ДОДАТОК А

Фрагменти коду програми

```
final class FilesViewController: UIViewController {

    @IBOutlet private var collectionView: UICollectionView!
    @IBOutlet private var label: UILabel!
    @IBOutlet private var spinner: UIActivityIndicatorView!

    private lazy var deleteButtonItem = UIBarButtonItem(barButtonSystemItem: .trash,
                                                        target: self,
                                                        action:
#selector(deleteObjects(_:)))

    private lazy var addButtonItem = UIBarButtonItem(barButtonSystemItem:
.add,
                                                     target: self,
                                                     action:
#selector(insertNewObject(_:)))

    private var detailViewController: DetailViewController?

    private var ids: [Network.FileID] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        navigationItem.leftBarButtonItems = [editButtonItem,
deleteButtonItem]
        navigationItem.rightBarButtonItems = [addButtonItem]
        observeFileUpdates()
        prepare()
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        setEditing(false, animated: false)
    }

    override func viewDidDisappear(_ animated: Bool) {
        super.viewDidDisappear(animated)
        collectionView.indexPathsForSelectedItems?.forEach { indexPath in
            collectionView.deselectItem(at: indexPath, animated: false)
        }
    }

    private func observeFileUpdates() {
        NotificationCenter.default.addObserver(forName:
Network.FileChangedNotification, object: nil, queue: nil) {
            [weak self] notification in
            guard let self = self else { return }
```

```
            let changedIdsArray: [Network.FileID] =
notification.userInfo?[Network.FileChangedNotificationIDsKey] as?
[Network.FileID] ?? []
            let changedIds = Set(changedIdsArray)

            let newIds = DataManager.sharedInstance.network.ids()

            let oldIdsSet = Set(self.ids)
            let newIdsSet = Set(newIds)

            assert(oldIdsSet.count == self.ids.count && newIdsSet.count ==
newIds.count)

            let insertions = newIdsSet.subtracting(oldIdsSet)
            let deletions = oldIdsSet.subtracting(newIdsSet)
            let refreshes =
newIdsSet.intersection(oldIdsSet).intersection(changedIds)

            var insertionCommands: [IndexPath] = []
            var deletionCommands: [IndexPath] = []
            var refreshCommands: [IndexPath] = []

            var i = 0
            while i < self.ids.count {
                if refreshes.contains(self.ids[i]) {
                    refreshCommands.append(IndexPath(row: i, section: 0))
                }
                if deletions.contains(self.ids[i]) {
                    deletionCommands.append(IndexPath(row: i, section: 0))
                }
                i += 1
            }

            var j = 0
            while j < newIds.count {
                if insertions.contains(newIds[j]) {
                    insertionCommands.append(IndexPath(row: j, section: 0))
                }
                j += 1
            }

            self.ids = newIds

            self.collectionView.performBatchUpdates({
                self.collectionView.reloadItems(at: refreshCommands)
                self.collectionView.deleteItems(at: deletionCommands)
                self.collectionView.insertItems(at: insertionCommands)
            }, completion: nil)

            if let id = self.detailViewController?.id,
deletions.contains(id) {
                self.navigationController?.popViewController(animated:
true)
            }
        }
    }

    private func setInteractionEnabled(_ isEnabled: Bool) {
```

```
        navigationItem.leftBarButtonItems?.forEach { $0.isEnabled =
isEnabled }
        navigationItem.rightBarButtonItems?.forEach { $0.isEnabled =
isEnabled }
        collectionView.isUserInteractionEnabled = isEnabled
        collectionView.allowsSelection = isEnabled
    }

    @objc private func deleteObjects(_ sender: Any) {
        deleteSelectedItems()
    }

    @objc private func insertNewObject(_ sender: Any) {
        create()
    }

    // MARK: - Segues

    private var pendingMemoryId: Memory.InstanceID?

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "showDetail", let controller =
segue.destination as? DetailViewController {
            if let indexPath =
collectionView.indexPathsForSelectedItems?.first, let memoryId =
pendingMemoryId {
                let crdt =
DataManager.sharedInstance.memory.getInstance(memoryId)
