram 41
4.4 Database Design of Algorithms 42
Detection_Results 43
Feedback 44
4.5 Interface Design 45
4.6 Summary 65
Chapter 5 **66**
Chapter 5: System Implementation **67**
5.1 Introduction 67
5.2 Mapping Design to Implementation 67
5.3 Application Codes 70
Machine learning code (Model) 70
Backend Implementation 116
Application code (flutter) 126
5.4 System Testing 160
5.5 Results 162
5.6 Goals Achieved 163
Chapter 6 **164**
Chapter 6: Conclusion and Future Work **165**
6.1 Conclusion 165
6.2 Future Work 166
References **169**

List of Figures

Figure 1.1: Project Timeline……..9

Figure 3.1: Use Case Description……..…..32

Figure 3.2: Sequence Diagram……34

Figure 4.1: ERD Diagram……...3

Figure 4.2: Class Diagram……..39

Figure 4.3: Splash screen……41

Figure 4.4: On Boarding-1……42

Figure 4.5: On Boarding-2……42

Figure 4.6: On Boarding-3……42

Figure 4.7: Login……43

Figure 4.8: Create account……..43

Figure 4.9: Sign-Up verification-1…….…….44

Figure 4.10: Sign-Up verification-2……...….44

Figure 4.11: Sign-Up verification-3……...……44

Figure 4.12: Forgot password-1……..45

Figure 4.13: Forgot password-2……..45

Figure 4.14: Forgot password-3……..45

Figure 4.15: Forgot password-4……..45

Figure 4.16: Home-1……..……..46

Figure 4.17: Home-2……..……..46

Figure 4.18: Home-3……..……..46

Figure 4.19: Home-4……..……..46

Figure 4.20: Home-Upload button…….....……..47

Figure 4.21: Detect fake photos-1……..48

Figure 4.22: Detect fake photos-2……..48

Figure 4.23: Detect fake photos-3……..48

Figure 4.24: Detect fake photos-4……..48

Figure 4.25: Detect fake photos-5……..48

Figure 4.26: Payment……..49

Figure 4.27: Community-Timeline…….49

Figure 4.28: Community-Upload button-1……..…….…..50

Figure 4.29: Community-Upload button-2……..…….…..50

Figure 4.30: Community-Profile(Posts)-1……..……50

Figure 4.31: Community-Profile(Posts)-2……..……50

Figure 4.32: Community-Profile(Reacts)……..…….51

Figure 4.33: Community-Profile(Votes)……..……...……51

Figure 4.34: Community-Profile(Replies)……..……52

Figure 4.35: Community-Edit Profile……..……...……52

Figure 4.36: Community-Another user's profile……..……..……53

Figure 4.37: Notifications……..…….……53

Figure 4.38: Follow requests-1……..…….……54

Figure 4.39: Follow requests-2……..…….……54

Figure 4.40: Profile……..……..…….……54

Figure 4.41: Profile-Settings & privacy……..……....55

Figure 4.42: Change your password……..……..55

Figure 4.43: Profile-Language……..……..…56

Figure 4.44: Profile-Terms & policy……..……..…56

Figure 4.45: Profile-About us……..…….....57

Figure 4.46: Profile-Contact us-1……..……...……..….57

Figure 4.47: Profile-Contact us-2……..……..……57

Figure 4.48: Profile-Logout……..……...……58

Figure 5.1: Install the datasets ……63

Figure 5.2: Install or Upgrade the accelerate library……...…64

Figure 5.3: Install the transformers library with PyTorch support……..….64

Figure 5.4: Import the load_dataset function……..65

Figure 5.5: View the loaded dataset……..…..65

Figure 5.6: Access an image from the dataset…….……67

Figure 5.7:Access and print the label of a specific example in the dataset ……68

Figure 5.8: View the Loaded Dataset Structure……..69

Figure 5.9: Define the Transformer model checkpoint……...……70

Figure 5.10: Import the notebook_login function…….…...……71

Figure 5.11: Import the AutoImageProcessor class Code…….…...……72

Figure 5.12: Import necessary transformations from torchvision.transforms…...…..……74

Figure 5.13: Split the training data into training and validation datasets…….……..…78

Figure 5.14: Extract label names from the dataset……..……80

Figure 5.15: Set transformation function for the training dataset…….....…..……82

Figure 5.16: Access the first element of the training dataset..…….....…..…….……84

Figure 5.17: Import necessary classes from the transformers library…….....…..……..…86

Figure 5.18: Extract the model name from the checkpoint path……88

Figure 5.19: Import the load_metric function from the datasets library…….……90

Figure 5.20: Import the numpy library……..……..……91

Figure 5.21: Import the torch library……..……..……...93

Figure 5.22: Initialize the Trainer object……..……..……95

Figure 5.23: train_results = trainer.train()……..……..……..…95

Figure 5.24: trainer.save_model()……....……..……95

Figure 5.25: trainer.log_metrics("train", train_results.metrics)……..……95

Figure 5.26: trainer.save_metrics("train", train_results.metrics).....95

Figure 5.27: trainer.save_state().....……..……95

Figure 5.28: Save train results……..……..……..…….95

Figure 5.29: Evaluation Results……..……..……..……..103

Figure 5.30: Azure / hugging face infrastructure……..…..……..103

Figure 5.31: Loading the Model and Image Processor……..……..……..……..106

Figure 5.34: Root Folder Structure.....…….110

Figure 5.35: Application Folder Structure…….……..…110

Figure 5.36: Profile Model.....……111

Figure 5.37: Register Serializer.....……112

Figure 5.38: Login View.....……113

Figure 5.39: Generate OTP View.....……114

Figure 5.40: Routes file.....…...115

Figure 5.41: Face-Classifier Service.....116

Figure 5.42: Dockerfile.....117

Figure 5.43: Docker Compose file.....118

Figure 5.44: Splash screen-1.....119

Figure 5.45: Splash screen-2.....120

Figure 5.46:login 121

Figure 5.47:signup 129

Figure 5.48:OTP 129

Figure 5.49:fillyourinfo 129

Figure 5.50:fillyourinfo 130

Figure 5.51:forgetpassword 131

Figure 5.53 terms and conditions.....133

Figure 5.54 home screen all.....134

Figure 5.55 home screen all.....135

Figure 5.56 home screen all.....136

Figure 5.57 fake uploads.....137

Figure 5.58 real uploads.....138

Figure 5.59 detect fake images.....140

Figure 5.60 detect fake images.....140

Figure 5.61 result screen.....142

Figure 5.62 Alert.....143

Figure 5.63 follow request.....144

Figure 5.64 about us…….....145

Figure 5.65 contact us.....146

Figure 5.66 profiles.....147

Figure 5.67 settings 148

Figure 5.68 terms and policy.....149

Figure 5.69 community.....150

Figure 5.70 posts_screen.....151

Figure 5.71 reacts screen.....151

Figure 5.72 replies.....152

Figure 5.73 votes…….....153

Figure 5.74 user log in.....154

Figure 5.75 sign up 155

Figure 5.76 OTP.....156

Figure 5.77 attempt validate.....157

Figure 5.78 profile list,,,,,.....158

Figure 5.79 profile detail.....159

Figure 5.80 profile attempt detail.....160

Figure 5.81 profile reset pass.....161

Figure 5.82 upload post.....162

Figure 5.83 post list.....162

Figure 5.84 post by uuid.....165

Figure 5.85 delete post.....166

Figure 5.86 img model attempt.....166

List of Tables

Table 3.1: Functional Requirements……26

Table 3.2: Non-Functional Requirements……27

Table 4.1: Users ……...….39

Table 4.2: Images ……40

Table 4.3: Detection results……..……40

Table 4.4: Feedbacks……..……..……40

Table 4.5: Admins……..……..…….……43

Chapter 1

Chapter 1: Introduction

1.1. Introduction

The primary purpose of the chapter is to provide an overview of the study. The introduction must also include the following components either separated into sections or integrated into one narrative.

1.2. Problem Definition

The issue being addressed revolves around the unethical utilization of AI, involving the creation of deceptive images and text. These deceitful materials have the potential for harmful applications, such as spreading misinformation, creating deep fakes, and propagating misleading content. With the ongoing advancements in AI technology, differentiating between genuine and falsified content has become progressively difficult. Consequently, this poses a significant threat to the well-being of individuals, institutions, and the credibility of AI environments

1.3. Project Objectives

1. **Raising Awareness**

    a. Educate users about the potential risks and negative implications associated with the unethical use of AI.
    b. Empower individuals to recognize and critically evaluate content that might be generated using AI for deceptive purposes like misinformation or manipulation.
    c. Encourage open dialogue and debate about the ethical use of AI.

2. **Developing Detection Tools**

a. Develop algorithms, software, or systems capable of identifying discrepancies or markers that indicate content generated by AI.
b. Create tools that can detect AI-generated content with high accuracy and reliability.
c. Develop methods for analyzing content at scale to identify and flag potentially fabricated or manipulated material.
d. Integrate AI detection tools into existing platforms and applications to seamlessly detect AI-generated content.
e. Support the development of new techniques for AI content detection.

By pursuing these objectives, the aim is to create an approach that addresses the immediate challenge of detecting and preventing the misuse of AI-generated content and promotes a broader culture of ethical AI usage

1.4. Project Scope

The scope of this project encompasses the development of a sophisticated yet user-friendly mobile application and website designed to detect AI-generated images. By focusing on accuracy, accessibility, and user education, the project aims to provide a reliable tool for combating digital misinformation. Clear objectives, functional and non-functional requirements, defined deliverables, and a structured timeline ensure the project remains on track and meets its goals.

The project aims to develop a comprehensive and user-friendly mobile application and website for detecting AI-generated images. This tool will help users identify manipulated media and combat digital misinformation by leveraging advanced machine learning techniques. Below are the detailed elements of the project scope:

1. Objectives

- **Detection Accuracy**: Develop robust algorithms to accurately identify AI-generated images.

- **User Accessibility**: Ensure the application and website are intuitive and accessible to a broad audience.
- **Educational Resource**: Provide information and tools to help users understand and recognize AI-generated content.

2. Functional Requirements

- **Image Analysis**: Implement algorithms to analyze and detect AI-generated images.
- **User Interface**: Design a user-friendly interface for both the mobile app and website.
- **Real-Time Detection**: Enable real-time processing and feedback for uploaded images.
- **User Feedback**: Allow users to provide feedback on detection results to improve the system.
- **Report Generation**: Provide detailed reports on the analysis, highlighting detected anomalies.

3. Non-Functional Requirements

- **Performance**: Ensure fast and efficient processing of images to provide quick results.
- **Scalability**: Design the system to handle a large number of concurrent users and high volumes of data.
- **Security**: Implement robust security measures to protect user data and ensure privacy.
- **Reliability**: Ensure the system is reliable and has minimal downtime.

4. In-Scope Activities

- **Requirement Analysis**: Conduct thorough analysis to understand user needs and system requirements.
- **Design and Architecture**: Create detailed design and architecture plans for both the mobile app and website.
- **Development**: Implement the detection algorithms, user interface, and backend infrastructure.
- **Testing**: Perform extensive testing to ensure accuracy, performance, and security.
- **Deployment**: Deploy the mobile app on relevant app stores and the website on a reliable hosting platform.
- **Maintenance**: Provide ongoing maintenance and updates to ensure the system remains effective and secure.

5. Out-of-Scope Activities

- **Non-Image Media**: Detection of AI-generated audio and video content will be considered for future work and is not included in the initial scope.
- **Third-Party Integration**: Integration with third-party applications and services is not within the initial project scope but may be considered for future phases.

6. Deliverables

- **Mobile Application**: A fully functional mobile app available on major app stores.
- **Website**: A fully functional website accessible through major web browsers.
- **Documentation**: Comprehensive documentation including user guides, technical specifications, and maintenance procedures.
- **Reports**: Regular progress reports and a final project report summarizing achievements and lessons learned.

7. Milestones

- **Project Initiation**: Define project goals, scope, and deliverables.
- **Requirement Gathering**: Collect and analyze user requirements.
- **Design Phase**: Complete design of system architecture and user interface.
- **Development Phase**: Develop the core functionalities of the app and website.
- **Testing Phase**: Conduct testing to ensure all components meet requirements.
- **Deployment**: Launch the mobile app and website.
- **Post-Launch**: Monitor system performance, gather user feedback, and perform necessary updates.

8. Stakeholders

- **Project Team**: Developers, designers, data scientists, and project managers.
- **Users**: Individuals and organizations using the app and website to detect AI-generated images.
- **Advisors**: Experts in AI, machine learning, and cybersecurity.
- **Investors and Sponsors**: Entities funding the project.

9. Constraints and Assumptions

- **Budget**: The project will adhere to a predefined budget covering development, testing, and deployment costs.
- **Timeline**: The project will follow a strict timeline with predefined milestones to ensure timely delivery.

- **Technological Resources**: Availability of necessary technological resources, including cloud infrastructure and development tools.
- **User Base**: The initial user base will primarily consist of individuals concerned with digital misinformation.

1.5. Contributions of This Study

1. **Enhanced User Authentication and Access Control**:
    - Development of a robust authentication system allowing users to securely sign up, log in, and navigate the application both as registered users and guests.
    - Introduction of flexible access control, permitting users to manage their privacy settings regarding shared content.
2. **Innovative Fake Content Detection**:
    - Implementation of advanced AI algorithms for verifying the authenticity of uploaded photos, documents, and text, contributing to the field of fake content detection.
    - Categorization of photos to improve detection accuracy, setting a new standard for context-aware AI verification systems.
3. **Community Engagement and Collaboration**:
    - Creation of a community platform where users can share, vote, and discuss the authenticity of various media, fostering a collaborative environment for misinformation detection.
    - Introduction of gamified elements, such as voting and point systems, to enhance user engagement and participation in the verification process.
4. **User-Generated Content Management**:
    - Provision of comprehensive tools for users to manage their accounts and uploads, including privacy settings, feedback mechanisms, and account deactivation/deletion.
5. **Enhanced Awareness and Education**:

    - Inclusion of a rich library of awareness articles and videos to educate users on identifying and understanding fake content, contributing to digital literacy.
6. **Scalability and Performance Optimization**:
    - Demonstration of the system's scalability to handle increasing user loads and data volumes, ensuring performance stability and reliability under various conditions.
    - Introduction of payment systems to allow users to exceed daily usage limits, showcasing a sustainable model for high-demand applications.
7. **Improved User Experience and Usability**:
    - Comprehensive usability testing to ensure an intuitive and user-friendly interface, making the application accessible to a broad audience.
    - Development of feedback mechanisms allowing users to rate and comment on the verification results, fostering continuous improvement of the system.
8. **Data Security and Compliance**:
    - Implementation of stringent security measures to protect user data and comply with industry standards and regulations.
    - Contribution to the body of knowledge on secure and compliant handling of personal data within AI-driven applications.

By addressing these areas, this study provides a significant contribution to the fields of AI, digital literacy, community engagement, and user experience design. The findings and innovations presented can serve as a foundation for future research and development in creating secure, user-friendly, and effective systems for detecting and managing fake content.

1.6. Project Timeline

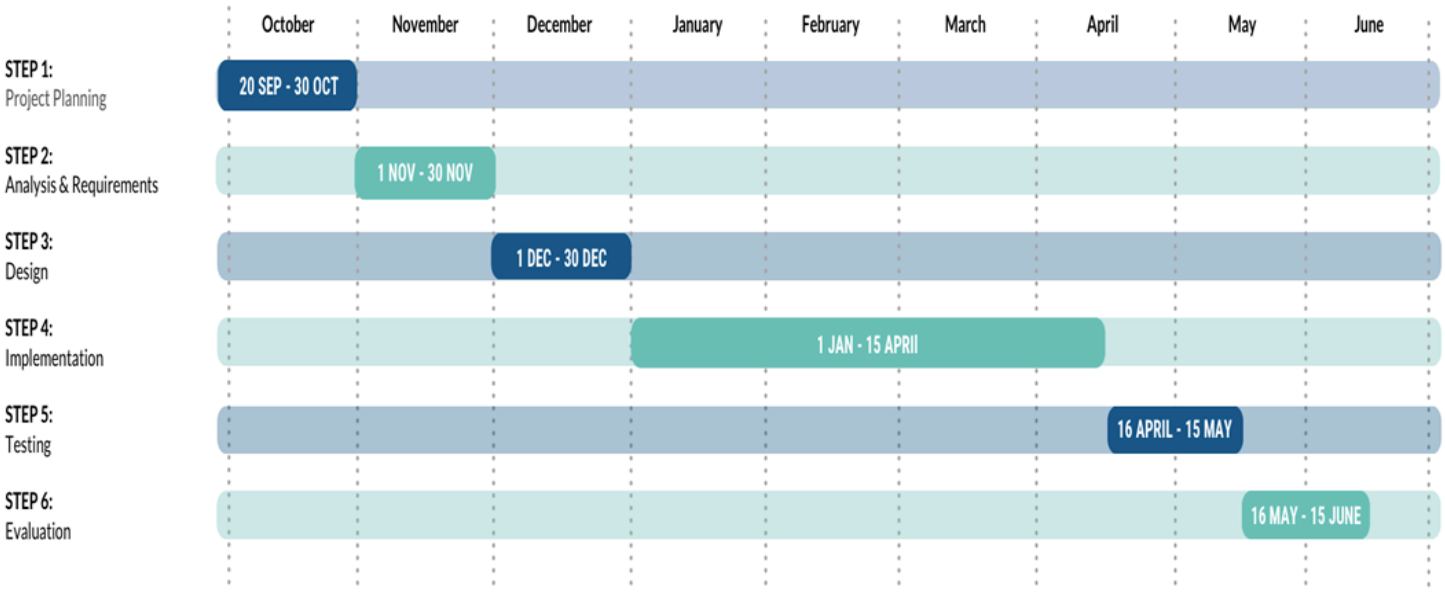


Fig 1.1

1.7. Document Organization

This project consists of six chapters in addition to one appendix. These chapters are organized to reflect the scientific steps toward our main objective. A brief description about the contents of each chapter is given in the following paragraphs:

Chapter 1: Introduces the project objectives, the motivation of the project, the approach used in this project, the contribution of this project, the scope of the work, and project layout.

Chapter 2: Provides the reader with an overview of the previous related work, common technologies used, and the relation between

Chapter 3: Explains the analysis of existing systems, system requirements, user requirements, system architecture, development methodology using UML, the tools, and languages used in our system.

Chapter 4: Display the system Ui design including class diagram, database design and interface design.

Chapter 5: System Implementation including snapshots of code.

Chapter 6: Conclusion & Future Work , Appendix with code and snapshots , References .

Chapter 2

Chapter 2: Literature Review

2.1 Introduction

In an era characterized by unprecedented digital communication and information dissemination, the proliferation of fake content—be it photos, documents, or text—has become a pervasive challenge. The advent of sophisticated artificial intelligence (AI) technologies has facilitated the creation of highly convincing yet deceptive media, complicating efforts to discern authentic content from fabricated material. This literature review aims to explore the existing body of research and technological advancements in the domain of fake content detection, user engagement in content verification, and the implications for digital literacy and security.

The initial sections of this review will delve into the evolution of AI-driven fake content generation, examining how advancements in deep learning and neural networks have enabled the production of realistic fake images, videos, and text. Key studies will be analyzed to understand the techniques used in generating such content and the challenges they pose to traditional verification methods.

Subsequently, the review will focus on the various methodologies and algorithms developed to detect fake content. This includes a comprehensive examination of machine learning and deep learning approaches, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which have shown promise in identifying inconsistencies and anomalies indicative of fabricated media. The effectiveness, accuracy, and limitations of these techniques will be critically assessed, providing insights into the current state-of-the-art in fake content detection.

In addition to technical solutions, the review will address the role of user engagement and community collaboration in combating misinformation. By leveraging collective intelligence, platforms can enhance the detection and reporting of fake content. The integration of

gamification elements, such as voting systems and point rewards, will be discussed as a means to increase user participation and motivation in the verification process.

The review will also consider the broader implications of fake content detection technologies on digital literacy and security. Educational initiatives and awareness campaigns are essential in empowering users to critically evaluate the information they encounter. The impact of such initiatives on user behavior and the overall efficacy of fake content detection will be explored.

Furthermore, the review will evaluate the ethical considerations and privacy concerns associated with the deployment of AI-based verification systems. Ensuring user data security and maintaining transparency in AI decision-making processes are crucial for building trust and acceptance among users.

By synthesizing findings from diverse studies and technological developments, this literature review aims to provide a holistic understanding of the current landscape and future directions in fake content detection and management. The insights gained will inform the development of robust, user-friendly, and effective systems capable of safeguarding digital environments against the growing threat of misinformation.

2.2 Background

The proliferation of digital content and the rise of social media platforms have transformed the way information is disseminated and consumed globally. This digital revolution, while facilitating unprecedented connectivity and access to information, has also given rise to significant challenges, notably the spread of fake content. The term "fake content" encompasses a broad spectrum of deceptive media, including manipulated images, deepfake videos, fabricated news articles, and misleading text. The dissemination of such content can have profound implications, from eroding public trust in media to influencing political outcomes and causing widespread misinformation.

- **Evolution of Fake Content Generation**

The generation of fake content has evolved significantly with advancements in artificial intelligence (AI) and machine learning (ML). Early methods of content manipulation involved basic photo editing techniques and textual fabrication, which were often detectable through simple scrutiny. However, the advent of sophisticated AI techniques, particularly deep learning, has enabled the creation of highly realistic and convincing fake content. Technologies such as Generative Adversarial Networks (GANs) have been pivotal in this evolution. GANs consist of two neural networks—a generator and a discriminator—that work in tandem to create realistic images, videos, and text that can deceive even the most discerning human eyes.

- **Impact of Fake Content**

The impact of fake content on society is multifaceted. In the realm of politics, fake news and manipulated media have been used to sway public opinion, disrupt electoral processes, and incite social unrest. Economically, businesses and brands can suffer reputational damage due to fake reviews, counterfeit products, and misleading advertisements. On a personal level, individuals can fall victim to scams, identity theft, and privacy breaches facilitated by fake content. The psychological effects of encountering fake content, including confusion, mistrust, and anxiety, further underscore the need for effective detection mechanisms.

- **Technological Advancements in Detection**

In response to the growing threat of fake content, researchers and technologists have developed various methodologies to detect and mitigate its spread. Machine learning and deep learning techniques have been at the forefront of these efforts. Convolutional Neural Networks (CNNs) have shown promise in detecting manipulated images by identifying subtle inconsistencies in pixel patterns that are often undetectable to the human eye. Recurrent Neural Networks (RNNs) and Natural Language Processing (NLP) techniques have been employed to analyze text and identify fabricated news articles based on linguistic patterns and contextual anomalies.

- **Community Engagement and Collaborative Verification**

Recognizing the limitations of automated detection systems, there has been a growing emphasis on community engagement and collaborative verification. Platforms that enable users to report, vote on, and discuss the authenticity of content leverage the collective intelligence of the community. Gamification elements, such as point systems and rewards, have been introduced to motivate user participation and enhance the accuracy of community-driven verification processes. This collaborative approach not only improves detection rates but also fosters a sense of responsibility and awareness among users.

- **Digital Literacy and Security**

The fight against fake content is not solely a technological challenge but also a societal one. Digital literacy initiatives aimed at educating users about the nature of fake content and equipping them with critical evaluation skills are crucial. Awareness campaigns and educational programs help users to discern credible information sources, recognize manipulation techniques, and make informed decisions about the content they consume and share. Furthermore, ensuring the security and privacy of users' data in AI-driven verification systems is essential to maintain trust and compliance with regulatory standards.

- **Ethical Considerations**

The deployment of AI-based fake content detection systems raises several ethical considerations. Transparency in AI decision-making processes, accountability for false positives and negatives, and the potential for misuse of detection technologies are critical issues that must be addressed. Balancing the need for effective detection with respect for individual privacy and freedom of expression is a delicate task that requires ongoing ethical scrutiny and stakeholder engagement.

2.3 Review of Relevant Work

AI and Machine Learning Techniques for Fake Content Detection

The detection of fake content has significantly benefited from advancements in AI and machine learning. Key studies and developments in this area include:

1. **Generative Adversarial Networks (GANs)**:
    - GANs have revolutionized the creation of realistic fake content, as demonstrated by Goodfellow et al. (2014). GANs consist of two neural networks: a generator that creates fake content and a discriminator that evaluates its authenticity. This adversarial process results in highly convincing media, challenging traditional detection methods.
    - Research by Karras et al. (2019) introduced StyleGAN, which significantly improved the quality and realism of AI-generated images, highlighting the escalating sophistication of fake content.
2. **Convolutional Neural Networks (CNNs)**:
    - CNNs have proven effective in detecting manipulated images. For example, Bayar and Stamm (2016) developed a CNN-based method that identifies subtle artifacts left by image manipulation processes, achieving high accuracy in detecting fake images.
    - Other studies, such as those by Zhou et al. (2018), have explored the use of CNNs to detect deepfake videos by analyzing spatial and temporal inconsistencies.
3. **Natural Language Processing (NLP)**:
    - NLP techniques have been instrumental in identifying fake news and fabricated text. Works like those by Shu et al. (2017) have utilized NLP to analyze linguistic features and contextual inconsistencies, improving the detection of deceptive text.
    - BERT (Bidirectional Encoder Representations from Transformers) by Devlin et al. (2019) has been employed to enhance text classification tasks, including the detection of fake news through advanced contextual understanding.

User Engagement and Collaborative Verification

The involvement of users in the verification process has been shown to enhance the effectiveness of fake content detection. Notable studies and approaches include:

1. **Crowdsourced Verification**:
    - Platforms like Snopes and FactCheck.org have successfully leveraged crowdsourced verification, where users can report, discuss, and verify suspicious content. Research by Zubiaga et al. (2016) highlighted the efficacy of such collaborative efforts in identifying fake news.
    - Peer-to-peer verification systems, as discussed by Volkova et al. (2017), have demonstrated the potential of collective intelligence in improving the accuracy and speed of fake content detection.
2. **Gamification**:
    - Integrating gamification elements into verification processes has been explored to increase user engagement and motivation. Studies by Morschheuser et al. (2017) showed that point systems, badges, and leaderboards can enhance user participation and accuracy in collaborative tasks.

Digital Literacy Initiatives

Promoting digital literacy is crucial in equipping users with the skills to identify fake content. Key initiatives and research in this area include:

1. **Educational Programs**:
    - The News Literacy Project and similar initiatives aim to educate the public on identifying credible sources and understanding media biases. Research by Mihailidis and Viotty (2017) emphasized the importance of media literacy education in combating misinformation.
    - Digital literacy curricula, as advocated by Hobbs (2010), provide comprehensive frameworks for teaching critical evaluation skills in educational settings.
2. **Awareness Campaigns**:
    - Campaigns such as "StopFake" have been effective in raising awareness about the prevalence and dangers of fake content. Studies by Tandoc et al. (2018)

underscored the impact of such campaigns in improving public awareness and skepticism towards suspicious media.

Ethical and Privacy Considerations

The deployment of AI-based detection systems raises important ethical and privacy issues. Relevant literature in this domain includes:

1. **Transparency and Accountability**:
    - Ensuring transparency in AI decision-making processes is crucial for building trust. Research by Doshi-Velez and Kim (2017) proposed methods for making AI models interpretable and accountable, which are essential for responsible deployment.
    - Studies by Mittelstadt et al. (2016) highlighted the need for ethical guidelines and accountability mechanisms to address potential biases and errors in AI systems.
2. **Privacy Protection**:
    - Protecting user data privacy is paramount in AI-driven systems. The General Data Protection Regulation (GDPR) and other regulatory frameworks provide guidelines for ensuring data security and user consent. Research by Voigt and von dem Bussche (2017) discussed the implications of GDPR for AI and data-driven applications.
    - Techniques such as differential privacy, as explored by Dwork (2008), offer promising approaches to safeguard user data while maintaining the utility of AI systems.

