React Native - Imports, Paths, Envs e Locale

Adonai Pinheiro

1- Ordenação de imports

Para já deixarmos um padrão de importação, vamos utilizar alguns plugins do eslint e algumas configurações no nosso arquivo `.eslintrc.js`.

1. Instalar as libs `yarn add -D eslint-plugin-import eslint-import-resolver-typescript`
2. Modificar o arquivo `.eslintrc.js`:

```
module.exports = {
  root: true,
  plugins: ['@typescript-eslint', 'prettier', 'import'],
  extends: [
    '@react-native',
    'plugin:import/recommended',
    'plugin:import/typescript',
  ],
  rules: {
    'sort-imports': [
      'error',
      {
        ignoreCase: false,
        ignoreDeclarationSort: true,
        ignoreMemberSort: false,
        memberSyntaxSortOrder: ['none', 'all', 'multiple',
'single'],
        allowSeparatedGroups: true,
      },
    ],
    'import/no-unresolved': 'error',
    'import/no-named-as-default-member': 'off',
    'import/order': [
      'error',
```

```
      {
        groups: [
          'builtin',
          'external',
          'internal',
          ['sibling', 'parent'],
          'index',
          'unknown',
        ],
        'newlines-between': 'always',
        alphabetize: {
          order: 'asc',
          caseInsensitive: true,
        },
      },
    ],
  },
  settings: {
    'import/resolver': {
      typescript: {
        project: './tsconfig.json',
      },
    },
  },
};
```

1. Adicionar configuração no arquivo `settings.json` do VSCode

```
{
  "editor.formatOnSave": false,
  "eslint.validate": [
    "typescript"
  ],
  "editor.codeActionsOnSave": {
    "source.fixAll": true}
}
```

1. Adicionar comandos de lint no `package.json`

```
{
    "lint": "eslint --ext ts,tsx ./src",
    "lint:fix": "npm run lint -- --fix",
}
```

2- Configuração inicial dos paths alias

Estaremos seguindo a documentação do React Native.

1. Alteramos o arquivo `tsconfig.json`:

```
{
  "extends": "@tsconfig/react-native/tsconfig.json",
  "compilerOptions": {
    "baseUrl": ".",
    "paths": {
      "@screens": ["src/screens"]
      // Outros paths da aplicação
    }
  }
}
```

1. Instalamos o plugin `module-resolver`

`yarn add --dev babel-plugin-module-resolver`

1. Configuramos o aquivo `babel.config.js`, lembrando que toda vez que adicionarmos um path novo, devemos reiniciar o bundle com o comando `yarn start --reset-cache` para que ele recarregue os paths configurados

```
module.exports = {
  presets: ['module:metro-react-native-babel-preset'],
  plugins: [
    [
      'module-resolver',
      {
        root: ['.'],
        extensions: ['.ios.js', '.android.js', '.js', '.t
```

```
s', '.tsx', '.json'],
        alias: {
          '@screens': './src/screens',
        },
      },
    ],
  ],
};
```

Para que possamos testar, vamos criar uma tela `src/screens/SignIn/index.tsx`.

3- Variáveis de ambiente

No projeto usaremos `react-native-config` para armazenar nossas variáveis de ambiente, podendo ser elas: API_KEYs, URLs de acesso de API, CLIENT_IDs etc. Dessa forma, além de centralizarmos essas informações sensíveis do projeto em um único só lugar, também estaremos prevenindo acessos indevidos e quebras de segurança do nosso projeto. Já que ao buildarmos uma versão para android, o arquivo `index.android.bundle` contém tudo que compõe nossa aplicação do lado Javascript, sendo nossa aplicação inteira minificada, fará referência a CHAVE utilizada, mas não estará vinculada a um VALOR.

6.1. Instalação

https://www.npmjs.com/package/react-native-config?activeTab=readme

`yarn add react-native-config@1.5.1` e `cd ios && pod install`

Já no nosso projeto, criaremos um arquivo `.env` na `root`, no qual armazenaremos todas as nossas chaves da nossa aplicação no modelo CHAVE→VALOR.

```
APP_LANGUAGE=ptBR
ENCRYPT_KEY=5m1l3s4R
```

Como estamos lidando com typescript e queremos tudo no nosso projeto o mais tipado possível, criaremos também um arquivo `react-native-config.d.ts` na `root` do nosso projeto e para cada nova chave criada, adicionaremos ela e o tipo neste arquivo também.

```
declare module 'react-native-config' {
  export interface NativeConfig {
    APP_LANGUAGE?: string;
    ENCRYPT_KEY?: string;
  }

  export const Config: NativeConfig;
  export default Config;
}
```

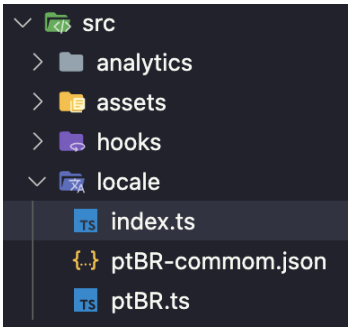
```
import {MMKV} from 'react-native-mmkv';

import Config from 'react-native-config';

export const userStorage = new MMKV({
  id: 'user-storage',
  encryptionKey: Config.ENCRYPT_KEY, // <- aqui acessamos de
});
```

4- Localização

Para os textos e localização na nossa aplicação, usaremos as bibliotecas `i18n` e `react-i18next`. Cada tela deverá conter seu próprio arquivo de strings localizadas, porém teremos na `src` uma pasta que será responsável por centralizar todas as strings em um só lugar.




```
/* eslint-disable import/no-named-as-def
import i18n from 'i18next';
import {initReactI18next} from 'react-i1

import Config from 'react-native-config'

import {ptBR} from './ptBR';

i18n.use(initReactI18next).init({
  compatibilityJSON: 'v3',
```

```
  resources: {
    ptBR,
  },
  lng: Config.APP_LANGUAGE,
  fallbackLng: Config.APP_LANGUAGE,
  interpolation: {
    escapeValue: false,
  },
});
```

Nota-se que não estamos exportando nada neste arquivo, pois o que queremos é que ele seja instanciado assim que a aplicação inicialize e todas as strings estejam disponíveis em tempo de execução. Então, para garantirmos isto, basta adicionar a importação diretamente no entry point da aplicação, que todas as strings serão carregadas corretamente.

Passo extra Android

```
android/app/build.gradle
```

```
apply from: project(':react-native-config').projectDir.getPat
```