ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

```
@objc(PostListViewController)
class PostListViewController: UIViewController, UITableViewDelegate {

  // [START define_database_reference]
  var ref: DatabaseReference!
  // [END define_database_reference]
  var encryptionEngine = EncryptionEngine.sharedInstance

  var dataSource: FUITableViewDataSource?

  @IBOutlet weak var tableView: UITableView!

  override func viewDidLoad() {
    super.viewDidLoad()

    // [START create_database_reference]
    ref = Database.database().reference()
    // [END create_database_reference]

    let identifier = "post"
    let nib = UINib(nibName: "PostTableViewCell", bundle: nil)
    tableView.register(nib, forCellReuseIdentifier: identifier)

    dataSource = FUITableViewDataSource(query: getQuery()) { (tableView,
indexPath, snap) -> UITableViewCell in
      let cell = tableView.dequeueReusableCell(withIdentifier: identifier, for:
indexPath) as! PostTableViewCell

      guard let post = Post(snapshot: snap) else { return cell }
      cell.authorImage.image = UIImage(named: "ic_account_circle")
      cell.authorLabel.text = post.author
      var imageName = "ic_star_border"
      if (post.stars?[self.getUid()]) != nil {
        imageName = "ic_star"
      }
      cell.starButton.setImage(UIImage(named: imageName), for: .normal)
      if let starCount = post.starCount {
        cell.numStarsLabel.text = "\(starCount)"
      }
```

```
        cell.postTitle.text = post.title

        // DECRYPT POST BODY:
//      var decryptedBody = post.body
//      do {
//          decryptedBody = try self.encryptionEngine.decryptOwnPost(encryptedPost:
decryptedBody)
//      } catch {
//        // error ¯\_(ツ)_/¯
//        // was printed to console
//        // can't decrypt? fine, use as is
//        decryptedBody = post.body
//      }

        cell.postBody.text = post.body
        return cell
      }

      dataSource?.bind(to: tableView)
      tableView.delegate = self
    }

    override func viewWillAppear(_ animated: Bool) {
      super.viewWillAppear(animated)
      self.tableView.reloadData()
    }

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath)
{
      performSegue(withIdentifier: "detail", sender: indexPath)
    }

    func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath)
-> CGFloat {
      return 150
    }

    func getUid() -> String {
      return (Auth.auth().currentUser?.uid)!
    }

    func getQuery() -> DatabaseQuery {
      return self.ref
    }
```

```
  override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    guard let indexPath: IndexPath = sender as? IndexPath else { return }
    guard let detail: PostDetailTableViewController = segue.destination as?
PostDetailTableViewController else {
      return
    }
    if let dataSource = dataSource {
      detail.postKey = dataSource.snapshot(at: indexPath.row).key
    }
  }
}
@objc(NewPostViewController)
class NewPostViewController: UIViewController, UITextFieldDelegate {

  var ref: DatabaseReference!

  var encryptionEngine = EncryptionEngine.sharedInstance

  @IBOutlet weak var bodyTextView: UITextView!
  @IBOutlet weak var titleTextField: UITextField!

  // UIView lifecycle methods
  override func viewDidLoad() {
    super.viewDidLoad()

    // [START create_database_reference]
    self.ref = Database.database().reference()
    // [END create_database_reference]

    let doneBar = UIToolbar(frame: CGRect(x: 0, y: 0, width: 320, height: 44))
    doneBar.isTranslucent = false
    doneBar.barTintColor = UIColor.purple
    doneBar.autoresizingMask = .flexibleWidth
    let flex = UIBarButtonItem(barButtonSystemItem: .flexibleSpace, target: nil,
action: nil)
    let done = UIBarButtonItem(title: "Post", style: .plain, target: self, action:
#selector(didTapPost))
    done.tintColor = UIColor.yellow
    doneBar.items  = [flex, done, flex]
    doneBar.sizeToFit()

    bodyTextView.inputAccessoryView = doneBar
    titleTextField.inputAccessoryView = doneBar
  }
```

```
  @IBAction func didTapPost(_ sender: AnyObject) {
    // [START single_value_read]
    let userID = Auth.auth().currentUser?.uid
    ref.child("users").child(userID!).observeSingleEvent(of: .value, with: { (snapshot) in
      // Get user value
      let value = snapshot.value as? NSDictionary
      let username = value?["username"] as? String ?? ""
      let user = AppUser(username: username)

      // ENCRYPT POST BODY:
      var postBody = self.bodyTextView.text!

