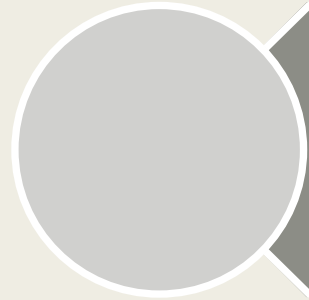


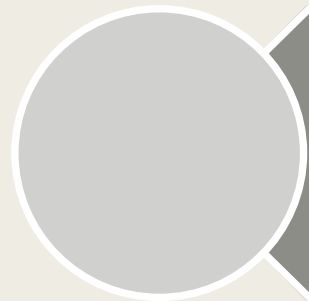
Актуальність теми



Зручність покращує
привабливість



Безпека



Контроль статистики
покращує продуктивність


Мета та призначення

Мета: покращення зручності користування системами пропуску.


Призначення: створити програмно апаратний комплекс, що дозволяє користувачам розблоковувати замок за допомогою ключ-карти, відбитку пальця та за допомогою мобільного застосунку.

Задачі


Для досягнення мети потрібно



Створити систему контролю доступу, дані якої збережені у локальній мережі пристрою керування замком

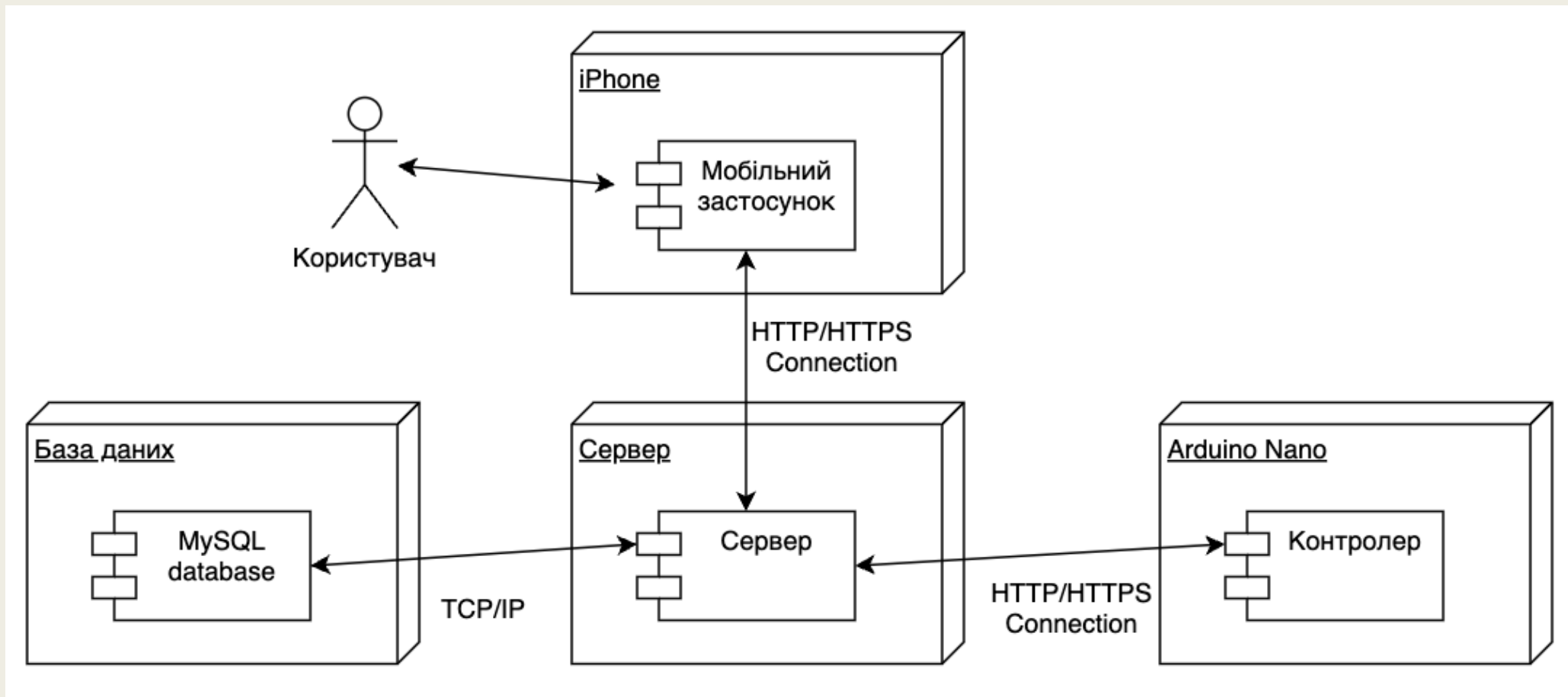


Створити мобільний застосунок, за допомогою якого користувачі зможуть розблоковувати замок



Додати у мобільний застосунок функціонал управління пропусками

Архітектура



Засоби розробки



SwiftUI
Better apps. Less code.

