# Trabalho Prático Android Kotlin

Relatório técnico

*Arquiteturas Móveis*

*João Vieira | nº 2019127230*

*Pedro Vieira | nº 2019127241*

*Sérgio Soares | nº 2016014425*

*Licenciatura em Engenharia Informática*

*Ramo de Desenvolvimento de Aplicações*

*Instituto Superior de Engenharia de Coimbra*

*Instituto Politécnico de Coimbra*

*Janeiro 2025*

isec
Engenharia

isec Engenharia

# Índice

# Índice de figuras

# Índice de tabelas

# Acrónimos

| Acrónimo | Definição |
|---|---|
| MVVM | *Model-View-ViewModel* |
| UDF | *Unidirectional data flow* |

# 1 Introdução

O presente relatório descreve o desenvolvimento da aplicação *Quizec*, projetada para criar e gerir questionários. Este trabalho foi desenvolvido no âmbito da unidade curricular Arquiteturas Móveis, utilizando Kotlin, Jetpack Compose e Firebase.

A aplicação permite que utilizadores registados criem, editem e respondam a questionários permitindo ainda a criação de perguntas com os seguintes formatos diferentes:

- Perguntas de Sim/não
- Perguntas de escolha múltipla com uma resposta certa
- Perguntas de escolha múltipla com várias respostas certas
- Perguntas de correspondência
- Perguntas de ordenação
- Perguntas de preenchimento de espaços em branco
- Perguntas de associação
- Perguntas com resposta baseada na indicação de palavras

Este relatório documenta os principais passos do projeto, incluindo a análise de requisitos, arquitetura e implementação técnica.

# 2 Requisitos da aplicação

Este capítulo apresenta os requisitos essenciais para o desenvolvimento da aplicação *Quizec*. Os requisitos foram identificados a partir da análise detalhada do enunciado do projeto e têm como objetivo assegurar a funcionalidade e usabilidade da aplicação. Estes foram organizados em duas categorias principais, Requisitos funcionais que descrevem as funcionalidades que a aplicação deve possuir, e os Requisitos não funcionais, que especificam características de qualidade e restrições técnicas.

## 2.1 Requisitos funcionais

Os requisitos funcionais descrevem as operações e funcionalidades que a aplicação deve oferecer para atender às necessidades dos utilizadores. A Tabela 1 lista os requisitos funcionais identificados assim como o seu estado, implementado ou não implementado.

| ID | Descrição | Estado |
|---|---|---|
| RF1 | Gestão de questionários (Criação, edição, remoção) | Implementado |
| RF2 | Gestão de perguntas (Criação, edição, remoção) | Implementado |
| RF3 | Suporta diferentes tipos de perguntas | Implementado |
| RF4 | Permite acesso ao questionário imediato ou apenas quando criador autoriza | Não Implementado |
| RF5 | Ecrã de espera para utilizadores conectados antes da autorização | Não Implementado |
| RF6 | Suporta restrição de localização | Não Implementado |
| RF7 | Permite definir se resultados são apresentados imediatamente ou após final do tempo | Não Implementado |
| RF8 | Criador pode encerrar questionário a qualquer momento | Não Implementado |
| RF9 | Todos os utilizadores devem estar autenticados | Implementado |
| RF10 | Contém histórico de questionários participados | Implementado |
| RF11 | Permite duplicação de perguntas de questionários participados | Implementado |

*Tabela 1. Requisitos funcionais da aplicação*

## 2.2 Requisitos não funcionais

Os requisitos não funcionais definem os tributos de qualidade, como desempenho e usabilidade.

| ID | Descrição | Estado |
|---|---|---|
| RNF1 | Desenvolvimento usando *jetpack Compose* | Implementado |
| RNF2 | Informação armazenada usando serviços (*Firebase*) | Implementado |
| RNF3 | Requer registo e autenticação dos utilizadores | Implementado |
| RNF4 | Idioma principal inglês e suporta português | Implementado |
| RNF5 | Suporta *Portrait* e *Landscape* | Não Implementado |

*Tabela 2. Requisitos não funcionais da aplicação*

# 3 Arquitetura da aplicação

A aplicação *Quizec* segue a arquitetura *Model-View-ViewModel* (MVVM) para garantir a modularidade, escalabilidade facilidade de manutenção. A arquitetura está organizada por duas camadas principais: a camada de dados (*Data layer*) e a camada de apresentação dos dados (*UI layer*) como ilustrado na Figura 1 [1].


