ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки
Кафедра ЕОМ

Кваліфікаційна робота
Перший (бакалаврський) рівень

Програмний модуль
«Розпізнавання та автопунктуація тексту»

Автор:
Ростислав Потапов
студент групи КІУКІ-21-6

Керівник:
Володимир Федорченко
доц. каф. ЕОМ

Харків, 2025

Мета і завдання роботи

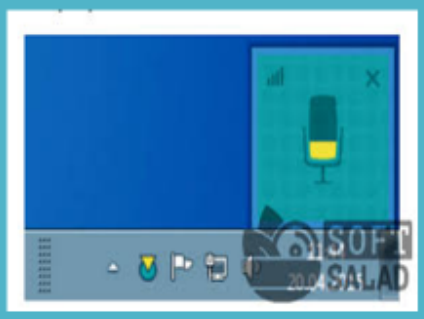
Мета:

Проєктування та створення мобільного застосунку для обробки людської мови

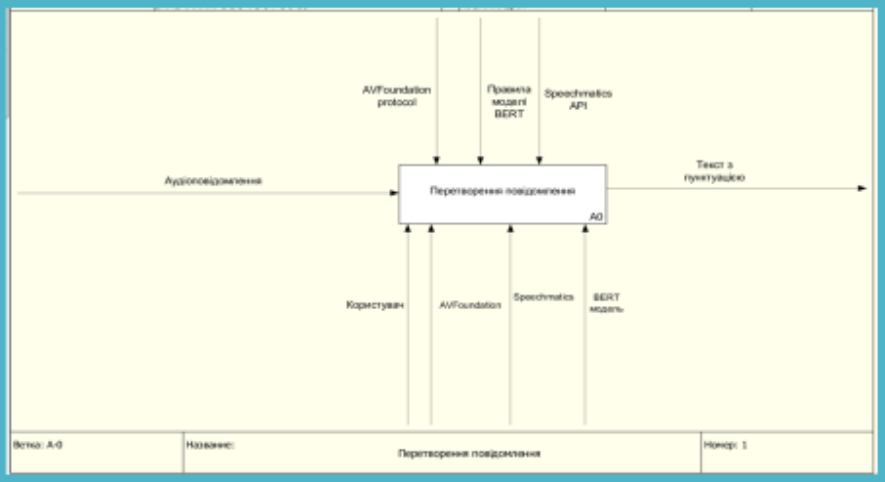
Завдання:

- дослідження бізнес-процесів запису аудіо, розпізнавання мови та автопунктуації тексту
- здійснення аналізу інтерфейсу та функціональності вже існуючих програмних продуктів-аналогів
- розроблення функціональних та нефункціональних вимог до модуля
- розроблення користувацького інтерфейсу
- розроблення програмного продукту для запису, розпізнавання та автопунктуації тексту з подальшим зберіганням результатів
- тестування модуля