      do {
        postBody = try self.encryptionEngine.encryptOwnPost(postBody: postBody).base64String
      } catch {
        // encryption error, post as is
      }

      // [START_EXCLUDE]
      // Write new post
      self.writeNewPost(withUserID: userID!, username: user.username, title: self.titleTextField.text!, body: postBody)
      // Finish this Activity, back to the stream
      _ = self.navigationController?.popViewController(animated: true)
      // [END_EXCLUDE]
      }) { (error) in
        print(error.localizedDescription)
    }
    // [END single_value_read]
  }

  func writeNewPost(withUserID userID: String, username: String, title: String, body: String) {
    // Create new post at /user-posts/$userid/$postid and at
    // /posts/$postid simultaneously
    // [START write_fan_out]
    let key = ref.child("posts").childByAutoId().key
    let post = ["uid": userID,
                "author": username,
                "title": title,
                "body": body,
                "starCount" : "0"] as [String : Any]
    let childUpdates = ["/posts/\(key)": post,
```

```
                    "/user-posts/\(userID)/\(key)/": post]
    ref.updateChildValues(childUpdates)
    // [END write_fan_out]
  }

  func textFieldShouldReturn(_ textField: UITextField) -> Bool {
    textField.resignFirstResponder()
    return false
  }
}
@objc(PostDetailTableViewController)
class PostDetailTableViewController: UITableViewController, UITextFieldDelegate
{

  let kSectionComments = 2
  let kSectionSend = 1
  let kSectionPost = 0

  var postKey = ""
  var comments: Array<DataSnapshot> = []
  var commentField: UITextField? = nil
  var post: Post = Post()
  lazy var ref: DatabaseReference = Database.database().reference()
  var postRef: DatabaseReference!
  var commentsRef: DatabaseReference!
  var refHandle: DatabaseHandle?

  var encryptionEngine = EncryptionEngine.sharedInstance

  // UITextViewDelegate protocol method
  func textFieldShouldReturn(_ textField: UITextField) -> Bool {
    textField.resignFirstResponder()
    return true
  }

  override func viewDidLoad() {
    super.viewDidLoad()
    postRef = ref.child("posts").child(postKey)
    commentsRef = ref.child("post-comments").child(postKey)
    let nib = UINib(nibName: "PostTableViewCell", bundle: nil)
    tableView.register(nib, forCellReuseIdentifier: "post")
  }

  override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
```

```
    comments.removeAll()
    // [START child_event_listener]
    // Listen for new comments in the Firebase database
    commentsRef.observe(.childAdded, with: { (snapshot) -> Void in
      self.comments.append(snapshot)
        self.tableView.insertRows(at: [IndexPath(row: self.comments.count-1, section:
self.kSectionComments)], with: UITableView.RowAnimation.automatic)
    })
    // Listen for deleted comments in the Firebase database
    commentsRef.observe(.childRemoved, with: { (snapshot) -> Void in
      let index = self.indexOfMessage(snapshot)
      self.comments.remove(at: index)
        self.tableView.deleteRows(at: [IndexPath(row: index, section:
self.kSectionComments)], with: UITableView.RowAnimation.automatic)
    })
    // [END child_event_listener]

    // [START post_value_event_listener]
    refHandle = postRef.observe(DataEventType.value, with: { (snapshot) in
      let postDict = snapshot.value as? [String : AnyObject] ?? [:]
      // [START_EXCLUDE]
    let newPost = Post(snapshot: snapshot)
      self.post = newPost!
      self.tableView.reloadData()

      self.navigationItem.rightBarButtonItem?.isEnabled = !self.isMyPost(postAuthor:
self.post.author)

      // [END_EXCLUDE]
    })
    // [END post_value_event_listener]
  }

  func indexOfMessage(_ snapshot: DataSnapshot) -> Int {
    var index = 0
    for  comment in self.comments {
      if snapshot.key == comment.key {
        return index
      }
      index += 1
    }
    return -1
  }

  override func viewWillDisappear(_ animated: Bool) {
```

```
    super.viewWillDisappear(animated)
    if let refHandle = refHandle {
      postRef.removeObserver(withHandle: refHandle)
    }
    commentsRef.removeAllObservers()
    if let uid = Auth.auth().currentUser?.uid {
      Database.database().reference().child("users").child(uid).removeAllObservers()
    }
  }

  // UITableViewDataSource protocol methods
  override func numberOfSections(in tableView: UITableView) -> Int {
    return 3
  }

  override func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
    switch section {
    case kSectionPost, kSectionSend:
      return 1
    case kSectionComments:
      return comments.count
    default:
      return 0
    }
  }

  @IBAction func didTapSend(_ sender: UIButton) {
    _ = textFieldShouldReturn(commentField!)
    commentField?.isEnabled = false
    sender.isEnabled = false
    if let uid = Auth.auth().currentUser?.uid {
      Database.database().reference().child("users").child(uid).observeSingleEvent(of:
.value, with: { (snapshot) in
        if let commentField = self.commentField, let user = snapshot.value as? [String :
AnyObject] {
          let comment = [
            "uid": uid,
            "author": user["username"] as? String ?? "",
            "text": commentField.text!