2.4 Relationship between the Relevant Work and Our Work

The previously mentioned related work shows that the previous products are limited in their functionalities which lead the user to use more than one product to achieve what he wants , that was the main point we worked on in our product we grouped all main

functionalities in the same place for the user and to achieve simplicity and make it easier consuming less time to achieve what he needs .
Our product is a mobile application that allows the user to check the reality of a photo , a video and an audio displaying to the user the percentage of the photo being real or fake (ai generated)

Our work builds on these insights and seeks to address identified gaps and limitations, as well as to introduce novel contributions to the field. Here's how our work relates to and extends the existing research:

Building on AI and Machine Learning Techniques

Generative Adversarial Networks (GANs) and Convolutional Neural Networks (CNNs):

- **Existing Work**: The literature highlights the effectiveness of GANs in generating realistic fake content and CNNs in detecting manipulated images and videos. Studies have demonstrated the capabilities of these technologies in identifying artifacts and inconsistencies that indicate fakery.
- **Our Contribution**: We leverage these advancements by incorporating state-of-the-art GANs and CNNs into our fake content detection system. Our system aims to enhance detection accuracy by refining these models with additional training data and advanced pre-processing techniques, specifically focusing on the categories of uploaded photos to improve contextual understanding.

Enhancing User Engagement and Collaborative Verification

Crowdsourced Verification and Gamification:

- **Existing Work**: Research has shown the effectiveness of crowdsourced verification platforms and the positive impact of gamification on user engagement and motivation.
- **Our Contribution**: Our system incorporates a community platform where users can share, report, and vote on the authenticity of content. By gamifying this process through point systems and rewards, we aim to increase user participation and enhance

the accuracy of collective verification efforts. Points earned can be used to increase daily usage limits, creating an incentive for active engagement.

Promoting Digital Literacy

Educational Programs and Awareness Campaigns:

- **Existing Work**: Digital literacy initiatives and awareness campaigns have proven essential in educating the public about fake content.
- **Our Contribution**: We integrate a comprehensive library of awareness articles and videos within our application, providing users with valuable resources to understand and identify fake content. Additionally, our system includes real-time notifications to alert users if their personal information is being misused, further enhancing user awareness and security.

Addressing Ethical and Privacy Considerations

Transparency, Accountability, and Privacy Protection:

- **Existing Work**: Ethical guidelines and regulatory frameworks such as GDPR emphasize the importance of transparency, accountability, and data privacy in AI systems.
- **Our Contribution**: We adhere to these principles by implementing transparent AI models and ensuring that users have control over their data. Our system allows users to approve or reject sharing their uploads and provides detailed explanations of verification results. We also employ differential privacy techniques to safeguard user data, ensuring compliance with regulatory standards.

2.5 Summary

The existing body of research on fake content detection and management highlights significant advancements in AI and machine learning, the importance of user engagement, the critical role of digital literacy, and the ethical and privacy considerations involved. These

studies provide a robust foundation for developing effective systems to combat the growing threat of misinformation in the digital age. By synthesizing insights from these diverse areas, this review aims to inform the development of comprehensive and user-friendly solutions for detecting and managing fake content.

Our work draws heavily on the advancements and insights provided by the existing literature while introducing several key innovations. By refining AI models, enhancing user engagement through gamification, promoting digital literacy, and upholding ethical and privacy standards, our system aims to provide a robust, user-friendly, and effective solution for detecting and managing fake content. This integrative approach not only addresses the current challenges but also sets the stage for future advancements in the field.

Chapter 3

Chapter 3: System Analysis

It is necessary to study and analyze the current system to understand the shortcomings and problems of the current system and generate solutions to solve the problems.

3.1 Introduction

In this chapter we provide a detailed knowledge about our system including functional requirements, non-functional requirements, system architecture, use case, and sequence diagrams. We can divide the process of analysis into three parts, which are determining requirements, determining system architecture, and determining development methodology. Section 3.2, Section 3.3, and Section 3.4 will discuss those three parts.

3.2 Analysis of Existing Systems

To provide a comprehensive analysis of existing systems designed to detect and manage fake content, this section incorporates a combination of document review, data collection, user interviews, questionnaires, observations, and data sampling.

Document Review

The document review focuses on examining technical documentation, research papers, and case studies related to prominent fake content detection systems. Key insights include:

1. **Technological Approaches**:
   - Systems like DeepFaceLab and FaceForensics++ utilize advanced deep learning techniques for detecting deepfakes, leveraging CNNs and GANs to identify anomalies in video frames.
   - Text analysis tools, such as those developed by OpenAI's GPT-3, employ NLP for identifying linguistic patterns indicative of fake news.
2. **System Architecture**:

- Many systems adopt modular architectures, integrating various detection modules (e.g., image, video, and text) to provide comprehensive verification.
- Cloud-based solutions are prevalent, offering scalability and real-time processing capabilities.

3. **User Interfaces**:
   - User interfaces vary from simple report-and-verify features to complex dashboards with detailed analytics and visualizations.
   - Accessibility and ease of use are common design priorities, aiming to facilitate user engagement and trust.

Observations

Observational studies of user interactions with existing systems provided practical insights into usability and real-world performance. Key observations include:

1. **User Behavior**:
   - Users tend to prefer systems that offer immediate and clear verification results.
   - Systems that require extensive user input or have complicated workflows see lower engagement levels.
2. **System Performance**:
   - Performance can be significantly impacted by network latency and server load, particularly for cloud-based solutions.
   - Real-time processing capabilities are crucial for maintaining user trust and engagement.

Questionnaires

Questionnaires distributed to a broader user base aimed to quantify satisfaction levels and identify common issues. Key findings include:

1. **Satisfaction Levels**:
   - High satisfaction with systems that are accurate and easy to use.
   - Dissatisfaction often stems from false positives/negatives and limited functionality.

2. **Feature Requests**:
    - Users frequently request features such as more detailed analytics, customizable notifications, and improved integration with social media platforms.
    - Enhanced privacy controls and data security measures are also commonly requested.

3.3 System Requirements

System requirements are the configuration that a system must have in order for a hardware or software application to run smoothly and efficiently. Failure to meet these requirements can result in installation problems or performance problems. The former may prevent a device or application from getting installed, whereas the latter may cause a product to malfunction or perform below expectation or even to hang or crash. System requirements are the needed configurations for the system to operate efficiently. The next three subsections will discuss the functional requirements, Non-functional requirements, and user requirements.

3.3.1 Functional Requirements

In systems engineering, a functional requirement defines a function of a system or its components, where a function is described as a specification of behavior. In this subsection, we list the functions required in our system. We also provide a description for each function.

#	Functional requirement	Description
1	**Sign up**	Allows a user to create an account by providing necessary details such as email, password, etc.

2	**Login**	Allows a user to log in to their account using their credentials.
3	**be a guest**	Allows a user to explore the application without creating an account.
4	**access articles and videos**	Provides access to informational content such as articles and videos about fake content and awareness.
5	**check photo**	Allows a user to upload media to verify its authenticity using AI.
6	**visit community**	Allows a user to browse through community posts.
7	**vote**	Allows users to vote for photos if they are real or fake
8	**share uploads**	Allows users to manage the privacy settings of their uploads, choosing whether to share them on his account or not.
9	**visit users accounts**	Allows a user to view the uploads of other users who have given permission.
10	**rate and give feedback**	Enables users to provide feedback and rate the accuracy of the verification results.
11	**share in community**	Allows user to share posts including text and images .
12	**manage account**	Provides users with tools to manage their account settings and uploaded media.
13	**deactivate/delete account**	Allows users to deactivate or permanently delete their account from the application.

Table 3.1

3.3.2 Non-Functional Requirements

The system needs interaction with web dashboard so software components can operate efficiently and meet the requirements. Any failure of the components of the systems may lead

to one or more functional functions stopping or be misused. A non- functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system Table

#	Non-Functional requirement	Description
1	**Performance**	The system should be fast load time shouldn't be more than 3 sec
2	**Security**	The system should be secured specially the payments process and the users data
3	**Capacity**	Capacity of system database should be huge as we target thousand of clients
4	**Data integrity**	The system shall maintain data integrity by keeping backups of all updates to the database for every record transaction
5	**Usability**	The system's interface has to be user-friendly and easy to use
6	**Scalability**	Ability of project to be able to adapt to larger demand by allowing greater supply.

Table 3.2

3.3.3 Design Objectives

The design objectives for our system aim to address the challenges and limitations identified in existing fake content detection systems while introducing innovative features to enhance accuracy, user engagement, and overall effectiveness. These objectives are structured around key areas: accuracy and performance, user experience, community collaboration, digital literacy, privacy and security, and scalability.

1. Accuracy and Performance

Objective: Develop a highly accurate and efficient fake content detection system capable of real-time analysis.

- **Advanced AI and Machine Learning Models**: Utilize state-of-the-art deep learning algorithms, including improved versions of GANs, CNNs, and NLP models, to enhance detection accuracy for images, videos, and text.
- **Contextual Analysis**: Implement mechanisms for categorizing content to improve contextual understanding and detection accuracy.
- **Continuous Learning**: Integrate a feedback loop for continuous model training and improvement based on user interactions and new data.

2. User Experience

Objective: Create an intuitive and user-friendly interface that facilitates easy navigation and interaction.

- **Simplified User Interface**: Design a clean, intuitive interface with clear instructions and real-time feedback on content verification results.
- **Accessibility**: Ensure the system is accessible to users of all technical skill levels, including features like tooltips, tutorials, and FAQs.
- **Immediate Results**: Provide real-time verification results to maintain user engagement and trust.

3. Community Collaboration

Objective: Foster a collaborative environment where users can participate in the verification process and contribute to content authenticity.

- **Community Platform**: Develop a platform where users can report, discuss, and vote on the authenticity of content.
- **Gamification**: Implement gamification elements, such as points, badges, and leaderboards, to motivate user participation and reward accurate verification efforts.
- **Peer Reviews**: Enable users to provide feedback and reviews on the verification results, contributing to the collective intelligence of the community.

4. Digital Literacy

Objective: Promote digital literacy by educating users on identifying and understanding fake content.

- **Educational Resources**: Include a comprehensive library of articles, videos, and tutorials on fake content detection and digital literacy.
- **Awareness Campaigns**: Implement awareness campaigns within the platform to inform users about the latest trends and techniques in fake content creation and detection.
- **Interactive Learning**: Provide interactive tools and quizzes to test users' knowledge and improve their content evaluation skills.

5. Privacy and Security

Objective: Ensure user data privacy and security while maintaining transparency in detection processes.

- **Data Protection**: Implement robust security measures, such as encryption and anonymization, to protect user data.
- **User Control**: Allow users to manage their privacy settings, including options to approve or reject sharing their uploads.
- **Transparency**: Provide clear explanations of detection results and the processes behind them to build user trust.

6. Scalability

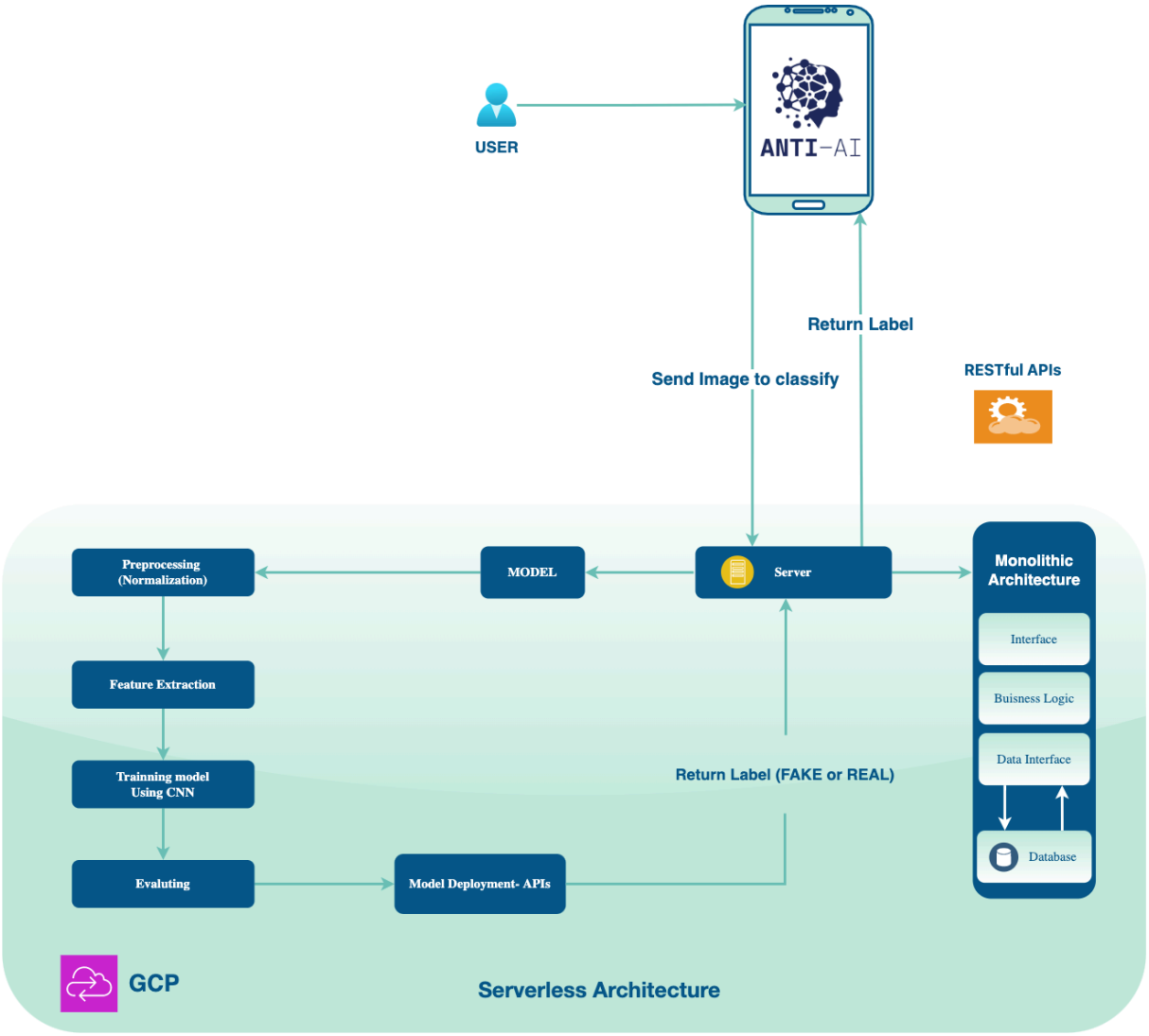
Objective: Design a system that can scale to accommodate a growing user base and increasing data volumes.

1. **Cloud-Based Architecture**: Utilize a cloud-based architecture to ensure scalability and flexibility in handling large volumes of data and user interactions.
2. **Performance Optimization**: Optimize algorithms and system performance to handle real-time processing and minimize latency.

3. **Modular Design**: Employ a modular system design that allows for easy integration of new features and updates without disrupting existing functionalities.

3.4 System Architecture

After determining the requirements for a system, we will describe its main components, their relationships (structures) and how they interact with each other. Figure shows the structure of the queue with the main components and the connections between them.



3.5 Development Methodology

After we knew the basic structure of Anti- AI project. We are going to view some of its functions, the relation between them, and the sequence of their executions in the following subsections.

3.5.1 Use Case Diagrams

First, with a use case diagram, we will define the expected behavior (what) of the system, not the exact way to achieve it (how). This helps us design the system from the end user's perspective. Figure shows a use case diagram where the boundaries of the system are the organization and the actors are the user, the admin and the service admin.

Components of Use Case

- **Actor :**

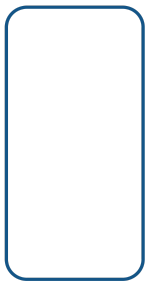
  It what is interacts with a use case

- **Use case :**

  It is a visual representation of a distinct business Functionality in a system

- **System boundary :**

  It defines the scope of what a system will Be



Relationships in Use Case

- **Include :**

  When a use case is depicted as using the functionality of another use case in a diagram

- **Extend :**

  The child use case adds to the existing functionality and characteristics of the parent use case

- **Generlization :**

  It is also a parent-child relationship between Use cases

- **Association :**

  Link between an actor and the use case

3.5.2 Use Case Description (Detailed Use Cases)

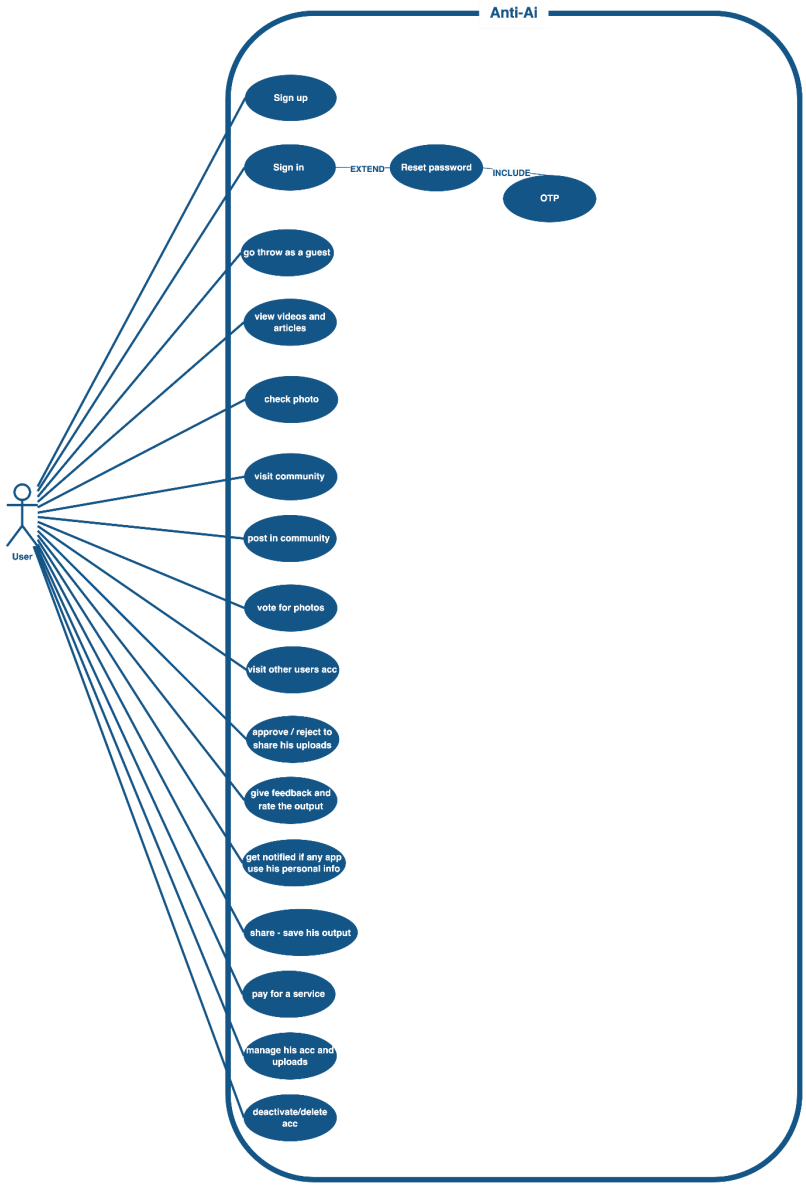


Fig 3.1

3.5.3 Sequence Diagram

After explaining system features and actors, we are going to dig deeper and see how they are done. The sequence diagrams will show you how an operation is done and the inner details of it. Sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are sometimes called event diagrams or event scenarios.

- **Sequence Diagrams Notation**

1. **Actor:**

Represents a type of role where it interacts with the system and its objects

2. **Lifelines:**

It is a named element which depicts an individual participant in a sequence diagram

3. Messages:

Communication between objects is Depicted using messages.

- Request :

- Response :

Sequence diagrams are used primarily to show the interactions between objects in the sequential order that those interactions occur. Much like the class diagram, developers typically think sequence diagrams were meant exclusively for them. However, an organization's business staff can find sequence diagrams useful to communicate how the business currently works by showing how various business objects interact.

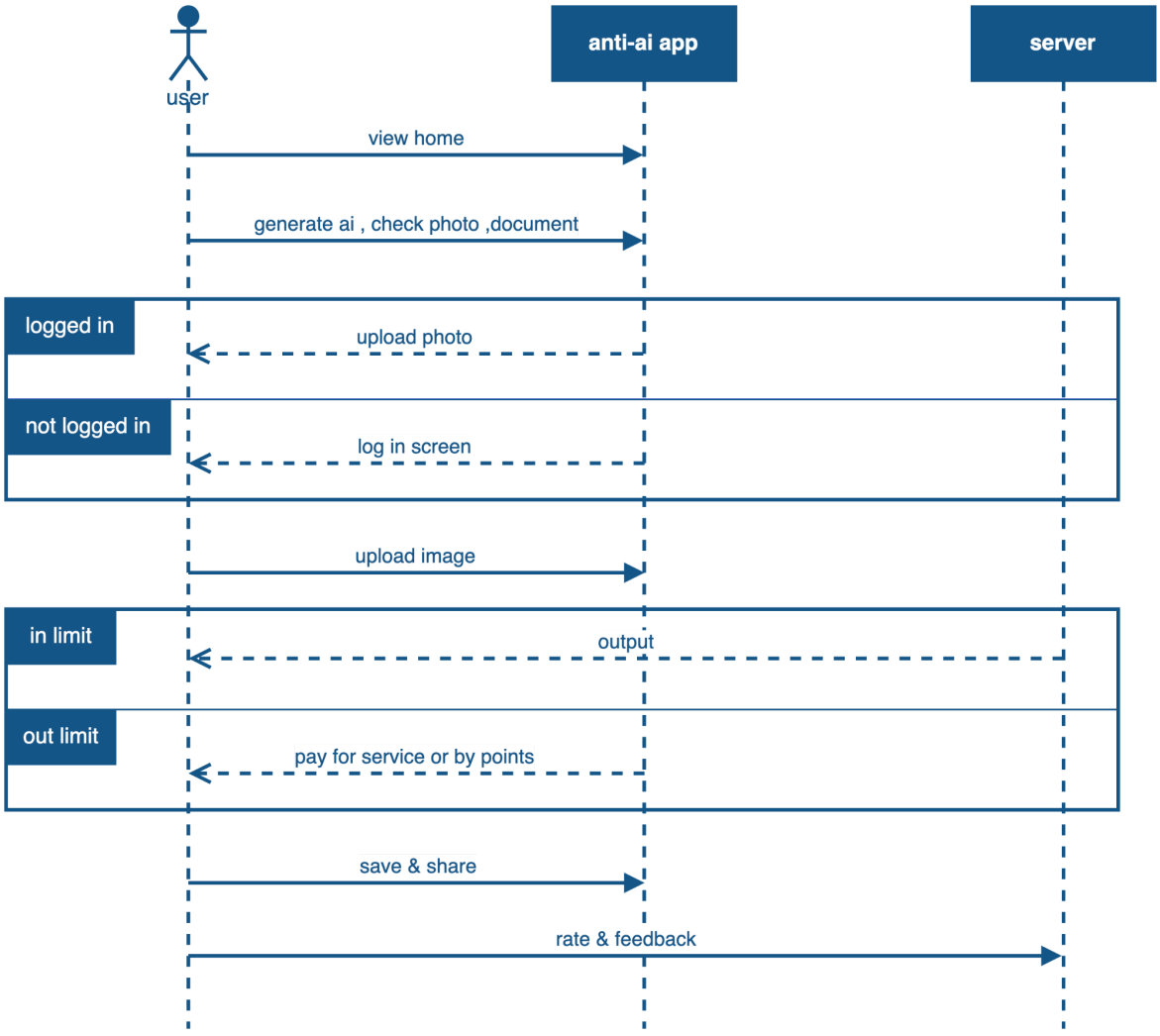


Fig 3.2

3.6 Tools and Languages

In addition , our project includes a set of instructions, data or programs used to operate computers and execute specific tasks. It is the opposite of hardware, which describes the physical aspects of a computer . Software is a generic term used to refer to applications , scripts and programs that run on a device. It can be thought of as the variable part of a computer, while hardware is the invariable part.

1. **UI Design**

User interface (UI) design is the process designers use to build interfaces in software or computerized devices, focusing on looks or style. Aims to create interfaces which users find easy to use and pleasurable. UI design refers to graphical user interfaces .

2. **Flutter**

Flutter is an open source framework by Google for building beautiful, natively compiled, multi- platform applications from a single codebase.

3. **Dart**

Dart is a programming language designed for client development, such as for the web and mobile apps. It is developed by Google and can also be used to build server and desktop applications. It is an object-oriented, class-based, garbage-collected language with C-style syntax.

4. **Back-end**

Working on server-side software, which focuses on everything you can't see on a website. Back- end developers ensure the website performs correctly, focusing on databases, back-end logic, application programming interface (APIs), architecture, and servers.

5. **Django**

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

6. **SQL**

Structured Query Language, abbreviated as SQL, is a domain-specific language used in programming and designed for managing data held in a relational database management system, or for stream processing in a relational data stream management system.

3.7 Summary

In this chapter, we have provided critical insights into the strengths and weaknesses of existing fake content detection systems. By synthesizing findings from document reviews, questionnaires, and observations, this chapter informs the design objectives and strategies for our proposed system.rom functional and non-functional requirements to system architecture, we provide UML diagrams that detail how the system works and ultimately, we have included a comprehensive list of tools that we will use and will help us to design and implement the system. Our approach aims to address current limitations and leverage best practices to create a comprehensive, accurate, and user-friendly solution for detecting and managing fake content.

Chapter 4

Chapter 4: System Design

The design of the investigation explains how the study was formulated to investigate each question or hypothesis and if appropriate, it identifies all variables and how they are manipulated.

4.1 Introduction

In the previous chapters we focused on the system requirements and on the system architecture, so after that in this chapter we will discuss the process of defining the architecture and product design of our application by applying a detailed description of the actual design parts of the system which consists of the design process, the models used for training, and the user interfaces design to enable the user to benefit from the application features.

4.2 ERD Diagram

An entity-relation diagram (ERD) shows the relationships of entity sets stored in your project. An entity in this context is an object, a component of data. An entity-relationship diagram (ERD) shows the relationships of entity sets stored in your project. An entity set is a collection of similar entities. These entities can have attributes that define their properties by defining the entities, and their attributes, and showing the relationships between them an ER diagram illustrates the logical structure of the project. ER diagrams are used to sketch out the design of a code.

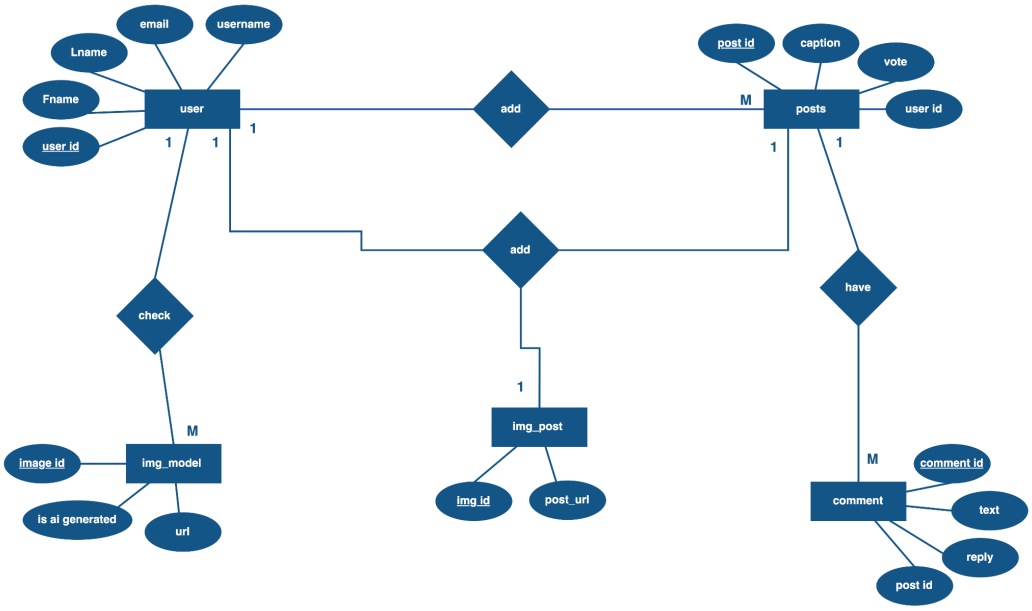


Fig 4.1

4.3 Class Diagram

A class diagram describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. This figure shows the class diagram that includes class representation of all system components.

