Food Delivery Android App Creation

Team ID : Team-590948

Team members:

- Kaushal Francis J | 21BPS1438
- Kavin S Kumar | 21BCE1737
- Agatsya Yadav | 21BCE1902
- Jayminkumar Mukeshkumar Panchal | 21BCE1573

1. INTRODUCTION:

1.1 Project overview:

This Project aims to create a food delivery service android app where costumers can order food online and the food will be delivered to them.

1.2 Purpose:

The Purpose is to make the delivery process easier using an android app where we can eat delicacies and all kinds of foods with the luxury of not going there and eating.

2. Literature Survey:

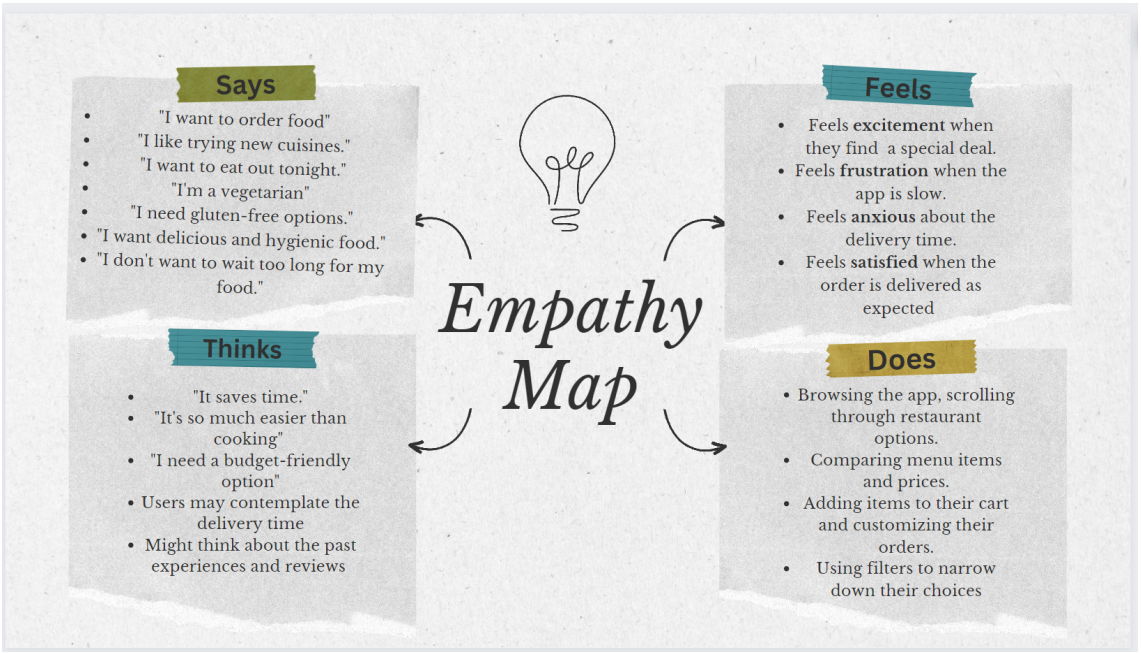
2.1 Existing Problem:

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	In today's fast-paced world, the process of ordering food can be time-consuming and inefficient, especially when it involves standing in long queues at restaurants or food establishments. This not only wastes valuable time for customers but also leads to frustration and dissatisfaction. Therefore, there is a need for a solution that enables customers to order food conveniently, either online or in-store, while minimizing the waste of time spent waiting in queues.

2.	Idea / Solution description	Making an application for ordering food online or instore(less waste of time standing in a queue)
3.	Novelty / Uniqueness	No login required. Just search for your favorite restaurant and order.
4.	Social Impact / Customer Satisfaction	Easy to use.
5.	Business Model (Revenue Model)	Commission-based model: The application can charge a commission fee from the restaurants or food establishments for each order placed through the app.
6.	Scalability of the Solution	Partnering with more restaurants: The application can onboard more restaurants and food establishments onto its platform, providing customers with a wider selection of options and increasing the number of orders.