          ]
          self.commentsRef.childByAutoId().setValue(comment)
          commentField.text = ""
          commentField.isEnabled = true
          sender.isEnabled = true
```

```
      }
     })
    }
  }

  override func tableView(_ tableView: UITableView,
                          cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell: UITableViewCell

    switch indexPath.section {
    case kSectionPost:
      cell = tableView.dequeueReusableCell(withIdentifier: "post", for: indexPath)
      if let uid = Auth.auth().currentUser?.uid {
        guard let postcell = cell as? PostTableViewCell else {
          break
        }

        let postAuthor = post.author

        let imageName = post.stars == nil || post.stars![uid] == nil ? "ic_star_border" :
"ic_star"
        postcell.authorLabel.text = postAuthor
        postcell.postTitle.text = post.title

        // DECRYPT POST BODY:

        var postBody = post.body

        do {
          if (isMyPost(postAuthor:postAuthor)) {
            postBody = try self.decryptBodyOfMyPost(encryptedBody:
EncryptedData(base64String: postBody)!)
          } else {
            postBody = try self.decryptBodyOfOtherPost(encryptedBody:
EncryptedData(base64String: postBody)!, author: postAuthor)
          }
        } catch {
          // decryption error, show encrypted text
        }

        postcell.postBody.text = postBody

        postcell.starButton.setImage(UIImage(named: imageName), for: .normal)
        if let starCount = post.starCount {
          postcell.numStarsLabel.text = "\(starCount)"
```

```
        }
        postcell.postKey = postKey
      }
    case kSectionComments:
      cell = tableView.dequeueReusableCell(withIdentifier: "comment", for: indexPath)
      let commentDict = comments[indexPath.row].value as? [String : AnyObject]
      if let text = cell.textLabel, let detail = cell.detailTextLabel,
        let author = commentDict?["author"], let commentText = commentDict?["text"] {
        detail.text = String(describing: author)
        text.text = String(describing: commentText)
      }
    default: // kSectionSend
      cell = tableView.dequeueReusableCell(withIdentifier: "send", for: indexPath)
      commentField = cell.viewWithTag(7) as? UITextField
      break
    }
    return cell
  }

  override func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
    if indexPath.section == kSectionPost {
      return 160
    }
    return 56
  }
}

// MARK: - decryption
extension PostDetailTableViewController {

  func isMyPost(postAuthor: String) -> Bool {
    let username = Auth.auth().currentUser?.displayName
    return (postAuthor == username)
  }

  func decryptBodyOfMyPost(encryptedBody: EncryptedData) throws -> String {
    return try encryptionEngine.decryptOwnPost(encryptedPost: encryptedBody)
  }

  func decryptBodyOfOtherPost(encryptedBody: EncryptedData, author: String) throws -> String {
```

```
    return try encryptionEngine.decryptSomebodyPost(encryptedPost: encryptedBody,
author: author)
  }
}
@objc(PublicKeysViewController)
class PublicKeysViewController: UIViewController, UITableViewDelegate {

  // [START define_database_reference]
  var ref: DatabaseReference!
  // [END define_database_reference]

  var encryptionEngine = EncryptionEngine.sharedInstance
  var dataSource: FUITableViewDataSource?

  @IBOutlet weak var tableView: UITableView!

