ТЕХНОЛОГИЧНО УЧИЛИЩЕ
ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

ДИПЛОМНА РАБОТА

Тема: Мебелен магизин с AR

Дипломант:
Ивайло Банков

Научен ръководител:
Петър Чакъров

София, 2021

Мнение на научен ръководител

Увод

Глава 1.

Методи и технологии за развитие на мобилни приложения

1.1 Основни принципи, технологии и развойни среди за реализация на мобилни приложения

1.1.1 Мобилно приложение

Мобилното приложение е компютърна програма, създадена, за да работи на мобилни устройства като смартфони и таблети. Най-често срещаните мобилни операционни системи са Android на Google и iOS на Apple Inc. Обикновено приложенията са достъпни на онлайн платформи за диситрибуция като App Store, Google Play Store, Galaxy Store и др. Обикновено платформата е построена и се оправлява от компанията собственик на операционната система на устройството.

1.1.2 Езици за програмиране на Android приложения

Приложенията за Android могат да се пишат на езиците Kotlin, Java и C++, с помощта на комплекта за разработка на софтуер за Android - Android SDK, като същевременно е възможно и използване на други езици. Всички езици, които не са използват виртуалната машина Java (напр.: Go, JavaScript, C и т.н.) се нуждаят от помощта на интерпретатор. (Фиг. 1.1)

1.1.3 Езици за програмиране на iOS приложения

При разработката на мобилно iOS приложение освен популярните езици базирани на C (C++/C#) се използва и Swift. Swift е програмен език със свободен код, създаден през 2014 от Apple Inc. специално за разработка на iOS приложения. Базиран е на езикът C и е разработен така че да има огромен спектър на функционалност, тъй като светът на мобилните приложения е много широк. Swift е компилируем език, който се характеризира с това, че превръщат програмния код в машинен. (Фиг. 1.1)

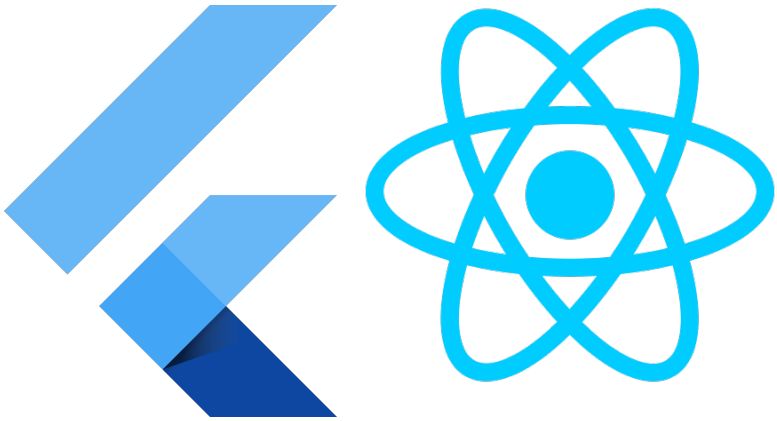
Фигура 1.1: Логота на Android и iOS

1.1.4 Езици за междуплатформени приложения

Когато се отнася до мобилно програмиране, има налични софтуерни рамки, които предоставят възможността за разработка едновременно и за двете гореспоменати мобилни платформи. Примери за това са Flutter, React Native, Xamarin и други. При подобен софтуер за разработка от една кодова база се компилира приложение за много платформи.

Flutter е комплект за разработка на софтуер за потребителски интерфейс с отворен код, създаден от Google. Използва се за разработване на крос-платформени приложения за Android, iOS, Linux, Mac, Windows, Google Fuchsia, както и на уеб страници от единна кодова база. Flutter използва обектно-ориентирания език Dart при разработка, който също е създаден от Google. (Фиг. 1.2)

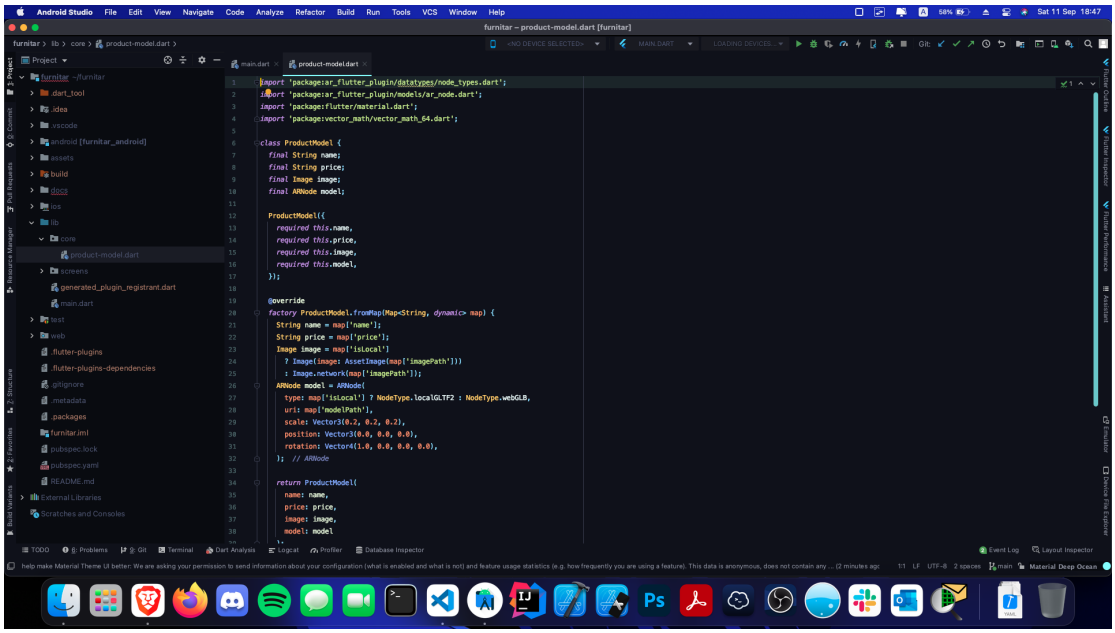
React Native е софтуерна рамка за потребителски интерфейс с отворен код, създадена от Facebook, Inc. Използва се за разработване на приложения за Android, Android TV, iOS, macOS tvOS, Web, Windows и UWP, като дава възможност на разработчиците да използват рамката React заедно с локалните за мобилното устройство функционалности. Често случаи е предпочитана от хора и организации, които се занимават с програмирането на уеб приложения на React, поради голямата сходност между технологиите. (Фиг. 1.2)