                controller.crdt = crdt
                controller.id = self.ids[indexPath.row]
                controller.navigationItem.leftItemsSupplementBackButton =
true
            }
            detailViewController = controller
            pendingMemoryId = nil
        }
    }

    override func setEditing(_ editing: Bool, animated: Bool) {
        super.setEditing(editing, animated: animated)
        deleteButtonItem.isEnabled = editing
    }
}

// MARK: - Actions

extension FilesViewController {

    private func prepare() {
        setInteractionEnabled(false)

        handleAction("Logging in...")

        DataManager.sharedInstance.network.login { error in
            if let error = error {
                self.handleError("Could not log in: \(error)")
            } else {
                self.handleSuccess("Data synchronized!")
```

```
                self.ids = DataManager.sharedInstance.network.ids()
                self.collectionView.reloadData()

                self.setInteractionEnabled(true)
            }
        }
    }

    private func create() {
        setInteractionEnabled(false)
        handleAction("Creating file...")

        let id = DataManager.sharedInstance.memory.create(withString: "Edit
me! Created on \(Date().description) by \(DataManager.sharedInstance.id)",
orWithData: nil)


DataManager.sharedInstance.memoryNetworkLayer.sendInstanceToNetwork(id,
createIfNeeded: true) { _, error in
            defer {
                // file was only opened to save
                DataManager.sharedInstance.memory.close(id)

DataManager.sharedInstance.memoryNetworkLayer.tempUnmap(memory: id)
            }

            if let error = error {
                self.handleError("Could not create file: \(error)")
                self.setInteractionEnabled(true)
            } else {
                self.handleSuccess("Created file, good to go!")

                // notification will have arrived at this point to alter
table
                self.setInteractionEnabled(true)
            }
        }
    }

    private func deleteSelectedItems() {
        collectionView.indexPathsForSelectedItems?.forEach { indexPath in
            delete(ids[indexPath.item])
        }
    }

    private func delete(_ id: Network.FileID) {
        setInteractionEnabled(false)

        handleAction("Deleting file...")

        DataManager.sharedInstance.memoryNetworkLayer.delete(id) { e in
            if let error = e {
                self.handleError("Could not delete file: \(error)")
                self.setInteractionEnabled(true)
            } else {
                self.handleFileDelete("Deleted file!")
```

```
                // notification will have arrived at this point to alter
table
                self.setInteractionEnabled(true)
            }
        }
    }
}

// MARK: - Loading Indicator

extension FilesViewController {

    private func handleError(_ message: String) {
        spinner.stopAnimating()
        spinner.isHidden = true

        label.textColor = .red
        label.text = message
    }

    private func handleSuccess(_ message: String) {
        spinner.stopAnimating()
        spinner.isHidden = true

        var hue: CGFloat = 0
        UIColor.green.getHue(&hue, saturation: nil, brightness: nil, alpha:
nil)
        label.textColor = UIColor(hue: hue, saturation: 0.8, brightness:
0.7, alpha: 1.0)
        label.text = message
    }

    private func handleAction(_ message: String) {
        spinner.startAnimating()
        spinner.isHidden = false

        var hue: CGFloat = 0
        UIColor.blue.getHue(&hue, saturation: nil, brightness: nil, alpha:
nil)
        label.textColor = UIColor(hue: hue, saturation: 0.9, brightness:
1.0, alpha: 1.0)
        label.text = message
    }

    private func handleFileDelete(_ message: String) {
        spinner.stopAnimating()
        spinner.isHidden = true

        var hue: CGFloat = 0
        UIColor.blue.getHue(&hue, saturation: nil, brightness: nil, alpha:
nil)
        label.textColor = UIColor(hue: hue, saturation: 0.9, brightness:
1.0, alpha: 1.0)
        label.text = message
    }
}

// MARK: - Collection View
```

```
extension FilesViewController: UICollectionViewDataSource,
UICollectionViewDelegateFlowLayout {

    func collectionView(_ collectionView: UICollectionView,
numberOfItemsInSection section: Int) -> Int {
        return ids.count
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt
indexPath: IndexPath) -> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
"Cell", for: indexPath) as! FileCollectionViewCell

        let metadata =