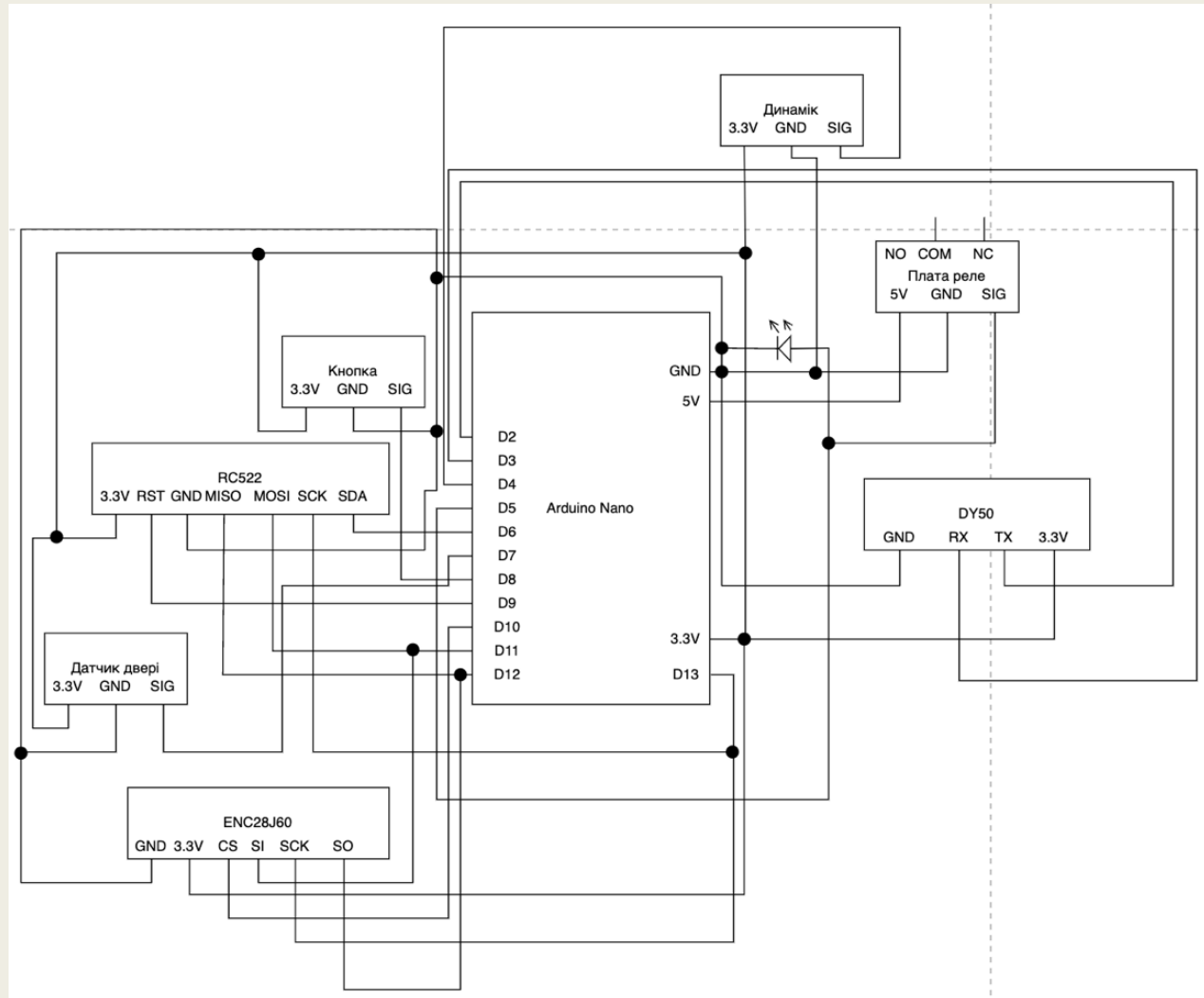


Fluent