Фигура 1.2: Логота на Flutter и React Native

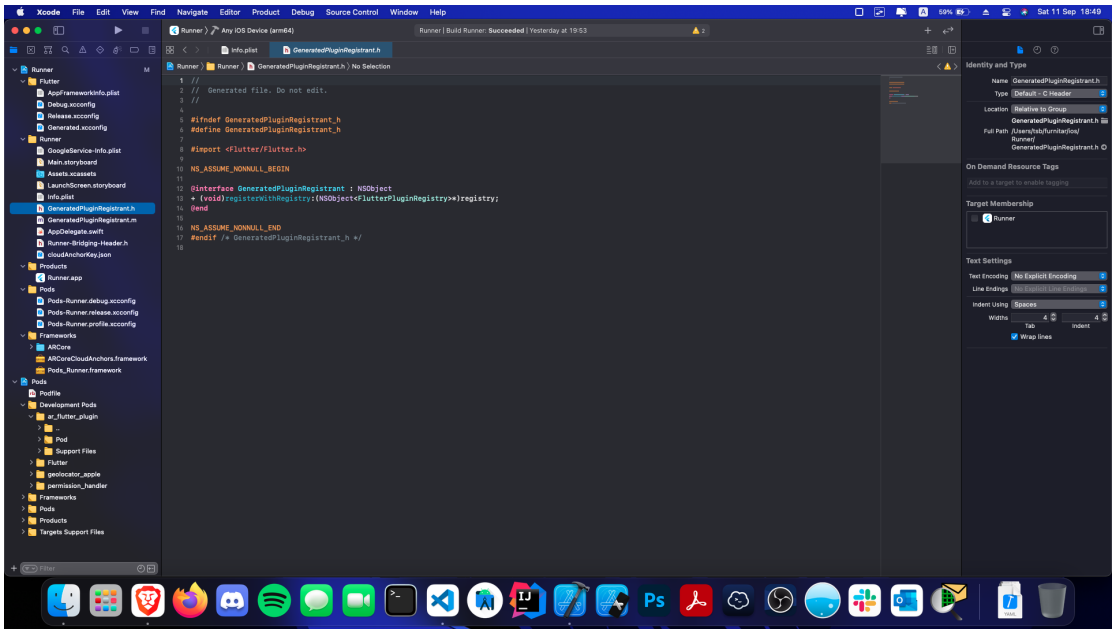
1.1.5 Развойни среди за разработка на мобилни приложения

Основната среда за разработка на Android приложения е Android Studio. Android Studio е разширение на IntelliJ IDEA редактора на JetBrains и се използва едининствено за разработката на мобилни приложения. Редактора предоставя множество инструменти, като Android SDK, Android Virtual Device (ADV) и д.р., които да улеснят работата на разработчиците. (Фиг. 1.3)



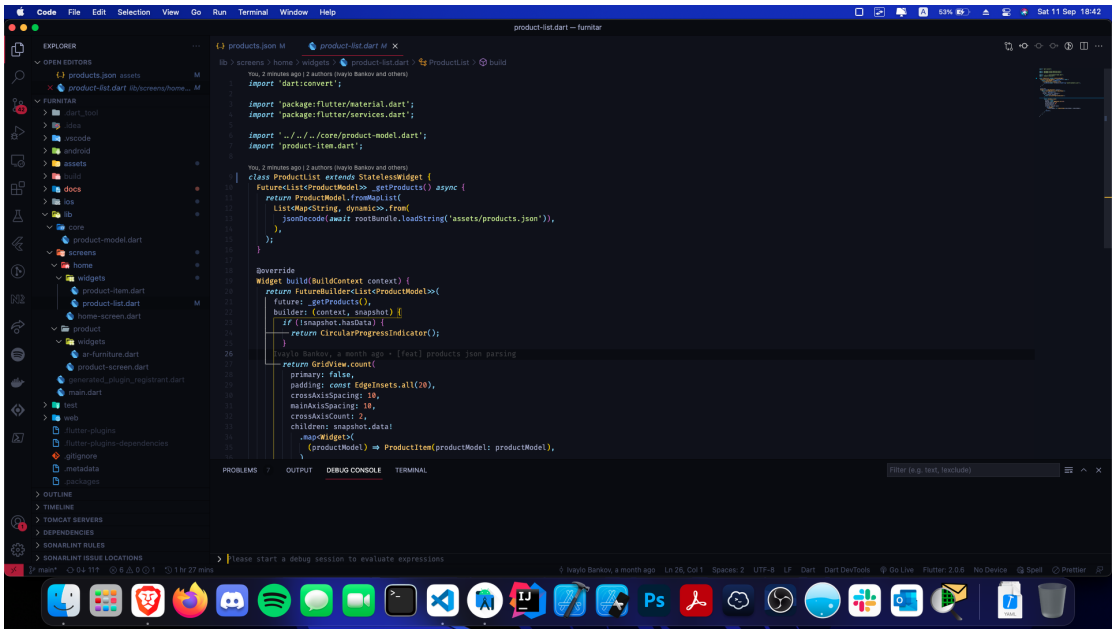
Фигура 1.3: Снимка от Android Studio

Основната среда за разработка на iOS приложения на е Xcode. Xcode е платформа, създадена от Apple Inc., ексклузивно за разработването на приложения и програми за устройства, използващи операционните системи на компанията. Тази платформа се използва масово, защото функционалността на платформата е максимално съвместима с операционните системи на компанията, изграждайки много добра връзка между устройството и кода. Xcode също така предоставя симулации на голяма част от мобилните устройства на компанията, което значително улеснява разработката на приложения за тях. (Фиг. 1.4)