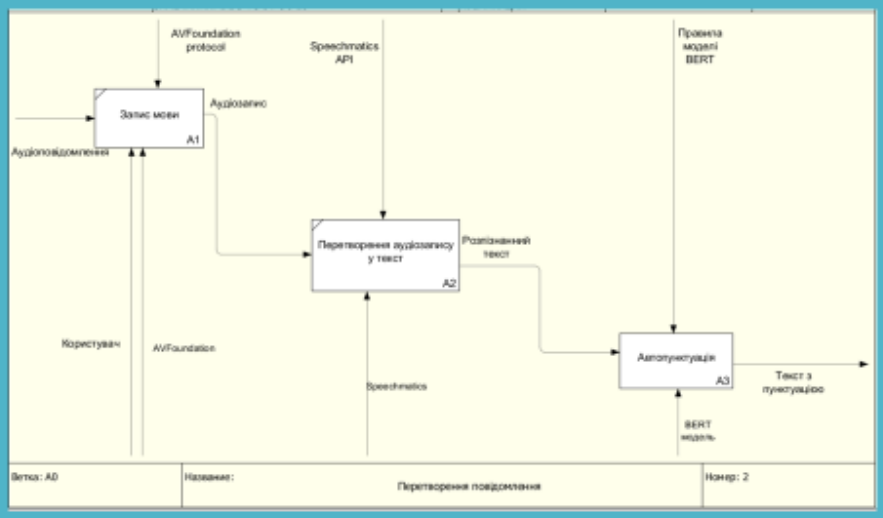
ОГЛЯД АНАЛОГІВ



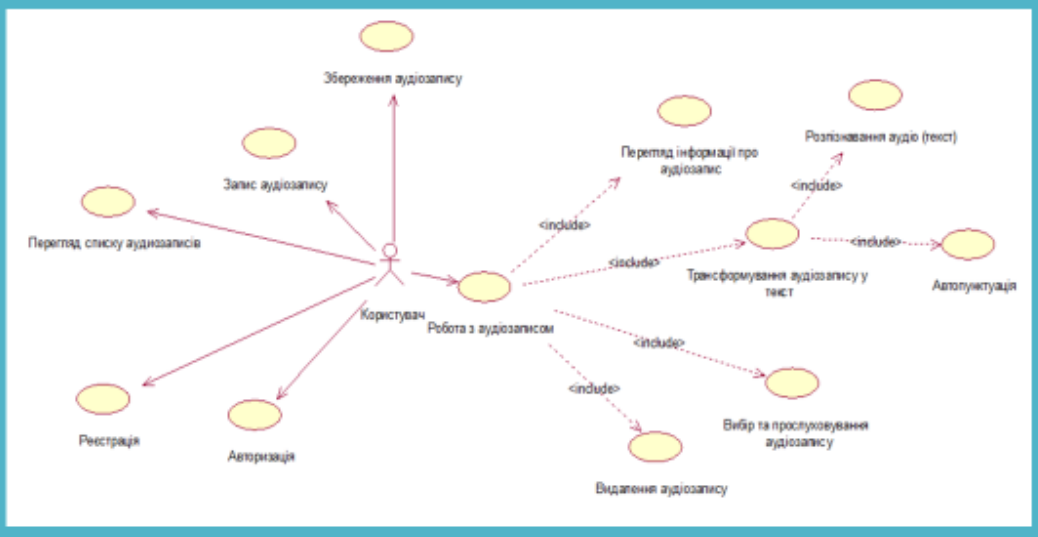
Контекстна діаграма IDEF0 бізнес - процесу «Перетворення повідомлення»



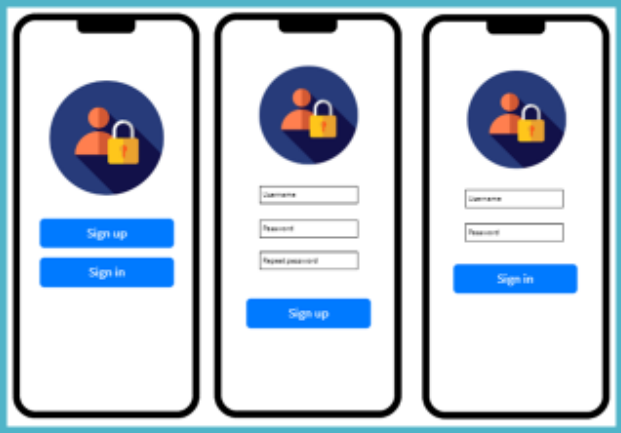
Діаграма декомпозиції IDEF0



Діаграма варіантів використання



Проєктування інтерфейсу



7

Проєктування інтерфейсу

8

Використані технології

Firebase

SPEECHMATICS

Google BERT

9

Використані технології

SPEECHMATICS

Переваги:

- швидка;
- проста у використанні;
- точна;
- підтримує кілька мов;
- підтримує декількох співрозмовників;
- підтримуються кілька форматів файлів;
- інтегрується через REST API.

Недоліки:

- платна;
- не є точною при поганому мікрофоні або шумі.