3. Ideation Phase:

3.1 Empathy Map:



3.2 Brainstorming Ideas (From High to low priority):

1. Design a **User-Friendly, lag-free** interface that simplifies the process of browsing restaurants, placing orders, and making payments.
2. **Filtering** out the restaurants according to the need of the user
3. **Secure payment** options with **various methods** to pay.
4. **Tracking orders in real time** with the contact of the deliverer.
5. Provide a robust **user rating system** where user shares their experiences about the restaurant's food quality
6. **Diverse selection of cuisines** from different restaurants, with a specific focus on vegetarian and gluten-free options.
7. **Delivering on time** to ensure costumer's urgency is protected and hunger is rescued.
8. **Personalize recommendations** for user based on the past order and favorites of the user.
9. **Show all discount** available and some eye-catching sales.

4. Requirement Analysis

4.1 Functional Requirements:

- The app must allow users to browse a list of restaurants.
- The app must allow users to view the menus of restaurants.
- The app must allow users to place orders for food.
- The app must allow users to track the status of their orders.
- The app must allow users to pay for their orders.

4.2 Non-Functional Requirements:

- The app must be available 24/7.
- The app must be able to handle a large number of users.

- The app must be secure.
- The app must be easy to use.

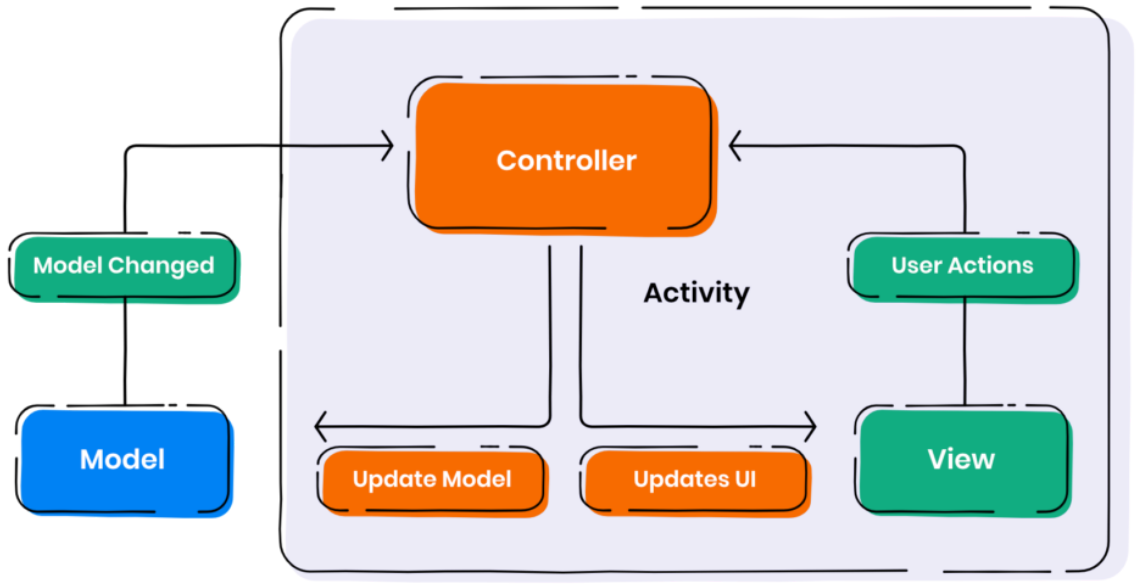
4.3 Additional Requirements:

- The app should allow users to filter restaurants by cuisine, price range, and other criteria.
- The app should allow users to save their favorite restaurants.
- The app should allow users to receive push notifications for order updates.
- The app should allow users to rate and review restaurants.