Фигура 1.4: Снимка от XCode

Visual Studio Code е интегрирана среда за разработка, направена от Microsoft за Windows, Linux и macOS. Характеристиките включват поддръжка за отстраняване на грешки, подчертаване на синтаксиса, интелигентно завършване на код, фрагменти, рефакторинг на код и вграден Git. Потребителите могат да променят темата, клавишните комбинации, предпочитанията и да инсталират разширения, които добавят допълнителна функционалност. (Фиг. 1.5)



Фигура 1.5: Снимка от Visual Studio Code

1.1.6 ARCore

ARCore, известен също като Google Play Services за AR, е комплект за разработка на софтуер, разработен от Google, който позволява да се създават приложения за разширена реалност. ARCore използва три ключови технологии за интегриране на виртуално съдържание с реалния свят, видян през камерата на Android мобилното устройство:

- Т.нар “Шест степени на свобода” (на англ. “Six degrees of freedom”) позволяват на телефона да разбира и проследява позицията си спрямо света.
- Екологичното разбиране, наблюдавано от телефона, позволява на телефона да открива размера и местоположението на плоски хоризонтални повърхности като земята или масичка за кафе.

- Оценката на светлината позволява на телефона да прецени текущите условия на осветеност на околната среда.

ARCore е интегриран в множество мобилни Android устройства. (Фиг. 1.6)

1.1.7 ARKit

ARKit съчетава проследяване на движението на iOS устройствата, заснемане на камера, усъвършенствана обработка на сцени и удобства на дисплея, за да опрости задачата за изграждане на AR приложения. Технологията може да създава много видове AR с тези технологии, като се използва предната или задната камера на iOS устройство. (Фиг. 1.6)

Фигура 1.6: Логота на ARCore и ARKit

1.1.8 XML

Extensible Markup Language (накратко XML) е стандарт (метаезик), дефиниращ правила за създаване на специализирани маркиращи езици, както и синтаксисът, на който тези езици трябва да се подчиняват. В последните няколко години, XML губи популярност като формат за съхранение на данни и бива заменен от JSON. (Фиг. 1.7)

```
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
 <glossary><title>example glossary</title>
  <GlossDiv><title>S</title>
   <GlossList>
    <GlossEntry ID="SGML" SortAs="SGML">
     <GlossTerm>Standard Generalized Markup Language</GlossTerm>
     <Acronym>SGML</Acronym>
     <Abbrev>ISO 8879:1986</Abbrev>
     <GlossDef>
      <para>A meta-markup language, used to create markup
languages such as DocBook.</para>
      <GlossSeeAlso OtherTerm="GML">
      <GlossSeeAlso OtherTerm="XML">
     </GlossDef>
     <GlossSee OtherTerm="markup">
    </GlossEntry>
   </GlossList>
  </GlossDiv>
 </glossary>
```

Фигура 1.7: Примерен XML документ

1.1.9 JSON

JavaScript Object Notation (накратко JSON), е текстово базиран отворен стандарт създаден за човешки четим обмен на данни. Произлиза от скриптовия език JavaScript, за да представя прости структури от данни и асоциативни масиви, наречени обекти. Въпреки своята връзка с JavaScript, това е езиково независима спецификация, с анализатори, които могат да преобразуват много други езици в JSON. Основно JSON се използва при разработката на REST приложнопрограни интерфейси. (Фиг. 1.8)

```
{
    "glossary": {
        "title": "example glossary",
        "GlossDiv": {
            "title": "S",
            "GlossList": {
                "GlossEntry": {
                    "ID": "SGML",
                    "SortAs": "SGML",
                    "GlossTerm": "Standard Generalized Markup Language",
                    "Acronym": "SGML",
                    "Abbrev": "ISO 8879:1986",
                    "GlossDef": {
                        "para": "A meta-markup language, used to create
                            markup languages such as DocBook.",
                        "GlossSeeAlso": [
                            "GML",
                            "XML"
                        ]
                    },
                    "GlossSee": "markup"
                }
            }
        }
    }
}
```