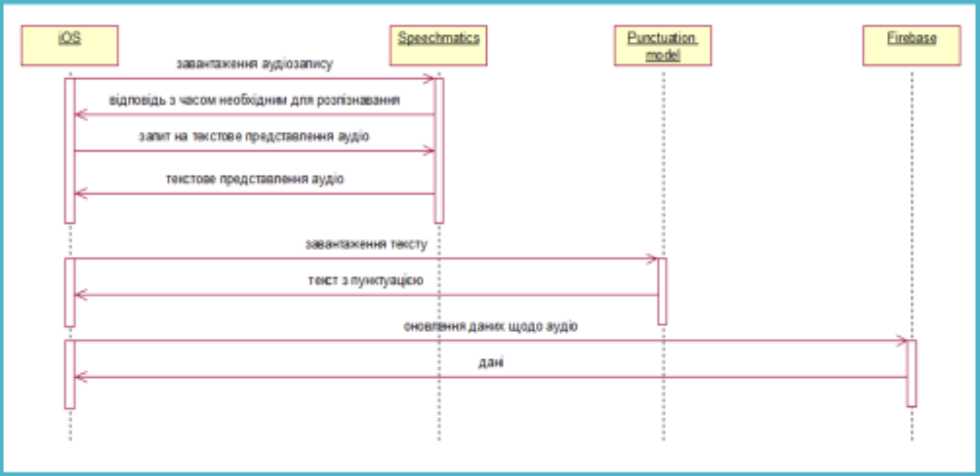
10

Використані технології

Проривним моментом в роботі BERT є його здатність навчати мовні моделі на основі всього набору слів в додатку або запиті (двонаправленне навчання), тоді як в традиційному навчанні аналізується впорядкована послідовність слів (зліва направо або справа наліво). BERT дозволяє мовній моделі розуміти контекст слова на основі оточуючих її слів, а не тільки того слова, яке йому передує або слід відразу за ним. Google називає BERT «глибоко двонаправленним», оскільки контекстні уявлення слів починаються «з самого низу глибокої нейронної мережі».