DataManager.sharedInstance.network.metadata(ids[indexPath.item])!

        let formatter = DateFormatter()
        formatter.dateFormat = "yyyy-MM-dd HH:mm:ss Z"

        if let date = formatter.date(from: metadata.name) {
            let prettyFormatter = DateFormatter()
            prettyFormatter.dateFormat = "MMM d, h:mm a"

            cell.nameLabel.text = prettyFormatter.string(from: date)
        } else {
            cell.nameLabel.text = metadata.name
        }

        var hue: CGFloat = 0
        UIColor.green.getHue(&hue, saturation: nil, brightness: nil, alpha:
nil)
        let green = UIColor(hue: hue, saturation: 0.9, brightness: 0.8,
alpha: 1.0)

        if metadata.remoteShared {
            cell.nameLabel.textColor = green
        } else {
            cell.nameLabel.textColor = metadata.associatedShare != nil ?
.blue : .black
        }

        return cell
    }

    func collectionView(_ collectionView: UICollectionView,
                        layout collectionViewLayout:
UICollectionViewLayout,
                        sizeForItemAt indexPath: IndexPath) -> CGSize {
        let interItemSpacing: CGFloat = 16 * 2
        let sectionInset = (collectionViewLayout as!
UICollectionViewFlowLayout).sectionInset.left + (collectionViewLayout as!
UICollectionViewFlowLayout).sectionInset.right
        let contentWidth = collectionView.bounds.width - interItemSpacing -
sectionInset
        let itemSize = contentWidth / 3
        return CGSize(width: itemSize, height: itemSize)
    }
```

```
    func collectionView(_ collectionView: UICollectionView, didSelectItemAt
indexPath: IndexPath) {
        guard          !isEditing,          let          indexPath          =
collectionView.indexPathsForSelectedItems?.first else {
            return
        }

        let id = ids[indexPath.row]

        var memoryId: Memory.InstanceID?
        var errorValue: Error?

        let group = DispatchGroup()
        group.enter()


DataManager.sharedInstance.memoryNetworkLayer.sendNetworkToInstance(id,

createIfNeeded: true,

continuingAfterMergeConflict: false) {

identifier, error in
            if let error = error {
                errorValue = error
            } else {
                memoryId = identifier
            }
            group.leave()
        }

        group.notify(queue: .main) {
            if let error = errorValue {
                self.handleError("Error retrieving file: \(error)")
            } else {
                self.pendingMemoryId = memoryId
                self.performSegue(withIdentifier:   "showDetail",   sender:
nil)
            }
        }
    }
}

public final class CRDTTextStorage: NSTextStorage {

    public private(set) var stringWrapper: CausalTreeStringWrapper

    private var backedString: NSMutableAttributedString!

    private var defaultTextAttributes: [NSAttributedString.Key: Any] {
        let paragraphStyle = NSMutableParagraphStyle()
        paragraphStyle.lineSpacing = 2.5
        return [
            .font: UIFont.systemFont(ofSize: 16),
            .foregroundColor: UIColor.darkText,
            .paragraphStyle: paragraphStyle
        ]
```

```
    }

    public var revision: CausalTreeString.WeftT? {
        didSet {
            stringWrapper.revision = revision
            reloadData()
        }
    }

    private var isFixingAttributes = false

    // AB: a new container is sometimes created on paste — presumably to
hold the intermediary string — so we have
    // to do this slightly ugly hack; this CT is merely treated like an
ordinary string and does not merge with anything
    var _kludgeCRDT: CausalTreeString?

    // MARK: - Init

    public override convenience init() {
        let kludge = CausalTreeString(site: UUID.zero, clock: 0)
        self.init(withCRDT: kludge)
        self._kludgeCRDT = kludge
    }

    public required init(withCRDT crdt: CausalTreeString) {
        self.stringWrapper = CausalTreeStringWrapper()
        self.stringWrapper.initialize(crdt: crdt)

        super.init()

        self.backedString        =        NSMutableAttributedString(string:
self.stringWrapper as String, attributes: defaultTextAttributes)
    }

    public required init?(coder aDecoder: NSCoder) {
        fatalError()
    }

    public required init?(pasteboardPropertyList propertyList: Any, ofType
type: UIPasteboard.Type) {
        fatalError()
    }

    // MARK: - Reload

    public func reloadData() {
        self.beginEditing()
        let oldLength = backedString.length
        let newString = stringWrapper