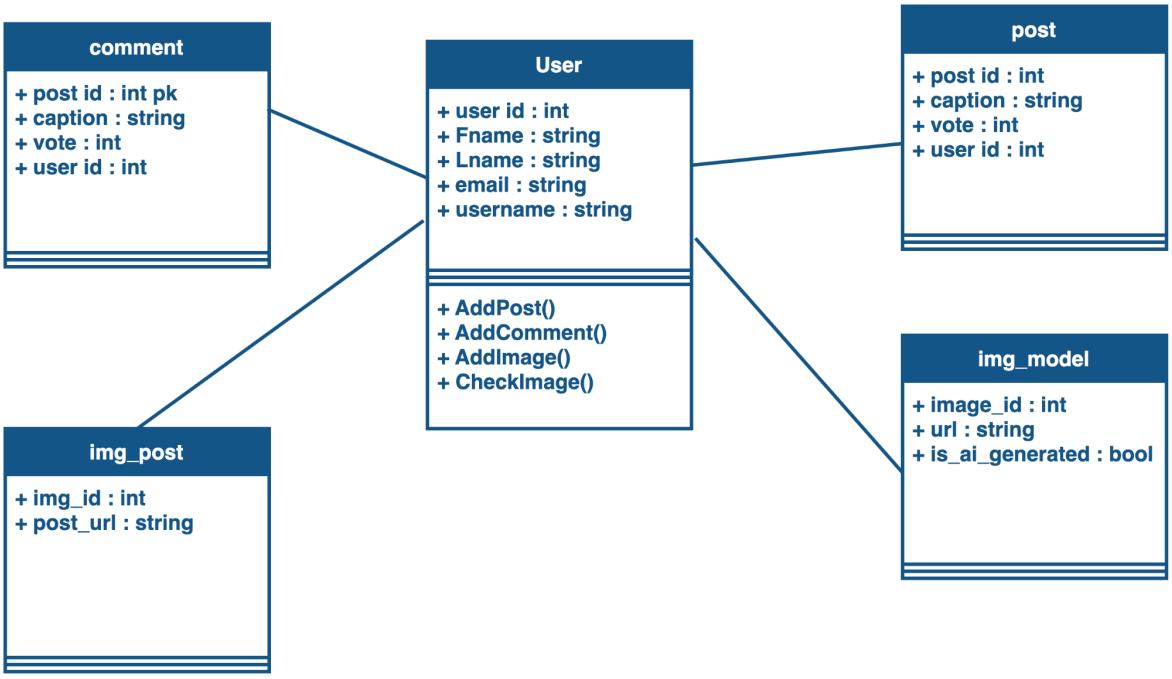


Fig 4.2

4.4 Database Design of Algorithms

Users

user_id	username	email	created_at	created_at	updated_at
1	user1	user2@example.com	hashed_password_1	2024-01-01 12:00:00	2024-01-01 12:00:00
2	user2	user2@example.com	hashed_password_2	2024-01-01 13:00:00	2024-01-01 13:00:00

Table 4.1

Images

image_id	user_id	image_url	upload_date	detection_status	created_at	updated_at
1	1	https://example.com/img1	2024-01-01 12:00:00	Processed	2024-01-01 12:00:00	2024-01-01 12:00:00
2	2	https://example.com/img2	2024-01-01 13:00:00	Pending	2024-01-01 13:00:00	2024-01-01 13:00:00

Table 4.2

Detection_Results

result_id	image_id	is_ai_generated	confidence_score	detection_date	created_at	updated_at
1	1	TRUE	0.98	2024-01-01 12:00:00	2024-01-01 12:00:00	2024-01-01 12:00:00

Table 4.3

Feedback

feedback_id	user_id	image_id	feedback_text	rating	created_at	updated_at
1	1	1	Great detection tool!	5	2024-01-01 12:00:00	2024-01-01 12:00:00

Table 4.4

Admins

admin_id	username	email	created_at	created_at	updated_at
1	admin1	admin1@example.com	hashed_password_1	2024-01-01 12:00:00	2024-01-01 12:00:00

Table 4.5

4.5 Interface Design

4.5.1 Splash screen

The logo appears when you start using the application, we used Adobe illustrator in this design.

Fig 4.3

4.5.2 On boarding

When the user opens the application, these pages appear showing the most important features of the application.

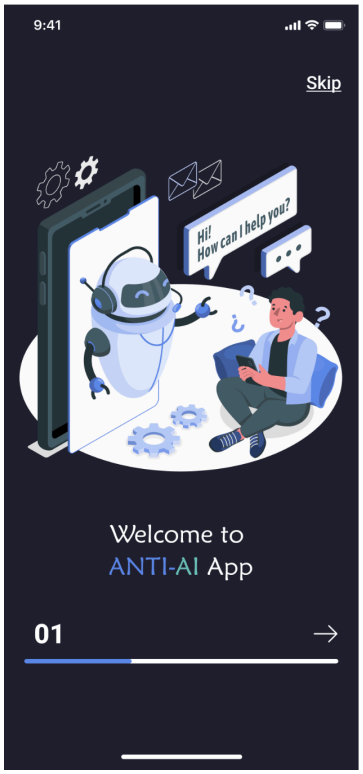


Fig 4.4

Fig 4.5

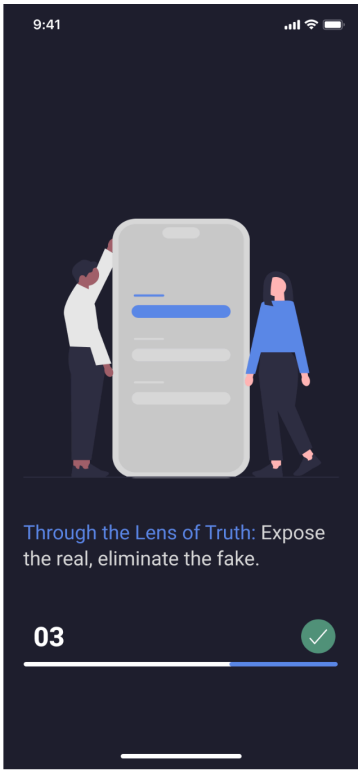


Fig 4.6

4.5.3 Login

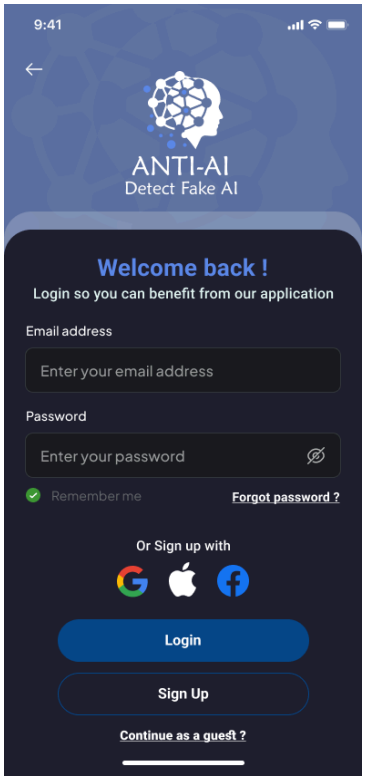


Fig 4.7

4.5.4 Create account

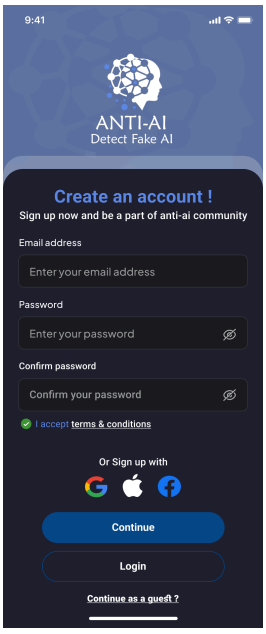


Fig 4.8

4.5.5 Sign up verification

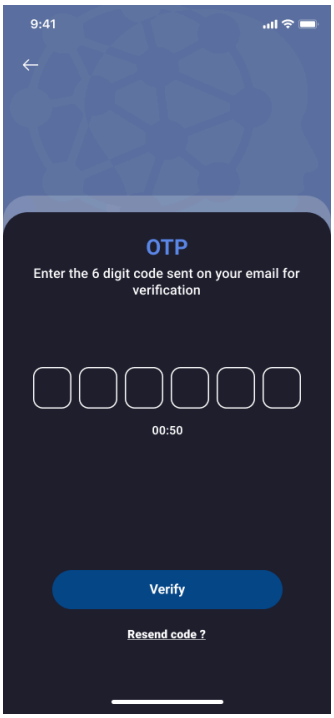


Fig 4.9

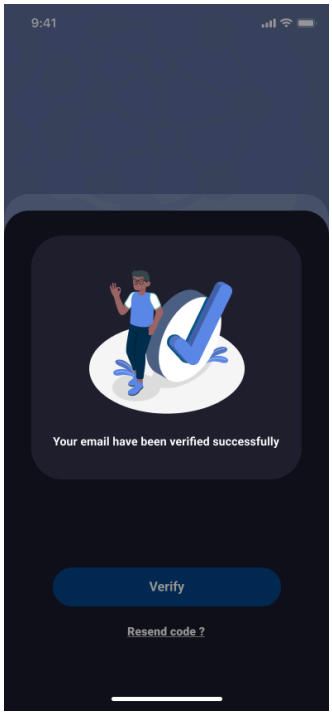


Fig 4.10

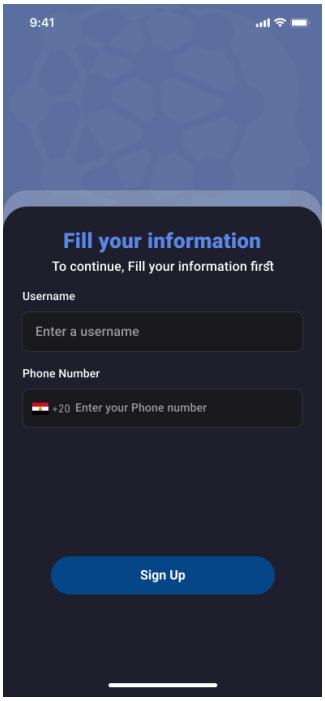


Fig 4.11

4.5.6 Forgot password

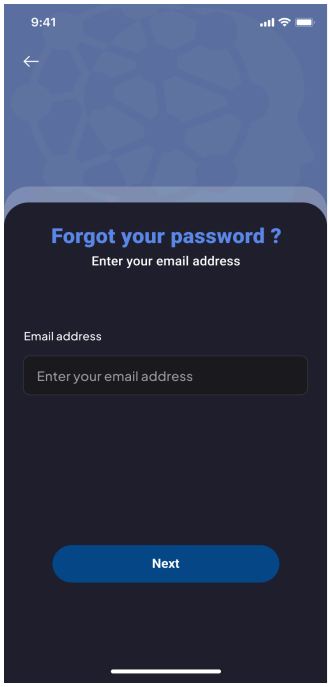


Fig 4.12

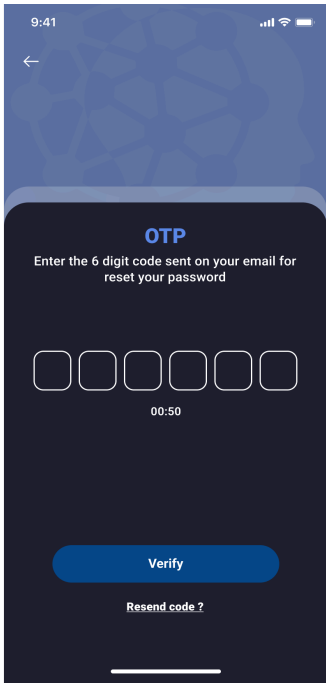


Fig 4.13

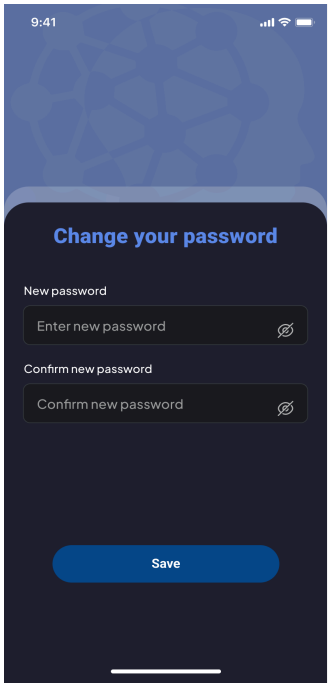


Fig 4.14

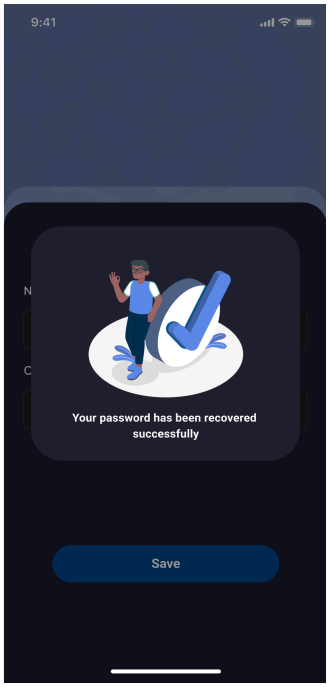


Fig 4.15

4.5.7 Home

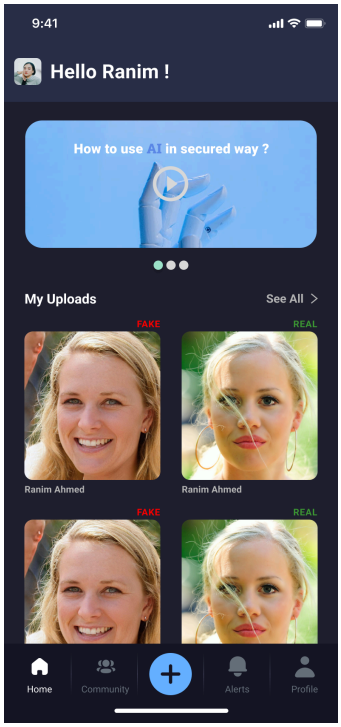


Fig 4.16

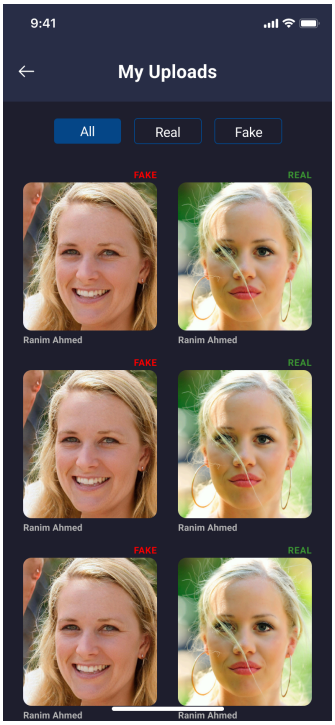


Fig 4.17

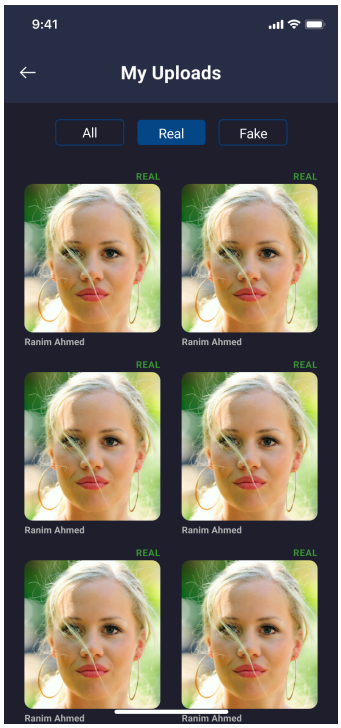


Fig 4.18

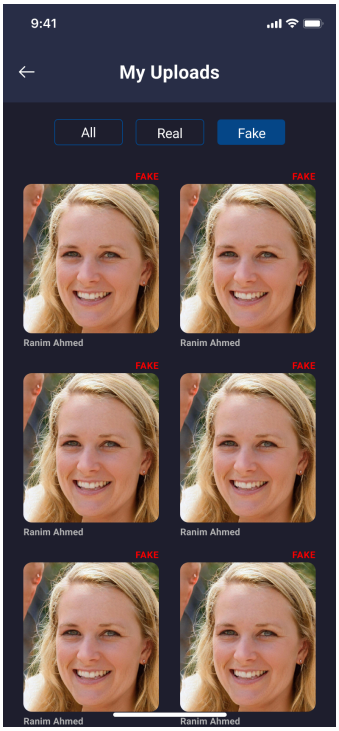


Fig 4.19

4.5.8 Home - upload button

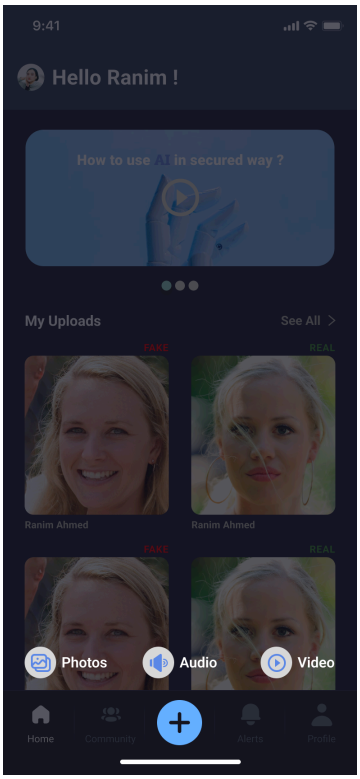


Fig 4.20

4.5.9 Detect fake photos

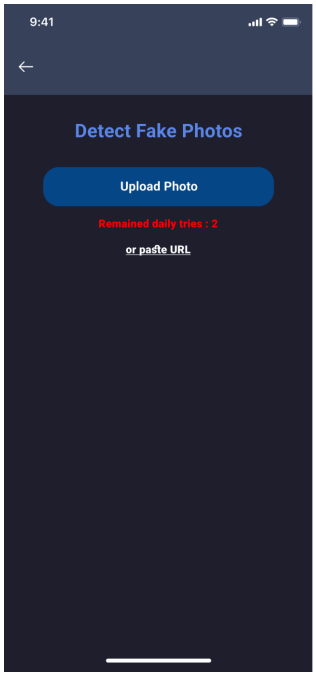


Fig 4.21

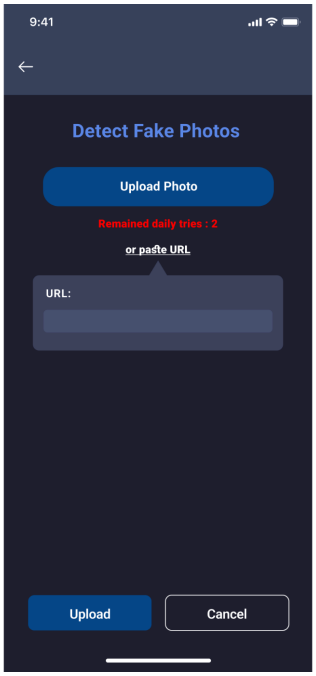


Fig 4.22

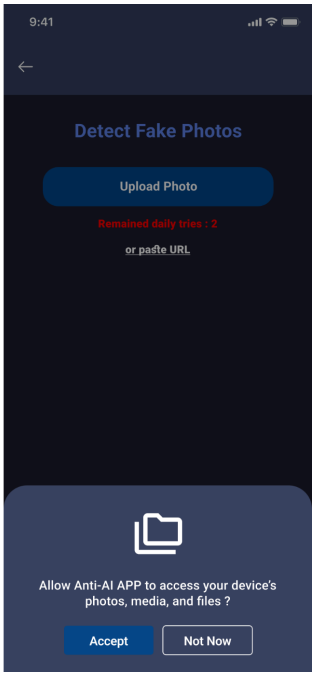


Fig 4.23

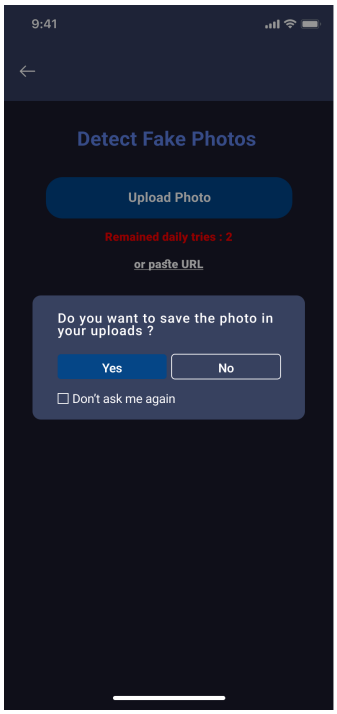


Fig 4.24

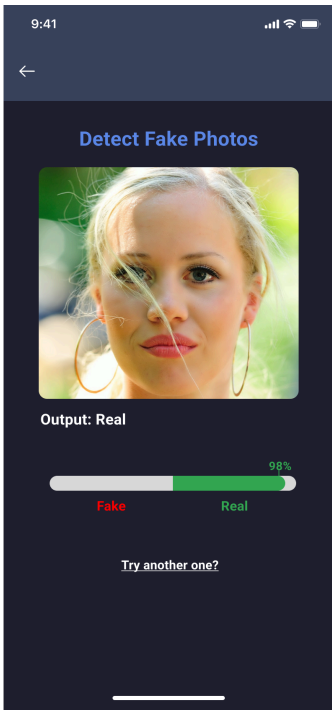


Fig 4.25

4.5.10 Payment

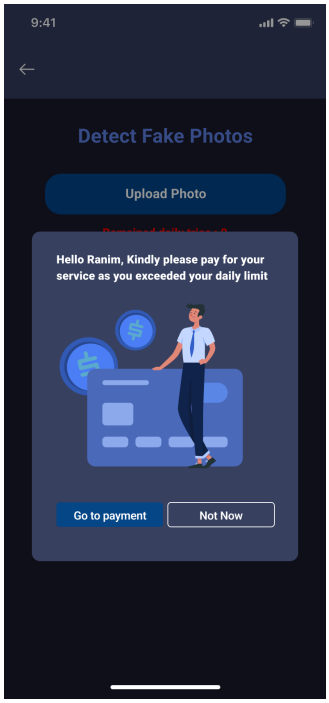


Fig 4.26

4.5.11 Community - Timeline

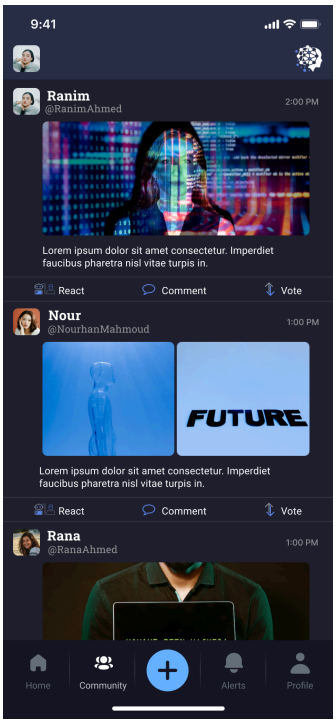
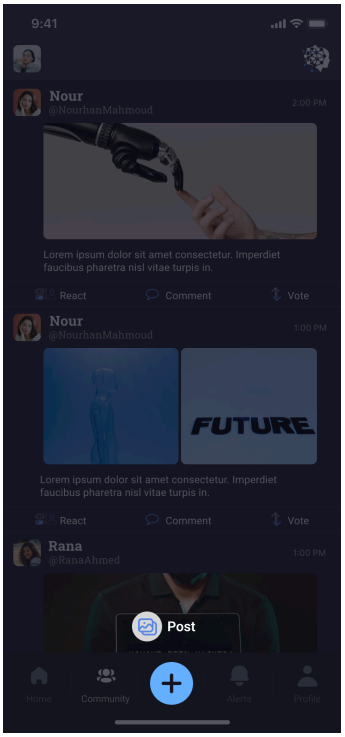


Fig 4.27

4.5.12 Community - Upload button



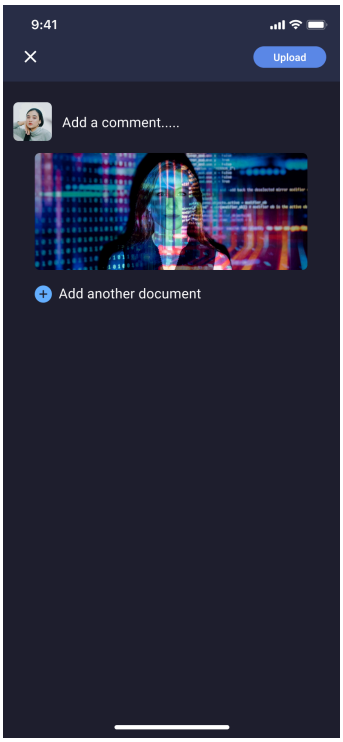
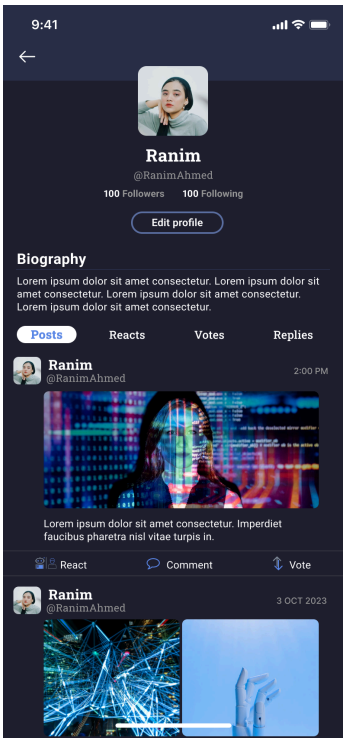