*Figura 1.Arquitetura da aplicação*

## 3.1 Camada de Apresentação

A camada de apresentação é composta por *Views* criadas com *jetpack Compose*, responsáveis por renderizar a interface gráfica e interagir diretamente com o utilizador. Por sua vez, *os ViewModels* são responsáveis por armazenar e gerir o estado das *Views,* permitindo que os dados persistam durante mudanças de configuração.

## 3.2 Camada de dados

A camada de dados é composta por Repositórios, que encapsulam a lógica de acesso aos dados. As fontes de dados, são responsáveis por lidar diretamente com a origem dos dados, como serviços (*Firebase*) ou bases de dados locais. Os modelos representam os objetos manipulados pela aplicação, assegurando consistência na troca de informações entre camadas.

A aplicação adota o padrão de *design Unidirectional data flow* (UDF) [2] onde os eventos são gerados pela interface de utilizador e são enviados para os *ViewModels*. Os *ViewModels* processam os eventos, atualizam os estados relevantes e notificam as *Views* de qualquer alteração. Por sua vez, as *Views* exibem os dados atualizados de forma reativa. A Figura 2 ilustra o ciclo de interação entra os *ViewModels* e as *Views*.

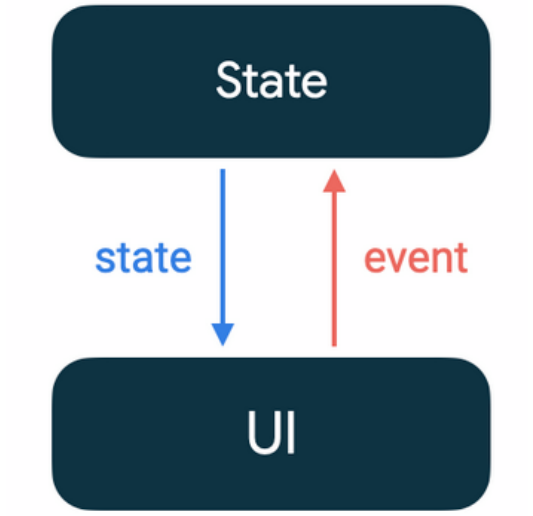


*Figura 2. Fluxo de dados unidirecional (UDF)*

# 4 Implementação

## 4.1 HomePage

O composable HomePage serve como ponto de partida após a autenticação do utilizador, fornece quase que a estrutura base para a aplicação e gere como a navegação é feita entre as diferentes abas, tem:

- TopAppBar que mostra o título da aba atual e o botão para logout.
- ButtonBar que permite a navegação entre as diferentes abas.
- FloatingActionButton que atualiza dinamicamente as suas ações com base na aba selecionada.

```
Scaffold(
    topBar = {
        TopAppBar(
            title = { Text(selectedTab.value.title) },
            actions = {
                IconButton(
                    onClick = {
                        navController.navigate("Login")
                        userViewModel.logOut()
                    },
                    enabled = isUserLoggedIn
                ) {
                    Icon(
                        Icons.AutoMirrored.Filled.ExitToApp,
                        contentDescription = stringResource(R.string.logout_description)
                    )

                }
            }
        )
    },
    bottomBar = {
        BottomBar(onTabSelected = {selected -> selectedTab.value = selected})
    },
    floatingActionButton = {
            FloatingButton(selectedTab.value.floatingActions(navController))
    }
) { paddingValues ->
```

Quanto ao conteúdo, este é renderizado de forma dinâmica consoante a aba selecionada, exemplos de abas:

- CreatedQuestionnaires: Exibe uma lista de questionários criados pelo utilizador.
- ParticipatedQuestionnaires: Mostra questionários em que o utilizador participou.
- UserQuestions: Lista as perguntas guardadas pelo utilizador.
- Settings: Atualmente serve apenas para que o utilizador consiga responder a um questionário.