5. Project Design Phase:

5.1 Solution Architecture Diagram:

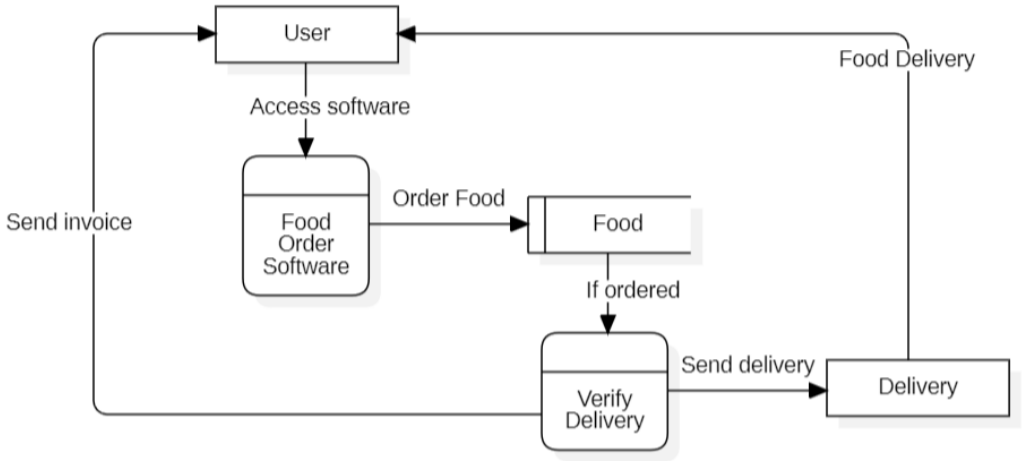
Diagram 1 - MVC (Model - View - Controller)



Scalable Path 2020

5.2 Data flow Diagram:

Level 1:



6. Project Planning:

6.1 Technology Stack:

- **Frontend:**
    - **Jetpack Compose:** Jetpack Compose is a declarative UI toolkit for Android developed by Google. It is based on the Compose Kotlin compiler plugin and provides a declarative way to build UI components.
    - **Firebase Authentication:** Firebase Authentication is a service that provides secure user authentication for Android apps. It allows you to add sign-in and sign-up functionality to your app using various methods, such as email and password, phone number, and social media accounts.
- **Backend:**
    - **Firebase Cloud Firestore:** Firebase Cloud Firestore is a NoSQL document database that allows you to store and retrieve data in real time. It is a fully managed cloud service, so you don't need to worry about managing servers or infrastructure.
    - **Firebase Cloud Functions:** Firebase Cloud Functions is a serverless platform that allows you to run code in response to events, such as changes to data in Cloud Firestore. This allows you to build real-time applications that can react to changes in data without having

to manage your own servers.

- **Firebase Cloud Messaging:** Firebase Cloud Messaging is a service that allows you to send push notifications to your app users. This allows you to keep your users up-to-date with the latest information, such as new orders or delivery updates.
- **Firebase Analytics**: Firebase Analytics is a service that allows you to collect and analyze data about how your app is being used. This data can help you to understand your users, improve your app, and make better business decisions.

- **Other tools and technologies:**
  - **Kotlin**: Kotlin is a programming language that is fully interoperable with Java and is the preferred language for Android development.
  - **Android Studio:** Android Studio is the official integrated development environment (IDE) for Android app development. It is based on IntelliJ IDEA and provides a variety of tools and features to help you develop Android apps.