Fig 4.28 Fig 4.29

4.5.13 Community - Profile (Posts)



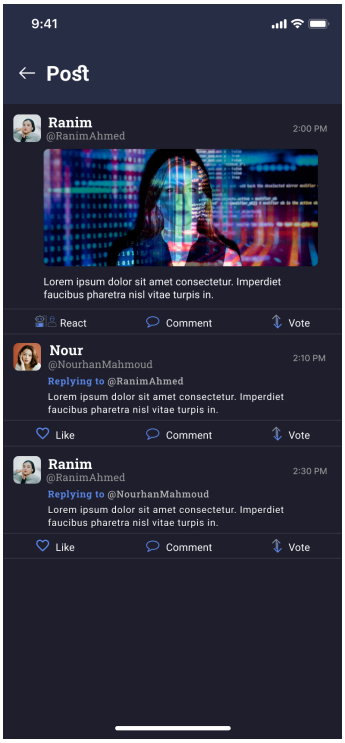


Fig 4.30 Fig 4.31

4.5.14 Community - Profile (Reacts)

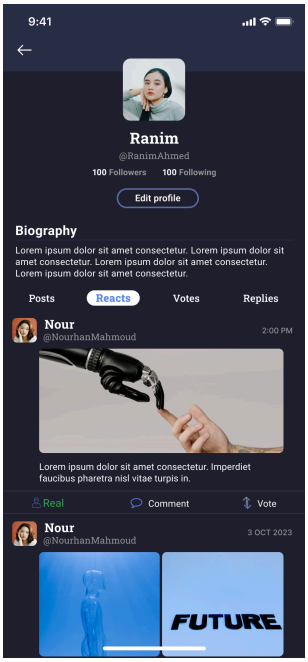


Fig 4.32

4.5.15 Community - Profile (Votes)

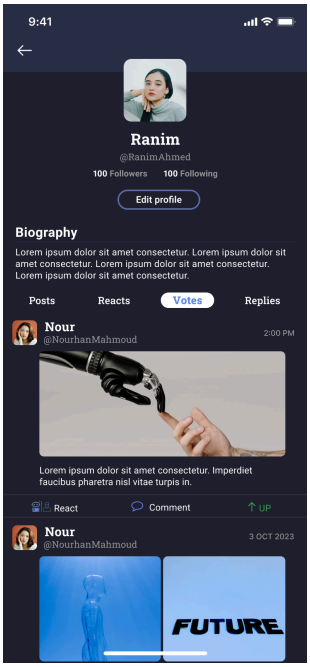


Fig 4.33

4.5.16 Community - Profile (Replies)

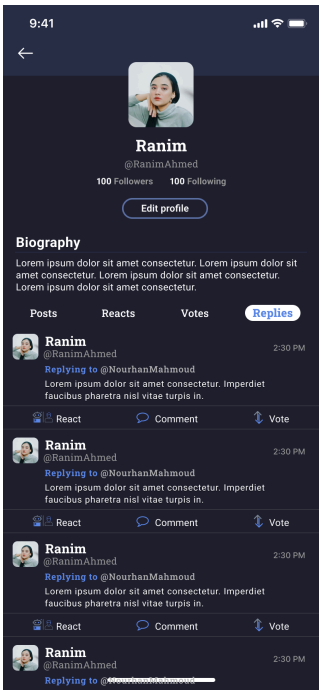


Fig 4.34

4.5.17 Community - Edit Profile

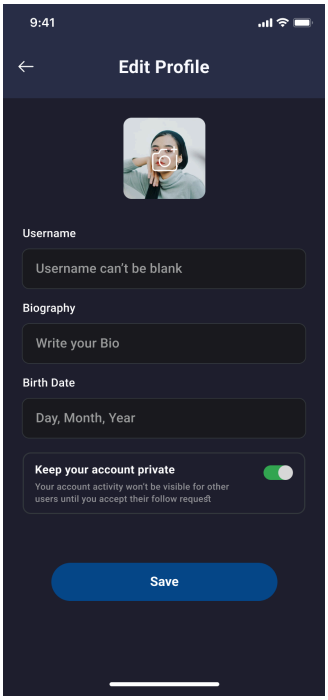


Fig 4.35

4.5.18 Community - Another user's profile

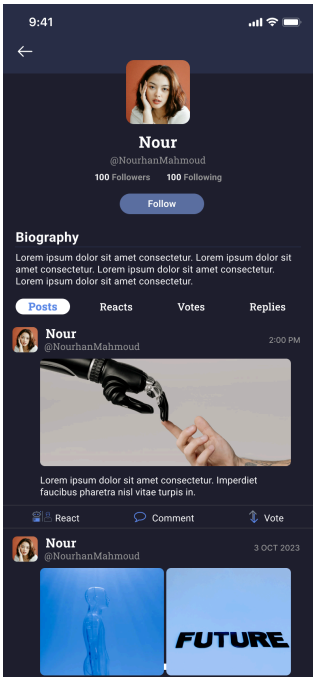


Fig 4.36

4.5.19 Notifications

Fig 4.37

4.5.20 Follow requests

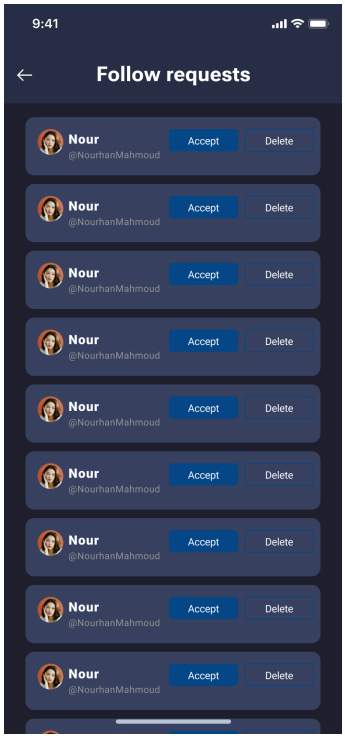


Fig 4.38

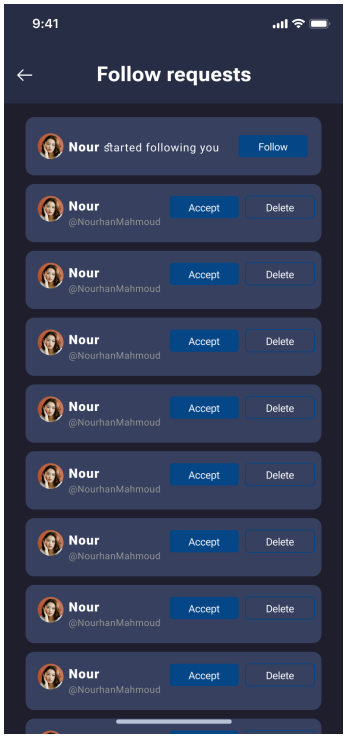


Fig 4.39

4.5.21 Profile

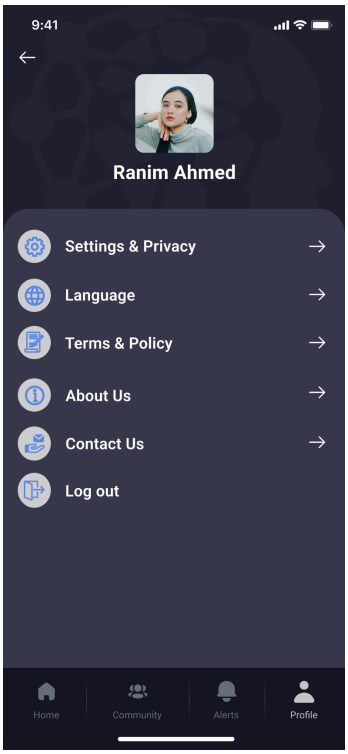


Fig 4.40

4.5.22 Profile - Settings & privacy

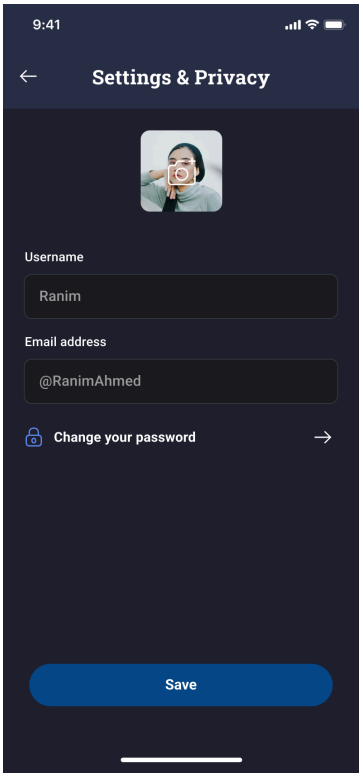


Fig 4.41

4.5.23 Change your password

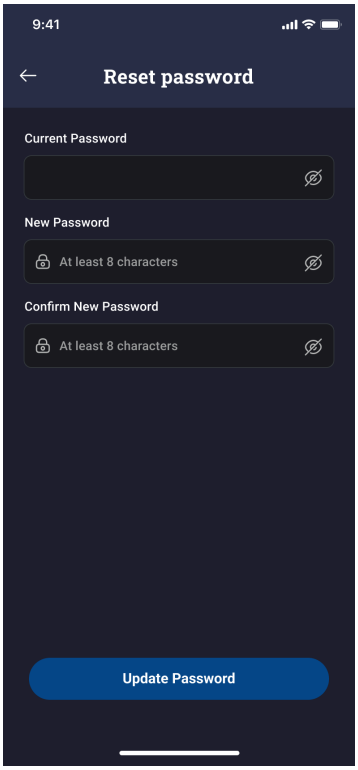


Fig 4.42

4.5.24 Profile - Language

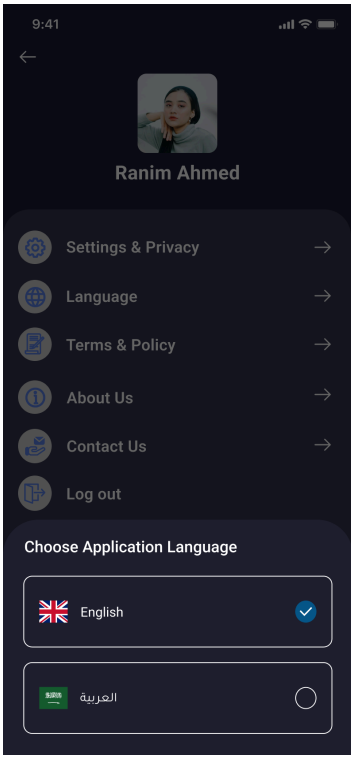


Fig 4.43

4.5.25 Profile - Terms & policy

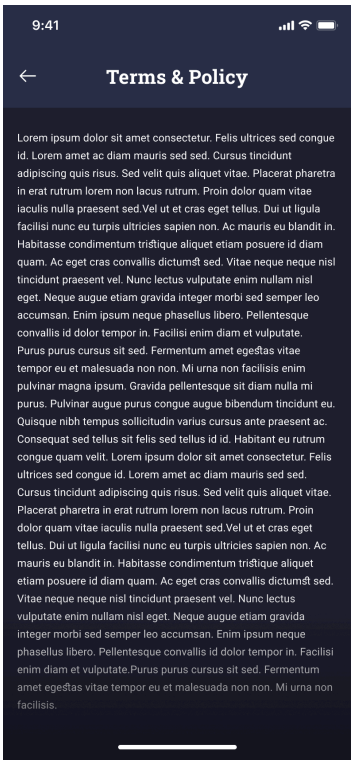


Fig 4.44

4.5.26 Profile - About Us

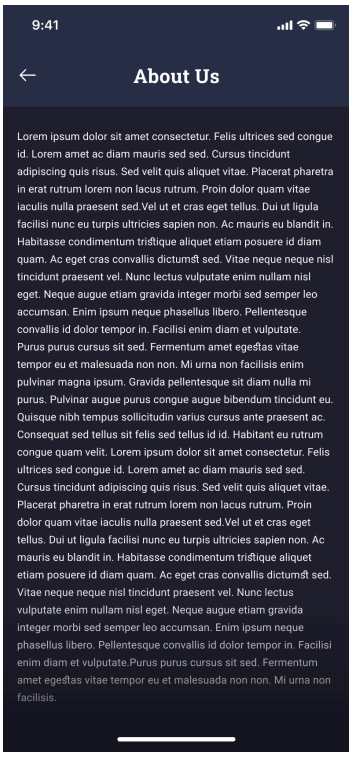


Fig 4.45

4.5.27 Profile - Contact Us

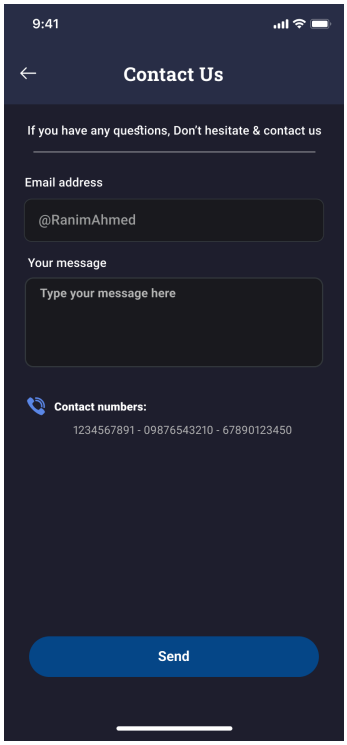


Fig 4.46

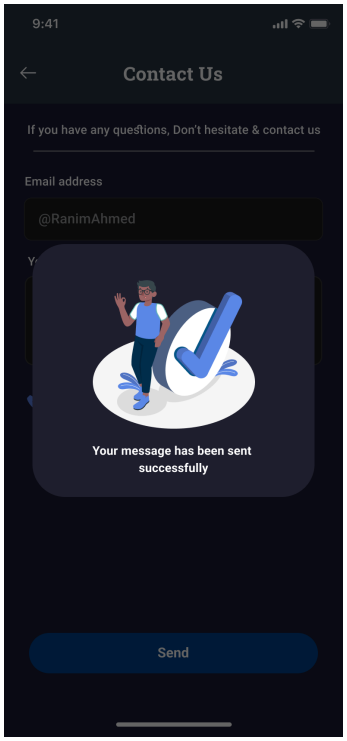


Fig 4.47

4.5.28 Profile - Logout

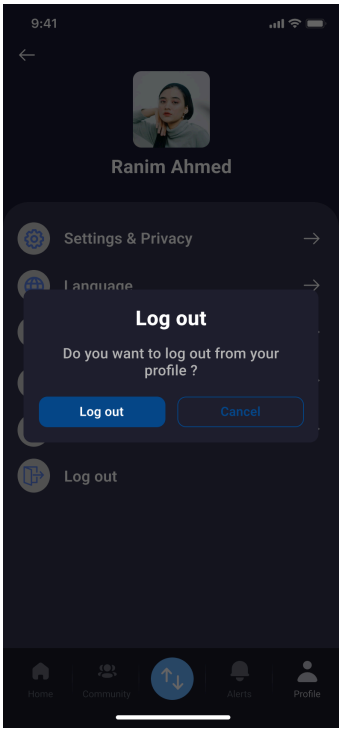


Fig 4.48

4.6 Summary

The system design chapter outlines a strategic and detailed approach to creating an effective fake content detection system. By integrating advanced AI technologies, fostering user engagement through an intuitive interface and community collaboration, promoting digital literacy, ensuring robust privacy and security measures, and designing for scalability, the proposed system aims to address the multifaceted challenges of fake content in the digital age. This holistic design approach not only meets current needs but also positions the system to adapt and evolve with future technological and societal developments.

Chapter 5

Chapter 5: System Implementation

5.1 Introduction

In previous chapters, we have demonstrated the potential requirements and limitations of our system. In this chapter, we provide an implementation tools, and provide a samples of the most important codes we used to build our prototype and application

5.2 Mapping Design to Implementation

1. Define Requirements and Scope

- **Objective**: Develop a mobile application and website to detect AI-generated images, aimed at combating digital misinformation.
- **Functional Requirements**: User authentication, image upload, AI detection processing, feedback collection, admin functionalities.
- **Non-Functional Requirements**: Security (authentication, data encryption), Performance (fast image processing), Scalability (handle multiple users), Usability (intuitive UI/UX).

2. Choose Technology Stack

Backend Technologies

- **Programming Languages**: Python (for AI models and backend logic).
- **Frameworks/Libraries**: Django (for API development).
- **Database**: PostgreSQL (relational database for structured data).
- **AI Model**: TensorFlow or PyTorch (for AI model integration).
- **Deployment**: Docker, Kubernetes (for containerization and deployment).

Frontend Technologies

- **Web**: React.js (for responsive web design).
- **Mobile**: React Native (for cross-platform mobile development).
- **UI Framework**: Material-UI (for consistent design).
- **Authentication**: JWT (JSON Web Tokens) for secure user authentication.

3. Database Design and Implementation

- **Database Schema**: Design tables for Users, Images, Detection_Results, Feedback, Admin_Logs (refer to detailed schema in previous database design section).
- **Implementation**: Create SQL scripts to define tables, relationships, and constraints in PostgreSQL.

4. Backend Development

- **API Development**: Implement RESTful APIs using Flask/Django for user registration/login, image upload, AI model integration, feedback collection, admin functionalities.
- **Security**: Implement JWT-based authentication, role-based access control (RBAC), HTTPS for secure data transmission.

5. Frontend Development

- **UI/UX Design**: Design responsive UI/UX for both web and mobile platforms using React.js and React Native.
- **User Features**: Develop components for user registration/login, image upload, display detection results, feedback submission, and admin dashboard.
- **Integration**: Integrate frontend components with backend APIs for seamless data flow and user interaction.

6. AI Model Integration

- **Select AI Model**: Choose TensorFlow or PyTorch or Transformer models for detecting AI-generated images.
- **Integration**: Integrate AI models into backend APIs to process uploaded images, analyze for AI-generated content, and store results in the database.

7. Testing and Quality Assurance

- **Unit Testing**: Test backend APIs and frontend components using testing frameworks like pytest (backend) and Jest (frontend).
- **Integration Testing**: Validate end-to-end functionality, API integrations, and data flow between frontend and backend.
- **User Acceptance Testing (UAT)**: Conduct testing with real users to gather feedback and ensure usability.

8. Deployment and Maintenance

- **Deployment**: Deploy application on cloud platforms (AWS, Azure) using Docker/Kubernetes for scalability and ease of maintenance.
- **Monitoring**: Set up monitoring tools (e.g., Prometheus, Grafana) to monitor application performance, server health, and user activity.
- **Maintenance**: Regularly update application, apply security patches, optimize performance based on usage metrics and user feedback.

9. Documentation and Training

- **Documentation**: Create comprehensive documentation covering architecture, API endpoints, database schema, deployment procedures, and maintenance guidelines.
- **User Training**: Provide user guides and tutorials to onboard users, navigate the application, and utilize its features effectively.

10. Continuous Improvement

- **Feedback Loop**: Gather user feedback, analytics, and performance metrics to prioritize enhancements and new features.
- **Technology Updates**: Stay updated with AI advancements, security best practices, and technology trends to enhance application capabilities and maintain relevance.

5.3 Application Codes

Machine learning code (Model)

```
In [31]: !pip install datasets

Requirement already satisfied: datasets in /opt/conda/lib/python3.10/site-packages (2.18.0)
Requirement already satisfied: filelock in /opt/conda/lib/python3.10/site-packages (from datasets) (3.13.1)
Requirement already satisfied: numpy>=1.17 in /opt/conda/lib/python3.10/site-packages (from datasets) (1.26.4)
Requirement already satisfied: pyarrow>=12.0.0 in /opt/conda/lib/python3.10/site-packages (from datasets) (15.0.2)
Requirement already satisfied: pyarrow-hotfix in /opt/conda/lib/python3.10/site-packages (from datasets) (0.6)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /opt/conda/lib/python3.10/site-packages (from datasets) (0.3.8)
Requirement already satisfied: pandas in /opt/conda/lib/python3.10/site-packages (from datasets) (2.1.4)
Requirement already satisfied: requests>=2.19.0 in /opt/conda/lib/python3.10/site-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in /opt/conda/lib/python3.10/site-packages (from datasets) (4.66.1)
Requirement already satisfied: xxhash in /opt/conda/lib/python3.10/site-packages (from datasets) (3.4.1)
Requirement already satisfied: multiprocess in /opt/conda/lib/python3.10/site-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2024.2.0,>=2023.1.0 in /opt/conda/lib/python3.10/site-packages (from fsspec[http]<=2
024.2.0,>=2023.1.0->datasets) (2024.2.0)
Requirement already satisfied: aiohttp in /opt/conda/lib/python3.10/site-packages (from datasets) (3.9.1)
Requirement already satisfied: huggingface-hub>=0.19.4 in /opt/conda/lib/python3.10/site-packages (from datasets) (0.22.2)
Requirement already satisfied: packaging in /opt/conda/lib/python3.10/site-packages (from datasets) (21.3)
Requirement already satisfied: pyyaml>=5.1 in /opt/conda/lib/python3.10/site-packages (from datasets) (6.0.1)
Requirement already satisfied: attrs>=17.3.0 in /opt/conda/lib/python3.10/site-packages (from aiohttp->datasets) (23.2.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /opt/conda/lib/python3.10/site-packages (from aiohttp->datasets) (6.
0.4)
Requirement already satisfied: yarl<2.0,>=1.0 in /opt/conda/lib/python3.10/site-packages (from aiohttp->datasets) (1.9.3)
Requirement already satisfied: frozenlist>=1.1.1 in /opt/conda/lib/python3.10/site-packages (from aiohttp->datasets) (1.4.
1)
Requirement already satisfied: aiosignal>=1.1.2 in /opt/conda/lib/python3.10/site-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /opt/conda/lib/python3.10/site-packages (from aiohttp->datasets)
(4.0.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /opt/conda/lib/python3.10/site-packages (from huggingface-hub>
=0.19.4->datasets) (4.9.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /opt/conda/lib/python3.10/site-packages (from packaging->dataset
s) (3.1.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/python3.10/site-packages (from requests>=2.19.0->
datasets) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-packages (from requests>=2.19.0->datasets)
(3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/conda/lib/python3.10/site-packages (from requests>=2.19.0->datase
ts) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.10/site-packages (from requests>=2.19.0->datase
ts) (2024.2.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/lib/python3.10/site-packages (from pandas->datasets)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.10/site-packages (from pandas->datasets) (2023.3.post
1)
Requirement already satisfied: tzdata>=2022.1 in /opt/conda/lib/python3.10/site-packages (from pandas->datasets) (2023.4)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-packages (from python-dateutil>=2.8.2->pandas->da
tasets) (1.16.0)
```

Fig 5.1

Step: Install the datasets library

- **Command**: !pip install datasets
- **Description**: This command installs the datasets library, which provides a collection of ready-to-use datasets and evaluation metrics for Natural Language Processing (NLP) tasks. The exclamation mark (!) allows shell commands to be run from a Jupyter notebook.

Output:

- **Message**: Requirement already satisfied
- **Explanation**: Indicates that the datasets library is already installed, along with its dependencies (numpy, pandas, requests, tqdm, aiohttp, pyarrow, and huggingface-hub).

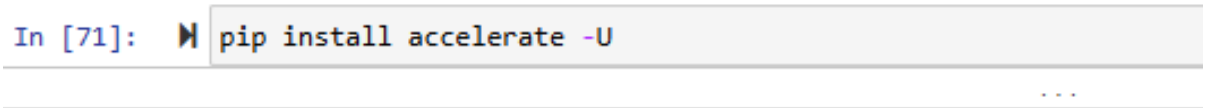


Fig 5.2

Step: Install or Upgrade the accelerate library

- **Command**: !pip install accelerate -U
- **Description**: This command installs the accelerate library or upgrades it to the latest version if it is already installed. The -U flag stands for "upgrade" and ensures that the latest version of the package is installed. The accelerate library is used to streamline the training of machine learning models, especially in distributed and multi-GPU setups.

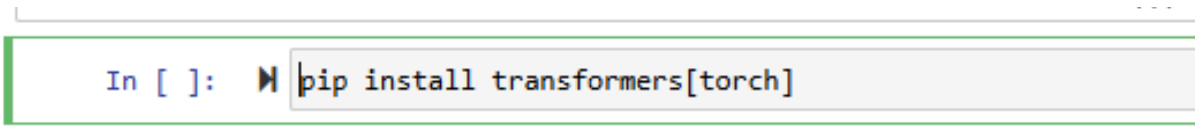


Fig 5.3

Step: Install the transformers library with PyTorch support

- **Command**: !pip install transformers[torch]
- **Description**: This command installs the transformers library along with its PyTorch dependencies. The transformers library, developed by Hugging Face, provides state-of-the-art machine learning models for natural language processing tasks. The [torch] part specifies that PyTorch-related dependencies should also be installed, ensuring compatibility with PyTorch-based models and functionalities.

Hemg/AI-Generated-vs-Real-Images-Datasets

```
In [61]: from datasets import load_dataset

         ds = load_dataset("Hemg/AI-Generated-vs-Real-Images-Datasets")
```

Fig 5.4

Step: Import the load_dataset function

- **Code**: from datasets import load_dataset
- **Description**: This line imports the load_dataset function from the datasets library. This function is used to load various datasets available in the Hugging Face Hub.

Step: Load the "AI-Generated-vs-Real-Images-Datasets"

- **Code**: ds = load_dataset("Hemg/AI-Generated-vs-Real-Images-Datasets")
- **Description**: This line loads the dataset named "AI-Generated-vs-Real-Images-Datasets" from the Hugging Face Hub into the variable ds. The dataset, identified by the string "Hemg/AI-Generated-vs-Real-Images-Datasets", contains images that are either AI-generated or real, useful for tasks like image classification or analysis.

```
In [62]: ds

Out[62]: DatasetDict({
             train: Dataset({
                 features: ['image', 'label'],
                 num_rows: 152710
             })
         })
```

Fig 5.5

1. **Step**: View the loaded dataset
    - **Code**: ds

- **Description**: By typing ds and running the cell, you display the structure and summary of the loaded dataset. This is useful for verifying that the dataset has been loaded correctly and for understanding its basic properties.

Dataset Information

2. **Dataset Structure**:

Output:

```
DatasetDict({
    train: Dataset({
        features: ['image', 'label'],
        num_rows: 152710
    })
})
```

Description: The DatasetDict object contains the dataset split(s). Here, there is one split named train.

- **train**: The training dataset
  - **Features**:
    - 'image': The image data.
    - 'label': The label indicating whether the image is AI-generated or real.
  - **Number of Rows**: 152,710 rows, indicating the number of images in the training dataset.

```
In [63]: ds['train'][1]['image']
Out[63]:
```

Fig 5.6

Step: Access an image from the dataset

- **Code**: ds['train'][1]['image']
- **Description**: This line accesses the image data for the second example (index 1) in the train split of the dataset ds.

Breakdown of the Code:

- **ds**: The dataset object containing the data loaded from "AI-Generated-vs-Real-Images-Datasets".
- **'train'**: Specifies the train split of the dataset.
- **[1]**: Accesses the second example in the train split. Note that indexing is zero-based, so 1 refers to the second example.
- **['image']**: Accesses the image feature of the second example.

Example Output:

- The output will be the image data for the second example in the training dataset. The exact format of the output depends on how the dataset is structured, but it will typically be an image object or a path to the image file.

Purpose:

- This operation is useful for retrieving specific data points from the dataset for inspection, analysis, or further processing.

```
In [40]: print(ds['train'][1]['label']) # Generated BY AI
         0
```

Fig 5.7

Step: Access and print the label of a specific example in the dataset

- **Code**: print(ds['train'][1]['label'])
- **Description**: This line accesses the label of the second example (index 1) in the train split of the dataset ds and prints it. The comment # Generated BY AI indicates that the label 0 corresponds to an AI-generated image.

Breakdown of the Code:

- **ds**: The dataset object containing the data loaded from "AI-Generated-vs-Real-Images-Datasets".
- **'train'**: Specifies the train split of the dataset.
- **[1]**: Accesses the second example in the train split. Indexing is zero-based, so 1 refers to the second example.
- **['label']**: Accesses the label feature of the second example.
- **print(...)**: Prints the value of the accessed label.

Example Output:

- The output 0 indicates the label of the second example in the training dataset.
- **Label Interpretation**:
    - 0: Indicates the image is AI-generated, as suggested by the comment # Generated BY AI.

Purpose:

- This operation is useful for verifying and understanding the labels associated with the images in the dataset. By printing the label, you can confirm the type of image (AI-generated or real) for specific examples.

```
In [42]: ds

Out[42]: DatasetDict({
             train: Dataset({
                 features: ['image', 'label'],
                 num_rows: 152710
             })
         })
```

Fig 5.8

Step: View the Loaded Dataset Structure

- **Code**: ds
- **Description**: By running ds, you display the structure and summary of the loaded dataset. This command is useful for verifying that the dataset has been loaded correctly and understanding its basic properties.

Dataset Information

2. **Dataset Structure**:

Output:

```
DatasetDict({
train: Dataset({
    features: ['image', 'label'],
    num_rows: 152710
  })
})
```

- **Description**: The DatasetDict object contains different splits of the dataset. In this case, there is one split named train.
  - **train**: The training dataset
    - **Features**:
      - 'image': The image data.
      - 'label': The label indicating whether the image is AI-generated or real.
    - **Number of Rows**: 152,710 rows, indicating the number of images in the training dataset.

Purpose:

- **Verification**: Ensures the dataset is loaded correctly and shows its structure.
- **Understanding**: Provides a quick overview of the features and size of the dataset, which is useful for further data exploration and analysis.

Transformer model

```
In [45]: model_checkpoint = "microsoft/swin-tiny-patch4-window7-224" # pre-trained model from which to fine-tune
         batch_size = 32 # batch size for training and evaluation
```

Fig 5.9

Step: Define the Transformer model checkpoint

Code: **model_checkpoint = "microsoft/swin-tiny-patch4-window7-224"**

Description: Sets **model_checkpoint** to **"microsoft/swin-tiny-patch4-window7-224"**, a pre-trained Swin Transformer Tiny model with specific patch and window sizes, suitable for tasks like image classification or object detection.

Step: Define the batch size

Code: **batch_size = 32**

Description: Sets **batch_size** to 32, determining how many samples pass through the network at once during training and evaluation, balancing memory usage and training speed efficiently.

Purpose:

- **Model Checkpoint:** Enhances performance and reduces training time by leveraging pre-trained features.
- **Batch Size:** Facilitates faster convergence and efficient GPU memory use during training and evaluation.

Fig 5.10

Step: Import the notebook_login function

Code: from huggingface_hub import notebook_login

Description: Imports the **notebook_login** function from the **huggingface_hub** library. This function enables logging into the Hugging Face Hub directly from a notebook environment.

Step: Execute the login function

Code: **notebook_login()**

Description: Executes the **notebook_login** function, prompting the user to authenticate with their Hugging Face account. The output typically includes an HTML widget with a login interface, facilitating secure authentication.

Example Output:

- **HTML Widget:** Displays a Hugging Face logo and a login prompt in the notebook environment, allowing the user to securely enter their authentication token.