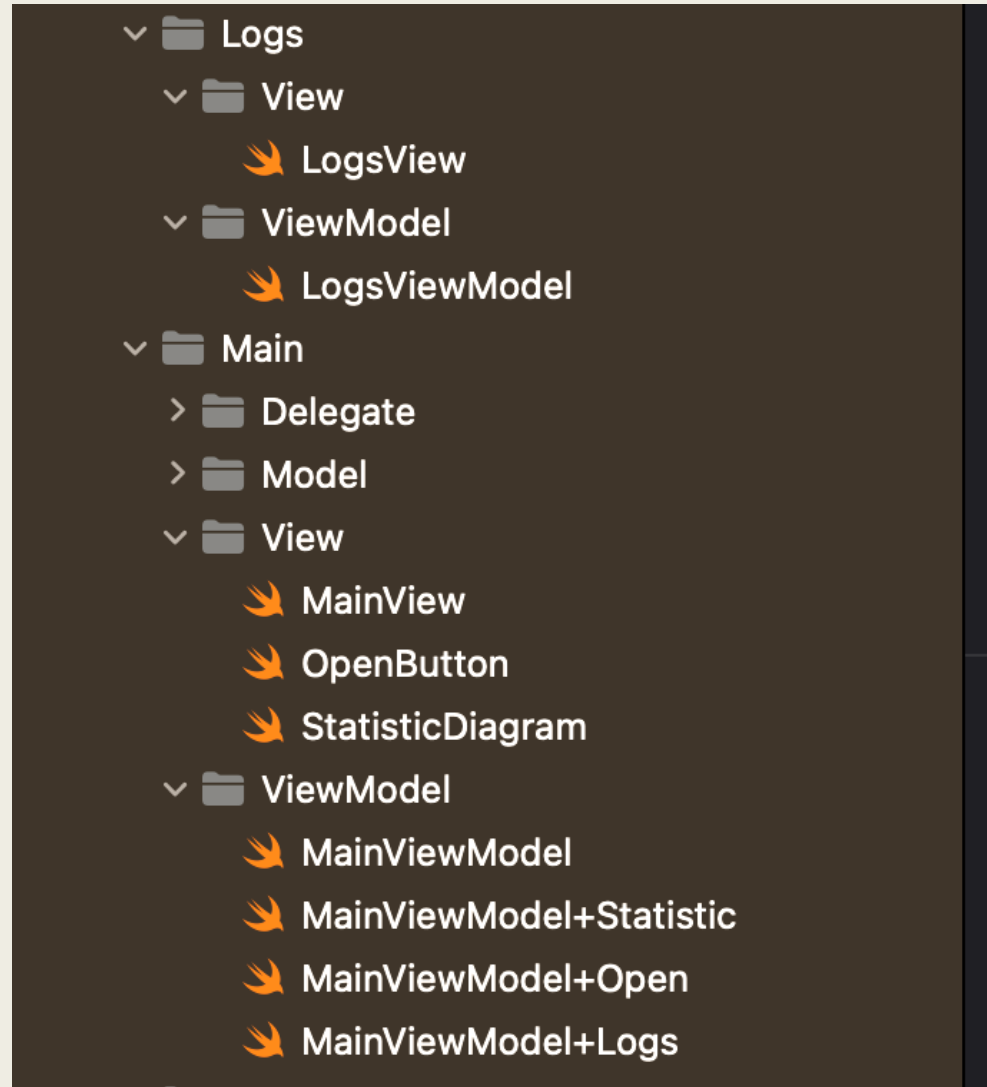
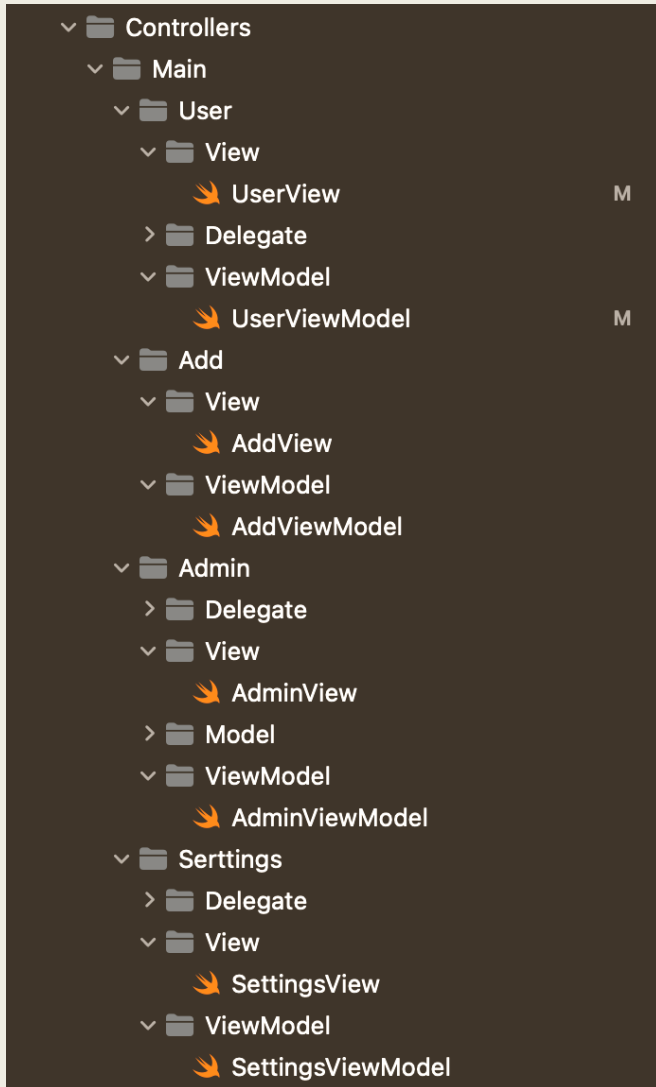
MySQL