6.2 Sprint Planning and estimation:

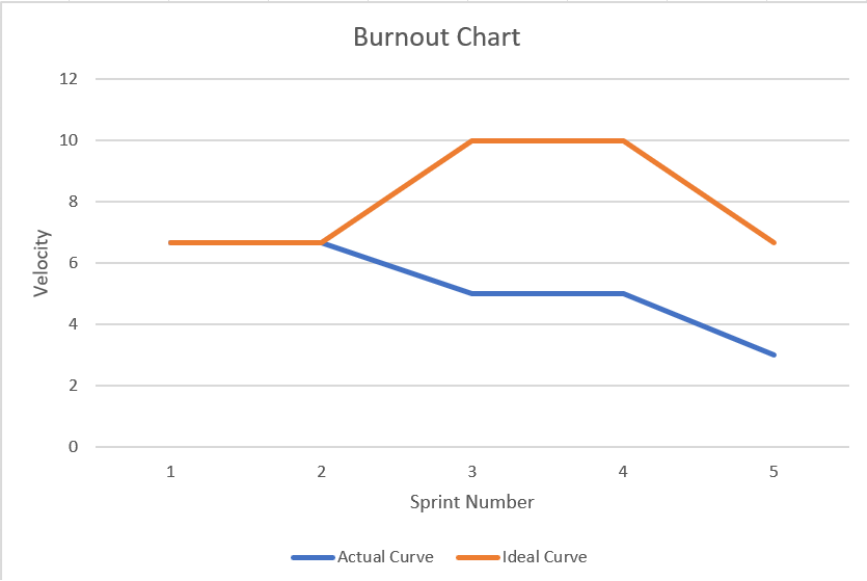
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Home Screen	USN-1	As a user, I want a pleasant looking app	1	High	Kavin S Kumar, Agatsya Yadav, Kaushal Francis J, Jayminkumar
Sprint-2	Restaurant Menu	USN-2	As a user, I want to look at all the available foods of the restaurant.	2	High	Kavin S Kumar, Agatsya Yadav, Kaushal Francis J, Jayminkumar

Sprint-3	Restaurant List	USN-3	As a user, I want to choose which restaurant I want to order food from.	2	High	Kavin S Kumar, Agatsya Yadav, Kaushal Francis J, Jayminkumar
Sprint-4	Launch Screen	USN-4	As a user, I want a good-looking launching animation	1	Low	Kavin S Kumar, Agatsya Yadav, Kaushal Francis J, Jayminkumar
Sprint-5	View Bill	USN-5	As a user, I can see the bill of the food that I ordered	2	Medium	Kavin S Kumar, Agatsya Yadav, Kaushal Francis J, Jayminkumar

6.3 Sprint Delivery Schedule:

Sprint	Total Story Points	Duration (Planned)	Duration (Actual)	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)	Average Velocity Of the Project
Sprint-1	20	3 Days	3 Days	27 Oct 2023	29 Oct 2023	20	29 Oct 2023	6.67
Sprint-2	20	3 Days	3 Days	29 Oct 2023	01 Nov 2023	20	01 Nov 2023	6.67
Sprint-3	20	2 Days	3 Days	01 Nov 2023	02 Nov 2023	15	03 Nov 2023	5
Sprint-4	20	2 Days	3 Days	02 Nov 2023	04 Nov 2023	15	05 Nov 2023	5
Sprint-5	20	3 Days	5 Days	04 Nov 2023	06 Nov 2023	15	09 Nov 2023	3

Burnout Chart:



7. Coding and Solutioning:

7.1 MainActivity.kt:

This Activity file is the main file where the app is executed.

```
class MainActivity : AppCompatActivity(),
RestaurantListAdapter.RestaurantListClickListener {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val actionBar: ActionBar? = supportActionBar
        actionBar?.setTitle("Restaurant List")

        val restaurantModel = getRestaurantData()
        initRecyclerView(restaurantModel)
    }

    private fun initRecyclerView(restaurantList: List<RestaurentModel?>?) {
        val recyclerViewRestaurant =
findViewById<RecyclerView>(R.id.recyclerViewRestaurant)
```

```
        recyclerViewRestaurant.layoutManager = LinearLayoutManager(this)
        val adapter = RestaurantListAdapter(restaurantList, this)
        recyclerViewRestaurant.adapter =adapter
    }

    private fun getRestaurantData(): List<RestaurentModel?>? {
        val inputStream: InputStream =
resources.openRawResource(R.raw.restaurant)
        val writer: Writer = StringWriter()
        val buffer = CharArray(1024)
        try {
            val reader: Reader = BufferedReader(InputStreamReader(inputStream,
"UTF-8"))
            var n : Int
            while (reader.read(buffer).also { n = it } != -1) {
                writer.write(buffer, 0, n)

            }

        }catch (e: Exception){}
        val jsonStr: String = writer.toString()
        val gson = Gson()
        val restaurantModel = gson.fromJson<Array<RestaurentModel>>(jsonStr,
Array<RestaurentModel>::class.java).toList()

        return restaurantModel
    }

    override fun onItemClick(restaurantModel: RestaurentModel) {
       val intent = Intent(this@MainActivity,
RestaurantMenuActivity::class.java)
        intent.putExtra("RestaurantModel", restaurantModel)
        startActivity(intent)
    }
 }
```