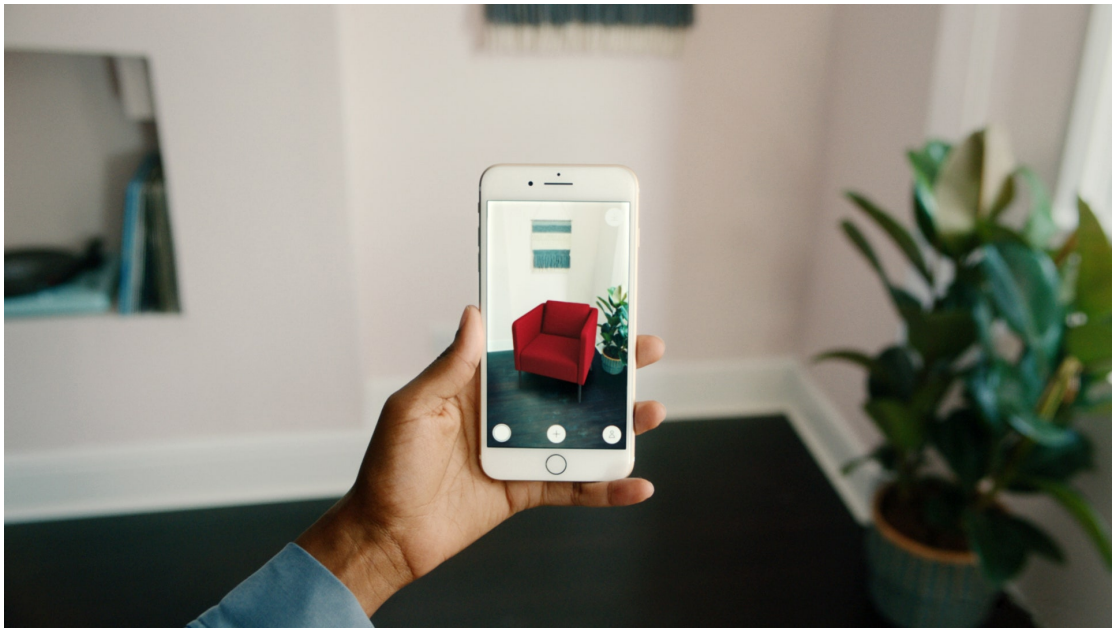
Фигура 1.8: Примерен JSON документ

1.2 Съществуващи решения и реализации

В световен мащаб, идеята за проектиране на мебели посредством технологии за разширена реалност, не е нещо непознато. Компании в мебелния бизнес, като IKEA, Home Depot и др, предоставят

подобни решения, които целят да улеснят клентите при избора им на предмети за дома.

IKEA Place е приложение на IKEA, което позволява на потребителите да “поставят” продуктите на IKEA във жилището си. Предлагано в App Store за iOS устройства, IKEA Place позволява визуализирането на обзавеждане, от дивани и лампи, до килими и маси, всички продукти в IKEA Place са 3D и в мащаб.Това помага на клентите да се уверят, че мебелта, която поръчват, е подходящият размер, с подходящия дизайн и функционалност. (Фиг. 1.9)



Фигура 1.9: Мобилното приложение IKEA Place

Приложението, разглеждано в тази дипломна работа, има преимуществото да се изпълнява и на Android, освен на iOS устройства.

Глава 2.

Проектиране на структура на мебелен магазин използващ AR

2.1 Функционални изисквания към мебелен магазин с AR

- **Прототип на Android приложение с AR интеграция показващо 1 статична картинка**

  Изискването предполага генерирането на ново Android приложение и създаването на екран, който да използва камерата и AR възможностите на устройството, за да визуализира мебели в реалното пространство. Изискването е изпълнено, като освен изпълнение на Android, приложението работи и на iOS.

- **Създаване на каталог с мебели**

  Изискването предполага добавянето на нов екран към приложението, създадено в предходната стъпка. На този екран трябва да се представят няколко възможни мебели, от които потребителя да може да избере. Изискването е изпълнено.

- **Създаване на API за достъп до каталога през Android**

приложението

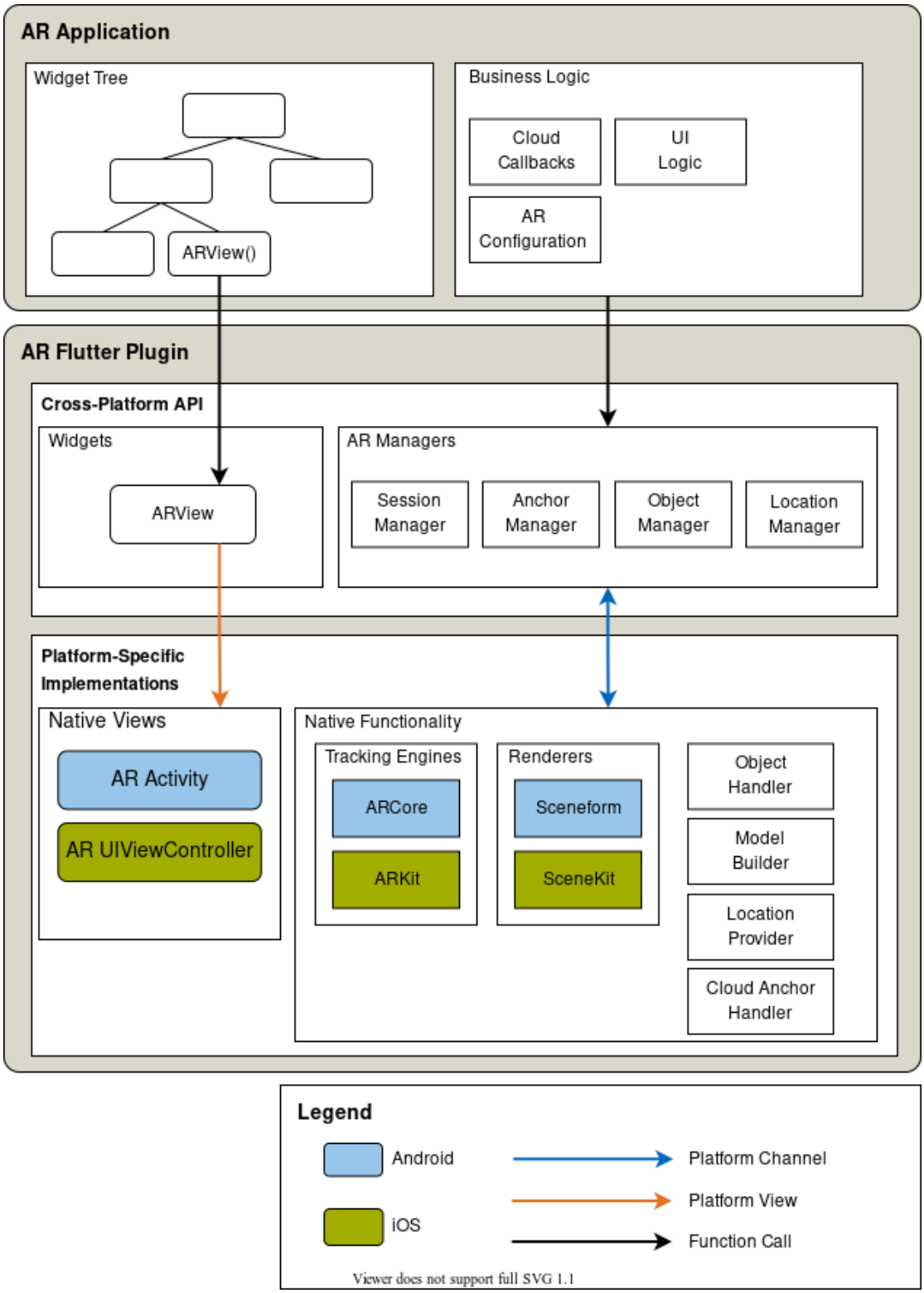
Предпоставя създаване на сървър предоставящ списък с предлаганите мебели в магазина. Също така се очаква предоставянето на статични ресурси, като 3D моделите и снимките на мебелите. Изискването не е изпълнено.