        backedString.replaceCharacters(in:  NSRange(location:  0,  length:
oldLength), with: newString as String)
        let newLength = backedString.length
        edited(.editedCharacters,  range:  NSRange(location:  0,  length:
oldLength), changeInLength: newLength - oldLength)
        endEditing()
    }
```

```
    // MARK: - Parent Methods

    public override var string: String {
        return backedString.string
    }

    public override func replaceCharacters(in range: NSRange, with text:
String) {
        stringWrapper.replaceCharacters(in: range, with: text)

        let oldCacheLength = backedString.length
        backedString.replaceCharacters(in: range, with: text)
        let newCacheLength = backedString.length

        edited(.editedCharacters, range: range, changeInLength:
newCacheLength - oldCacheLength)
    }

    public override func setAttributes(_ attributes:
[NSAttributedString.Key: Any]?, range: NSRange) {
        if isFixingAttributes {
            backedString.setAttributes(attributes, range: range)
            edited(.editedAttributes, range: range, changeInLength: 0)
        }
    }

    public override func fixAttributes(in range: NSRange) {
        isFixingAttributes = true
        super.fixAttributes(in: range)
        isFixingAttributes = false
    }

    public override func processEditing() {
        isFixingAttributes = true
        setAttributes(nil, range: editedRange)
        setAttributes(defaultTextAttributes, range: editedRange)
        isFixingAttributes = false
        super.processEditing()
    }

    public override func attributes(at location: Int, effectiveRange range:
NSRangePointer?) -> [NSAttributedString.Key: Any] {
        return backedString.attributes(at: location, effectiveRange: range)
    }
}
public final class CRDTTextEditor: NSCopying, Codable {
    public typealias CRDTMapType = CRDTMap<CausalTreeString.SiteUUIDT,
AtomId, CausalTreeString.SiteUUIDT>

    public private(set) var tree: CausalTreeString
    public private(set) var cursorMap: CRDTMapType

    public init(site: CausalTreeString.SiteUUIDT) {
        self.tree = CausalTreeString(site: site, clock:
Clock(CACurrentMediaTime() * 1000))
        self.cursorMap = CRDTMap(withOwner: site)
    }
```

```
    public func updateCursor(to atomId: AtomId) {
        cursorMap.setValue(atomId)
    }

    public func transfer(to uuid: CausalTreeString.SiteUUIDT, clock: Clock)
-> ([SiteId: SiteId]) {
        let remap = tree.transferToNewOwner(withUUID: uuid, clock: clock)

        for pair in cursorMap.map {
            if let newSite = remap[pair.value.value.site] {
                cursorMap.setValue(AtomId(site:       newSite,       index:
pair.value.value.index), forKey: pair.key, updatingId: false)
            }
        }
        cursorMap.owner = uuid

        return remap
    }

    public func incrementTimestamp() {
        let _ = tree.weave.lamportTimestamp.increment()
        let _ = cursorMap.lamportTimestamp.increment()
    }

    public static func ==(lhs: CRDTTextEditor, rhs: CRDTTextEditor) -> Bool
{
        return lhs.tree == rhs.tree && lhs.cursorMap == rhs.cursorMap
    }

    public func hash(into hasher: inout Hasher) {
        hasher.combine(tree)
        hasher.combine(cursorMap)
    }

    public func copy(with zone: NSZone? = nil) -> Any {
        let returnCopy = CRDTTextEditor(site: tree.ownerUUID())

        returnCopy.tree = tree.copy() as! CausalTreeString
        returnCopy.cursorMap = cursorMap.copy() as! CRDTMap

        return returnCopy
    }
}

// MARK: - CvRDT

extension CRDTTextEditor: CvRDT {

    public func integrate(_ v: inout CRDTTextEditor) {
        let remaps = tree.integrateReturningSiteIdRemaps(&v.tree)

        for pair in cursorMap.map {
            if let newSite = remaps.localRemap[pair.value.value.site] {
                cursorMap.setValue(AtomId(site:       newSite,       index:
pair.value.value.index), forKey: pair.key, updatingId: false)
            }
        }
        for pair in v.cursorMap.map {
```

```
            if let newSite = remaps.remoteRemap[pair.value.value.site] {
                v.cursorMap.setValue(AtomId(site:      newSite,      index:
pair.value.value.index), forKey: pair.key, updatingId: false)
            }
        }
        cursorMap.integrate(&v.cursorMap)
    }

    public func isSuperset(of v: inout CRDTTextEditor) -> Bool {
        return  tree.isSuperset(of:  &v.tree)  &&  cursorMap.isSuperset(of:
&v.cursorMap)
    }

    public func validate() throws -> Bool {
        return try tree.validate() && cursorMap.validate()
    }
}
```