7.2 SplashActivity.kt

This activity is used to create a launching animation which increases the richness of the application.

```
class SplashActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        setContentView(R.layout.activity_splash)

        val actionBar: ActionBar? = supportActionBar
        actionBar?.hide()



        Handler().postDelayed(Runnable {
            startActivity(Intent(this@SplashActivity,
MainActivity::class.java))
            finish()
        }, 2000)
    }
}
```

7.3 RestaurantModel.kt

This activity consists of the design of the restaurant's menu which is consistent throughout the application.

```
data class RestaurentModel(val name: String?, val address: String?, val
delivery_charge: String?,
                 val image: String?, val hours: Hours?, var menus:
List<Menus?>?) : Parcelable {
    constructor(parcel: Parcel) : this(
        parcel.readString(),
        parcel.readString(),
        parcel.readString(),
        parcel.readString(),
        parcel.readParcelable(Hours::class.java.classLoader),
        parcel.createTypedArrayList(Menus)
    ) {
    }

    override fun writeToParcel(parcel: Parcel, flags: Int) {
        parcel.writeString(name)
        parcel.writeString(address)
        parcel.writeString(delivery_charge)
        parcel.writeString(image)
        parcel.writeParcelable(hours, flags)
        parcel.writeTypedList(menus)
    }

    override fun describeContents(): Int {
        return 0
```

```
    }

    companion object CREATOR : Parcelable.Creator<RestaurentModel> {
        override fun createFromParcel(parcel: Parcel): RestaurentModel {
            return RestaurentModel(parcel)
        }

        override fun newArray(size: Int): Array<RestaurentModel?> {
            return arrayOfNulls(size)
        }
    }
 }

data class Hours(val Sunday: String?, val Monday: String?, val Tuesday:
String?, val Wednesday: String?,
            val Thursday: String?, val Friday: String?, val Saturday: String?)
: Parcelable {
    constructor(parcel: Parcel) : this(
        parcel.readString(),
        parcel.readString(),
        parcel.readString(),
        parcel.readString(),
        parcel.readString(),
        parcel.readString(),
        parcel.readString()
    ) {
    }

    override fun writeToParcel(parcel: Parcel, flags: Int) {
        parcel.writeString(Sunday)
        parcel.writeString(Monday)
        parcel.writeString(Tuesday)
        parcel.writeString(Wednesday)
        parcel.writeString(Thursday)
        parcel.writeString(Friday)
        parcel.writeString(Saturday)
    }

    override fun describeContents(): Int {
        return 0
    }

    companion object CREATOR : Parcelable.Creator<Hours> {
        override fun createFromParcel(parcel: Parcel): Hours {
```

```
            return Hours(parcel)
        }

        override fun newArray(size: Int): Array<Hours?> {
            return arrayOfNulls(size)
        }
    }
 }


data class Menus(val name: String?, val price: Float,  val url: String?, var
totalInCart: Int) :
    Parcelable {
    constructor(parcel: Parcel) : this(
        parcel.readString(),
        parcel.readFloat(),
        parcel.readString(),
        parcel.readInt()
    ) {
    }

    override fun writeToParcel(parcel: Parcel, flags: Int) {
        parcel.writeString(name)
        parcel.writeFloat(price)
        parcel.writeString(url)
        parcel.writeInt(totalInCart)
    }

    override fun describeContents(): Int {
        return 0
    }

    companion object CREATOR : Parcelable.Creator<Menus> {
        override fun createFromParcel(parcel: Parcel): Menus {
            return Menus(parcel)
        }

        override fun newArray(size: Int): Array<Menus?> {
            return arrayOfNulls(size)
        }
    }
 }
```

7.4 PlaceYourOrderActivity.kt

This activity activates the payment page and facilitates the payment after getting the costumer's address, name, contact, etc.