- **Интеграция на Android с приложението с API**

  Предполага връзката между сървър от предходната точка и мобилното приложение. Изискването е частично изпълнено, тъй като обработката на данни в JSON формата е имплементирана в приложението.

2.2 Съобръжения за избор на програмни средства и развойна среда

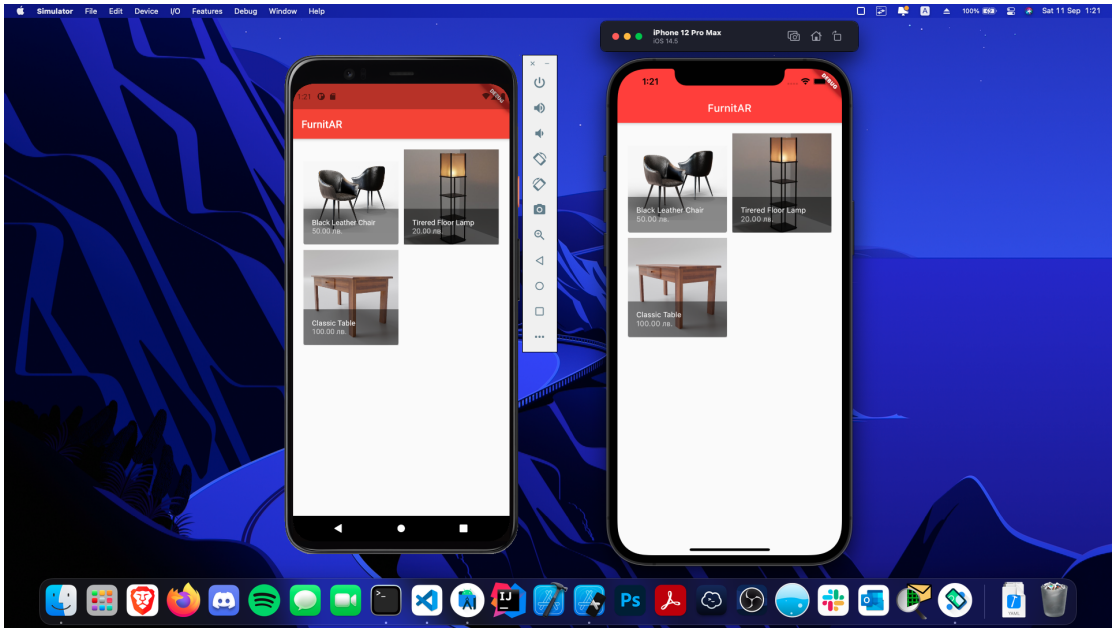
За създаване на мобилното приложение беше избран комплекта за разработване на софтуер Flutter на програмния език Dart. Това решение има предимството на гореспоменатите междуплатформени технологии, а именно, че приложението може да се изпълнява както на Android, така и на iOS устройства. Друго съображение за избора на Flutter пред други технологии за разработка, беше наличието на вече разработената библиотеката ar-flutter-plugin. Тази библиотека служи за имплементиране на аугментирана реалност, като за целта превежда подадените ѝ инструкции (описани в т. 3.2.) да работят със съответния локален приложно програмен интерфейс (ARCore, ARKit) за съответната система (Android, iOS). (Фиг. 2.1)



Фигура 2.1: Диаграма на архитектурата на ar-flutter-plugin

За съхранение на информацията за мебелите, беше избран JSON формата. Това се дължи главно на лекотата, с която може да се промени кода на приложението, при изпълнение на условието от заданието за разработка на приложнопрограненен интерфейс. JSON също така има преимуществото да използва по-малко пространство при съжраняване на същото количество данни спрямо XML.

По време на разработка, програмата Visual Studio Code беше използвана като развойна среда, за писане на програмния код на мобилното приложение. Основната причина за това беше наличието на добавки към редактора, за поддръжка на всички технологии използвани в проекта (Flutter, TeX, т.н.) Въпреки това, използвани бяха и симулационните компоненти Android Virtual Device (ADV) и Simulator, от Android Studio и XCode съответно, за тестване на приложението. (Фиг. 2.2)



Фигура 2.2: Android Virtual Device (ADV) и XCode Simulator изпълняващи мобилното приложение разглеждано в тази дипломна работа

Глава 3.