ДОДАТОК Б

Слайди презентації

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

АТЕСТАЦІЙНА РОБОТА МАГІСТРА

Дослідження моделей безконфліктних реплікованих типів даних в колаборативних офлайн застосунках

Виконав:
ст. гр. ІПЗм-18-4
Полторацький А.О.

Науковий керівник:
проф. д.т.н. Єрохін А.Л.

Мета роботи

- Дослідження моделей безконфліктних реплікованих типів даних (CRDT) для роботи з текстом. Їх застосування у колаборативних офлайн застосунках для редагування текстових документів.
- Програмна реалізація мобільного додатку, що дозволить користувачу працювати з текстовими докуменами у колаборативному режимі з декількох пристроїв одночасно.

Аналіз предметної галузі

- Колаборативне програмне забезпечення є дуже поширеним: Google Docs, Evernote, Notes та інші
- Поширеним є недолік знань у проектуванні офлайн застосунків
- Існуючі програмні рішення є проприєтарними
- Ad-hoc рішення мають великі ризики та складний протокол комунікації
- Є необхідність у математичному підґрунті

3

Постановка задачі

- Дослідити моделі роботи з текстом
- Реалізувати програмну систему, що задовольняє наступним вимогам:
  - редагування документів без затримки, навіть при відсутності підключення до мережі; робота в офлайн режимі
  - синхронізація у фоновому режимі
  - автоматичне злиття внесених змін
  - уся необхідна для синхронізації інформація повинна зберігатися у самому документі
  - максимально спрощений протокол комунікації
- Винести моделі та операції для роботи з безконфліктними реплікованими типами даних у окремий програмний модуль

4

Види CRDT: переваги та недоліки

- CvRDT (state-based)
  - Невимогливість до протоколу передачі даних
  - Простота реалізації
  - Асоціативна, комутативна та ідемпотентна функція злиття
- CmRDT (operation-based)
  - Високі вимоги до протоколу передачі даних
  - Складність реалізації

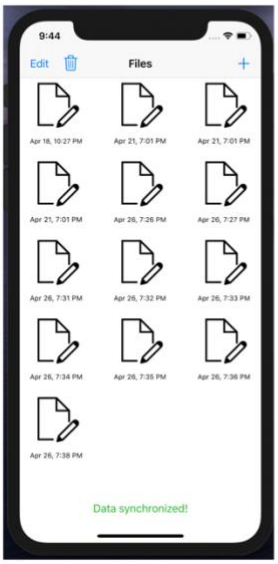
5

Існуючі структури даних

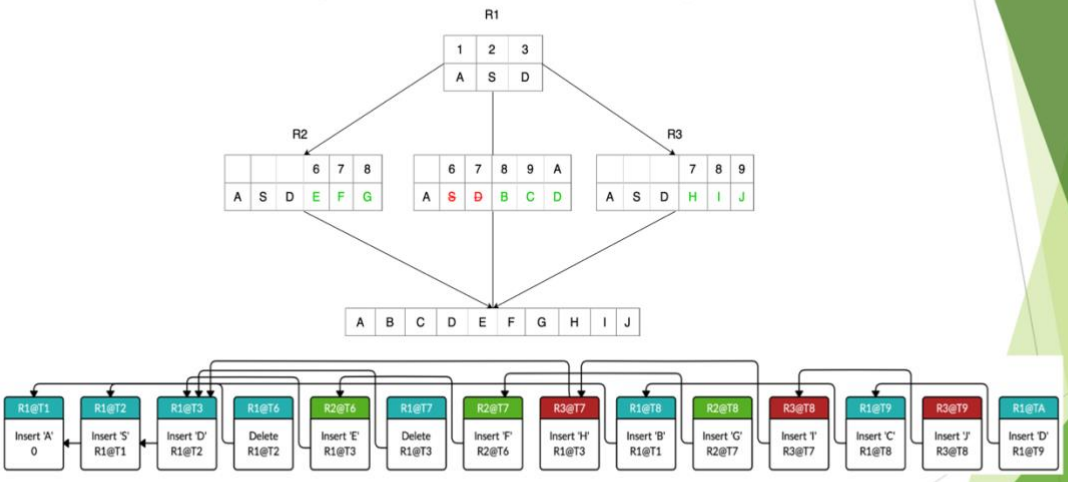
- Для роботи з текстом виділяють декілька існуючих моделей:
  - WOOT
  - Treedoc
  - Logoot
  - RGA
- Реалізація RGA – найбільш актуальна та використана для порівняння та аналізу в даній роботі
  - реалізація на базі суміжного масиву
  - реалізація на бази причинних дерев