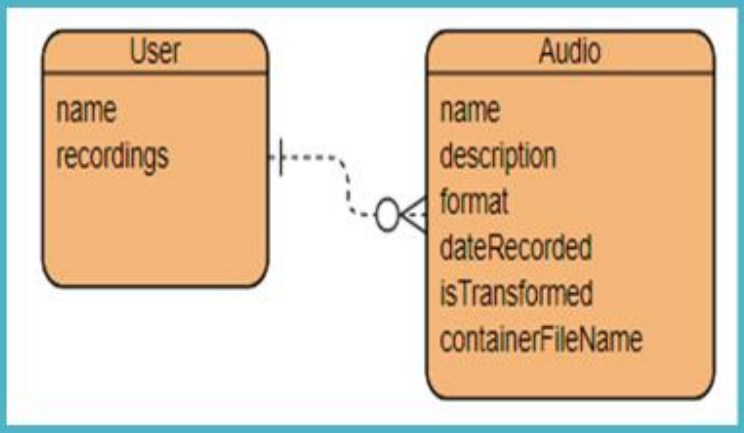
11

Діаграма взаємодії сервісів



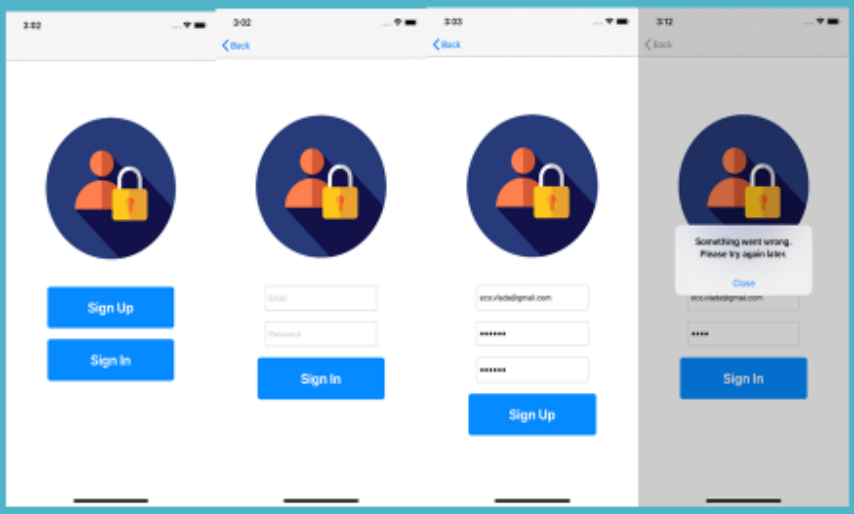
12

Логічна модель бази даних



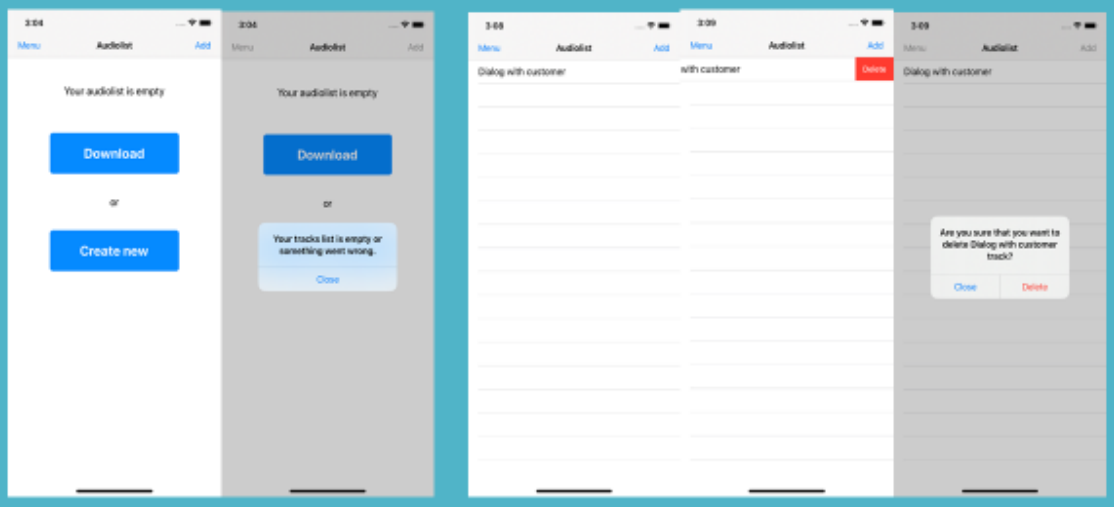
13

Інтерфейс програми



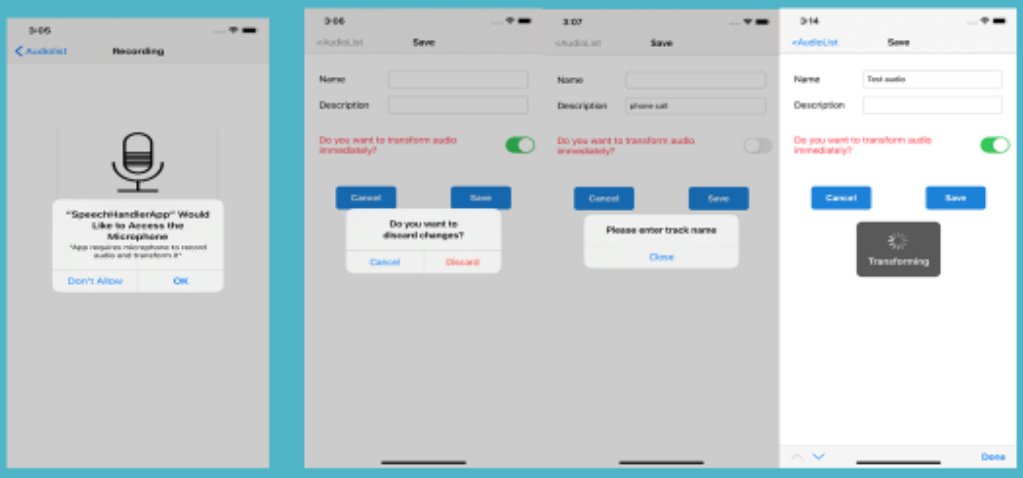
14

Інтерфейс програми



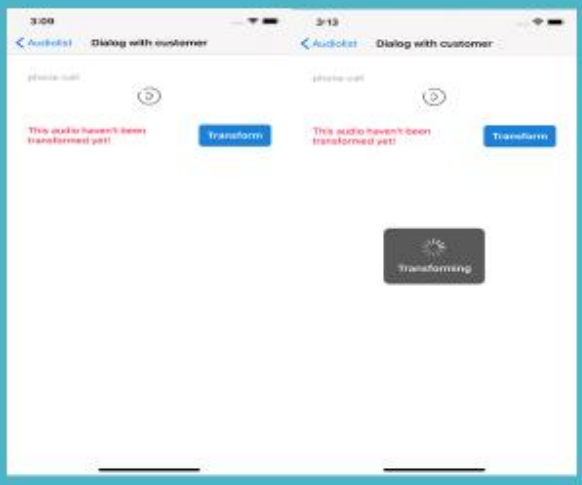
15

Інтерфейс програми

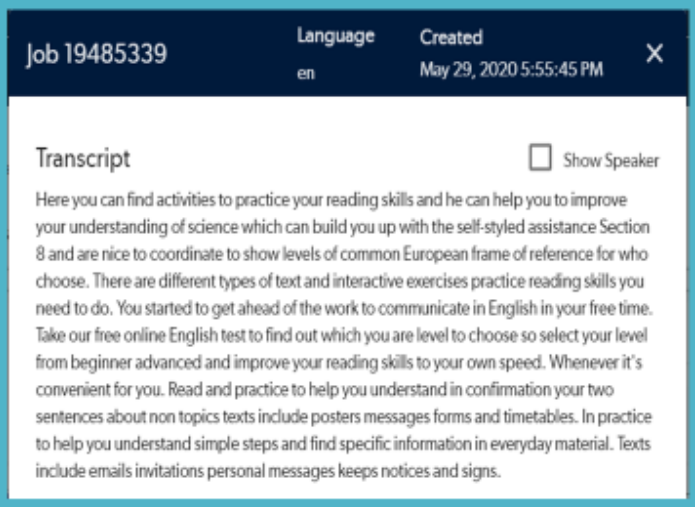


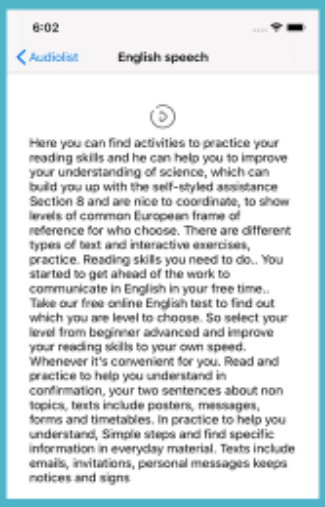
16

Інтерфейс програми



Інтерфейс програми





Висновки

У результаті виконання кваліфікаційної роботи було розроблено модуль «Розпізнавання та автопунктуація тексту» на базі мобільних технологій».
Також було досліджено бізнес-процеси запису аудіо, розпізнавання мови та автопунктуації тексту.
Здійснено аналіз інтерфейсу та функціональності вже існуючих програмних продуктів-аналогів.
Розроблено функціональні та нефункціональні вимоги до модуля
Розроблено користувацький інтерфейс.
Розроблено програмний продукт для запису, розпізнавання та автопунктуації тексту з подальшим зберіганням результатів.
Протестовано модуль.

19

Апробація

Потапов Р.О., Федорченко В.М. // Основні підходи до вирішення завдань розпізнавання та автопунктуації тексту // Матеріали п’ятнадцятої міжнародної науково-технічної конференції “Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління” . Том 3 : тези доповідей, 24 – 25 квітня 2025 р. – Харків : ХНУРЕ, 2025. – С. 54.