Програмна реализация на мебелен магизин с AR функционалност

3.1 Реализация на началния екран на приложението

Началният екран на мобилното приложение съдържа в себе си галерия с всичкимебели, които се предлагат. Всяка мебел се дефинира от 5 качества, а именно:

- **name**

  Низ от символи, който определя името на предлагания продукт.

- **price**

  Низ от символи, който определя името на цената продукт.

- **imagePath**

  Низ от символи, който определя локацията, на която снимка на мебелта може да бъде достъпена. Тази локация може да е локална, на устройството на потребителя или глобална, в интернет мрежата.

- **modelPath**

  Низ от символи, който определя локацията, на която 3D модела може да бъде достъпен. Тази локация може да е локална, на устройството на потребителя или глобална, в интернет мрежата.

- **isLocal**

  Булева променива, която определя дали изображението и 3D модела на предмета се намират локално на устройството или в мрежата.

Тези качества са структорирани в JSON файл. Примерно, ако трябва да се съхрани конфигурация, съдържаща в себе си 3 предмета (стол, маса и хладилник), структората на JSON документа би изглеждала така:

```
[
    {
        "name": "Black Leather Chair",
        "price": "$50.00",
        "imagePath": "assets/chair.jpeg",
        "modelPath": "assets/chair.glb",
        "isLocal": true
    },
    {
        "name": "Tirered Floor Lamp",
        "price": "$20.00",
        "imagePath": "assets/lamp.jpeg",
        "modelPath": "assets/lamp.glb",
        "isLocal": true
    },
    {
        "name": "Classic Table",
        "price": "$100.00",
        "imagePath": "assets/table.jpeg",
        "modelPath": "assets/table.glb",
```

```
        "isLocal": true
    }
]
```

В кода на приложението е дефиниран клас за подобни продукти - ProductModel, който съдържа в себе си обработената информация от JSON файла. Важно е да се отбележи, че пътя към снимката на предмета е заменен с обект съдържащ самата снимка. Същото се отнася и за 3D модела на мебелта, която по-късно ще се визуализира. Променливата *isLocal* не се съхранява, но се използва при създаването на обектите на изображението и 3D модела:

```
class ProductModel {
  final String name;
  final String price;
  final Image image;
  final ARNode model;

  ProductModel({
    required this.name,
    required this.price,
    required this.image,
    required this.model,
  });
```

В приложението, също така, е дефиниран метод, който да итерира през JSON списаци с мебели и да ги конвертира към ProductModel обекти:

```
  Future<List<ProductModel>> _getProducts() async {
    return ProductModel.fromMapList(
      List<Map<String, dynamic>>.from(
        jsonDecode(await rootBundle.loadString('assets/products.json')),
      ),
    );
  }
```

За целта се използва метода на ProductModel - fromMapList,

който получава списък от карти, съдържащи в себе си двойки ключ-стойност (на англ. “key-value pairs”). Този списък представлява зареденият JSON файл. В метода се създава нов списък, получения като аргумент списък се обхожда, като на всеки елемент се извиква метода *ProductModel.fromMap* и резултата се запазва в новия списък:

```
1  static List<ProductModel> fromMapList(List<Map<String, dynamic>>
       mapList) {
2    List<ProductModel> products = [];
3    mapList.forEach((mapItem) {
4      products.add(ProductModel.fromMap(mapItem));
5    });
6    return products;
7  }
```

Статичния метод ProductModel.fromMap получава една карта, съдържащи в себе си двойки ключ-стойност и създава ProductModel обект от нея. Името и цената остават непроменени. При създаване на обекта за изображението на предмета, първо се проверява да ли то е достъпно онлайн или в локалната памет на телефона, като за целта се използва *isLocal* променливата. След това се извиква съответния метод на вградения във Flutter клас *Image*. Аналогична е ситулацията при създаване на обекта за 3D модела на предмета. Единственото нещо, което трябва да се допълни тук, че всички модели на предмети имат едно и също оразмеряване, позиция (положението на телефона) и завъртане:

```
1  @override
2  factory ProductModel.fromMap(Map<String, dynamic> map) {
3    String name = map['name'];
4    String price = map['price'];
5    Image image = map['isLocal']
6      ? Image(image: AssetImage(map['imagePath']))
7      : Image.network(map['imagePath']);
8    ARNode model = ARNode(
```

```
      type: map['isLocal'] ? NodeType.localGLTF2 : NodeType.webGLB,
      uri: map['modelPath'],
      scale: Vector3(10.0, 10.0, 10.0),
      position: Vector3(0.0, 0.0, 0.0),
      rotation: Vector4(1.0, 0.0, 0.0, 0.0),
    );

    return ProductModel(
      name: name,
      price: price,
      image: image,
      model: model
    );
  }
```