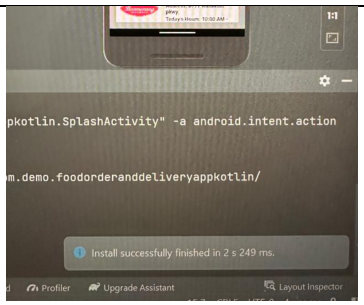
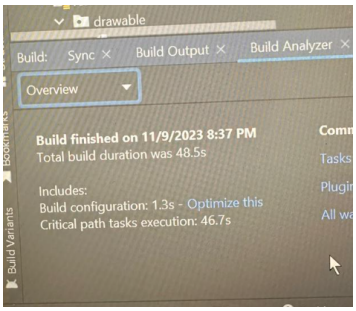
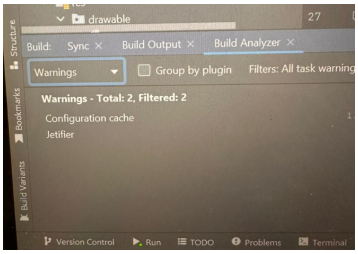
```
class PlaceYourOrderActivity : AppCompatActivity() {

    var placeYourOrderAdapter: PlaceYourOrderAdapter? = null
    var isDeliveryOn: Boolean = false
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_place_your_order)

        val restaurantModel: RestaurentModel? =
intent.getParcelableExtra("RestaurantModel")
        val actionbar: ActionBar? = supportActionBar
        actionbar?.setTitle(restaurantModel?.name)
        actionbar?.setSubtitle(restaurantModel?.address)
        actionbar?.setDisplayHomeAsUpEnabled(true)

        buttonPlaceYourOrder.setOnClickListener {
            onPlaceOrderButtonCLick(restaurantModel)
        }

        switchDelivery?.setOnCheckedChangeListener { buttonView, isChecked ->

            if(isChecked) {
                inputAddress.visibility = View.VISIBLE
                inputCity.visibility = View.VISIBLE
                inputState.visibility = View.VISIBLE
                inputZip.visibility = View.VISIBLE
                tvDeliveryCharge.visibility = View.VISIBLE
                tvDeliveryChargeAmount.visibility = View.VISIBLE
                isDeliveryOn = true
                calculateTotalAmount(restaurantModel)
            } else {
                inputAddress.visibility = View.GONE
                inputCity.visibility = View.GONE
                inputState.visibility = View.GONE
                inputZip.visibility = View.GONE
                tvDeliveryCharge.visibility = View.GONE
                tvDeliveryChargeAmount.visibility = View.GONE
                isDeliveryOn = false
                calculateTotalAmount(restaurantModel)
```