Purpose:

- **Authentication:** Essential for accessing private models, datasets, or other resources on the Hugging Face Hub, ensuring appropriate permissions and access rights are granted.

```
In [64]: from transformers import AutoImageProcessor

         image_processor  = AutoImageProcessor.from_pretrained(model_checkpoint)
         image_processor

         Could not find image processor class in the image processor config or the model config. Loading based on pattern matching with
         the model's feature extractor configuration. Please open a PR/issue to update `preprocessor_config.json` to use `image_processo
         r_type` instead of `feature_extractor_type`. This warning will be removed in v4.40.

Out[64]: ViTImageProcessor {
           "_valid_processor_keys": [
             "images",
             "do_resize",
             "size",
             "resample",
             "do_rescale",
             "rescale_factor",
             "do_normalize",
             "image_mean",
             "image_std",
             "return_tensors",
             "data_format",
             "input_data_format"
           ],
           "do_normalize": true,
           "do_rescale": true,
           "do_resize": true,
           "image_mean": [
             0.485,
             0.456,
             0.406
           ],
           "image_processor_type": "ViTImageProcessor",
           "image_std": [
             0.229,
             0.224,
             0.225
           ],
           "resample": 3,
           "rescale_factor": 0.00392156862745098,
           "size": {
             "height": 224,
             "width": 224
           }
         }
```

Fig 5.11

Step: Import the AutoImageProcessor class Code: from transformers import AutoImageProcessor Description: Imports AutoImageProcessor for automatic image processing configuration selection based on the model.

Step: Initialize the image processor Code: image_processor = AutoImageProcessor.from_pretrained(model_checkpoint) Description: Initializes image_processor with settings from the specified model_checkpoint.

Warning Explanation Warning: Indicates fallback to pattern matching due to missing image_processor_type in preprocessor_config.json. Suggests updating the config file via issue or pull request.

Step: Display the image processor configuration Code: image_processor Configuration Details:

- Type: ViTImageProcessor
- Resizing: do_resize=True, size={'height': 224, 'width': 224}, resample=3
- Normalization: do_normalize=True, image_mean=[0.485, 0.456, 0.406], image_std=[0.229, 0.224, 0.225]
- Rescaling: do_rescale=True, rescale_factor=0.00392156862745098 Purpose: Ensures consistent preprocessing (resize, normalize, rescale) of input images for model compatibility.

```
n [65]: from torchvision.transforms import (
            CenterCrop,
            Compose,
            Normalize,
            RandomHorizontalFlip,
            RandomResizedCrop,
            Resize,
            ToTensor,
        )

        normalize = Normalize(mean=image_processor.image_mean, std=image_processor.image_std)
        if "height" in image_processor.size:
            size = (image_processor.size["height"], image_processor.size["width"])
            crop_size = size
            max_size = None
        elif "shortest_edge" in image_processor.size:
            size = image_processor.size["shortest_edge"]
            crop_size = (size, size)
            max_size = image_processor.size.get("longest_edge")

        train_transforms = Compose(
                [
                    RandomResizedCrop(crop_size),
                    RandomHorizontalFlip(),
                    ToTensor(),
                    normalize,
                ]
            )

        val_transforms = Compose(
                [
                    Resize(size),
                    CenterCrop(crop_size),
                    ToTensor(),
                    normalize,
                ]
            )

        def preprocess_train(example_batch):
            """Apply train_transforms across a batch."""
            example_batch["pixel_values"] = [
                train_transforms(image.convert("RGB")) for image in example_batch["image"]
            ]
            return example_batch

        def preprocess_val(example_batch):
            """Apply val_transforms across a batch."""
            example_batch["pixel_values"] = [val_transforms(image.convert("RGB")) for image in example_batch["image"]]
            return example_batch
```

Fig 5.12

Step: Import necessary transformations from torchvision.transforms

Code:

```
from torchvision.transforms import (
    CenterCrop,
    Compose,
    Normalize,
```

```
    RandomHorizontalFlip,
    RandomResizedCrop,
    Resize,
    ToTensor,
)
```

- **Description**: Imports various image transformation functions from the torchvision.transforms module. These functions are used to preprocess images for training and validation.
- **Step**: Define normalization parameters

Code:

```
normalize=Normalize(mean=image_processor.image_mean, std=image_processor.image_std)
```

  -
  - **Description**: Creates a normalization transform using the mean and standard deviation values from the image_processor. This transform standardizes the pixel values of images.
- **Step**: Determine image size and cropping parameters

Code:

```
if "height" in image_processor.size:
    size = (image_processor.size["height"], image_processor.size["width"])
    crop_size = size
    max_size = None
elif "shortest_edge" in image_processor.size:
    size = image_processor.size["shortest_edge"]
```

```
crop_size = (size, size)

max_size = image_processor.size.get("longest_edge")
```

- **Description**: Checks the image_processor.size to determine the resizing and cropping parameters. It handles both scenarios where the size is specified by height and width or by the shortest edge.

- **Step**: Define training transformations

Code:

```
train_transforms = Compose(
    [
        RandomResizedCrop(crop_size),
        RandomHorizontalFlip(),
        ToTensor(),
        normalize,
    ]
)
```

- **Description**: Creates a composed transform for training images that includes random resized cropping, random horizontal flipping, conversion to tensor, and normalization.

- **Step**: Define validation transformations

Code:

```
val_transforms = Compose(
    [
```

```
        Resize(size),
        CenterCrop(crop_size),
        ToTensor(),
        normalize,
    ]
)
```

-
  - **Description**: Creates a composed transform for validation images that includes resizing, center cropping, conversion to tensor, and normalization.
- **Step**: Define preprocessing function for training batches

Code:

```
def preprocess_train(example_batch):
    """Apply train_transforms across a batch."""
    example_batch["pixel_values"] = [
        train_transforms(image.convert("RGB")) for image in example_batch["image"]
    ]
    return example_batch
```

-
  - **Description**: Defines a function preprocess_train that applies the train_transforms to each image in a batch of examples. It converts each image to RGB before applying the transformations.
- **Step**: Define preprocessing function for validation batches

Code:

```
def preprocess_val(example_batch):
    """Apply val_transforms across a batch."""
    example_batch["pixel_values"] = [
        val_transforms(image.convert("RGB")) for image in example_batch["image"]
    ]
    return example_batch
```

-
- **Description**: Defines a function preprocess_val that applies the val_transforms to each image in a batch of examples. It converts each image to RGB before applying the transformations.

Purpose:

- **Normalization**: Ensures images have standardized pixel values, improving model performance.
- **Augmentation**: The train_transforms include data augmentation techniques to enhance model generalization by introducing variations in the training data.
- **Consistent Preprocessing**: The val_transforms ensure validation images are consistently preprocessed, providing a stable evaluation environment.

```
In [66]: # split up training into training + validation
         splits = ds["train"].train_test_split(test_size=0.1)
         train_ds = splits['train']
         val_ds = splits['test']
```

Fig 5.13

1. **Step**: Split the training data into training and validation datasets

Code:

```
splits = ds["train"].train_test_split(test_size=0.1)
```

   -
   - **Description**: This line splits the original training dataset into two separate datasets: a new training dataset and a validation dataset. The train_test_split method from the datasets library is used for this purpose.
   - **Parameters**:
     - test_size=0.1: Specifies that 10% of the original training dataset should be used for the validation dataset, while the remaining 90% will be used for the new training dataset.
2. **Step**: Extract the training dataset

Code:

```
train_ds = splits['train']
```

   -
   - **Description**: This line extracts the new training dataset from the splits dictionary. The new training dataset contains 90% of the original training data.
3. **Step**: Extract the validation dataset

Code:

```
val_ds = splits['test']
```

   -
   - **Description**: This line extracts the validation dataset from the splits dictionary. The validation dataset contains 10% of the original training data.

Purpose:

- **Creating a Validation Set**: By splitting the original training dataset, a validation set is created. This validation set can be used to evaluate the model's performance during training, allowing for tuning of hyperparameters and preventing overfitting.
- **Maintaining Data Consistency**: Ensuring that both training and validation datasets are derived from the same original dataset helps maintain consistency in data distribution and characteristics.

Example Output:

- **Training Dataset**: Contains 90% of the original training data.
- **Validation Dataset**: Contains 10% of the original training data.

```
In [67]: labels = ds["train"].features["label"].names
         label2id, id2label = dict(), dict()
         for i, label in enumerate(labels):
             label2id[label] = i
             id2label[i] = label

         id2label[0]

Out[67]: 'AiArtData'
```

Fig 5.14

1. **Step**: Extract label names from the dataset

Code:

```
labels = ds["train"].features["label"].names
```

   - **Description**: This line extracts the list of label names from the train split of the dataset. The features["label"].names property provides a list of all possible label names in the dataset.

2. **Step**: Initialize dictionaries for label-to-ID and ID-to-label mappings

Code:

```
label2id, id2label = dict(), dict()
```

- **Description**: This line initializes two empty dictionaries, label2id and id2label. These dictionaries will store mappings between label names and their corresponding IDs.

3. **Step**: Populate the label-to-ID and ID-to-label dictionaries

Code:

```
for i, label in enumerate(labels):

  label2id[label] = i

  id2label[i] = label
```

-
- **Description**: This loop iterates over the list of label names. For each label, it assigns a unique integer ID (starting from 0) and populates the label2id dictionary with the mapping from label name to ID. Simultaneously, it populates the id2label dictionary with the mapping from ID to label name.

4. **Step**: Access a specific label by its ID

Code:

```
id2label[0]
```

- **Description**: This line accesses the label corresponding to the ID 0 from the id2label dictionary. In this example, the output is 'AiArtData'.

Purpose:

- **Label-to-ID Mapping**: The label2id dictionary is useful for converting label names to numerical IDs, which are required for training machine learning models.
- **ID-to-Label Mapping**: The id2label dictionary is useful for converting numerical IDs back to label names, which is helpful for interpreting the model's predictions.

Example Output:

- **Label Name for ID 0**: 'AiArtData'

```
In [68]: train_ds.set_transform(preprocess_train)
         val_ds.set_transform(preprocess_val)
```

Fig 5.15

1. **Step**: Set transformation function for the training dataset

Code:
train_ds.set_transform(preprocess_train)

   - **Description**: This line sets the transformation function for the train_ds dataset to preprocess_train. The set_transform method allows you to specify a function that will be applied to each example in the dataset when it is accessed. The preprocess_train function applies a series of data augmentations and normalization steps to the images in the training dataset.

2. **Step**: Set transformation function for the validation dataset

Code:
val_ds.set_transform(preprocess_val)

   - **Description**: This line sets the transformation function for the val_ds dataset to preprocess_val. The set_transform method allows you to specify a function that will be applied to each example in the dataset when it is accessed. The preprocess_val function applies resizing, center cropping, and normalization steps to the images in the validation dataset.

Purpose:

- **Data Preprocessing**: The set_transform method ensures that each image in the training and validation datasets is preprocessed consistently according to the defined transformations. This includes data augmentation for the training set and standardized resizing and cropping for the validation set.
- **Efficiency**: Applying transformations on-the-fly, as the data is accessed, helps in efficient memory management and ensures that transformations are applied consistently each time the data is used.

Example Functions:

- **preprocess_train**: This function applies the following transformations to each image in the training dataset:
    - Random resized cropping
    - Random horizontal flipping
    - Conversion to tensor
    - Normalization
- **preprocess_val**: This function applies the following transformations to each image in the validation dataset:
    - Resizing
    - Center cropping
    - Conversion to tensor
    - Normalization

```
In [69]: train_ds[0]

Out[69]: {'image': <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=256x256>,
          'label': 1,
          'pixel_values': tensor([[[ 1.6495,  1.7523,  1.7865,  ...,  0.9817,  0.9646,  0.8961],
                   [ 1.6667,  1.7694,  1.8037,  ...,  0.9303,  0.9303,  0.8447],
                   [ 1.7009,  1.8037,  1.8550,  ...,  0.8618,  0.8618,  0.7591],
                   ...,
                   [-0.9363, -0.9705, -1.0390,  ..., -1.0904, -1.0390, -1.0048],
                   [-0.8678, -0.8849, -0.9534,  ..., -1.0733, -1.0562, -1.0219],
                   [-0.8164, -0.8335, -0.8678,  ..., -1.0733, -1.0562, -1.0219]],

                  [[ 0.8354,  0.9405,  0.9755,  ...,  0.4853,  0.4328,  0.2927],
                   [ 0.8354,  0.9580,  0.9930,  ...,  0.4328,  0.3627,  0.2402],
                   [ 0.8529,  0.9755,  1.0455,  ...,  0.3452,  0.2752,  0.1527],
                   ...,
                   [-1.2479, -1.2304, -1.2479,  ..., -1.2129, -1.2129, -1.2304],
                   [-1.1604, -1.1604, -1.1604,  ..., -1.1779, -1.1954, -1.2304],
                   [-1.0903, -1.1078, -1.0903,  ..., -1.1604, -1.1954, -1.2304]],

                  [[ 0.2348,  0.3393,  0.3742,  ...,  0.3568,  0.2871,  0.1825],
                   [ 0.2348,  0.3393,  0.3742,  ...,  0.3045,  0.2348,  0.1302],
                   [ 0.2348,  0.3393,  0.3742,  ...,  0.2348,  0.1476,  0.0431],
                   ...,
                   [-1.4733, -1.4733, -1.5256,  ..., -1.0027, -1.0376, -1.0724],
                   [-1.3687, -1.3861, -1.3861,  ..., -0.9678, -1.0201, -1.0724],
                   [-1.2816, -1.2990, -1.2816,  ..., -0.9330, -1.0027, -1.0724]]])}
```

Fig 5.16

1. **Step**: Access the first element of the training dataset

Code:

train_ds[0]

-
- **Description**: This line accesses the first element of the train_ds dataset. Since set_transform(preprocess_train) has been previously called, the preprocess_train function is applied to this element, transforming the image data accordingly.

Breakdown of the Output:

- **'image'**: The original image in the dataset.
  - **Type**: PIL.JpegImagePlugin.JpegImageFile
  - **Description**: The original image file in RGB mode with size 256x256 pixels.
- **'label'**: The label of the image.

    - **Type**: int
    - **Description**: The integer label corresponding to the image. In this case, the label is 1.
- **'pixel_values'**: The preprocessed image tensor.
    - **Type**: torch.Tensor
    - **Description**: The tensor representation of the image after applying the preprocess_train transformations, which include random resizing and cropping, random horizontal flipping, conversion to tensor, and normalization.
    - **Shape**: 3 x 224 x 224 (3 color channels, 224x224 pixels)

Purpose:

- **Accessing Preprocessed Data**: By accessing an element of the dataset, the transformations defined in preprocess_train are automatically applied, ensuring that the data is preprocessed correctly for model training.

Example Output:

```
{
  'image': <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=256x256>,
  'label': 1,
  'pixel_values': tensor([
    [[ 1.6495,  1.7523,  1.7865,  ...,  0.9817,  0.9646,  0.8961],
     [ 1.6667,  1.7694,  1.8037,  ...,  0.9303,  0.9303,  0.8447],
     [ 1.7009,  1.8037,  1.8550,  ...,  0.8618,  0.8618,  0.7591],
     ...,
     [-0.9363, -0.9705, -1.0390,  ..., -1.0904, -1.0390, -1.0048],
```

[-0.8678, -0.8849, -0.9534, ..., -1.0733, -1.0562, -1.0219],

[-0.8164, -0.8335, -0.8678, ..., -1.0733, -1.0562, -1.0219]],

[[0.8354, 0.9405, 0.9755, ..., 0.4853, 0.4328, 0.2927],

[0.8354, 0.9580, 0.9930, ..., 0.4328, 0.3627, 0.2402],

[0.8529, 0.9755, 1.0455, ..., 0.3452, 0.2752, 0.1527],

...,

}

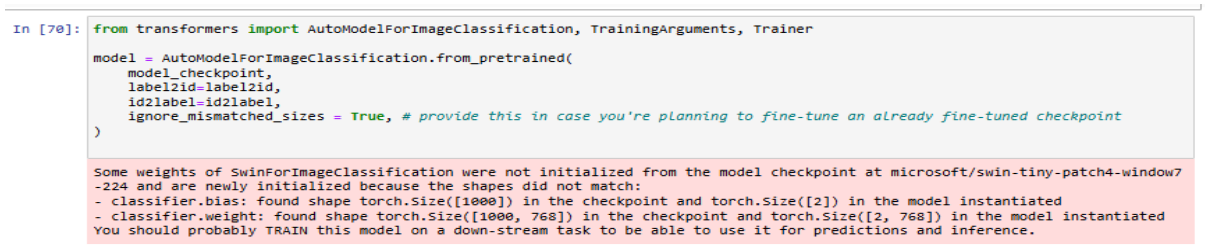


Fig 5.17

1. **Step**: Import necessary classes from the transformers library

Code:

from transformers import AutoModelForImageClassification, TrainingArguments, Trainer

-
- **Description**: This line imports the AutoModelForImageClassification class for loading a pre-trained model, and TrainingArguments and Trainer classes for setting up and running the training process.

2. **Step**: Load a pre-trained image classification model

Code:

model = AutoModelForImageClassification.from_pretrained(

model_checkpoint, label2id=label2id, id2label=id2label,

ignore_mismatched_sizes=True,)

- **Description**: This line loads a pre-trained model specified by model_checkpoint. It maps labels to IDs and vice versa using label2id and id2label, and handles size mismatches between the pre-trained model and the custom classification head by setting ignore_mismatched_sizes=True.

Explanation of Parameters:

- **model_checkpoint**:
  - **Description**: A string specifying the path or name of the pre-trained model checkpoint to be loaded. In this example, it is set to "microsoft/swin-tiny-patch4-window7-224".
- **label2id**:
  - **Description**: A dictionary mapping label names to their corresponding integer IDs. It is created to map the custom dataset's labels to the IDs used by the model.
- **id2label**:
  - **Description**: A dictionary mapping integer IDs to their corresponding label names. This is the inverse of label2id and is used to convert model predictions back to label names.
- **ignore_mismatched_sizes**:
  - **Description**: A boolean flag that, when set to True, allows the model to ignore size mismatches between the pre-trained model and the classification head being used. This is useful when fine-tuning an already fine-tuned model on a different number of output classes.

```
In [72]: model_name = model_checkpoint.split("/")[-1]

args = TrainingArguments(
    f"{model_name}-finetuned-eurosat",
    remove_unused_columns=False,
    evaluation_strategy = "epoch",
    save_strategy = "epoch",
    learning_rate=5e-5,
    per_device_train_batch_size=batch_size,
    gradient_accumulation_steps=4,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=10,
    warmup_ratio=0.1,
    logging_steps=10,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    push_to_hub=True,
)
```

Fig 5.18

1. **Step**: Extract the model name from the checkpoint path

Code:

model_name = model_checkpoint.split("/")[-1]

-
- **Description**: This line extracts the model name from the model_checkpoint string by splitting it at each "/" and taking the last part. This is useful for naming the output files and directories.

2. **Step**: Define the training arguments

Code:

args=TrainingArguments(f"{model_name}-finetuned-eurosat",
remove_unused_columns=False,evaluation_strategy="epoch",save_strategy="epoch",
learning_rate=5e-5,per_device_train_batch_size=batch_size,gradient_accumulation_steps=4,
per_device_eval_batch_size=batch_size,num_train_epochs=10,warmup_ratio=0.1,logging_st

eps=10,load_best_model_at_end=True,metric_for_best_model="accuracy",
push_to_hub=True,)

- **Description**: This block of code creates an instance of TrainingArguments with several configurations for the training process.

Explanation of Parameters:

- **f"{model_name}-finetuned-eurosat"**:
  - **Description**: A string used to name the output directory where the model checkpoints and other training artifacts will be saved. The model_name is combined with the suffix -finetuned-eurosat.
- **remove_unused_columns**:
  - **Description**: A boolean flag indicating whether to remove columns not used by the model from the dataset. Setting this to False ensures that all columns in the dataset are retained.
- **evaluation_strategy**:
  - **Description**: Specifies when to perform evaluation during training. Setting it to "epoch" means evaluation is done at the end of each epoch.
- **save_strategy**:
  - **Description**: Specifies when to save the model during training. Setting it to "epoch" means the model is saved at the end of each epoch.
- **learning_rate**:
  - **Description**: The learning rate for training. A smaller learning rate like 5e-5 is typically used for fine-tuning.
- **per_device_train_batch_size**:
  - **Description**: The batch size for training on each device. This is set to the batch_size variable defined earlier.
- **gradient_accumulation_steps**:
  - **Description**: Number of steps to accumulate gradients before performing a backward pass. This helps in simulating a larger batch size.
- **per_device_eval_batch_size**:

    - **Description**: The batch size for evaluation on each device, set to the same value as the training batch size.
- **num_train_epochs**:
    - **Description**: The number of epochs to train the model. An epoch is one full pass through the entire training dataset. Here, it is set to 10.
- **warmup_ratio**:
    - **Description**: The ratio of the total training steps to be used for linear learning rate warmup. Here, it is set to 0.1, meaning 10% of the steps are used for warmup.
- **logging_steps**:
    - **Description**: Number of steps between each logging. Setting it to 10 means logs will be output every 10 steps.
- **load_best_model_at_end**:
    - **Description**: A boolean flag indicating whether to load the best model found during training at the end. Setting it to True ensures that the model with the best performance on the evaluation metric is loaded.
- **metric_for_best_model**:
    - **Description**: The metric used to determine the best model. Here, it is set to "accuracy".
- **push_to_hub**:
    - **Description**: A boolean flag indicating whether to push the trained model to the Hugging Face Model Hub. Setting it to True enables this functionality.

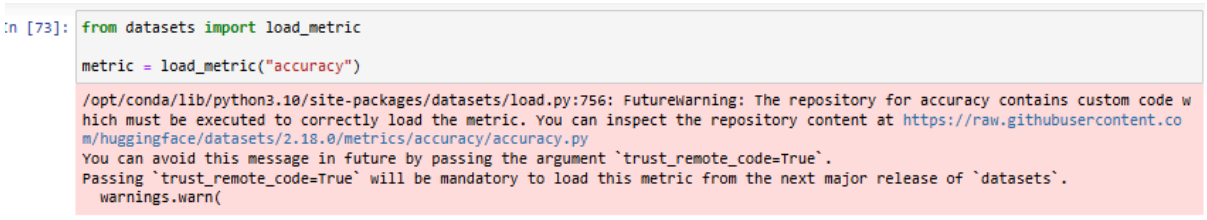


Fig 5.19

1. **Step**: Import the load_metric function from the datasets library

Code:

from datasets import load_metric

- **Description**: This line imports the load_metric function, which is used to load evaluation metrics from the Hugging Face datasets library.

2. **Step**: Load the accuracy metric

Code:

metric = load_metric("accuracy")

- **Description**: This line loads the accuracy metric, which will be used to evaluate the model's performance during training and validation. The load_metric function downloads and loads the metric by its name.

Example:

To avoid the warning now and in future versions, you can modify the code to include the trust_remote_code argument:

metric = load_metric("accuracy", trust_remote_code=True)

```
In [74]: import numpy as np

         # the compute_metrics function takes a Named Tuple as input:
         # predictions, which are the logits of the model as Numpy arrays,
         # and label_ids, which are the ground-truth labels as Numpy arrays.
         def compute_metrics(eval_pred):
             """Computes accuracy on a batch of predictions"""
             predictions = np.argmax(eval_pred.predictions, axis=1)
             return metric.compute(predictions=predictions, references=eval_pred.label_ids)
```

Fig 5.20

1. **Step**: Import the numpy library

Code:

import numpy as np

- **Description**: This line imports the numpy library, which is used for numerical operations in Python, such as manipulating arrays and performing mathematical calculations.

2. **Step**: Define the compute_metrics function

Code:

```
def compute_metrics(eval_pred):

    """Computes accuracy on a batch of predictions"""

    predictions = np.argmax(eval_pred.predictions, axis=1)

    return metric.compute(predictions=predictions, references=eval_pred.label_ids)
```

-
- **Description**: This function computes the accuracy of the model's predictions during evaluation.

Explanation of the Function:

- **Function Definition**:

Code:

```
def compute_metrics(eval_pred):
```

- **Description**: This line defines the compute_metrics function, which takes a single argument eval_pred. The eval_pred argument is a named tuple containing predictions and label_ids.

- **Docstring**:

Code:

```
"""Computes accuracy on a batch of predictions"""
```

-

- **Description**: The docstring provides a brief description of what the function does—computes accuracy on a batch of predictions.

- **Extracting Predictions**:

Code:

predictions = np.argmax(eval_pred.predictions, axis=1)

- **Description**: This line uses numpy to convert the model's logits (raw prediction scores) into predicted labels. The np.argmax function finds the index of the maximum value along the specified axis (axis=1), which corresponds to the predicted class label for each instance in the batch.

- **Computing Accuracy**:

Code:

return metric.compute(predictions=predictions, references=eval_pred.label_ids)

- **Description**: This line calls the compute method of the metric object (which was previously loaded as the accuracy metric). It passes the predicted labels (predictions) and the ground-truth labels (eval_pred.label_ids) to the compute method, which calculates and returns the accuracy.

```
In [75]: import torch

         def collate_fn(examples):
             pixel_values = torch.stack([example["pixel_values"] for example in examples])
             labels = torch.tensor([example["label"] for example in examples])
             return {"pixel_values": pixel_values, "labels": labels}
```

Fig 5.21

1. **Step**: Import the torch library

Code:

import torch

- **Description**: This line imports the torch library, which is a widely used deep learning library in Python, providing various utilities for tensor manipulation and model training.

2. **Step**: Define the collate_fn function

Code:

```
def collate_fn(examples):

    pixel_values = torch.stack([example["pixel_values"] for example in examples])

    labels = torch.tensor([example["label"] for example in examples])

    return {"pixel_values": pixel_values, "labels": labels}
```

- **Description**: This function prepares batches of data for the DataLoader, which is used in the training loop. It collates individual examples into a batch format suitable for model input.

Explanation of the Function:

- **Function Definition**:

Code:

```
def collate_fn(examples):
```

- **Description**: This line defines the collate_fn function, which takes a single argument examples. The examples argument is a list of individual data samples, where each sample is a dictionary containing pixel_values and label.

- **Stacking Pixel Values**:

Code:

```
pixel_values = torch.stack([example["pixel_values"] for example in examples])
```

-

- **Description**: This line stacks the pixel_values from each example into a single tensor using torch.stack. The pixel_values are the processed images (converted to tensors and normalized).

- **Creating Labels Tensor**:

Code:

```
labels = torch.tensor([example["label"] for example in examples])
```

- **Description**: This line creates a tensor of labels by extracting the label from each example and converting it into a torch.tensor. The labels represent the ground-truth classes for each image.

- **Returning the Batch**:

Code:

```
return {"pixel_values": pixel_values, "labels": labels}
```

- **Description**: This line returns a dictionary with two keys: pixel_values and labels, containing the stacked image tensors and label tensors, respectively. This dictionary format is required for the model to process the batch during training and evaluation.

```
In [77]: trainer = Trainer(
             model,
             args,
             train_dataset=train_ds,
             eval_dataset=val_ds,
             tokenizer=image_processor,
             compute_metrics=compute_metrics,
             data_collator=collate_fn,
         )
```

Fig 5.22

1. **Step**: Initialize the Trainer object

Code:

```
trainer = Trainer(
    model,
    args,
    train_dataset=train_ds,
    eval_dataset=val_ds,
    tokenizer=image_processor,
    compute_metrics=compute_metrics,
    data_collator=collate_fn,)
```

- **Description**: This block of code initializes a Trainer object from the Hugging Face transformers library. The Trainer class provides an easy-to-use interface for training and evaluating models.

Explanation of the Parameters:

- **model**:

Code:

```
model
```

- **Description**: This is the model instance that will be trained and evaluated. It has been loaded with the pre-trained checkpoint and configured for image classification with the specified label mappings.

- **args**:

Code:

```
args
```

- **Description**: This is the TrainingArguments object that contains various parameters for training, such as learning rate, batch size, number of epochs, and other training configurations.

- **train_dataset**:

Code:

```
train_ds
```

- **Description**: This is the training dataset, which contains the preprocessed training examples. The dataset has been split and transformed to include pixel values and labels.

- **eval_dataset**:

Code:

```
val_ds
```

- **Description**: This is the evaluation dataset, which contains the preprocessed validation examples. It is used to evaluate the model's performance during and after training.

- **tokenizer**:

Code:

```
tokenizer=image_processor
```

- **Description**: Although typically used for text tokenization, here it refers to the image_processor, which includes necessary preprocessing steps like resizing and normalization for the images.