20

ДОДАТОК Б

Лістинг програми

Б.1 Код інтерфейсу та логіки для створення аудіо.

```
import UIKit
import Async
import AVFoundation

protocol RecordingViewControllerDelegate: class {

    func recordingViewController(_ viewController:
RecordingViewController, recordAudioAt path: URL)
}

class RecordingViewController: BaseViewController {

    weak var delegate: RecordingViewControllerDelegate?

    private var recordingSession: AVAudioSession!
    private var audioRecorder: AVAudioRecorder!
    private var filePath: URL?

    @IBOutlet private weak var recordingButton: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()

        title = "Recording"
        self.setupSession()
    }

    @IBAction func didTapRecordingButton(_ sender: UIButton) {

        guard recordingSession.recordPermission != .granted
        else {
            audioRecorder == nil ? startRecording() :
finishRecording(success: true)
            return
        }

        recordingSession.requestRecordPermission() { [weak self]
allowed in
            Async.main {

                guard let strongSelf = self else { return }
```

```
                guard allowed
                else {
                    strongSelf.showAlert(message: "App need
record permissions for recording. Please allow it and try
again.")
                    return
                }

                strongSelf.startRecording()
            }
        }
    }
}

// MARK: - AVAudioRecorderDelegate
extension RecordingViewController: AVAudioRecorderDelegate {

    func audioRecorderDidFinishRecording(_ recorder:
AVAudioRecorder, successfully flag: Bool) {
        if !flag { finishRecording(success: false) }
    }
}

// MARK: - Private
private extension RecordingViewController {

    func startRecording() {

        let fileName = "\(Date())file.m4a"
        filePath =
self.getDocumentsDirectory().appendingPathComponent(fileName)

        let settings = [
            AVFormatIDKey: Int(kAudioFormatMPEG4AAC),
            AVSampleRateKey: 12000,
            AVNumberOfChannelsKey: 1,
            AVEncoderAudioQualityKey:
AVAudioQuality.high.rawValue
        ]

        do {
            guard let path = filePath else { return }

            audioRecorder = try AVAudioRecorder(url: path,
settings: settings)
            audioRecorder.delegate = self
            audioRecorder.prepareToRecord()
            audioRecorder.record()
            recordingButton.setTitle("Stop recording", for:
.normal)
        } catch {
            self.finishRecording(success: false)
        }
```

```
    }

    func setupSession() {

        recordingSession = AVAudioSession.sharedInstance()

        do {
            try recordingSession.setCategory(.record, mode:
.default)
            try recordingSession.setActive(true)
        } catch {
            self.showAlert(message: "Something went wrong.
Please try again later.")
        }
    }

    func finishRecording(success: Bool) {

        audioRecorder.stop()
        audioRecorder = nil

        if success, let path = filePath {
            delegate?.recordingViewController(self,
recordAudioAt: path)
        } else {
            recordingButton.setTitle("Tap to Re-record", for:
.normal)
        }
    }

    func getDocumentsDirectory() -> URL {

        let paths = FileManager.default.urls(for:
.documentDirectory, in: .userDomainMask)
        return paths[0]
    }
}
```

Б.2 Код інтерфейсу та логіки для розпізнавання та трансформування аудіо.

```
SaveAudioViewController.swift

import UIKit
import JGProgressHUD

protocol SaveAudioViewControllerDelegate: class {

    func saveAudioViewControllerDidCancel(_ viewController:
SaveAudioViewController)
```

```
    func saveAudioViewController(_ viewController:
SaveAudioViewController, didSave model: TrackModel)
}

class SaveAudioViewController: BaseViewController {

    var filePath: URL!
    var networkManager: NetworkManager!
    var firebaseService: FirebaseService!

    weak var delegate: SaveAudioViewControllerDelegate?

    @IBOutlet weak var nameTextField: UITextField!
    @IBOutlet weak var descriptionTextField: UITextField!
    @IBOutlet weak var transformSwitch: UISwitch!

    override func viewDidLoad() {
        super.viewDidLoad()

        self.setupInitialUI()
    }

    @IBAction func didTapSaveButton(_ sender: UIButton) {

        guard let trackName = nameTextField.text,
              !trackName.isEmpty
        else {
            self.showAlert(message: "Please enter track name")
            return
        }

        var trackModel = TrackModel(name: trackName,
                                    description:
descriptionTextField.text,
                            containerFileName:
filePath.lastPathComponent,
                                isTransformed: transformSwitch.isOn)

        guard transformSwitch.isOn
        else {
            let docId = self.firebaseService.save(model:
trackModel)
            self.firebaseService.upload(filePath: filePath)
            trackModel.trackId = docId

            self.delegate?.saveAudioViewController(self,
didSave: trackModel)

            return
        }

        let hud = JGProgressHUD(style: .dark)
        hud.textLabel.text = "Transforming"
```

```
        hud.show(in: self.view)

        networkManager.postData(input: trackModel,
                                data: try! Data(contentsOf:
filePath),
                                fileName:
filePath.lastPathComponent) { response in

                                    Timer.scheduledTimer(withTim
eInterval: TimeInterval(response?.check_wait ?? 30),
                                                         repeats
: false) { timer in

                                        let job = Job(id:
response?.id ?? 0)

                                        self.networkManager.getD
ata(input: job) { response in

                                            let text =
ConvertedText(text: response ?? "")
                                            trackModel.text =
response

                                            self.networkManager.
getData(input: text) { response in

                                                trackModel.text
= response
                                                let docId =
self.firebaseService.save(model: trackModel)
                                                self.firebaseSer
vice.upload(filePath: self.filePath)
                                                trackModel.track
Id = docId

                                                hud.dismiss()

                                                self.delegate?.s
aveAudioViewController(self, didSave: trackModel)
                                            }
                                        }
                                    }
        }
    }

    @IBAction func didTapCancelButton(_ sender: UIButton) {

        self.showDiscardChangesAlert { [weak self] in

            guard let strongSelf = self else { return }

            try? FileManager.default.removeItem(at:
```

```
strongSelf.filePath)

            strongSelf.delegate?.saveAudioViewControllerDidCance
l(strongSelf)
        }
    }
}

private extension SaveAudioViewController {

    func setupInitialUI() {

        title = "Save"
        navigationItem.leftBarButtonItem =
UIBarButtonItem(title: "<AudioList", style: .plain, target:
self, action: #selector(didTapCancelButton))
    }
}
```