Схема апаратної частини



MVVM iOS App



Flow iOS App

```

46 extension Flow {
47     @objc func changeScreenWhenServerLinkUpdated() {
48         if UserDefaults.serverLink == nil {
49             showEnterServer()
50         } else {
51             showLoading { [weak self] in
52                 guard let self else { return }
53                 goToRegistrationOrMain()
54             }
55         }
56     }
57
58     private func goToRegistrationOrMain() {
59         if NetworkMonitor().checkInternetConnectivity() {
60             if UserDefaults.expirationDate ?? .distantPast < .now {
61                 refreshToken()
62             } else {
63                 DispatchQueue.main.async { [weak self] in
64                     guard let self else { return }
65                     showMain()
66                 }
67             }
68         } else {
69             DispatchQueue.main.async { [weak self] in
70                 guard let self else { return }
71                 if UserDefaults.userInfo == nil {
72                     showRegistration()
73                 } else {
74                     showMain()
75                 }
76             }
77         }
78     }
79 }

```

```

10 class Flow {
11
12     // MARK: - Life
13
14     func start() {
15         subscribeToNotifications()
16         showPasscode()
17     }
18
19     private func subscribeToNotifications() {
20         NotificationCenter.default.addObserver(self, selector: #selector(changeScreenWhenServerLinkUpdated),
21             name: .serverLinkChanged, object: nil)
22     }
23 }

```

```

80     private func refreshToken() {
81         TokenRefresher().refreshToken { result in
82             DispatchQueue.main.async { [weak self] in
83                 guard let self else { return }
84                 switch result {
85                     case .success(_):
86                         showMain()
87                     case .failure(let error):
88                         print(error)
89                         showRegistration()
90                 }
91             }
92         }
93     }
94
95     private func refreshData() {
96         DataRefresher().refreshData { result in
97             DispatchQueue.main.async { [weak self] in
98                 guard let self else { return }
99                 switch result {
100                     case .success(_):
101                         showMain()
102                     case .failure(let error):
103                         print(error)
104                         showRegistration()
105                 }
106             }
107         }
108     }

```


Обгортки для зберігання даних iOS

```
10 @propertyWrapper
11 struct UserDefaultsValue<T: Equatable> {
12     let storageKey: String
13     let storageID: String?
14     let notificationKey: NSNotification.Name?
15
16     init(storageKey: String, notificationKey: NSNotification.Name? = nil, storageID: String? = nil) {
17         self.storageKey = storageKey
18         self.notificationKey = notificationKey
19         self.storageID = storageID
20     }
21
22     var wrappedValue: T? {
23         get {
24             if let storageID {
25                 return UserDefaults(suiteName: storageID)!.value(forKey: storageKey) as? T
26             } else {
27                 return UserDefaults.standard.value(forKey: storageKey) as? T
28             }
29         }
30         set {
31             let storage = storageID == nil ? UserDefaults.standard : UserDefaults(suiteName: storageID)!
32             let oldValue = storage.value(forKey: storageKey) as? T
33             storage.set(newValue, forKey: storageKey)
34             if let notificationKey, oldValue != newValue {
35                 DispatchQueue.main.async {
36                     NotificationCenter.default.post(name: notificationKey, object: nil)
37                 }
38             }
39         }
40     }
41 }
```