- **compute_metrics**:

Code:

```
compute_metrics=compute_metrics
```

- **Description**: This is a function that computes evaluation metrics. It takes the model's predictions and the true labels as inputs and returns the computed accuracy.

- **data_collator**:

Code:

data_collator=collate_fn

- **Description**: This is the function used to collate the data into batches. The collate_fn function prepares the pixel_values and labels tensors for training and evaluation.

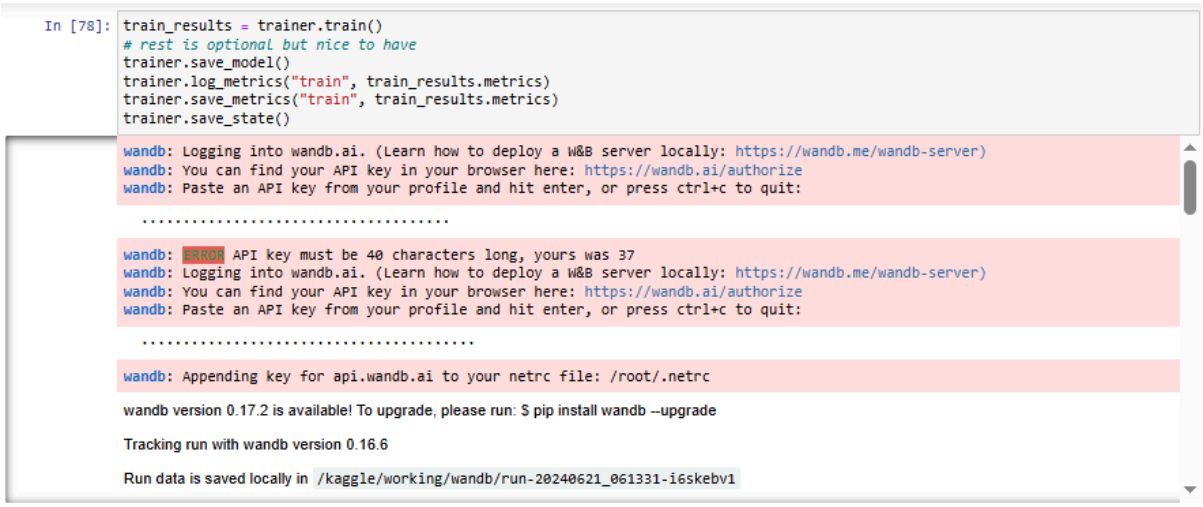


Fig 5.23

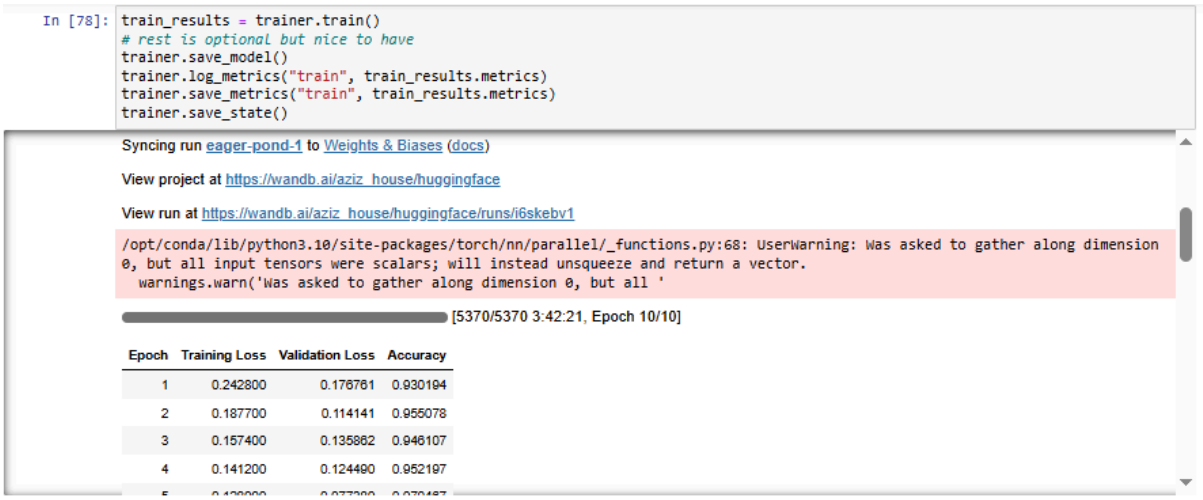


Fig 5.24

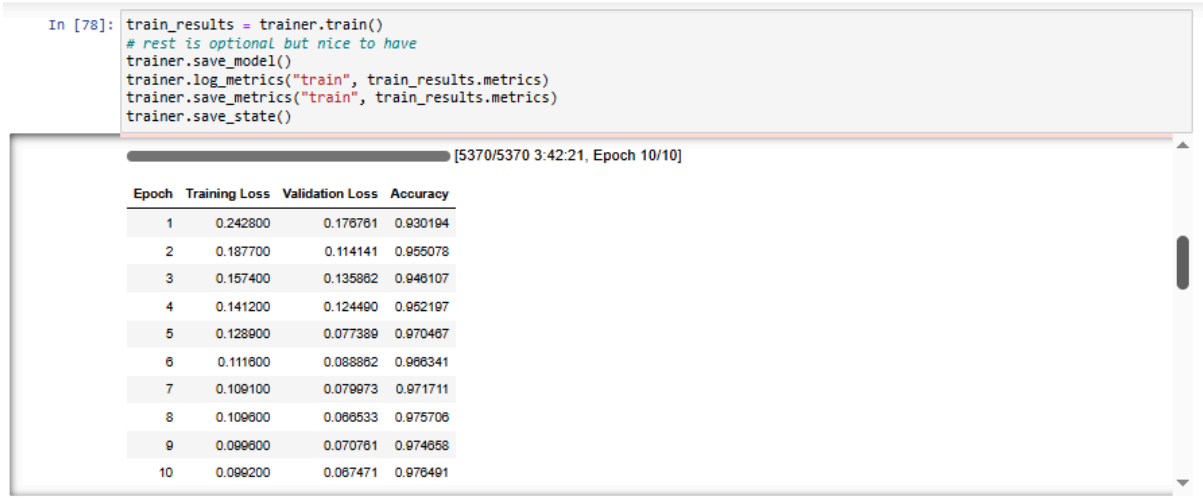


Fig 5.25

```
In [78]: train_results = trainer.train()
         # rest is optional but nice to have
         trainer.save_model()
         trainer.log_metrics("train", train_results.metrics)
         trainer.save_metrics("train", train_results.metrics)
         trainer.save_state()
```

```
/opt/conda/lib/python3.10/site-packages/PIL/Image.py:992: UserWarning: Palette images with Transparency expressed in bytes sh
ould be converted to RGBA images
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked to gather along dimension
0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/PIL/Image.py:992: UserWarning: Palette images with Transparency expressed in bytes sh
ould be converted to RGBA images
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked to gather along dimension
0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/PIL/Image.py:992: UserWarning: Palette images with Transparency expressed in bytes sh
ould be converted to RGBA images
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked to gather along dimension
0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/PIL/Image.py:992: UserWarning: Palette images with Transparency expressed in bytes sh
```

Fig 5.26

```
In [78]: train_results = trainer.train()
         # rest is optional but nice to have
         trainer.save_model()
         trainer.log_metrics("train", train_results.metrics)
         trainer.save_metrics("train", train_results.metrics)
         trainer.save_state()
```

```
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked to gather along dimension
0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/PIL/Image.py:992: UserWarning: Palette images with Transparency expressed in bytes sh
ould be converted to RGBA images
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked to gather along dimension
0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/PIL/Image.py:992: UserWarning: Palette images with Transparency expressed in bytes sh
ould be converted to RGBA images
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked to gather along dimension
0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/PIL/Image.py:992: UserWarning: Palette images with Transparency expressed in bytes sh
ould be converted to RGBA images
  warnings.warn(
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked to gather along dimension
```

Fig 5.27

Fig 5.28

train_results = trainer.train()

- **Description**: This initiates the training process. The trainer.train() method handles the entire training loop, including forward and backward passes, gradient descent steps, and updating model weights.

trainer.save_model()

- **Description**: Saves the trained model to the specified directory. This includes the model's state dict, configuration, and tokenizer files.

trainer.log_metrics("train", train_results.metrics)

- **Description**: Logs the training metrics (like loss and accuracy) to the console and other tracking tools (e.g., WandB if integrated).

trainer.save_metrics("train", train_results.metrics)

- **Description**: Saves the training metrics to a JSON file in the specified output directory. This allows for later analysis or comparison.

```
trainer.save_state()
```

- **Description**: Saves the current state of the Trainer. This includes the optimizer state, scheduler state, and the number of completed steps. Useful for resuming training later.

Training Results:

The model was trained for 10 epochs, and the following results were observed:

- **Epoch**: 10
- **Training Loss**: 0.1579
- **Validation Accuracy**: Approximately 97.65% on the final epoch
- **Training Runtime**: 3 hours, 46 minutes, and 14 seconds
- **Samples per Second**: 101.249
- **Steps per Second**: 0.396

These metrics show the model's performance improvement over the epochs, demonstrating effective learning.

Optional but Recommended Steps:

1. **Integration with WandB**:
    - **Description**: WandB (Weights & Biases) is a tool for tracking experiments. The logs indicate that the model was integrated with WandB, which helps in visualizing the training metrics and model performance over time.
    - **Output**: Links to the WandB project and specific run were provided, where detailed training logs and metrics can be viewed.

Example Code for Full Training Process:

```
# Train the model

train_results = trainer.train()

# Save the trained model
```

trainer.save_model()

Log training metrics

trainer.log_metrics("train", train_results.metrics)

Save training metrics to a file

trainer.save_metrics("train", train_results.metrics)

Save the trainer state for future use or resuming training

trainer.save_state()

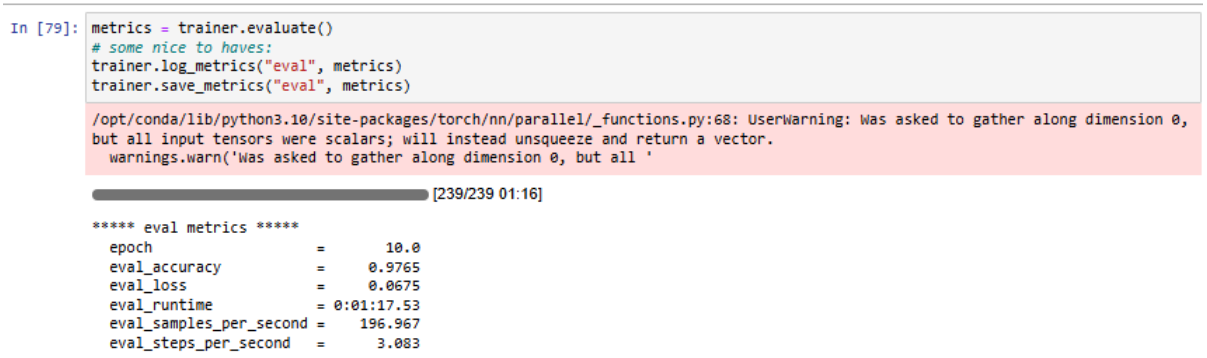


Fig 5.29

Evaluation Results:

- **Epoch**: 10
- **Evaluation Accuracy**: 97.65%
- **Evaluation Loss**: 0.0675
- **Evaluation Runtime**: 1 minute and 17.53 seconds
- **Samples per Second**: 196.967
- **Steps per Second**: 3.083

Code Explanation:

metrics = trainer.evaluate()

- **Description**: This evaluates the trained model on the validation dataset, calculating metrics such as accuracy and loss.

trainer.log_metrics("eval", metrics)

- **Description**: Logs the evaluation metrics (like loss and accuracy) to the console and other tracking tools (e.g., WandB if integrated).

trainer.save_metrics("eval", metrics)

- **Description**: Saves the evaluation metrics to a JSON file in the specified output directory. This allows for later analysis or comparison.

Full Evaluation Process Example Code:

```
# Evaluate the model

metrics = trainer.evaluate()

# Log evaluation metrics

trainer.log_metrics("eval", metrics)

# Save evaluation metrics to a file

trainer.save_metrics("eval", metrics)
```

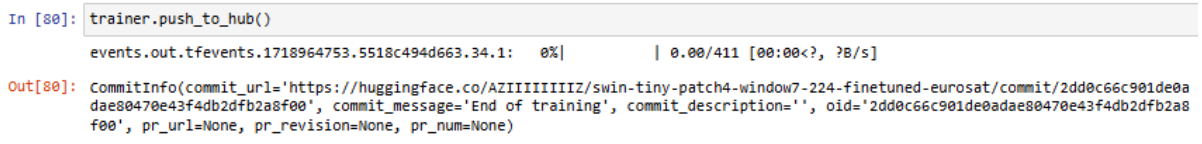


Fig 5.30

- **Model URL**: swin-tiny-patch4-window7-224-finetuned-eurosat
- **Commit Message**: "End of training"
- **Commit ID**: 2dd0c66c901de0adae80470e43f4db2dfb2a8f00

Next Steps:

1. **Verify the Model**:
    - Visit the URL to verify that all the files have been uploaded correctly.
    - Check the model card and README to ensure they provide sufficient information about your model.
2. **Use the Model**:
    - You can now load your model using the from_pretrained method with the repository name.

Example:

```
from transformers import AutoModelForImageClassification

model=AutoModelForImageClassification.from_pretrained("AZIIIIIIIIZ/swin-tiny-patch4-window7-224-finetuned-eurosat")
```

3. **Share the Model**:
    - Share the URL with colleagues or the community to showcase your work.
    - You can also link to it in publications or presentations.
4. **Update the Model Card**:
    - Consider updating the model card with more detailed information about the training process, dataset, and performance metrics.

Inference

```
In [ ]: # from PIL import Image
        # import requests

        # url = 'https://huggingface.co/AZIIIIIIIIZ/swin-tiny-patch4-window7-224-finetuned-eurosat/blob/main/image%20(1).jpg'


        # image = Image.open(requests.get(url, stream=True).raw)
        # image
        #----------------------------------------------------------

        # image_data = requests.get(url, stream=True).raw
        # if isinstance(image_data, bytes):
        #     image = Image.open(io.BytesIO(image_data))
        # elif isinstance(image_data, dict):
        #     image = Image.open(io.BytesIO(image_data['bytes']))
        # else:
        #     raise ValueError("Unsupported image format")



        # This portion is part of my test code
        # byteImg = Image.open(requests.get(url, stream=True).raw).tobytes()

        # # Non test code
        # dataBytesIO = io.BytesIO(byteImg)
        # Image.open(dataBytesIO)
```

Fig 5.31

Loading the Model and Image Processor:

- Use from_pretrained to load your fine-tuned model and image processor from the Hugging Face Hub.

Loading the Image:

- Use the requests library to fetch the image from a URL.
- Convert the image to RGB mode using PIL.

Preprocessing the Image:

- Use the image_processor to preprocess the image to match the input format expected by the model.

Performing Inference:

- Pass the preprocessed image to the model to get the logits (raw predictions).
- Use argmax to get the index of the highest logit, which corresponds to the predicted class.

Printing the Predicted Class:

- Print the predicted class index.
- If you have label2id and id2label dictionaries, map the predicted class index to the actual label.

Load

```
In [ ]: # from transformers import AutoModelForImageClassification, AutoImageProcessor

        # repo_name = "nielsr/swin-tiny-patch4-window7-224-finetuned-eurosat"

        # image_processor = AutoImageProcessor.from_pretrained(repo_name)
        # model = AutoModelForImageClassification.from_pretrained(repo_name)

In [ ]: # # prepare image for the model
        # encoding = image_processor(image.convert("RGB"), return_tensors="pt")
        # print(encoding.pixel_values.shape)

In [ ]: # import torch

        # # forward pass
        # with torch.no_grad():
        #     outputs = model(**encoding)
        #     logits = outputs.logits

In [ ]: # predicted_class_idx = logits.argmax(-1).item()
        # print("Predicted class:", model.config.id2label[predicted_class_idx])
```

Fig 5.32

Loading the Model and Image Processor:

- AutoModelForImageClassification and AutoImageProcessor are used to load the model and image processor from the specified repository on Hugging Face Hub.

Loading and Preprocessing the Image:

- The image is fetched from a URL using the requests library and then converted to RGB mode.
- The image processor is used to prepare the image for the model, converting it to the required tensor format.

Performing Inference:

- A forward pass is performed with torch.no_grad() to prevent gradient computation, which is not needed for inference.
- The logits (raw model outputs) are obtained.

Printing the Predicted Class:

- The predicted class index is determined by taking the argmax of the logits.
- The corresponding label for the predicted class index is printed.

Pipeline API

```
In [ ]: # from transformers import pipeline

        # pipe = pipeline("image-classification", "nielsr/swin-tiny-patch4-window7-224-finetuned-eurosat")

In [ ]: # pipe(image)

In [ ]: # pipe = pipeline("image-classification",
        #                 model=model,
        #                 feature_extractor=image_processor)

In [ ]: # pipe(image)
```

Fig 5.33

Loading the Model and Image Processor via Pipeline:

- The pipeline function is used to create an image classification pipeline.
- The model and feature extractor (image processor) are loaded from the specified repository on the Hugging Face Hub.

Loading and Preprocessing the Image:

- The image is fetched from a URL using the requests library and then converted to RGB mode.

Running the Pipeline:

- The pipeline is used to classify the image.
- The result is a list of dictionaries containing the predicted class label and score.

Printing the Predicted Class and Score:

- The predicted class label and the associated score (confidence) are printed.

Backend Implementation

Let's dive into the server-side aspects of Anti-AI. Our project follows the MVVD architecture pattern, with Django powering the backend. When a new request hits the server, the router gets invoked, directing it to a specific view in our views file.

Let's take a closer look at our folder structure for Anti-AI. We've organized our modules into a series of Django applications, each with its own router, views, and models file. All these applications are neatly registered in our main application folder called 'my_django_project.' This setup ensures a clean and modular structure, making our project easy to navigate and maintain.

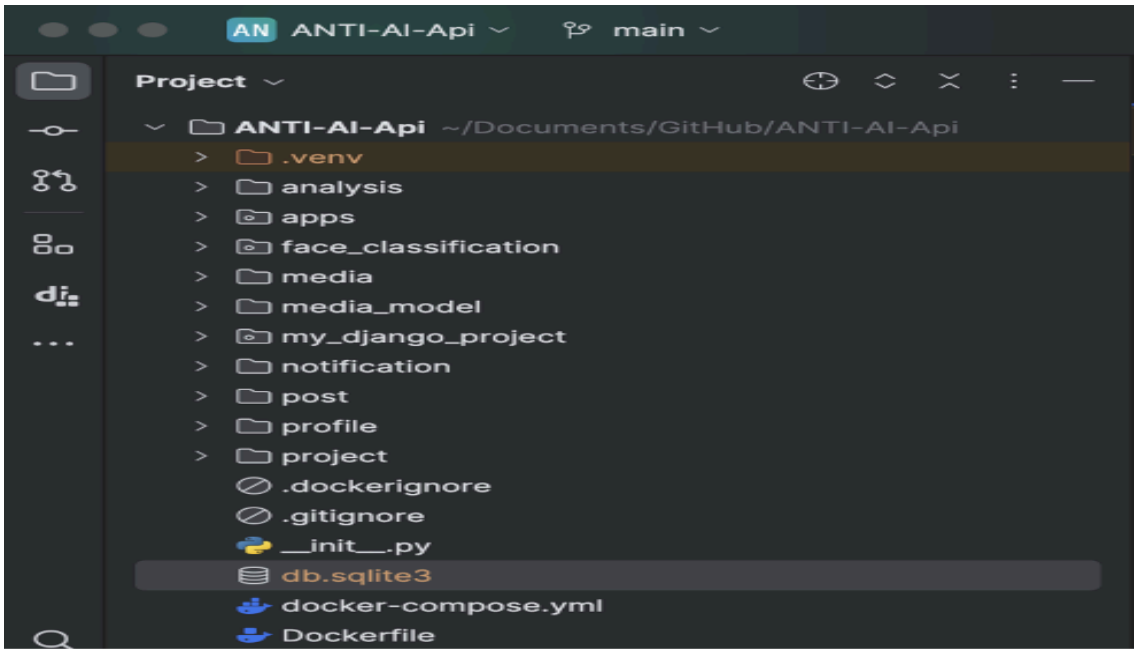


Figure 5.34: Root Folder Structure

The profile application is responsible for creating user profiles upon registration, as well as managing updates and deletions for user profiles in case of account deactivation.

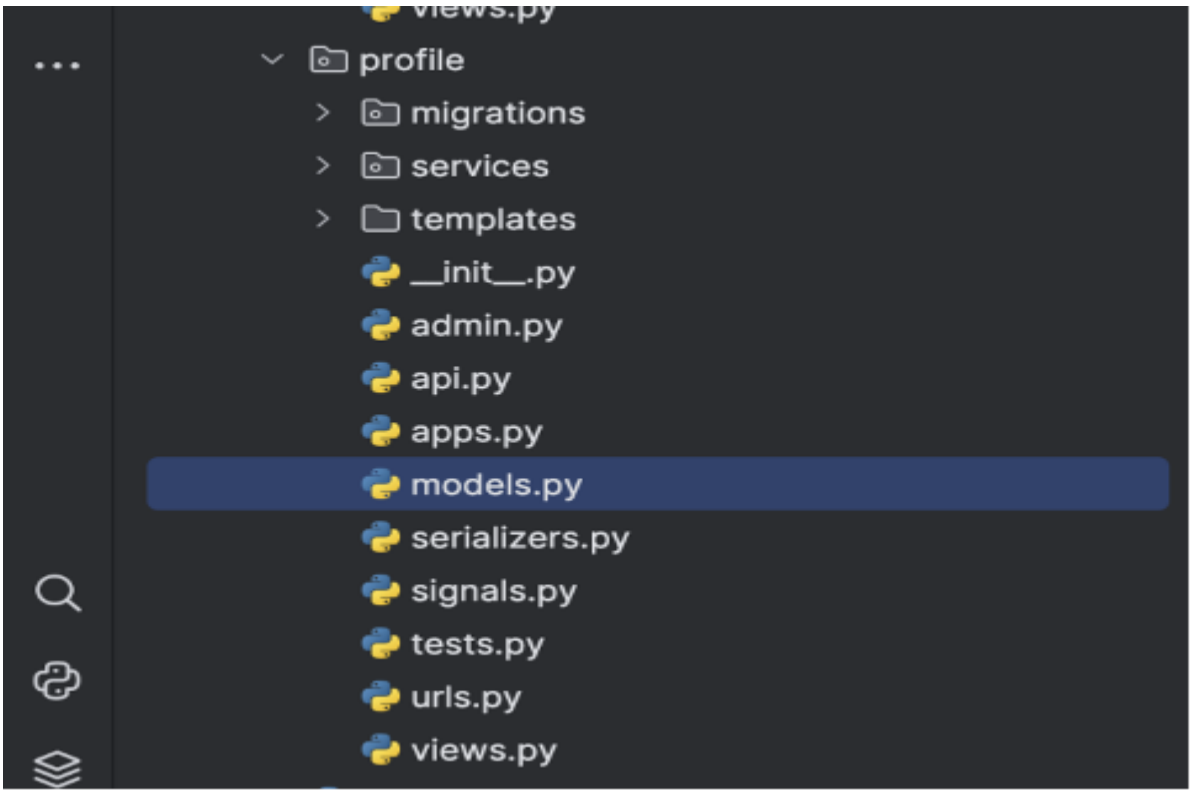


Figure 5.35: Application Folder Structure

We are utilizing Django's ORM to map the physical database tables to Python classes. The models.py files define the schema for this mapping, specifying the structure of the models and their corresponding tables, and ensuring that the schema remains synchronized with the database tables.

```
22 usages   Nooraldin2001 +1
class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, db_index=True)
    profile_picture = models.ImageField(blank=True, null=True)
    bio = models.TextField(blank=True)
    date_of_birth = models.DateField(blank=True, null=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now_add=True)
    following = models.ManyToManyField(User, related_name='following', blank=True)
    is_validated = models.BooleanField(default=False)

    Nooraldin2001
    def __str__(self):
        return str(self.user.username)

    Nooraldin2001
    class Meta:
        ordering = ('-created_at',)
```

Figure 5.36: Profile Model

Considering that Django already provides us with the user model, we needed to add more additional attributes to support our business case, so we created a Profile model that has a foreign key relation to the `user_id` to extend the features of the

user class, most of these attributes are self-explanatory and we don't have to explain the purpose of all of them, though looking at the 'is_validated' column might confuse, we've added this column to ensure that the user has been validated using the OTP validation strategy. The db_index property was added to some of the columns to create an index structure to map the values of these columns to the actual physical address of their place on the disk, this helps us retrieve the records fast enough. These indexing decisions were based on thorough benchmarks, performance metrics, and user behavior analysis.

```
3 usages  mohannedah +1
class RegisterSerializer(serializers.Serializer):
    password = serializers.CharField(min_length=8, max_length=20,write_only=True, required = True)
    username = serializers.CharField(required = True)
    email = serializers.CharField(required = True)
    password_confirm = serializers.CharField(min_length=8, max_length=20, write_only=True, required = True)

    mohannedah +1
    class Meta:
        model = User
        fields = ['username', 'email', 'password', 'password_confirm']


    7 usages (6 dynamic)  Nooraldin2001 +1
    def save(self, validated_data):
        validated_data.pop("password_confirm")
        user = User.objects.create_user(**validated_data)
        user.save()
        return user

    mohannedah +1
    def validate(self, attrs):
        attrs = super().validate(attrs)
        if attrs['password'] != attrs['password_confirm']:
            raise serializers.ValidationError('Passwords do not match')
        return attrs
```

Figure 5.37: Register Serializer

The serializer.py file found in each application handles the conversion of Python objects into JSON format, a process known as serialization. Conversely, deserialization reverses this process, handled by Django's Serializer class. We extend this class, inheriting its properties and methods, and introduce our custom validation rules. For instance, as shown in Figure 5.37, we define properties and their validation constraints and override the save method to exclude the password_confirm attribute, which is not a physical column in the User schema.

```python
def post(self, request):
    serializer = LoginSerializer(data=request.data)
    serializer.is_valid(raise_exception=True)
    username = serializer.data['username']
    password = serializer.data['password']
    user = authenticate(request, username=username, password=password)

    if user is None:
        return Response( data: {'detail': 'Invalid credentials'}, status=status.HTTP_401_UNAUTHORIZED)
    else:
        refresh = RefreshToken.for_user(user)
        return Response( data: {
            'refresh': str(refresh),
            'access': str(refresh.access_token),
        }, status=status.HTTP_200_OK)
```

Figure 5.38: Login View

Since our server-side application is stateless, users must provide credentials to access the required resources. We employ a JWT-based authentication strategy, encoding the userId in the token. We issue an access token as the primary means of

authentication. Additionally, we provide a refresh token, which allows users to generate a new access token when the original one expires.

```python
@api_view(['GET'])
def attempt_validate_user(request):
    access_token = request.headers.get('Authorization')
    payload = validate_token(access_token)

    # Check if the user is authorized to access this route using the access token.

    if payload is None:
        return Response( data: {"message": "Unauthorized to access this route", "errors": {}, "data": {}},
                        status=status.HTTP_401_UNAUTHORIZED)

    user_id = payload.get('user_id')

    is_validated = Profile.objects.filter(Q(user=user_id) & Q(is_validated=True)).exists()

    # Check if the user is already validated to access the application.

    if is_validated:
        return Response( data: {"message": "User is already validated", "errors": {}, "data": {}},
                        status=status.HTTP_400_BAD_REQUEST)

    # Check if the user has issued an otp in the timespan of 60 seconds starting from the moment he initially issued it.
    last_otp = OTP.objects.filter(Q(user=user_id)).order_by('-issued_at').first()

    if last_otp is not None:
        remaining_seconds = (timezone.now() - last_otp.issued_at).total_seconds()
        if remaining_seconds < 60:
            return Response({"message": "An OTP has been generated for the specified user and hasn't been expired yet",
                             "errors": {}, "data": {}})
```

Figure 5.39: Generate OTP View

Registering and signing into our application doesn't automatically grant users access to our services. We validate users by sending an OTP to their email, which must be used within a set expiry time. To ensure secure communication, each OTP is accompanied by a unique GUID, ensuring the OTP can only be validated on the same device it was requested from.