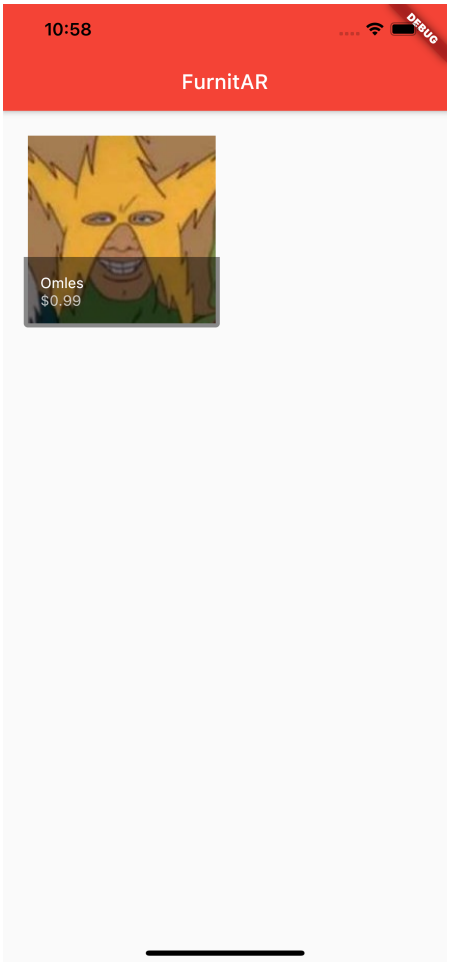
За клиентската част на приложението се използва вградения във Flutter FutureBuilder елемент. FutureBuilder се използва за визуализиране на асинхронни данни - функцията getProduct() е асинхронна, понеже самото четене на JSON файла от локалната памет е асинхронно. Докато данните още се четат от паметта, се визуализира индикатор за зареждане; в противен случай се показва самата галерия от мебели:

```
@override
Widget build(BuildContext context) {
  return FutureBuilder<List<ProductModel>>(
    future: _getProducts(),
    builder: (context, snapshot) {
      if (!snapshot.hasData) {
        return CircularProgressIndicator();
      }

      return GridView.count(
        primary: false,
        padding: const EdgeInsets.all(20),
        crossAxisSpacing: 10,
        mainAxisSpacing: 10,
```

```
            crossAxisCount: 2,
            children: snapshot.data!
              .map<Widget>(
                (productModel) => ProductItem(productModel: productModel),
              )
              .toList(),
          );
        }
    );
  }
```

Зареденатa версия на началния екран може да се наблюдава на фиг. 3.1.



Фигура 3.1: Начален екран на мобилното приложение

3.2 Реализация на AR функционалността

Екранът с за разглеждане на продукт получава като аргументи гореописаната информация за продукта.

```
class ARFurniture extends StatelessWidget {
  final ARNode model;

  ARFurniture({
    Key? key,
    required this.model,
  }) : super(key: key);

  late ARSessionManager arSessionManager;
  late ARObjectManager arObjectManager;
```

AR функционалност е разработен с помоща на ar-flutter-plugin библиотеката. Използва се ARView елемента, който се визуализира на цял екран. Той служи за създаване на "прозорец" за аугментирана реалност. ARView получава два аргумента:

- **onARViewCreated** Функция, която да се изпълни при зареждане на елемента;
- **PlaneDetectionConfig** Определя за какъв тип равнини трябва ARView да следи.

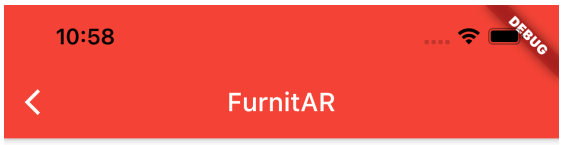
```
  @override
  Widget build(BuildContext context) {
    return Expanded(
      child: ARView(
        onARViewCreated: onARViewCreated,
        planeDetectionConfig: PlaneDetectionConfig.horizontalAndVertical,
      ),
    );
  }
```

Метода, който се изпълнява при създаване на AR прозореца, инициализира в себе си мениджър на настоящата сесия - *arSessionManager* и мениджър на обекти - *arObjectManager*. След това в управителя на 3D обекти се добавя модела на предмета, който трябва да се визуализира в пространството. Показването на информация, служеща за дебъгинг, като равнини, отправна точка и др. е изключена:

```
  void onARViewCreated(
    ARSessionManager arSessionManager,
    ARObjectManager arObjectManager,
    ARAnchorManager arAnchorManager,
    ARLocationManager arLocationManager,
  ) {
    this.arSessionManager = arSessionManager;
    this.arObjectManager = arObjectManager;

    this.arSessionManager.onInitialize(
          showFeaturePoints: false,
          showPlanes: false,
          showWorldOrigin: false,
          handleTaps: false,
        );
    this.arObjectManager.onInitialize();
    this.arObjectManager.addNode(this.model);
  }
}
```

Завършената версия на екрана с AR функционалност може да се наблюдава на фиг. 3.2.



Фигура 3.2: AR визуализация в мобилното приложение

Глава 4.

Ръководство за потребителя

4.1 Инсталация на приложението

За инсталиране на приложението на мобилно устройство, е нужно да се навигира в директорията на проекта и да се изпълни командата:

```
flutter install
```

За да се генерират инсталационните файлове за Android, трябва да се изпълни следната команда:

```
flutter build apk
```

Това ще генерира 3 *.apk* инсталационни файла, които могат да бъдат качени на мобилното устройство и инсталирани:

- */build/app/outputs/apk/release/app-armabi-v7a-release.apk*
- */build/app/outputs/apk/release/app-arm64-v8a-release.apk*
- */build/app/outputs/apk/release/app-x86-64-release.apk*

За да се построи приложението за iOS, командата, следната команда трябва да бъде изпълнена:

```
flutter build ipa
```