6

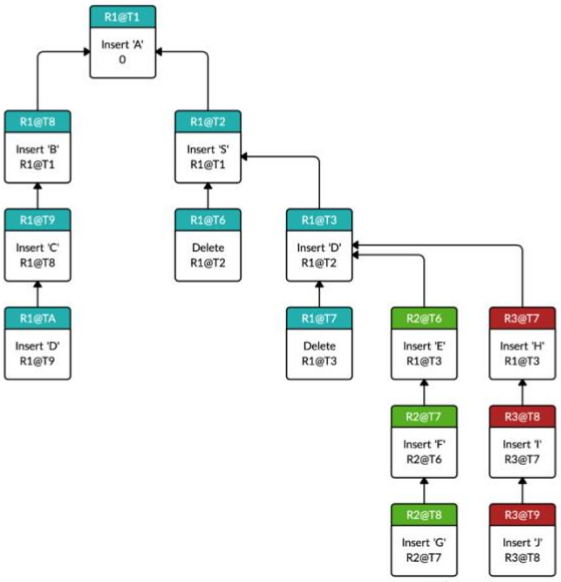
Інтерфейс продукту



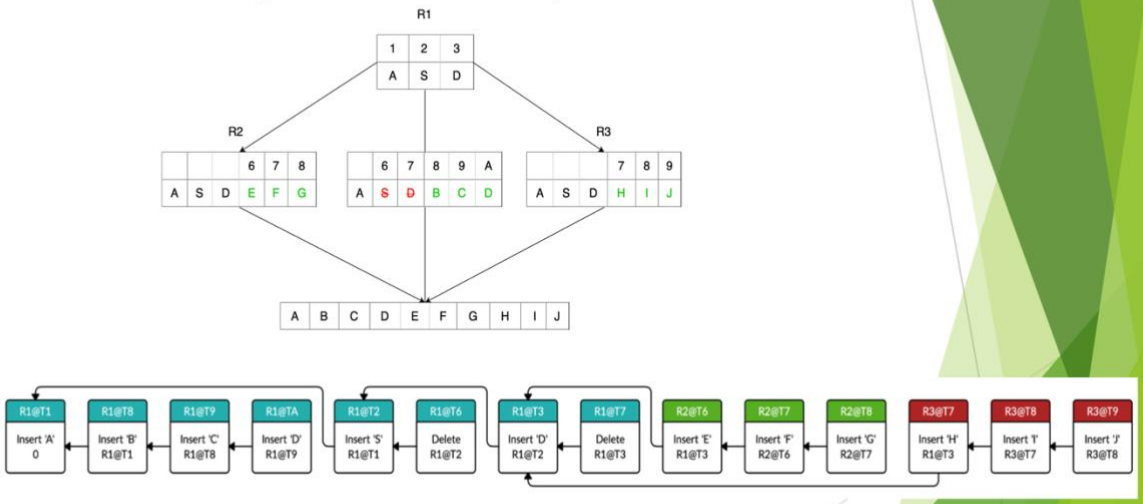
RGA на базі суміжного масиву



RGA на базі причинних дерев



RGA на базі причинних дерев

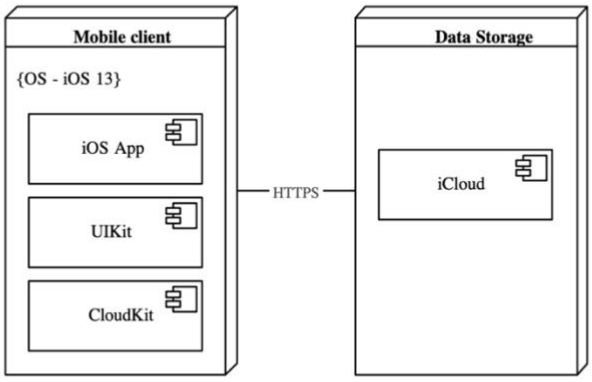


Стек технологій

- iOS 13
- Swift
- iCloud, CloudKit
- Foundation, UIKit
- Swift PM

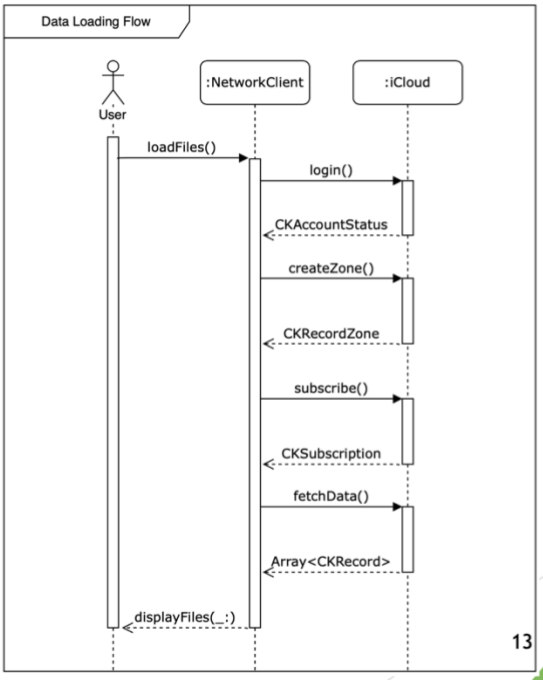
11

Діаграма розгортання



12

Схема завантаження даних з iCloud



Висновки

- Колаборативні застосунки потребують застосування офлайн моделі для роботи з даними
- Було досліджено особливості моделей безконфліктних реплікованих типів даних для роботи з текстом: RGA, WOOT та інші.
- Було порівняно 2 реалізації моделі RGA:
  - реалізації на базі суміжного масиву
  - реалізації на базі причинних дерев
- У результаті дослідження виявлено:
  - Якщо відомо, що операція злиття виконується рідко, рекомендованою є реалізація моделі RGA на базі суміжного масиву
  - Для режиму активного колаборативного редагування підходять структури даних на базі причинних дерев, тому що вони мають кращу реалізацію функції злиття документів
- Моделі безконфліктних реплікованих типів даних підходять для архітектури ПЗ з наявністю «товстого» клієнта
- Розроблене програмне забезпечення задовольняє усім вимогам поставленої задачі

14

ДОДАТОК В

Апробація результатів роботи

В ході науково-дослідної роботи було зроблено тези для участі у наукових конференціях:

– Полторацький А., Єрохін А. «Моделі синхронізації безконфліктних реплікованих типів даних в мобільних офлайн застосунках» / Радіоелектроніка та молодь у XXI столітті 2020.

– Полторацький А., Єрохін А. «Застосування безконфліктних реплікованих типів даних в мобільних офлайн застосунках» / інформаційні технології: наука, техніка, технологія, освіта, здоров’я. MicroCAD-2020.

У наступних підрозділах додатку наведено текст тез доповідей для даних конференцій.

В.1 Моделі синхронізації безконфліктних реплікованих типів даних в мобільних офлайн застосунках

There are a lot of ad-hoc and error-prone approaches for eventual consistency. This article reviews the solution of the problem of data synchronization in offline applications. The main goal is to describe main models and approaches of synchronization using existing conflict-free replicated data types (CRDTs). It describes main principles of CmRDT, CvRDT and their modifications. CRDT will be compared to Operational Transformation in scope of text editing tasks. Finally, we will make a conclusion about feasibility of implementation of CRDTs in mobile applications.