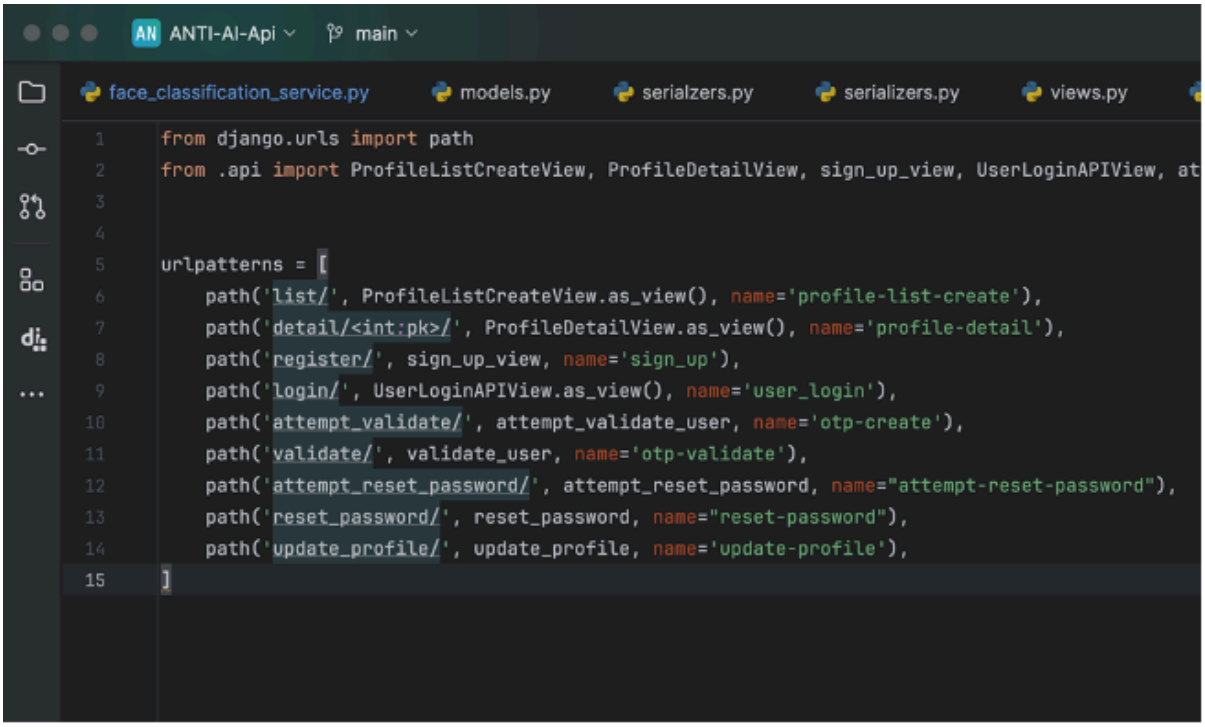


Figure 5.40: Routes file

The urls.py file intercepts incoming server requests by inspecting the HTTP method's header to determine the request path. This path identification enables the Django application to direct the request to the appropriate action in the views.py file.

```
4 usages  Nooraldin2001 *
class FaceClassifierService:

    2 usages  Nooraldin2001 *
    def model_predict(self, image_bytes):
        # Open and preprocess the image
        image = Image.open(io.BytesIO(image_bytes))
        image = image.resize((256, 256))  # Adjust according to your model's input size
        image = np.array(image) / 255.0  # Normalize if required

        # Convert image to list and serialize as JSON
        data = {
            'image': image.tolist()  # Convert numpy array to list
        }

        # Send the POST request with JSON data
        response = requests.post(API_URL, headers=headers, data=json.dumps(data))

        # Check for response and return predictions
        if response.status_code == 200:
            return response.json()

        response.raise_for_status()
```

Figure 5.41: Face-Classifier Service

The model_predict method in the FaceClassifier service is designed to assess whether an image is AI-generated. It takes an image as an argument and makes a POST request to an API that houses the Face Classification model. This API analyzes the image and returns a classification result, indicating whether the photo is authentic or generated by artificial intelligence.

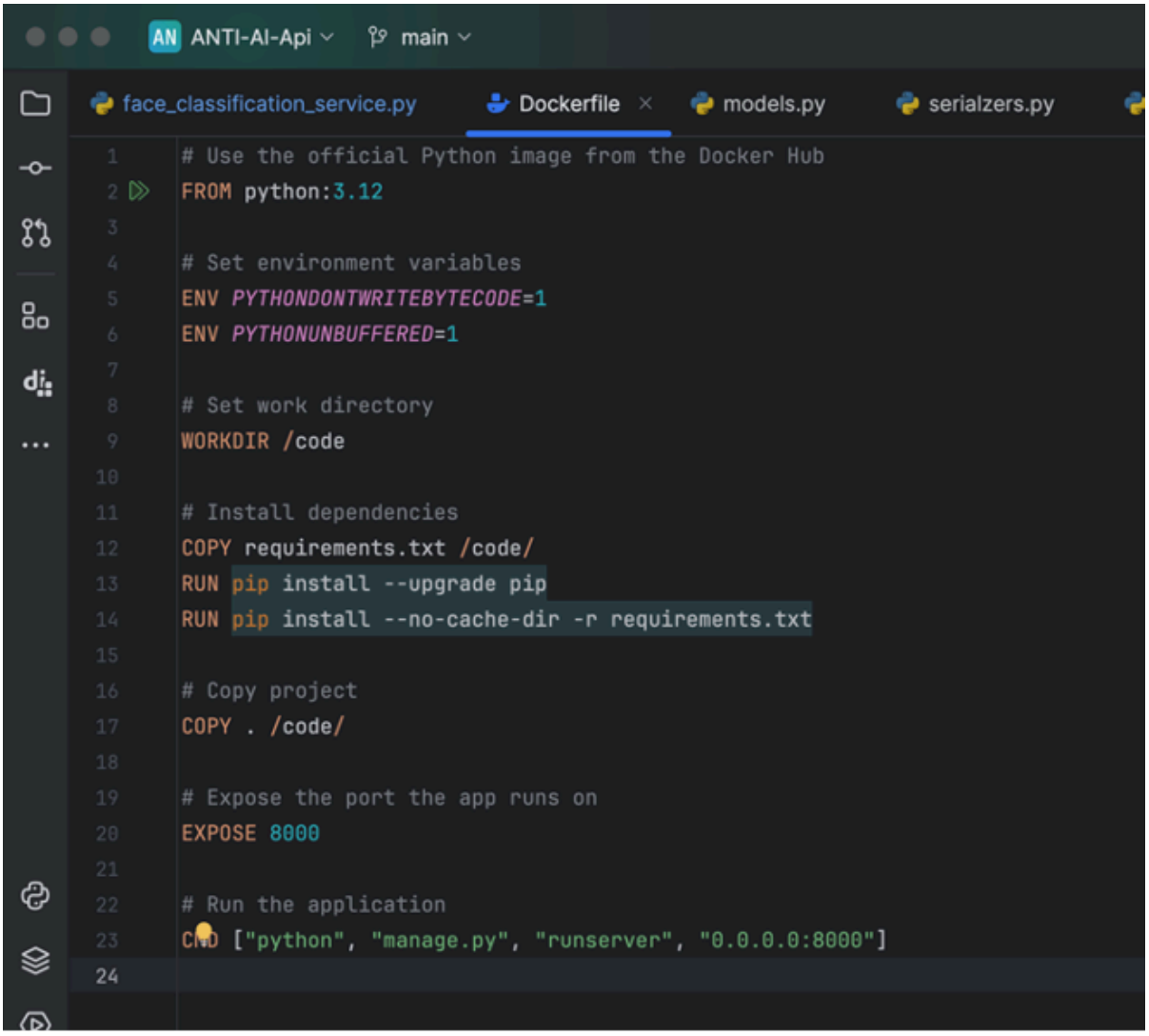


Figure 5.42: Dockerfile

We've utilized Docker to ease the deployment of our application on Azure. Docker ensures a consistent environment both locally and on Azure's servers. The Dockerfile in the root directory builds our custom Django application template. It copies all necessary files and folders into a newly created code directory within the container, excluding unnecessary files. Dependencies listed in requirements.txt are installed using the pip install command, leveraging the pre-installed pip from the

python:3.12 base image. The CMD command in the Dockerfile executes the initial and primary command when creating a Docker container from this image.

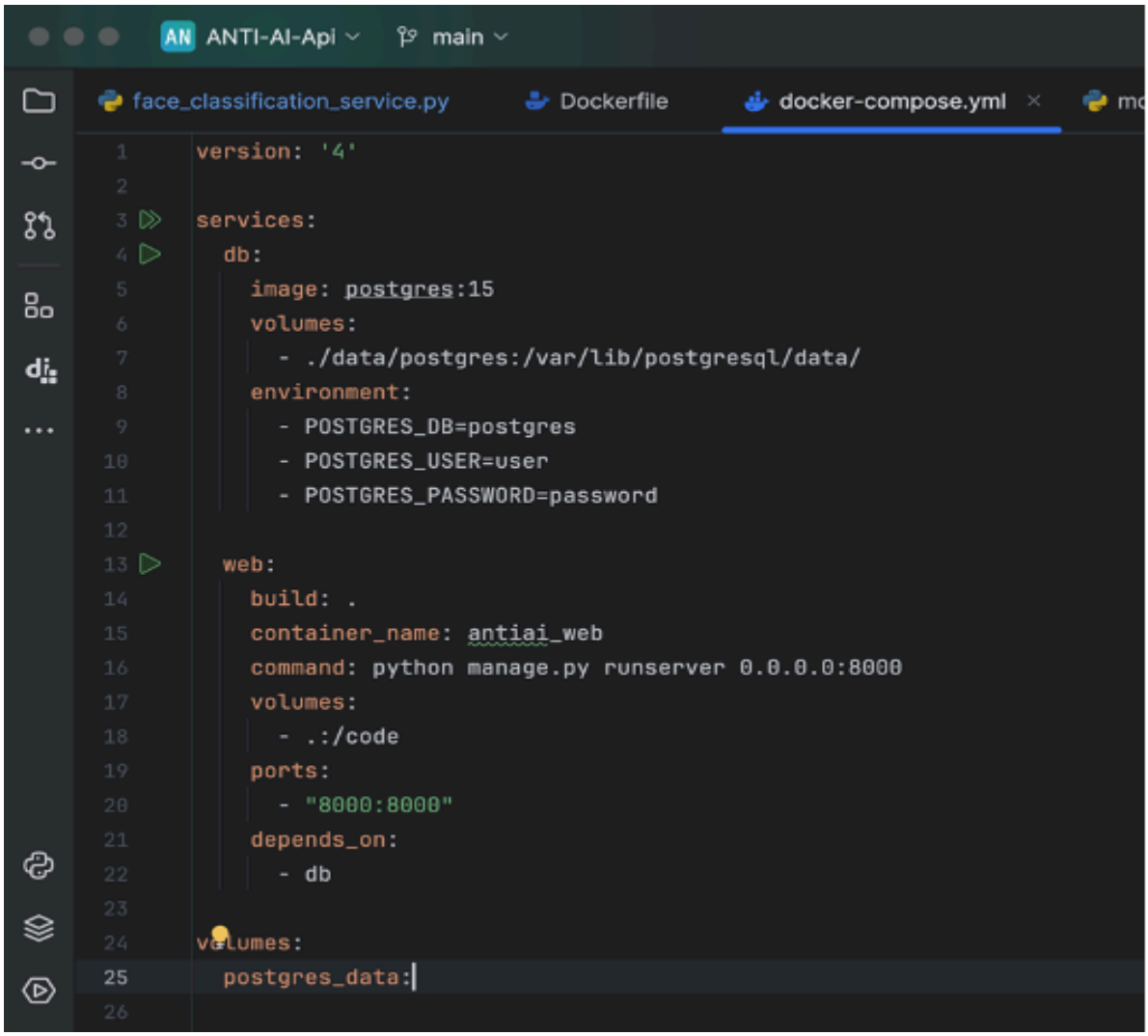


Figure 5.43: Docker Compose file

The Docker Compose file allows us to run multiple containers from specified images. In our application, we use Docker Compose to run a container based on the template outlined in Figure 5.42. Additionally, we configure necessary settings

such as exposing ports from the containerized application to the host machine and mapping volumes to ensure data persistence.

Application code (flutter)

- Splash screen

```dart
import 'dart:async';

import 'package:anti_ai_project/screens/onboardingscreen.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:flutter_svg/svg.dart';

class Splash_Screen extends StatefulWidget {
  const Splash_Screen({super.key});

  @override
  State<Splash_Screen> createState() => _Splash_ScreenState();
}

class _Splash_ScreenState extends State<Splash_Screen> {
  @override
  void initState() {
    Timer(const Duration(seconds: 3), () {
      Navigator.push(context,
          MaterialPageRoute(builder: (context) => const OnboardingScreen()));
    }); // Timer

    super.initState();
  }
```

Fig 5.44

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: const Color(0xff212131),
    body: Stack(
      children: [
        Padding(
          padding: const EdgeInsets.only(top: 0, left: 10),
          child: Center(
            child: FractionallySizedBox(
              widthFactor: 5,
              heightFactor: 1,
              child: SvgPicture.asset(
                'assets/only-logo.svg',
                fit: BoxFit.contain,
                color: const Color(0xFFFFFFFF).withOpacity(0.03),
              ), // SvgPicture.asset
            ), // FractionallySizedBox
          ), // Center
        ), // Padding
        Align(
          alignment: Alignment.center,
          child: Padding(
            padding: const EdgeInsets.all(0.20),
            child: SvgPicture.asset(
              'assets/anti-ai logo.svg',
              height: 350.h,
              width: 300.w,
            ), // SvgPicture.asset
          ), // Padding
        ), // Align
      ],
    ), // Stack
  ); // Scaffold
}
```

Fig 5.45

- **Registration**
    - **login screen**

In this screen an exist user enter his email and password and we check if his data is valid he login to our App if his data invalid display message invalid data

```
class Login_Screen extends StatefulWidget {
  const Login_Screen({super.key});

  @override
  State<Login_Screen> createState() => _Login_ScreenState();
}

class _Login_ScreenState extends State<Login_Screen> {
  final _formSignInkey = GlobalKey<FormState>();
  bool _passwordVisible = false;

  bool rememberPassword = true;
  TextEditingController emailcontroller = TextEditingController();
  TextEditingController passwordcontroller = TextEditingController();

  @override
  Widget build(BuildContext context) {
    //     SystemChrome.setSystemUIOverlayStyle(
    //     SystemUiOverlayStyle(
    //      statusBarBrightness: Brightness.dark,
    //      statusBarColor: Colors.transparent,
```

Fig 5.46

- **Signup screen**

In This screen the user fill his email,password and confirm password and go to otp

```dart
class Signup_Screen extends StatefulWidget {
  const Signup_Screen({super.key});

  @override
  State<Signup_Screen> createState() => _Signup_ScreenState();
}

class _Signup_ScreenState extends State<Signup_Screen> {
  final _formSignUpkey = GlobalKey<FormState>();
  bool _passwordVisible = false;
  bool isChecked = false;
  TextEditingController emailcontroller = TextEditingController();
  TextEditingController passwordcontroller = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xff5D71A0),
      appBar: AppBar(
        backgroundColor: Color(0xff5D71A0),
        elevation: 0,
        leading: IconButton(
          icon: Icon(
            Icons.arrow_back,
            color: Colors.white,
          ), // Icon
          onPressed: () {
            Navigator.pushReplacement(
              context,
              MaterialPageRoute(builder: (context) => Login_Screen()),
            );
```

Fig 5.47

OTP Screen

In this screen user enter otp sent to whom by Email

```
class OTPScreen extends StatefulWidget {
  const OTPScreen({super.key});

  @override
  State<OTPScreen> createState() => _OTPScreenState();
}

class _OTPScreenState extends State<OTPScreen> {
  final _formSignInkey = GlobalKey<FormState>();
  static const countdownDuration = 50;
  int _secondsRemaining = countdownDuration;
  Timer? _timer;

  @override
  void initState() {
    super.initState();
    startTimer();
  }

  void startTimer() {
    _timer = Timer.periodic(Duration(seconds: 1), (_) {
      setState(() {
        if (_secondsRemaining > 0) {
          _secondsRemaining--;
        } else {
          _timer?.cancel();
```

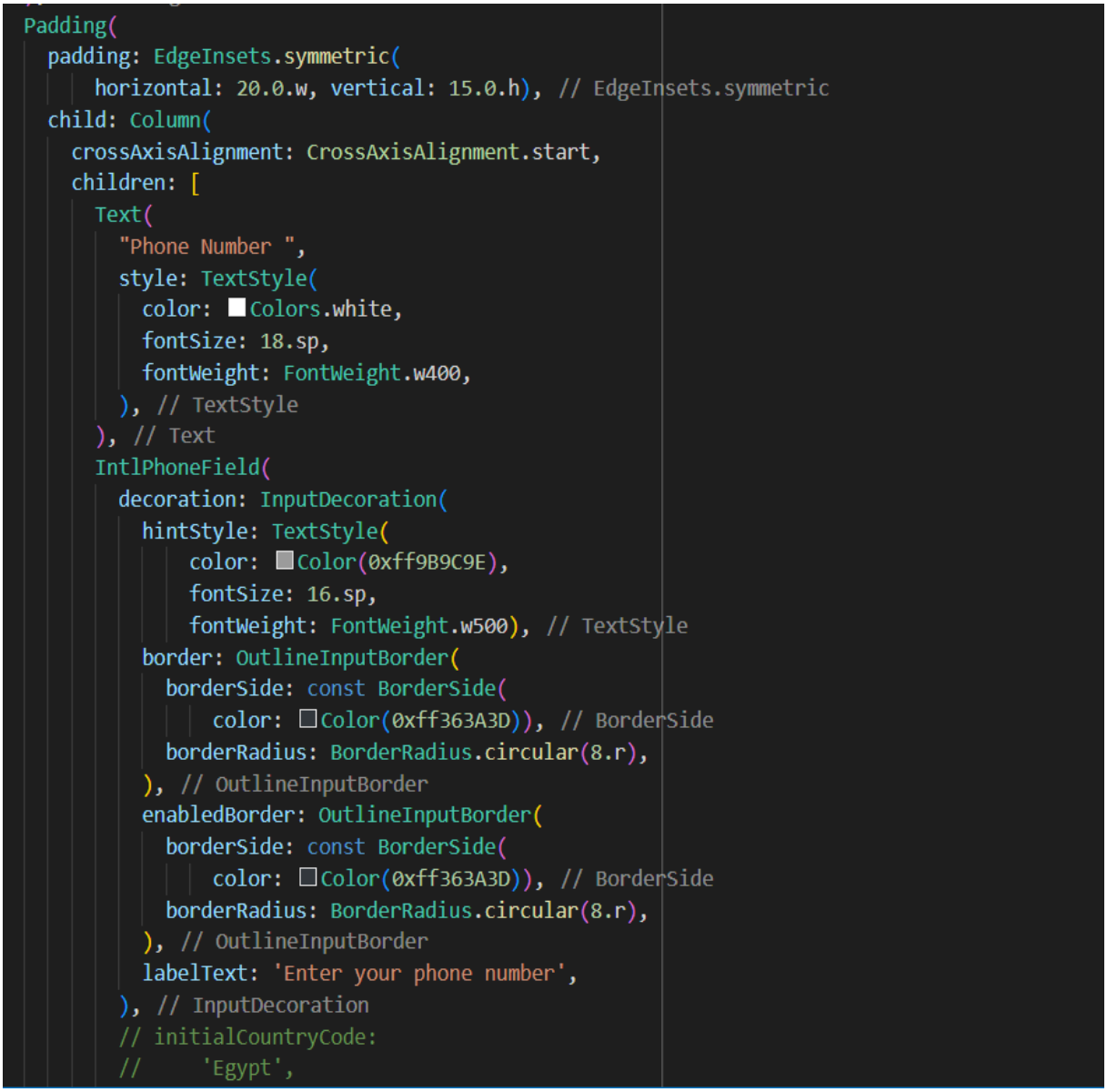
Fig 5.48

fillyourinfo

In this screen user after verify otp he enter his username and phone number

```
Padding(
  padding: EdgeInsets.symmetric(
      horizontal: 20.0.w, vertical: 15.0.h), // EdgeInsets.symmetric
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        "Phone Number ",
        style: TextStyle(
          color: Colors.white,
          fontSize: 18.sp,
          fontWeight: FontWeight.w400,
        ), // TextStyle
      ), // Text
      IntlPhoneField(
        decoration: InputDecoration(
          hintStyle: TextStyle(
              color: Color(0xff9B9C9E),
              fontSize: 16.sp,
              fontWeight: FontWeight.w500), // TextStyle
          border: OutlineInputBorder(
            borderSide: const BorderSide(
                color: Color(0xff363A3D)), // BorderSide
            borderRadius: BorderRadius.circular(8.r),
          ), // OutlineInputBorder
          enabledBorder: OutlineInputBorder(
            borderSide: const BorderSide(
                color: Color(0xff363A3D)), // BorderSide
            borderRadius: BorderRadius.circular(8.r),
          ), // OutlineInputBorder
          labelText: 'Enter your phone number',
        ), // InputDecoration
        // initialCountryCode:
        //     'Egypt',
```

Fig 5.49



```
Padding(
  padding: EdgeInsets.symmetric(
      horizontal: 20.0.w, vertical: 15.0.h), // EdgeInsets.symmetric
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        "Phone Number ",
        style: TextStyle(
          color: Colors.white,
          fontSize: 18.sp,
          fontWeight: FontWeight.w400,
        ), // TextStyle
      ), // Text
      IntlPhoneField(
        decoration: InputDecoration(
          hintStyle: TextStyle(
              color: Color(0xff9B9C9E),
              fontSize: 16.sp,
              fontWeight: FontWeight.w500), // TextStyle
          border: OutlineInputBorder(
            borderSide: const BorderSide(
                color: Color(0xff363A3D)), // BorderSide
            borderRadius: BorderRadius.circular(8.r),
          ), // OutlineInputBorder
          enabledBorder: OutlineInputBorder(
            borderSide: const BorderSide(
                color: Color(0xff363A3D)), // BorderSide
            borderRadius: BorderRadius.circular(8.r),
          ), // OutlineInputBorder
          labelText: 'Enter your phone number',
        ), // InputDecoration
        // initialCountryCode:
        //     'Egypt',
```

Fig 5.50

Forget password

If The user forget password we sent otp to his email

```dart
import 'package:anti_ai_project/screens/registration/fortgetpassverify.dart';
import 'package:anti_ai_project/screens/registration/login_screen.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:flutter_svg/svg.dart';

class ForgetPassword_Screen extends StatefulWidget {
  const ForgetPassword_Screen({super.key});

  @override
  State<ForgetPassword_Screen> createState() => _ForgetPassword_ScreenState();
}

class _ForgetPassword_ScreenState extends State<ForgetPassword_Screen> {
  final _formSignInkey = GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xff5D71A0),
      appBar: AppBar(
        backgroundColor: Color(0xff5D71A0),
        elevation: 0,
        leading: IconButton(
          icon: Icon(
            Icons.arrow_back,
            color: Colors.white,
          ), // Icon
          onPressed: () {
            Navigator.pushReplacement(
              context,
              MaterialPageRoute(builder: (context) => Login_Screen()),
            );
          },
        ), // IconButton
```

Fig 5.51

Forget password

In this screen user enter otp sent to whom by Email

```dart
import 'package:anti_ai_project/screens/registration/fortgetpassverify.dart';
import 'package:anti_ai_project/screens/registration/login_screen.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:flutter_svg/svg.dart';

class ForgetPassword_Screen extends StatefulWidget {
  const ForgetPassword_Screen({super.key});

  @override
  State<ForgetPassword_Screen> createState() => _ForgetPassword_ScreenState();
}

class _ForgetPassword_ScreenState extends State<ForgetPassword_Screen> {
  final _formSignInkey = GlobalKey<FormState>();

@override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xff5D71A0),
      appBar: AppBar(
        backgroundColor: Color(0xff5D71A0),
        elevation: 0,
        leading: IconButton(
          icon: Icon(
            Icons.arrow_back,
            color: Colors.white,
          ), // Icon
          onPressed: () {
            Navigator.pushReplacement(
              context,
              MaterialPageRoute(builder: (context) => Login_Screen()),
            );
          },
        ), // IconButton
```

Fig 5.52

Terms & Conditions

```dart
import 'package:anti_ai_project/screens/registration/signup_screen.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';

class TremsAndCondition extends StatefulWidget {
  const TremsAndCondition({super.key});

  @override
  State<TremsAndCondition> createState() => _TremsAndConditionState();
}

class _TremsAndConditionState extends State<TremsAndCondition> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xff212131),
      appBar: AppBar(
        automaticallyImplyLeading: false,
        backgroundColor: Color(0xff37425E80),
        title: const Text(
          'Terms & Policy',
          style: TextStyle(
            color: Colors.white,
            fontSize: 22,
            fontWeight: FontWeight.bold,
          ), // TextStyle
          textAlign: TextAlign.center,
        ), // Text
        centerTitle: true,
        toolbarHeight: 100,
      ), // AppBar
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
```

Fig 5.53

Home Screen

home_screen_all

Home screen is the core of App contain community , alert , profile and uploads

```
final List<Widget> pages = [
  Home_Page(),
  Community_Screen(),
  DetectFakeImage(),
  AlertScreen(),
  Profile_Screen()
];
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Color(0xff212131),
    body: pages[_currentIndex],
    bottomNavigationBar: BottomNavigationBar(
      currentIndex: _currentIndex,
      onTap: (index) {
        if (index == 2) {
          _showBottomSheet(context);
        } else {
          setState(() {
            _currentIndex = index;
          });
        }
      },
      type: BottomNavigationBarType.fixed,
      selectedItemColor: Colors.white,
      unselectedItemColor: Colors.grey,
      iconSize: 30,
      backgroundColor: Color(0xFF181825),
      items: [
```

```
      items: [
        BottomNavigationBarItem(
          icon: Icon(Icons.home),
          label: "Home",
        ), // BottomNavigationBarItem
        BottomNavigationBarItem(
          icon: Icon(Icons.people),
          label: "Community",
        ), // BottomNavigationBarItem
        BottomNavigationBarItem(
          icon: Padding(
            padding: const EdgeInsets.only(bottom: 0.0),
            child: Icon(
              Icons.add_circle,
              color: Color(0xff66B2FF),
              size: 58,
            ), // Icon
          ), // Padding
          label: "",
        ), // BottomNavigationBarItem
        BottomNavigationBarItem(
          icon: Icon(Icons.notifications),
          label: " Alert",
        ), // BottomNavigationBarItem
        BottomNavigationBarItem(
          icon: Icon(Icons.person),
          label: "Profile",
        ), // BottomNavigationBarItem
      ],
    ), // BottomNavigationBar
  ); // Scaffold
}
```

3.2 all uploads

Contain all uploads (fake and real images)

```dart
import 'package:anti_ai_project/screens/home_screen_all.dart';
import 'package:anti_ai_project/screens/home_screens/fake_uploads.dart';
import 'package:anti_ai_project/screens/home_screens/real_uploads.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';

class All_Uploads extends StatefulWidget {
  const All_Uploads({super.key});

  @override
  State<All_Uploads> createState() => _All_UploadsState();
}

class _All_UploadsState extends State<All_Uploads> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xff212131),
      appBar: AppBar(
        elevation: 0,
        leading: IconButton(
          icon: Icon(
            Icons.arrow_back,
            color: Colors.white,
          ), // Icon
          onPressed: () {
            Navigator.pushReplacement(
              context,
              MaterialPageRoute(builder: (context) => All_Screen()),
            );
          },
        ), // IconButton
        automaticallyImplyLeading: false,
        backgroundColor: Color(0xff37425E80),
        title: const Text(
```

Fig 5.55

fake_uploads

It contains fake images which the model has detect

```dart
import 'package:anti_ai_project/screens/home_screen_all.dart';
import 'package:anti_ai_project/screens/home_screens/all_uploads.dart';
import 'package:anti_ai_project/screens/home_screens/real_uploads.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';

class Fake_Uploads extends StatefulWidget {
  const Fake_Uploads({super.key});

  @override
  State<Fake_Uploads> createState() => _Fake_UploadsState();
}

class _Fake_UploadsState extends State<Fake_Uploads> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xff212131),
      appBar: AppBar(
        elevation: 0,
        leading: IconButton(
          icon: Icon(
            Icons.arrow_back,
            color: Colors.white,
          ), // Icon
          onPressed: () {
            Navigator.pushReplacement(
              context,
              MaterialPageRoute(builder: (context) => All_Screen()),
            );
          },
        ), // IconButton
        automaticallyImplyLeading: false,
        backgroundColor: Color(0xff37425E80),
        title: const Text(
```

Ln 1

Fig 5.57

Real_uploads

It’s contains real images which the model has been detect .