Esta navegação dinâmica é feita da seguinte maneira:

```
) { paddingValues ->
    Box(
        modifier = Modifier
            .fillMaxSize()
            .padding(paddingValues)
    ) {
        val userId = userViewModel.currentUser.value?.uid

        when (selectedTab.value) {
            HomeTab.CreatedQuestionnaires -> ListQuestionnaires(
                tab = 1,
                userId = userId ?: "",
                questionnaireViewModel = questionnaireViewModel,
                onCopyClick = { navController.navigate("HomePage") },
                onEditClick = { questionnaireId ->
                    navController.navigate("CreateQuestionnaire/$questionnaireId")
                },
                onGetQuestionnaire = {questionnaireViewModel.getQuestionnaires(creatorId = userViewModel.currentUs
            )
            HomeTab.ParticipatedQuestionnaires -> ListQuestionnaires(
                userId = userId ?: "",
                questionViewModel = questionViewModel,
                questionnaireViewModel = questionnaireViewModel,
                onCopyClick = {},
                onEditClick = {},
                onGetQuestionnaire = {
                    questionnaireViewModel.getQuestionnaires(
                        uid = userViewModel.currentUser.value!!.uid
                    )
                }
            )
            HomeTab.UserQuestions -> ListQuestions(
                navController = navController,
                questionViewModel = questionViewModel,
                questionnaireViewModel = questionnaireViewModel,
                userViewModel = userViewModel,
```

## 4.2 BasicOptionsScreen

Este composable trata da maioria dos tipos de perguntas incluindo Yes/No, Multiple Choice, Matching, Ordering e Association.

- Adapta-se aos diferentes tipos de perguntas ajustando a interface e a funcionalidade.
- É usado para criar, editar e responder.
- Utiliza radio button para as perguntas de escolha múltipla apenas uma correta e Yes/No e checkbox para perguntas de escolha múltipla, com varias corretas.

```
options.forEachIndexed { index, option ->
    Row(verticalAlignment = Alignment.CenterVertically) {
        if (type != QuestionType.MATCHING.name && type != QuestionType.CONCEPT_ASSOCIATION.name && type !=
            if (maxCorrectOptions == 1) {
                RadioButton(
                    selected = selectedOptions.contains(index),
                    onClick = {
                        if (isEditable) {
                            selectedOptions.clear()
                            selectedOptions.add(index)
                            correctOptions.clear()
                            correctOptions.add(index)
                            responsesInt?.add(index)
                        }
                    }
                )
            } else {
                Checkbox(
                    checked = selectedOptions.contains(index),
                    onCheckedChange = { isChecked ->
                        if (isEditable) {
                            if (isChecked) {
                                if (maxCorrectOptions == null || selectedOptions.size < maxCorrectOptions)
                                    selectedOptions.add(index)
                                    correctOptions.add(index)
                                    responsesInt?.add(index)
                                }
```

# 5 Conclusão

Quizec foi uma aplicação interessante de fazer por termos enfrentado alguns desafios que nos fizeram pensar e evoluir como desenvolvedores, como foi o caso de tentar criar um composable capaz gerir vários tipos de perguntas de uma forma reutilizável, embora estejamos conscientes que dava para fazer um trabalho melhor nesta questão, ficamos contentes com a evolução que tivemos.

Embora tenham ficado algumas coisas por desenvolver, e outras por otimizar acreditamos que ficamos aqui com um projeto que consegue cumprir o objetivo de servir como uma base para uma plataforma para criação e gestão de questionários dentro de uma sala de aula.

# 6 Referências

[1] “Recommended app architecture,” Google, [Online]. Available: https://developer.android.com/topic/architecture#recommended-app-arch. [Acedido em 19 12 2024].

[2] “Unidirectional data flow,” Google, [Online]. Available: https://developer.android.com/develop/ui/compose/architecture#udf. [Acedido em 19 12 2024].