Сучасні програмні продукти досить часто потребують підтримки роботи з даними у режимі офлайн. Такими є розподілені сховища даних, редактори тексту, календарі та інші колаборативні застосунки. У якості прикладу такого продукту

можна виділити Evernote, що є сервісом для роботи з нотатками. Однією з найважливіших вимог в даному випадку є відсутність конфліктів та блокувань для користувача, або їх мінімізація. Сучасним рішенням даної проблеми є conflict-free replicated data types (CRDT, безконфліктні репліковані типи даних).

Виходячи з CAP-теореми, для будь якої розподіленої системи можливо одночасно забезпечити лише дві з трьох властивостей [1]:

– узгодженість;

– доступність;

– стійкість до розподілення.

CRDT вирішує дану проблему завдяки забезпеченню принципу strong eventual consistency (SEC), допускаючи узгодженість даних у кінцевому рахунку та наявності певних алгоритмів вирішення конфліктів [2].

За моделями синхронізації CRDT розподіляються на два класи [3]:

– комутативні (operation-based, CmRDT);

– засновані на зберігання стану (state-based, CvRDT).

В комутативних CRDT усім реплікам відправляються лише дані про виконану операцію. Наприклад, при додаванні нового елементу до списку буде відправлений лише доданий елемент. Варто відзначити, що операції мають бути комутативними, щоб не залежати від порядку отримання змін.

Модель, що заснована на зберіганні стану, передбачає відправку цілої структури даних, а не окремих змін. Дані структури даних повинні підтримувати операції:

– query – операція читання без мутації стану;

– update – мутація стану;

– merge – злиття зі станом з іншої репліки.

Операція merge повинна буди асоціативною, комутативною та ідемпотентною [4].

Для успішного використання даного підходу також необхідно, щоб виконувались наступні вимоги [5]:

– дані мають складати напіврешітку;

– репліки мають складати зв’язний граф.

Було доведено, що будь-яку структуру CRDT можна представити у вигляді CmRDT та CvRDT. Комутативний підхід потребує налагодженого способу доставки повідомлень до реплік, у той час як CvRDT потребує лише запису та читання стану, але при цьому розмір стану може бути достатньо великим.