  override func viewDidLoad() {
    super.viewDidLoad()

    // [START create_database_reference]
    ref = Database.database().reference()
    // [END create_database_reference]

    let identifier = "post"
    let nib = UINib(nibName: "PostTableViewCell", bundle: nil)
    tableView.register(nib, forCellReuseIdentifier: identifier)

    dataSource = FUITableViewDataSource(query: getQuery()) { (tableView,
indexPath, snap) -> UITableViewCell in
      let cell = tableView.dequeueReusableCell(withIdentifier: identifier, for:
indexPath) as! PostTableViewCell

      guard let publicKey = PublicKey(snapshot: snap) else { return cell }
      cell.authorImage.image = UIImage(named: "ic_account_circle")
      cell.authorLabel.text = publicKey.author
      cell.postBody.text = publicKey.body

      cell.starButton.isHidden = true
      cell.numStarsLabel.isHidden = true
      cell.postTitle.isHidden = true

      return cell
    }

    dataSource?.bind(to: tableView)
```

```
    tableView.delegate = self
  }

  override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    self.tableView.reloadData()
  }

  func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath)
{
    guard let dataSource = dataSource else { return }
    guard let publicKeyModel = PublicKey(snapshot: dataSource.snapshot(at:
indexPath.row)) else { return }

    let publicKey = publicKeyModel.body
    let author = publicKeyModel.author

    let pasteBoard = UIPasteboard.general
    pasteBoard.string = publicKey

    self.showMessagePrompt("Public key is saved, and copied to clipboard")

    self.encryptionEngine.rememberPublicKey(user: author, publicKey:
Key(base64String: publicKey)!)
  }

  func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath)
-> CGFloat {
    return 150
  }

  func getUid() -> String {
    return (Auth.auth().currentUser?.uid)!
  }

  func getQuery() -> DatabaseQuery {
    let sharedKeysQuery = (ref?.child("public-keys")
      .queryOrdered(byChild: "timestamp") // not working
      .queryLimited(toFirst: 10))!
    return sharedKeysQuery
  }

  //  override func viewWillDisappear(_ animated: Bool) {
  //    super.viewWillDisappear(animated)
  //    getQuery().removeAllObservers()
```

```
  //  }
}
class NewPublicKeyViewController: UIViewController, UITextFieldDelegate {

    var ref: DatabaseReference!
    @IBOutlet weak var bodyTextView: UITextView!

    var encryptionEngine = EncryptionEngine.sharedInstance

    override func viewDidLoad() {
        super.viewDidLoad()

        self.tabBarController?.navigationItem.title = "Add Public Key"

        // [START create_database_reference]
        self.ref = Database.database().reference()
        // [END create_database_reference]

        let doneBar = UIToolbar(frame: CGRect(x: 0, y: 0, width: 320, height: 44))
        doneBar.isTranslucent = false
        doneBar.barTintColor = UIColor.purple
        doneBar.autoresizingMask = .flexibleWidth
        let flex = UIBarButtonItem(barButtonSystemItem: .flexibleSpace, target: nil,
action: nil)
        let done = UIBarButtonItem(title: "Post", style: .plain, target: self, action:
#selector(didTapPost))
        done.tintColor = UIColor.yellow
        doneBar.items  = [flex, done, flex]
        doneBar.sizeToFit()
        bodyTextView.inputAccessoryView = doneBar

        // POST MY OWN PUBLIC KEY
        var bodyText = ""
        if let myPubKey = try? encryptionEngine.getMyPublicKey() {
          bodyText = myPubKey.base64String
        }

        bodyTextView.text = bodyText
        bodyTextView.isEditable = false
    }

    @IBAction func didTapPost(_ sender: AnyObject) {
        // [START single_value_read]
        let userID = Auth.auth().currentUser?.uid
```

```
        ref.child("users").child(userID!).observeSingleEvent(of: .value, with: { (snapshot) in
            // Get user value
            let value = snapshot.value as? NSDictionary
            let username = value?["username"] as? String ?? ""
            let user = AppUser(username: username)

            // [START_EXCLUDE]
            // Write new PK
            self.writeNewPublicKey(withUserID: userID!, username: user.username, body: self.bodyTextView.text)
            // Finish this Activity, back to the stream
            _ = self.navigationController?.popViewController(animated: true)
            // [END_EXCLUDE]
        }) { (error) in
            print(error.localizedDescription)
        }
        // [END single_value_read]
    }

    func writeNewPublicKey(withUserID userID: String, username: String, body: String) {
        // Create new post at /user-posts/$userid/$postid and at
        // /posts/$postid simultaneously
        // [START write_fan_out]
        let key = ref.child("public-keys").childByAutoId().key
        let publicKey = ["uid": userID,
                         "author": username,
                         "body": body] as [String : Any]
        let childUpdates = ["/public-keys/\(key)": publicKey,
                            "/user-public-keys/\(userID)/\(key)/": publicKey]
        ref.updateChildValues(childUpdates)
        // [END write_fan_out]
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        textField.resignFirstResponder()
        return false
    }
}
```