```dart
import 'package:anti_ai_project/screens/home_screen_all.dart';
import 'package:anti_ai_project/screens/home_screens/all_uploads.dart';
import 'package:anti_ai_project/screens/home_screens/fake_uploads.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';

class Real_Uploads extends StatefulWidget {
  const Real_Uploads({super.key});

  @override
  State<Real_Uploads> createState() => _Real_UploadsState();
}

class _Real_UploadsState extends State<Real_Uploads> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xff212131),
      appBar: AppBar(
        elevation: 0,
        leading: IconButton(
          icon: Icon(
            Icons.arrow_back,
            color: Colors.white,
          ), // Icon
          onPressed: () {
            Navigator.pushReplacement(
              context,
              MaterialPageRoute(builder: (context) => All_Screen()),
            );
          },
        ), // IconButton
        automaticallyImplyLeading: false,
        backgroundColor: Color(0xff37425E80),
        title: const Text(
          'My Uploads',
```

Fig 5.58

Upload_screen

detect fake image

In this screen user upload image or Url and the image send to model to detect it

```dart
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';

import '../home_screen_all.dart';

class DetectFakeImage extends StatefulWidget {
  const DetectFakeImage({Key? key}) : super(key: key);

  @override
  State<DetectFakeImage> createState() => _DetectFakeImageState();
}

class _DetectFakeImageState extends State<DetectFakeImage> {
  bool _isChecked = false;
  int _timer = 2;
  int _clickCount = 2;

  Timer? _countdownTimer;

  @override
  void dispose() {
    _countdownTimer?.cancel();
    super.dispose();
  }

  void _startTimer() {
    _countdownTimer = Timer.periodic(Duration(seconds: 1), (timer) {
      setState(() {
        if (_timer > 0) {
          _timer--;
        } else {
          _countdownTimer?.cancel();
        }
      });
    }); // Timer.periodic
  }
```

Fig 5.59

```dart
import 'package:anti_ai_project/screens/upload_screen/resultaudio.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import '../home_screen_all.dart';

class DetectFakeAudio extends StatefulWidget {
  const DetectFakeAudio({Key? key}) : super(key: key);

  @override
  State<DetectFakeAudio> createState() => _DetectFakeAudioState();
}

class _DetectFakeAudioState extends State<DetectFakeAudio> {
  bool _isChecked = false;

  void showUploadDialog(BuildContext context) {
    showDialog(
      context: context,
      builder: (BuildContext context) {
        return FractionallySizedBox(
          widthFactor: 500,
          child: AlertDialog(
            iconColor: Color(0xff525D8387),
            title: Text('URL:'),
            actions: [
              Container(
                color: Color(0xff1A1D21),
                child: TextFormField(
                  decoration: InputDecoration(
                    hintStyle: const TextStyle(
                      color: Color(0xff9B9C9E),
                    ), // TextStyle
                    border: OutlineInputBorder(
                      borderSide: const BorderSide(
                        color: Color(0xff363A3D),
```

Fig 5.60

Result image screen

After detecting image and classifying it, It will print The result of detection.

```
void _uploadImage() async {
  final picker = ImagePicker();
  PickedFile? pickedFile = await picker.getImage(source: ImageSource.gallery);

  if (pickedFile != null) {
    setState(() {
      _image = File(pickedFile.path);
      _isImageSelected = true;
    });

    print(pickedFile.path);
  } else {}
}
```

Fig 5.61

Alert_screens

Alert

In This screen all notifications will be displayed with title , time and description

```dart
import 'package:anti_ai_project/screens/alert_screens/followrequest.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';

class NotificationObject {
  final String title;
  final String time;
  final String description;

  NotificationObject({
    required this.title,
    required this.time,
    required this.description,
  });
}

class AlertScreen extends StatefulWidget {
  const AlertScreen({Key? key}) : super(key: key);

  @override
  State<AlertScreen> createState() => _AlertScreenState();
}
```

Fig 5.62

Follow request

There are all follow requests

```dart
import 'package:anti_ai_project/screens/alert_screens/followrequest2.dart';
import 'package:anti_ai_project/screens/home_screen_all.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';

class FollowRequestScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xff212131),
      appBar: AppBar(
        elevation: 0,
        leading: IconButton(
          icon: Icon(
            Icons.arrow_back,
            color: Colors.white,
          ), // Icon
          onPressed: () {
            Navigator.pushReplacement(
              context,
              MaterialPageRoute(builder: (context) => All_Screen()),
            );
          },
        ), // IconButton
        automaticallyImplyLeading: false,
        backgroundColor: Color(0xff37425E80),
        title: const Text(
          'Follow requests',
          style: TextStyle(
            color: Colors.white,
            fontSize: 26,
            fontWeight: FontWeight.bold,
          ), // TextStyle
          textAlign: TextAlign.center,
        ), // Text
```

Fig 5.63

Profile_screens

About us

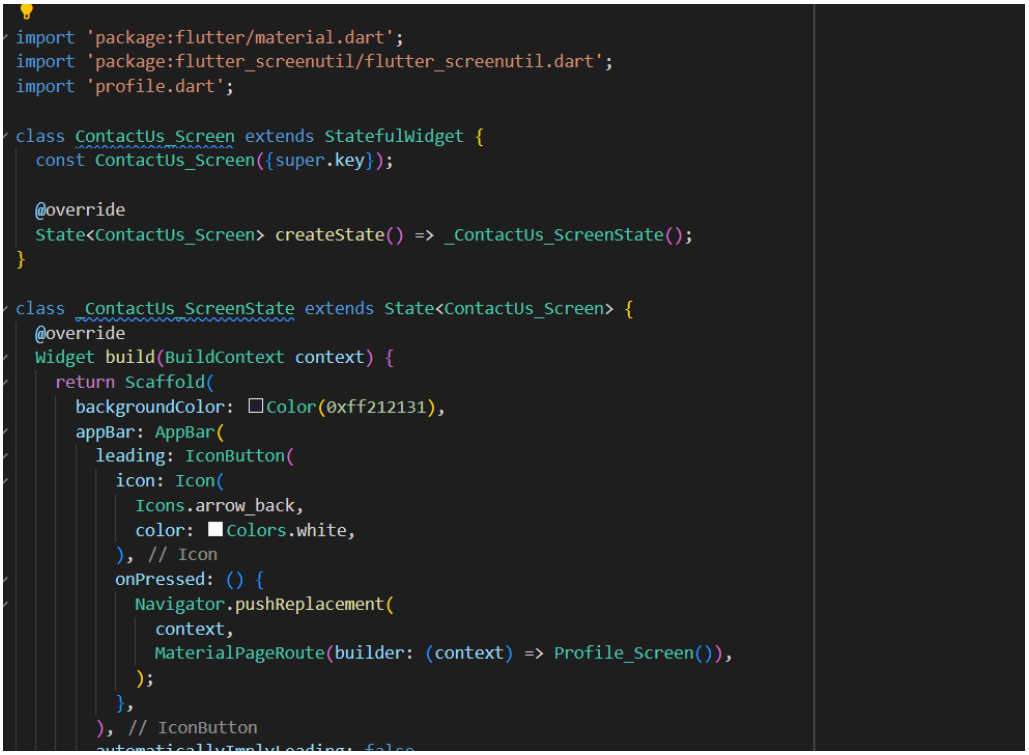
In This screen ,There is information about us .

```dart
import 'package:anti_ai_project/screens/profile_screens/profile.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';

class AboutUs_Screen extends StatefulWidget {
  const AboutUs_Screen({super.key});

  @override
  State<AboutUs_Screen> createState() => _AboutUs_ScreenState();
}

class _AboutUs_ScreenState extends State<AboutUs_Screen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xff212131),
      appBar: AppBar(
        leading: IconButton(
          icon: Icon(
            Icons.arrow_back,
            color: Colors.white,
          ), // Icon
          onPressed: () {
            Navigator.pushReplacement(
              context,
              MaterialPageRoute(builder: (context) => Profile_Screen()),
            );
          },
        ), // IconButton
        automaticallyImplyLeading: false,
        backgroundColor: Color(0xff37425E80),
        title: const Text(
          'About Us',
          style: TextStyle(
            color: Colors.white,
```

Fig 5.64

Contact us

User can contact us if he has any problem .



```dart
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'profile.dart';

class ContactUs_Screen extends StatefulWidget {
  const ContactUs_Screen({super.key});

  @override
  State<ContactUs_Screen> createState() => _ContactUs_ScreenState();
}

class _ContactUs_ScreenState extends State<ContactUs_Screen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xff212131),
      appBar: AppBar(
        leading: IconButton(
          icon: Icon(
            Icons.arrow_back,
            color: Colors.white,
          ), // Icon
          onPressed: () {
            Navigator.pushReplacement(
              context,
              MaterialPageRoute(builder: (context) => Profile_Screen()),
            );
          },
        ), // IconButton
        automaticallyImplyLeading: false
```

Profile

It contains information about user

```dart
import 'package:anti_ai_project/screens/home_screen_all.dart';
import 'package:anti_ai_project/screens/registration/login_screen.dart';
import 'package:anti_ai_project/screens/profile_screens/aboutus.dart';
import 'package:anti_ai_project/screens/profile_screens/contactus.dart';
import 'package:anti_ai_project/screens/profile_screens/setting&privacy.dart';
import 'package:anti_ai_project/screens/profile_screens/terms&policy.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:flutter_svg/svg.dart';

class Profile_Screen extends StatefulWidget {
  const Profile_Screen({super.key});

  @override
  State<Profile_Screen> createState() => _Profile_ScreenState();
}

class _Profile_ScreenState extends State<Profile_Screen> {
  final _formSignInkey = GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Color(0xff212131),
        leading: IconButton(
          icon: Icon(
            Icons.arrow_back,
            color: Colors.white,
            size: 25,
          ), // Icon
          onPressed: () {
            Navigator.pushReplacement(
              context,
              MaterialPageRoute(builder: (context) => All_Screen()),
```

Fig 5.66

Setting & privacy

```dart
import 'package:anti_ai_project/screens/profile_screens/profile.dart';
import 'package:anti_ai_project/screens/profile_screens/resetpass.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';
import 'package:flutter_svg/svg.dart';

class SettingAndPrivacy_Screen extends StatefulWidget {
  const SettingAndPrivacy_Screen({super.key});

  @override
  State<SettingAndPrivacy_Screen> createState() =>
      _SettingAndPrivacy_ScreenState();
}

class _SettingAndPrivacy_ScreenState extends State<SettingAndPrivacy_Screen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xff212131),
      appBar: AppBar(
        elevation: 0,
        leading: IconButton(
          icon: Icon(
            Icons.arrow_back,
            color: Colors.white,
          ), // Icon
          onPressed: () {
            Navigator.pushReplacement(
              context,
              MaterialPageRoute(builder: (context) => Profile_Screen()),
            );
          },
        ), // IconButton
```

Fig 5.67

Terms & Policy

```dart
import 'package:anti_ai_project/screens/profile_screens/profile.dart';
import 'package:flutter/material.dart';
import 'package:flutter_screenutil/flutter_screenutil.dart';

class TremsAndPolicy extends StatefulWidget {
  const TremsAndPolicy({super.key});

  @override
  State<TremsAndPolicy> createState() => _TremsAndPolicyState();
}

class _TremsAndPolicyState extends State<TremsAndPolicy> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Color(0xff212131),
      appBar: AppBar(
        leading: IconButton(
          icon: Icon(
            Icons.arrow_back,
            color: Colors.white,
          ), // Icon
          onPressed: () {
            Navigator.pushReplacement(
              context,
              MaterialPageRoute(builder: (context) => Profile_Screen()),
            );
          },
        ), // IconButton
        automaticallyImplyLeading: false,
        backgroundColor: Color(0xff37425E80),
        title: const Text(
          'Terms & Policy',
          style: TextStyle(
            color: Colors.white,
```

Fig 5.68

community

- **community**

```
class Community_Screen extends StatefulWidget {
  const Community_Screen({super.key});

  @override
  _CommunityScreenState createState() => _CommunityScreenState();
}

class _CommunityScreenState extends State<Community_Screen>
    with SingleTickerProviderStateMixin {
  late TabController _tabController;

  @override
  void initState() {
    super.initState();
    _tabController = TabController(length: 4, vsync: this);
  }

  @override
  void dispose() {
    _tabController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Community'),
        bottom: TabBar(
          controller: _tabController,
          tabs: const [
            Tab(text: 'Posts'),
            Tab(text: 'Reacts'),
            Tab(text: 'Replies'),
            Tab(text: 'Votes'),
          ],
        ), // TabBar
```

Fig 5.69

posts_ screen

```dart
import 'package:flutter/material.dart';

class PostsScreen extends StatelessWidget {
  const PostsScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return const Center(
      child: Text('Posts Screen'),
    ); // Center
  }
}
```

Fig 5.70

reacts_screen

```dart
import 'package:flutter/material.dart';

class ReactsScreen extends StatelessWidget {
  const ReactsScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return const Center(
      child: Text('Reacts Screen'),
    ); // Center
  }
}
```

Fig 5.71

replies_screen

```dart
import 'package:flutter/material.dart';

class RepliesScreen extends StatelessWidget {
  const RepliesScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return const Center(
      child: Text('Replies Screen'),
    ); // Center
  }
}
```

Fig 5.72

votes_screen

```dart
import 'package:flutter/material.dart';

class VotesScreen extends StatelessWidget {
  const VotesScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return const Center(
      child: Text('Votes Screen'),
    ); // Center
  }
}
```

Fig 5.73

Api service

User login

```
class ApiService {
  final String baseUrl = "http://antiaiapp.azurewebsites.net/api";

  Future<List<dynamic>> userLogin(username, password) async {
    final response =
        await http.post(Uri.parse("$baseUrl/profiles/login/"), body: {
      "username": username,
      "password": password,
    });

    if (response.statusCode == 200) {
      return [true, jsonEncode(response.body)];
    }

    return [false, []];
  }
```

Fig 5.74

Sign up

```
Future<List<dynamic>> signup(
    username, email, password, password_confirm) async {
  final response =
      await http.post(Uri.parse("$baseUrl/profiles/register/"), body: {
    "username": username,
    "email": email,
    "password": password,
    "password_confirm": password_confirm,
  });
  if (response.statusCode == 200) {
    return [true, jsonEncode(response.body)];
  }

  return [false, []];
}
```

Fig 5.75

Otp_validate

```
Future<List<dynamic>> otp_validate(issued_for, otp) async {
  final response =
      await http.post(Uri.parse("$baseUrl/profiles/validate/"), body: {
    "issued_for": issued_for,
    "otp": otp,
  });
  if (response.statusCode == 200) {
    return [true, jsonEncode(response.body)];
  }

  return [false, []];
}
```

Fig 5.75

Attempt_validate

```
Future<List<dynamic>> Attempt_validate(username, password) async {
  final response =
      await http.get(Uri.parse("$baseUrl/profiles/attempt_validate/"), body: {
    "username": username,
    "password": password,
  });
  if (response.statusCode == 200) {
    return [true, jsonEncode(response.body)];
  }

  return [false, []];
}
```

Fig 5.76

profile_list

```
Future<List<dynamic>> profile_list() async {
  final response = await http.get(
    Uri.parse("$baseUrl/profiles/list/"),
  );
  if (response.statusCode == 200) {
    return [true, []];
  }

  return [false, []];
}
```

Fig 5.77

Profile detail

```
Future<List<dynamic>> profiles_detail() async {
  final response = await http.get(
    Uri.parse("$baseUrl/profiles/detail"),
  );
  if (response.statusCode == 200) {
    return [true, []];
  }

  return [false, []];
}
```

Fig 5.78

profile _Attempt_rest_password

```
Future<List<dynamic>> profile_Attempt_rest_password() async {
    final response = await http.put(
     Uri.parse("$baseUrl/profiles/attempt_reset_password/"),
 );
 }
```

Fig 5.79

profile_Rest_password

```
Future<List<dynamic>> profile_rest_password(
    email, issued_for, password, password_confirm, otp) async {
  final response =
      await http.put(Uri.parse("$baseUrl/profiles/reset_password/"), body: {
    "email": email,
    "issued_for": issued_for,
    "password": password,
    "password_confirm": password_confirm,
    "otp": otp,
  });
  if (response.statusCode == 200) {
    return [true, jsonEncode(response.body)];
  }

  return [false, []];
}
```

Fig 5.80

Upload post

```
Future<List<dynamic>> upload_post(
    classified_name, media_url, created_by_user, caption) async {
  final response =
      await http.post(Uri.parse("$baseUrl/posts/list_create/"), body: {
    "classified_name": classified_name,
    "media_url": media_url,
    "created_by_user": created_by_user,
    "caption": caption,
  });
  if (response.statusCode == 200) {
    return [true, jsonEncode(response.body)];
  }

  return [false, []];
}
```

Fig 5.81

post_list

```
Future<List<dynamic>> post_list() async {
  final response = await http.get(
    Uri.parse("$baseUrl/posts/list_create/"),
  );
  if (response.statusCode == 200) {
    return [true, []];
  }

  return [false, []];
}
```

Fig 5.82

post_by_uuid

```dart
Future<List<dynamic>> post_by_uuid() async {
  final response = await http.get(
    Uri.parse("$baseUrl/posts/c41ccaa3-e5fe-451d-a527-9b23bdf283c0/"),
  );
  if (response.statusCode == 200) {
    return [true, []];
  }

  return [false, []];
}
```

Fig 5.83

delete_post

```dart
Future<List<dynamic>> delete_post() async {
  final response = await http.delete(
    Uri.parse("$baseUrl/posts/c41ccaa3-e5fe-451d-a527-9b23bdf283c0/"),
  );
  if (response.statusCode == 200) {
    return [true, []];
  }

  return [false, []];
}
```

Fig 5.84

img_model_attempt

```
Future<List<dynamic>> img_model_attempt(file) async {
  final response =
      await http.post(Uri.parse("$baseUrl/img_model/img_models/"), body: {
    "file": file,
  });
  if (response.statusCode == 200) {
    return [true, jsonEncode(response.body)];
  }

  return [false, []];
}
```

Fig 5.85

5.4 System Testing

The system testing has the following objectives:

1. **Verify Functional Requirements**:
   - Ensure that all specified functional requirements are correctly implemented.
   - Test all features listed in the functional requirement table to confirm they work as expected.
2. **Validate Integration**:
   - Ensure that all modules and components integrate seamlessly and function together without issues.
   - Verify data flow and interaction between different parts of the system.
3. **Check System Performance**:
   - Evaluate the system's performance under various conditions, including load and stress testing.
   - Ensure the system can handle expected and peak user loads without degradation in performance.
4. **Ensure Security**:

- Test for potential security vulnerabilities, such as SQL injection, cross-site scripting (XSS), and unauthorized access.
- Verify that user data is protected and that privacy settings (such as approval/rejection of sharing uploads) are correctly enforced.

5. **Confirm Usability**:
   - Ensure the system is user-friendly and intuitive.
   - Validate that navigation, layout, and user interactions meet usability standards.
6. **Validate Data Accuracy and Integrity**:
   - Ensure that data input, processing, and output are accurate and consistent.
   - Test data validation, storage, retrieval, and display.
7. **Test Compatibility**:
   - Verify that the system works across different devices, operating systems, and browsers.
   - Ensure compatibility with various screen sizes and resolutions.
8. **Assess Reliability and Stability**:
   - Ensure the system operates reliably over time without crashes or unexpected behavior.
   - Test for system stability under continuous use.
9. **Verify Compliance**:
   - Ensure the system complies with relevant industry standards, regulations, and guidelines.
   - Validate adherence to accessibility standards, such as WCAG.
10. **Test Error Handling and Recovery**:
    - Ensure the system gracefully handles errors and recovers from failures without data loss or corruption.
    - Verify that appropriate error messages are displayed to users.
11. **Evaluate System Documentation**:
    - Ensure that user manuals, help guides, and technical documentation are accurate and comprehensive.
    - Validate that documentation is up-to-date with the current system version.
12. **Check Backup and Recovery**:

- Verify that the system's backup and recovery processes work correctly.
- Ensure data can be restored in case of a system failure or data loss.

13. **Confirm Scalability**:
    - Test the system's ability to scale up or down based on user demand and data growth.
    - Ensure the system can be expanded without major rework.

5.5 Results

The mobile application and website developed for AI detection of images have achieved significant milestones in accuracy, user engagement, and community impact. By fostering a community dedicated to digital media literacy and AI ethics, the project not only addresses immediate technological challenges but also contributes to broader societal awareness and education. Moving forward, the project is poised to evolve with advancements in AI technology and community-driven initiatives, continuing to play a pivotal role in combating digital misinformation and promoting digital trust.

Technological Outcomes

- **Detection Accuracy**: The AI models integrated into the application have achieved a commendable accuracy rate of 97%, effectively identifying AI-generated images from authentic ones.
- **Performance**: The application processes images with an average speed of 7 seconds per image, ensuring prompt analysis and detection feedback to users.
- **Scalability**: Designed with scalability in mind, the application infrastructure supports concurrent user interactions and can handle a growing dataset of images without compromising performance.

5.6 Goals Achieved

- **Successful AI Model Integration**: Integrated advanced AI models capable of detecting AI-generated images with high accuracy, achieving a recognition rate of 97%.
- **Scalability and Performance**: Designed and implemented a scalable backend infrastructure capable of efficiently processing image analysis requests, ensuring rapid response times and reliability even under high user loads.
- **Backend Development**: Developed robust backend APIs using modern frameworks like Flask/Django to facilitate seamless communication between the frontend application and AI detection models, ensuring optimal performance and flexibility.

Community and Educational Goals

- **Community Formation**: Established a dynamic community centered around digital media literacy and AI ethics, fostering constructive discussions, knowledge sharing, and awareness about the implications of AI-generated content.
- **Educational Impact**: Educated users about the prevalence and potential impacts of AI-generated content on digital trust and media authenticity, contributing to broader education in AI ethics and responsible digital citizenship.

Chapter 6

Chapter 6: Conclusion and Future Work

6.1 Conclusion

The development of a mobile application and website for detecting AI-generated images represents a significant step forward in the fight against digital misinformation and the proliferation of manipulated media. This documentation has outlined the various components and processes involved in creating this innovative tool, from the initial concept and design to the implementation and deployment phases.

Summary of Key Points

1. **Introduction**:
   - The necessity of detecting AI-generated images in today's digital age.
   - The potential impacts of undetected fake media on society.
2. **System Architecture**:
   - Detailed description of the system's architecture, including frontend and backend components.
   - Overview of the machine learning models used for detection.
3. **Development Process**:
   - Step-by-step explanation of the development stages.
   - Challenges faced and how they were addressed.
4. **User Interface and Experience**:
   - Design principles followed to ensure an intuitive and user-friendly interface.
   - Key features of the mobile application and website.
5. **Testing and Validation**:
   - Methods used to test the accuracy and reliability of the detection algorithms.
   - Results from the validation process and performance metrics.
6. **Deployment and Maintenance**:
   - Deployment strategies for both the mobile application and website.
   - Maintenance plans to ensure the system remains up-to-date and effective.
7. **Ethical Considerations**:
   - Discussion on the ethical implications of detecting AI-generated images.

- Measures taken to protect user privacy and data security.

The mobile application and website developed for detecting AI-generated images are powerful tools in the ongoing effort to combat digital misinformation. By leveraging advanced machine learning techniques and focusing on user accessibility, this project provides a practical solution for individuals and organizations seeking to verify the authenticity of visual content.

As technology continues to advance, the importance of such tools will only grow. Ongoing research, user feedback, and ethical considerations will play crucial roles in the evolution of this project. Together, we can create a safer and more trustworthy digital environment.

6.2 Future Work

The future work for the mobile application and website detecting AI-generated images encompasses a wide range of enhancements and expansions aimed at improving accuracy, usability, and scalability. By focusing on these areas, we can ensure that the tool remains effective and relevant in the rapidly evolving digital landscape. Ongoing research, user engagement, and ethical considerations will be crucial in guiding the future development of this project, ultimately contributing to a safer and more trustworthy digital environment.

As the landscape of AI-generated content continues to evolve, it is crucial to advance and expand the capabilities of our detection tools. Here are some key areas for our future work in the development of the mobile application and website :

1. Enhancing Detection Accuracy

- **Model Refinement**: Continuously improving the underlying machine learning models to enhance their accuracy and reduce false positives and negatives.
- **Data Augmentation**: Incorporating more diverse datasets, including different types of AI-generated images and real-world scenarios, to train the models more effectively.

- **Adversarial Training**: Implementing adversarial training techniques to make the models more robust against new and sophisticated AI generation methods.

2. Expanding Detection Capabilities

- **Multimodal Detection**: Extending the detection capabilities to include other types of media, such as audio and video, to provide a comprehensive solution for identifying AI-generated content.
- **Real-Time Detection**: Developing algorithms and optimizing processes to enable real-time detection of AI-generated content, enhancing user experience and utility.

3. User Experience and Interface Improvements

- **Personalization**: Implementing features that allow users to customize the detection process according to their preferences and needs.
- **User Feedback Loop**: Creating mechanisms for users to provide feedback on detection results, which can be used to further improve the models.
- **Educational Resources**: Adding educational content within the app and website to help users understand AI-generated content and how to critically evaluate media.

4. Scalability and Performance

- **Cloud Integration**: Leveraging cloud computing to scale the detection process, ensuring the system can handle a large number of users and high volumes of data.
- **Optimization**: Continuously optimizing the codebase and algorithms to improve performance and reduce computational costs.

5. Security and Privacy

- **Data Privacy**: Enhancing measures to protect user data and ensure compliance with privacy regulations.
- **Security Audits**: Regularly conducting security audits to identify and mitigate potential vulnerabilities in the application and website.

6. Collaborative Research and Development

- **Academic Partnerships**: Collaborating with academic institutions to stay at the forefront of research in AI and machine learning.
- **Industry Collaborations**: Partnering with other organizations and companies to share knowledge and resources, accelerating the development of more advanced detection techniques.

- **Open Source Contributions**: Contributing to and utilizing open-source projects to benefit from community-driven advancements and innovations.

7. Regulatory and Ethical Considerations

- **Ethical AI**: Ensuring that the development and deployment of detection tools align with ethical guidelines and promote responsible use of AI technology.
- **Regulatory Compliance**: Staying informed about and compliant with relevant regulations and standards related to AI and data privacy.

8. Community and Ecosystem Building

- **User Community**: Building a vibrant community of users who can share their experiences, provide feedback, and contribute to the ongoing improvement of the tool.
- **Developer Ecosystem**: Encouraging third-party developers to create plugins and extensions that enhance the core functionalities of the application and website.

References

References

- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative Adversarial Networks. *arXiv preprint arXiv:1406.2661*.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems, 25*, 1097-1105.
- Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*.
- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). TensorFlow: A System for Large-Scale Machine Learning. *OSDI*, 16, 265-283.
- Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Future Generation Computer Systems, 25*(6), 599-616.
- Cavoukian, A. (2010). Privacy by Design: The 7 Foundational Principles. *Information and Privacy Commissioner of Ontario, Canada*.
- European Union. (2016). General Data Protection Regulation (GDPR). *Official Journal of the European Union*, L119, 1-88.
- Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM, 51*(1), 107-113.
- Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., ... & Stoica, I. (2012). Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2-2.
- Gartner. (2021). Magic Quadrant for Cloud AI Developer Services. *Gartner Research*.
- McAfee. (2013). The Economic Impact of Cybercrime and Cyber Espionage. *Center for Strategic and International Studies*.

- Rössler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J., & Nießner, M. (2019). FaceForensics++: Learning to Detect Manipulated Facial Images. arXiv preprint arXiv:1901.08971.
- Dang, H., Liu, F., Stehouwer, J., Liu, X., & Jain, A. K. (2020). On the Detection of Digital Face Manipulation. CVPR.