Для оптимізації state-based CRDT було створено окремий підклас delta-based CRDT, що об'єднує обидва попередні підходи, розсилаючи дельта-мутатори, що оновлюють стан згідно до попереднього часу синхронізації [5]. Але при першій синхронізації все одно необхідно передати увесь стан.

Існує також альтернатива CRDT для роботи з текстом – Operational Transformation. Але потрібно виділити, що підхід Operational Transformation більш вимогливий до ресурсів, складний та за результатами деяких досліджень – менш стабільний [4]. Отже, неможливо обрати однозначно кращий з даних двох інструментів.

Після аналізу існуючих інструментів роботи с CRDT було виявлено, що існує певна кількість готових відкритих програмних рішень для веб-застосунків. У мобільних застосунках, як правило, використовують ad-hoc або пропрієтарні рішення (наприклад, компанія Apple використовує CRDT у додатку Notes, але не надає публічний інтерфейс для роботи з подібними типами даних), тому є доцільним у майбутньому реалізувати відкриту нативну бібліотеку для роботи з CRDT та протестувати її.

Перелік використаних джерел:

1. Теорема CAP. URL: https://uk.wikipedia.org/wiki/Теорема_CAP (дата звернення: 15.02.2020).

2. Strong eventual consistency. URL: https://en.wikipedia.org/wiki/Eventual_consistency (дата звернення: 15.02.2020).

3. A comprehensive study of Convergent and Commutative Replicated Data Types. URL: https://hal.inria.fr/inria-00555588/document (дата звернення: 15.02.2020).

4. Репликация без конфликтов: CRDT в теории и на практике. URL: https://habr.com/ru/post/272987/ (дата звернення: 15.02.2020).

5. CRDT: Conflict-free Replicated Data Types. URL: https://habr.com/ru/post/418897/ (дата звернення: 15.02.2020).

В.2 Застосування безконфліктних реплікованих типів даних в мобільних офлайн застосунках

В роботі розглянуто питання застосування безконфліктних реплікованих типів даних (conflict-free replicated data types, CRDT) у мобільних офлайн застосунках.

В даний момент існує певний недолік знань пов’язаних з проектуванням офлайн застосунків, а більша частина мобільних клієнтів використовують ad-hoc рішення. Наслідки неправильного вибору моделі роботи з даними можуть мати великий вплив на кінцевий результат та досвід користувача. На етапі проектування дуже важливо правильно обрати модель роботи з даними. Виділяють 3 основні моделі: онлайн-модель, «frequently-connected» модель та офлайн-модель [1]. Дані моделі мають різну ступінь толерантності до відсутності підключення до мережі у порядку зростання від першої моделі до останньої. Основною помилкою є використання «frequently-connected» моделі замість офлайн-моделі.

Колаборативні застосунки, наприклад Evernote, повинні гарантувати постійну доступність даних та можливість їх оновлення незалежно від підключення до мережі. Подібні вимоги також присутні в деяких медичних застосунках. Виходячи з цього є необхідність використання офлайн-моделі, основними складностями якої є проблеми розподілення даних та реплікації.

Застосування безконфліктних реплікованих типів даних дозволяє вирішити проблему реплікації даних із використанням принципу «strong eventual

consistency» та наявності певного алгоритму уникнення або вирішення конфліктів [2]. Виділяють 2 основні типи CRDT: CvRDT (state-based) та CmRDT (operation-based), кожен з яких має недоліки та переваги [3]. CmRDT є більш гнучким інструментом, але накладає більш високі вимоги до якості підключення до мережі. CvRDT вимагає передавати цілий стан у процесі реплікації, а отже передача великих об’ємів даних є проблемою. Проте CvRDT має набагато менші вимоги до якості з’єднання. Відповідно до вимог, функція злиття станів двох об’єктів у CmRDT є комутативною та ідемпотентною, що дозволяє без проблем відправляти одне й те саме повідомлення декілька разів, а також отримувати їх у довільному порядку. У якості висновку було проаналізовано, що інструменти для роботи з CRDT на мобільних платформах є здебільшого пропрієтарними, а актуальною є розробка відкритого програмного модуля для роботи з ними.

Перелік використаних джерел:

1. Offline vs Online Application Design . URL: https://blogs.sap.com/2016/04/29/offline-vs-online-application-design/ (дата звернення 11.03.2020)

2. A comprehensive study of Convergent and Commutative Replicated Data Types. URL: https://hal.inria.fr/inria-00555588/document (дата звернення: 11.03.2020).

3. CRDT: Conflict-free Replicated Data Types. URL: https://habr.com/ru/post/418897/ (дата звернення: 11.03.2020).