Використання Singleton

```
10 class ThemeManager {
11
12     static let shared = ThemeManager()
13
14     var currentTheme = Theme(rawValue: UserDefaults.theme ?? 2) ?? .system {
15         didSet {
16             UserDefaults.theme = currentTheme.rawValue
17             applyTheme()
18         }
19     }
20
21     private init() {}
22
23     func applyTheme() {
24         guard let window = UIApplication.shared.windows.first else {
25             return
26         }
27         window.overrideUserInterfaceStyle = currentTheme.userInterfaceStyle
28     }
29
30 }
```











Використання Delegate
















```
10 protocol RegistrationShowerDelegate: AnyObject {
11     func showMain()
12     func showChangePasswordFromRegistration()
13 }
```

```
127 extension Flow: RegistrationShowerDelegate, SettingsShowerDelegate {
161     func showChangePasswordFromRegistration() {
162         showChangePassword(from: registrationVC)
163     }
164
165     func openChangePasswordFromSettings() { ... }
168
169     func showChangePassword(from vc: UIViewController?, _ email: String? = nil) { ... }
184
185     func showMain() {
186         guard let info = UserDefaults.userInfo else {
187             showLoading { [weak self] in
188                 guard let self else { return }
189                 refreshData()
190             }
191             return
192         }
193         showTabBar(info)
194     }
```

```
10 class RegistrationViewModel: ObservableObject {
11
12     @Published var email = ""
13     @Published var password = ""
14     @Published var isLoading = false
15     @Published var alert: AlertItem?
16
17     private weak var showerDelegate: RegistrationShowerDelegate?
18
19     init(delegate: RegistrationShowerDelegate) {
20         showerDelegate = delegate
21     }
```

MVC Server App

- ▼  Controllers
 - ▼  Auth
 - >  Authenticator
 - >  Constants
 - >  Model
 - >  Routes
 -  AuthController
 - ▼  Command
 - >  Routes
 -  CommantController

- ▼  Info
 - >  Model
 - >  Routes
 -  InfoController
- ▼  KeyVerifier
 - >  Model
 - >  Routes
 -  KeyVerifierController
- ▼  Open
 - >  Model
 - >  Routes
 -  OpenController
- ▼  ValidServer
 - >  Model
 -  ValidServerController

Використання менеджерів Server App

```

11 struct KeyVerifierController: RouteCollection {
12
13     private let cardCodeVerifier: CardCodeVerifier
14     private let fingerVerifier: FingerVerifier
15

```

```

10 class CardCodeVerifier {
11
12     let db: DBManager
13
14     init(db: DBManager) {
15         self.db = db
16     }
17
18     func verify(_ code: String) async throws -> Bool {
19         let hash = calculateHash(for: code)
20         let cards = try await db.getCards(forHash: hash)
21         guard let card = cards.first(where: { $0.code == code && $0.employerID != nil }),
22             let employerID = card.employerID else {
23             return false
24         }
25         addEnter(for: employerID)
26         return true
27     }

```

```

10 protocol DBManager: AnyObject {
11     func setup(_ app: Application)
12     func getCards(forHash hash: Int) async throws -> [CardDBModel]
13     func getFinger(forCode code: Int) async throws -> FingerDBModel?
14     func addEnter(for id: UUID) async throws
15     func getUser(for email: String) async throws -> any EmployerDBModel
16     func getUser(by id: UUID) async throws -> any EmployerDBModel
17     func updatePassword(for userID: UUID, newPassword: String) async throws
18     func getLogs(for userID: UUID, after date: Date?) async throws -> [EnterDBModel]
19     func getAllEmployers() async throws -> [EmployerDBModel]
20     func removeUser(with id: UUID) async throws
21     func addUser(_ user: EmployerModel, password: String) async throws
22     func hasCard(_ id: UUID) async throws -> Bool
23     func hasFinger(_ id: UUID) async throws -> Bool
24 }