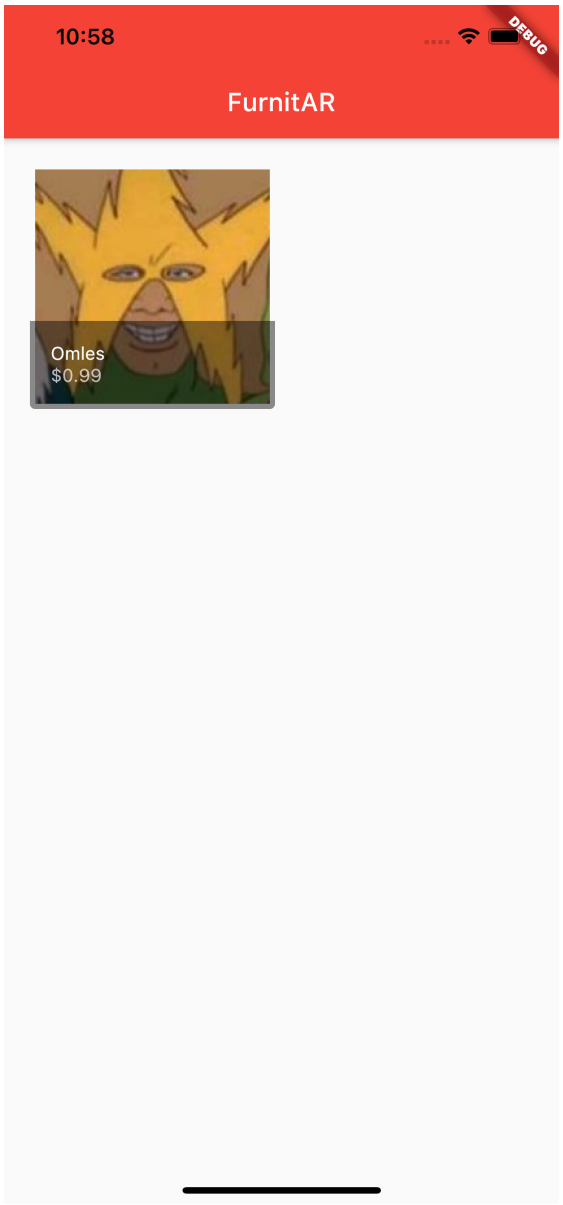
Това ще генерира */build/ios/archive/DRUN.xcarchive* архив, съдържащ инсталационните файлове.

Важно е да се отбележи, че приложението не е налично за всички модели телефони. Това се дължи на ограничената поддръжка на ARCore и на ARKit. За Android устройства също така е нужно да се инсталира *Google Play Services for AR* апликацията, която е налична в *Google Play Store*.

Списък със съвместими iOS устройства може да бъде достъпен на адрес: https://www.cnet.com/tech/mobile/these-iphones-and-ipads-work-with-arkit/. Списък със съвместими Android устройства може да бъде достъпен на адрес: https://developers.google.com/ar/devices/.

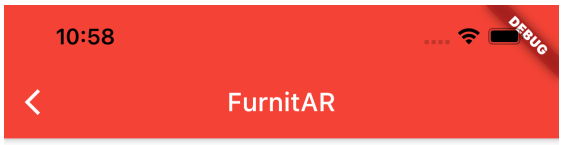
4.2 Употреба на приложението

При стартиране на приложението се показва галерия с всички налични предмети, името и цената им. (Фиг. 4.1)



Фигура 4.1: Начален екран на мобилното приложение

От тази галерия може да се избере продукта, който да бъде визуализиран посредством аугментирана реалност. Преди избор от менюто, мобилното устройство трябва да се постави малко над мястото, където е желано да се проектира 3D модела на мебелта. След избор, камерата може да бъде преместена, като проекцията запазва положението си в пространството. (Фиг. 4.2)



Фигура 4.2: AR визуализация в мобилното приложение

Заключение

Чрез разработването на мобилно визуализиране на мебели в пространството, разглеждано в контекста на тази дипломна работа беше постигнато едно функционално решение на проблем, който е приложим на множество хора, които са закупили ново или обзавеждат настоящето си жилище.

В рамките на дипломната работа бяха постигнати следните цели:

- Разработено беше мобилно приложение за Android и iOS с AR интеграция за визуализиране на мебели;
- Създаден беше каталог с мебели, от които потребителя да може да избира;
- Направен беше механизма за работене с мебелните данни структорирани в JSON формата. Това позволява лесната интеграция на приложението с приложно-програмен интерфейс разработен в бъдеще.

Бъдещи подобрения на приложението биха включвали, примерно:

- Разработката на централизиран сървър, който да предоставя списък с мебелите;
- Възможността за завъртане на мебелите в пространството;

- Възможността за сортиране и търсене по категориите на мебелите.

Библиография

[1] CGTrader. *3D Models for VR / AR and CG projects*. URL: https://www.cgtrader.com/ (дата на посещ. 11.09.2021).

[2] Oleksandr Leuschenko | Flutter Europe. *Augmented Reality in Flutter*. URL: https://www.youtube.com/watch?v=lmiRYar996w (дата на посещ. 11.09.2021).

[3] *Flutter ArKit Tutorial for Beginners - IOS AR*. URL: https://www.youtube.com/watch?v=Du_xxiDyo1A (дата на посещ. 11.09.2021).

[4] TeX Users Group. *Getting started with TeX, LaTeX, and friends*. URL: https://www.tug.org/begin.html (дата на посещ. 11.09.2021).

[5] Apple Inc. *Apple ARKit Documentation*. URL: https://developer.apple.com/documentation/arkit (дата на посещ. 11.09.2021).

[6] Apple Inc. *Apple Developer Documentation*. URL: https://developer.apple.com/documentation/ (дата на посещ. 11.09.2021).

[7] Carius Lars. *ar-flutter-plugin: Flutter Plugin for AR - GitHub*. URL: https://github.com/CariusLars/ar_flutter_plugin (дата на посещ. 11.09.2021).

[8] Google LLC. *Android Documentation*. URL: https://developer.android.com/ (дата на посещ. 11.09.2021).

[9] Google LLC. *ARCore Documentation*. URL: https://developers.google.com/ar (дата на посещ. 11.09.2021).

[10] Google LLC. *Flutter Documentation*. URL: `https://flutter.dev/docs` (дата на посещ. 11.09.2021).

[11] Google LLC. *Flutter Gallery*. URL: `https://gallery.flutter.dev/` (дата на посещ. 11.09.2021).

Съдържание