Б.3 Сервіс роботи з базою даних.

```
FirebaseService.swift

import Foundation
import Firebase
import FirebaseStorage

class FirebaseService {

    fileprivate var authService: AuthService
    fileprivate let db = Firestore.firestore()
    fileprivate let storage = Storage.storage()
    fileprivate var currentUserTracks: CollectionReference?

    init(authService: AuthService) {

        self.authService = authService
        currentUserTracks =
db.collection("users").document(authService.currentUser?.uid ??
"").collection("tracks")
    }

    func save(model: TrackModel) -> String? {

        let doc = currentUserTracks?.addDocument(data:
model.model())
        return doc?.documentID
    }

    func update(model: TrackModel) {
```

```
        guard let id = model.trackId else { return }
        currentUserTracks?.document(id).setData(model.model())
    }

    func deleteTrack(id: String) {

        currentUserTracks?.document(id).delete()
    }

    func downloadModels(completion: @escaping ([TrackModel])->()) {

        currentUserTracks?.getDocuments(completion: { (snapshot, error) in

            guard error == nil,
                  let docSnapshot = snapshot
            else {
                completion([])
                return
            }

            var models: [TrackModel] = []
            docSnapshot.documents.forEach { doc in
                let model = self.decodeTrack(id: doc.documentID, from: doc.data())

                if let model = model {
                    models.append(model)
                }
            }
            completion(models)
        })
    }

    func upload(filePath: URL) {

        let refPath = (authService.currentUser?.uid ?? "") + "/" + filePath.lastPathComponent
        let fileRef = storage.reference().child(refPath)

        _ = fileRef.putFile(from: filePath)
    }

    func download(fileName: String, completion: @escaping (URL?)-> ()) {

        let refPath = (authService.currentUser?.uid ?? "") + "/" + fileName
        let pathReference = storage.reference(withPath: refPath)

        let filePath = self.getDocumentsDirectory().appendingPathComponent(fileName)
```

```
        let downloadTask = pathReference.write(toFile: filePath)
{ url, error in
          if let url = url {
            completion(url)
          } else {
            completion(nil)
          }
        }
    }
}

private extension FirebaseService {

    func decodeTrack(id: String, from data: [String : Any]) ->
TrackModel? {

        guard let name = data["name"] as? String,
              let containerFileName = data["containerFileName"]
as? String,
              let isTransformed = data["isTransformed"] as? Bool
        else { return nil }

        let model = TrackModel(trackId: id,
                               name: name,
                        description: data["description"] as?
String,
                  containerFileName: containerFileName,
                      isTransformed: isTransformed,
                               text: data["text"] as? String)
        return model
    }
}
```