```

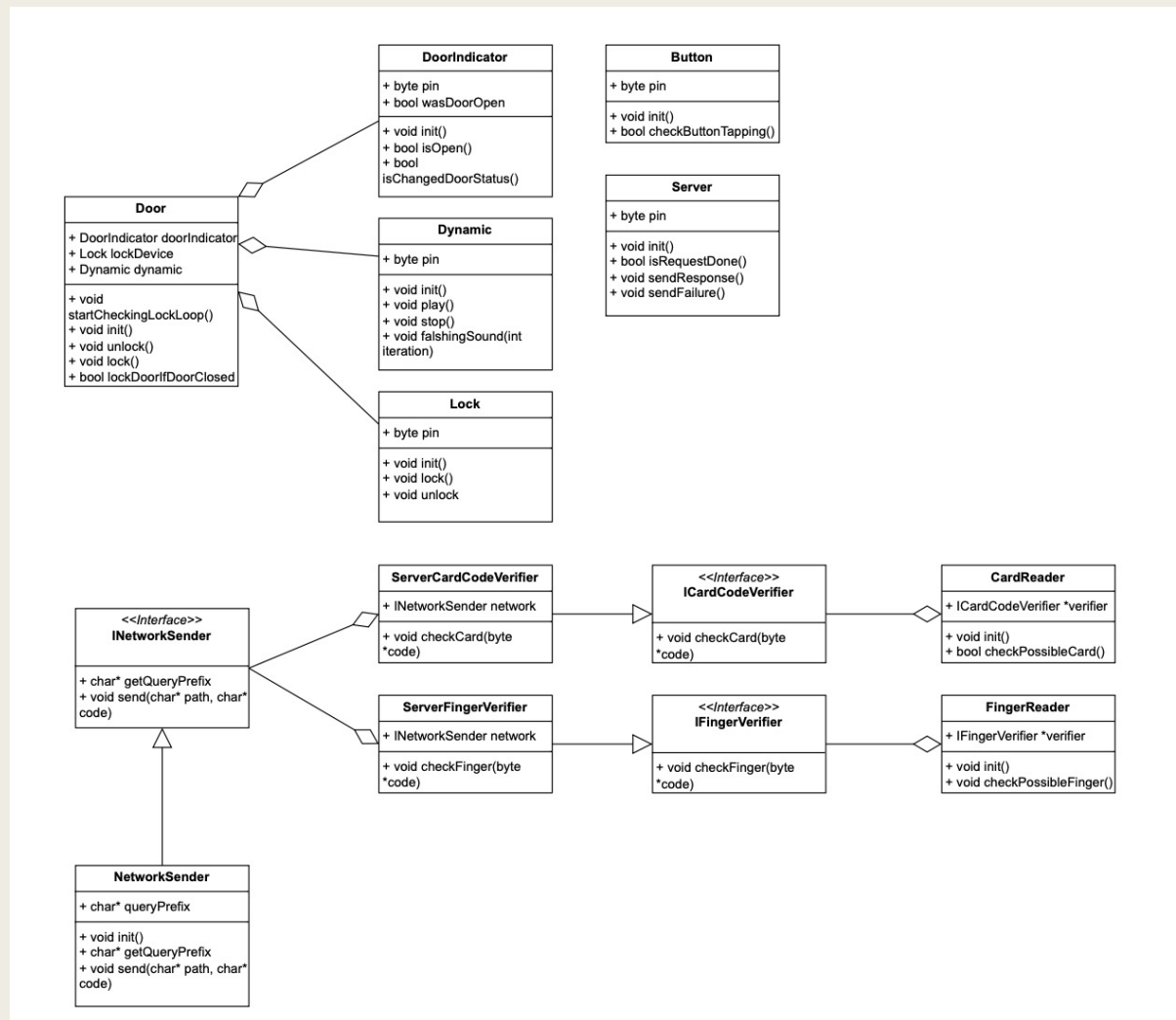
```

12 class MySQLManager: DBManager {
13
14     static let shared = MySQLManager()
15
16     private var db: Database?
17

```

Static property 'shared' is not concurrency-safe because non-'S

Схема класів для коду замка



```

void setup() {
    Serial.begin(9600);
    while (!Serial);
    Serial.println(F("Arduino start"));
    door.init();
    openButton.init();
    cardReader.init();
    fingerReader.init();
    inputServer.init();
    networkSender.init();
}

void loop() {
    door.lockDoorIfDoorClosed();
    if (NetworkSender::isRequestFinished) {
        checkButton();
        cardReader.checkPossibleCard();
        fingerReader.checkPossibleFinger();
    }
    checkServer();
    checkNetworkSender();
}
  
```