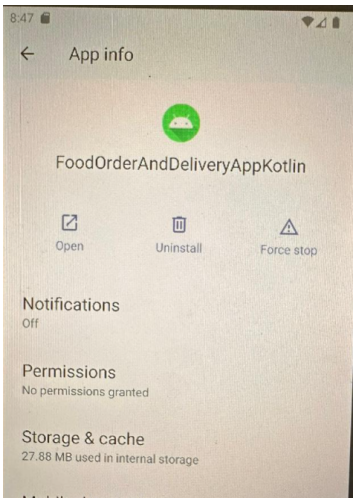
```
            }
        }

        initRecyclerView(restaurantModel)
        calculateTotalAmount(restaurantModel)
    }

    private fun initRecyclerView(restaurantModel: RestaurentModel?) {
        cartItemsRecyclerView.layoutManager = LinearLayoutManager(this)
        placeYourOrderAdapter = PlaceYourOrderAdapter(restaurantModel?.menus)
        cartItemsRecyclerView.adapter =placeYourOrderAdapter
    }

    private fun calculateTotalAmount(restaurantModel: RestaurentModel?) {
        var subTotalAmount = 0f
        for(menu in restaurantModel?.menus!!) {
            subTotalAmount += menu?.price!!  * menu?.totalInCart!!

        }
        tvSubtotalAmount.text = "Rs."+ String.format("%.2f", subTotalAmount)
        if(isDeliveryOn) {
            tvDeliveryChargeAmount.text = "Rs."+String.format("%.2f",
restaurantModel.delivery_charge?.toFloat())
            subTotalAmount += restaurantModel?.delivery_charge?.toFloat()!!
        }

        tvTotalAmount.text = "Rs."+ String.format("%.2f", subTotalAmount)
    }

    private fun onPlaceOrderButtonCLick(restaurantModel: RestaurentModel?) {
        if(TextUtils.isEmpty(inputName.text.toString())) {
            inputName.error =  "Enter your name"
            return
        } else if(isDeliveryOn &&
TextUtils.isEmpty(inputAddress.text.toString())) {
            inputAddress.error =  "Enter your address"
            return
        } else if(isDeliveryOn &&
TextUtils.isEmpty(inputCity.text.toString())) {
            inputCity.error =  "Enter your City Name"
            return
        } else if(isDeliveryOn && TextUtils.isEmpty(inputZip.text.toString()))
{
            inputZip.error =  "Enter your Zip code"
            return
        } else if( TextUtils.isEmpty(inputCardNumber.text.toString())) {
            inputCardNumber.error =  "Enter your credit card number"
```

```
            return
        } else if( TextUtils.isEmpty(inputCardExpiry.text.toString())) {
            inputCardExpiry.error =  "Enter your credit card expiry"
            return
        } else if( TextUtils.isEmpty(inputCardPin.text.toString())) {
            inputCardPin.error =  "Enter your credit card pin/cvv"
            return
        }
        val intent = Intent(this@PlaceYourOrderActivity,
SuccessOrderActivity::class.java)
        intent.putExtra("RestaurantModel", restaurantModel)
        startActivityForResult(intent, 1000)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
        if(requestCode == 1000) {
            setResult(RESULT_OK)
            finish()
        }
        super.onActivityResult(requestCode, resultCode, data)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when(item.itemId) {
            android.R.id.home -> finish()
            else -> {}
        }
        return super.onOptionsItemSelected(item)
    }

    override fun onBackPressed() {
        super.onBackPressed()
        setResult(RESULT_CANCELED)
        finish()
    }
 }
```

8. Performance:

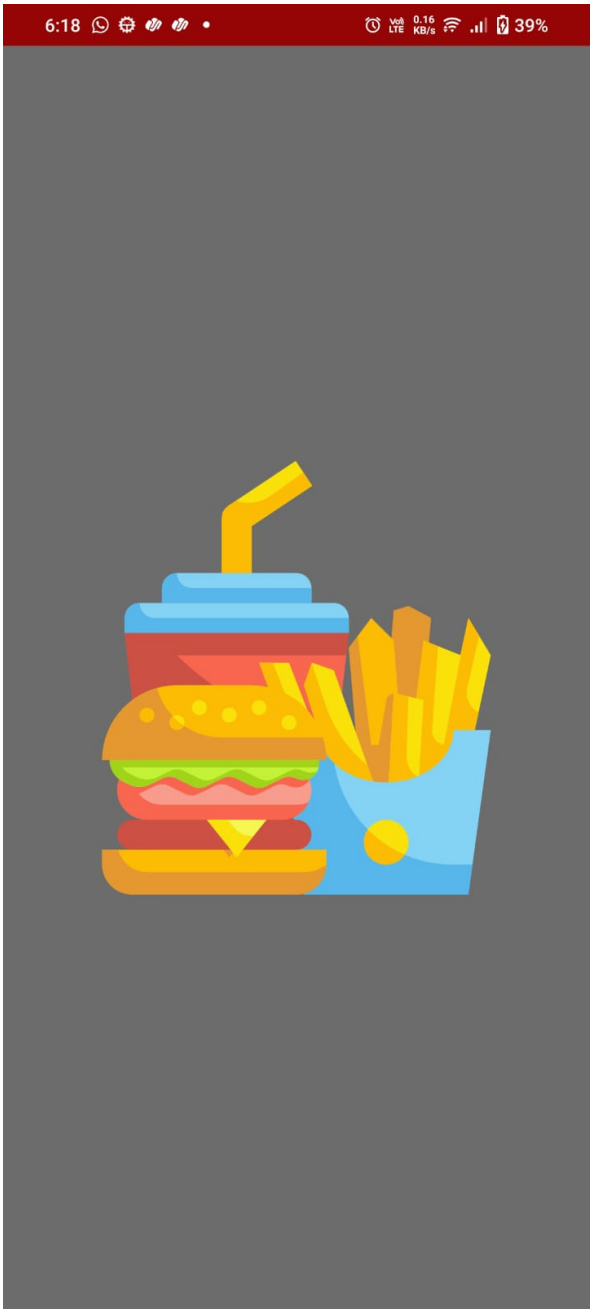
8.1. Performance metrics:

S.No.	Parameter	Values	Screenshot
1.	Metrics	App Launch Time-	
		Build Render Time-	
		Code Quality-	

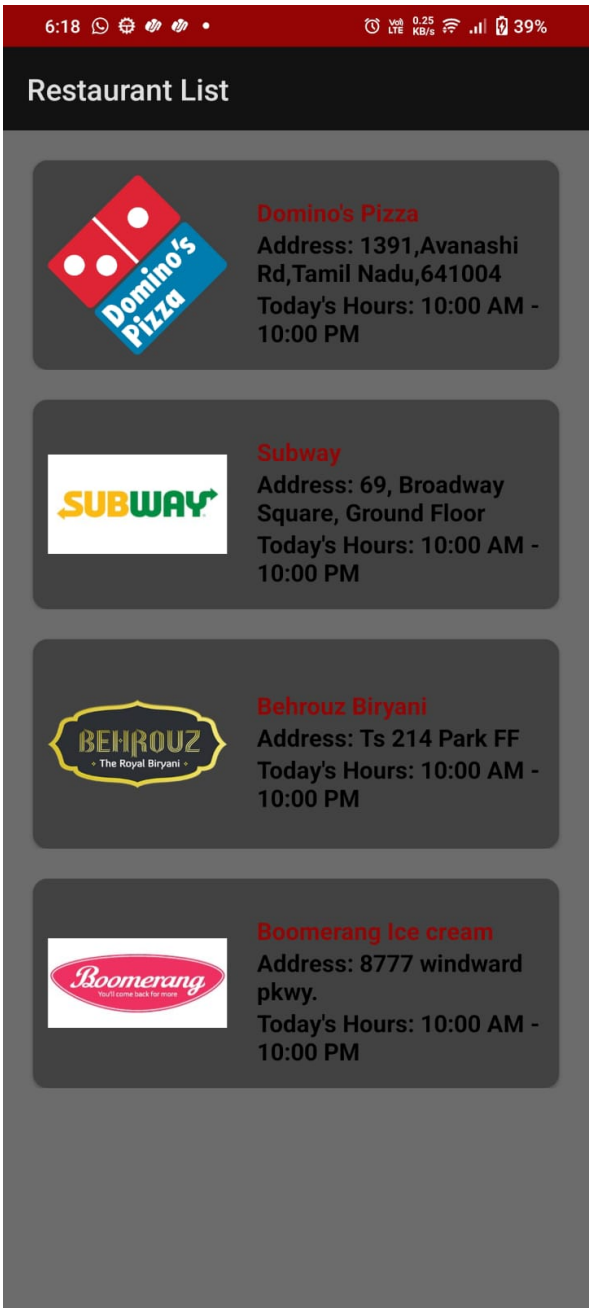
2.	Usage	App Size-	
3.	Performance	Error and Crash Rates-	None

9. Results:

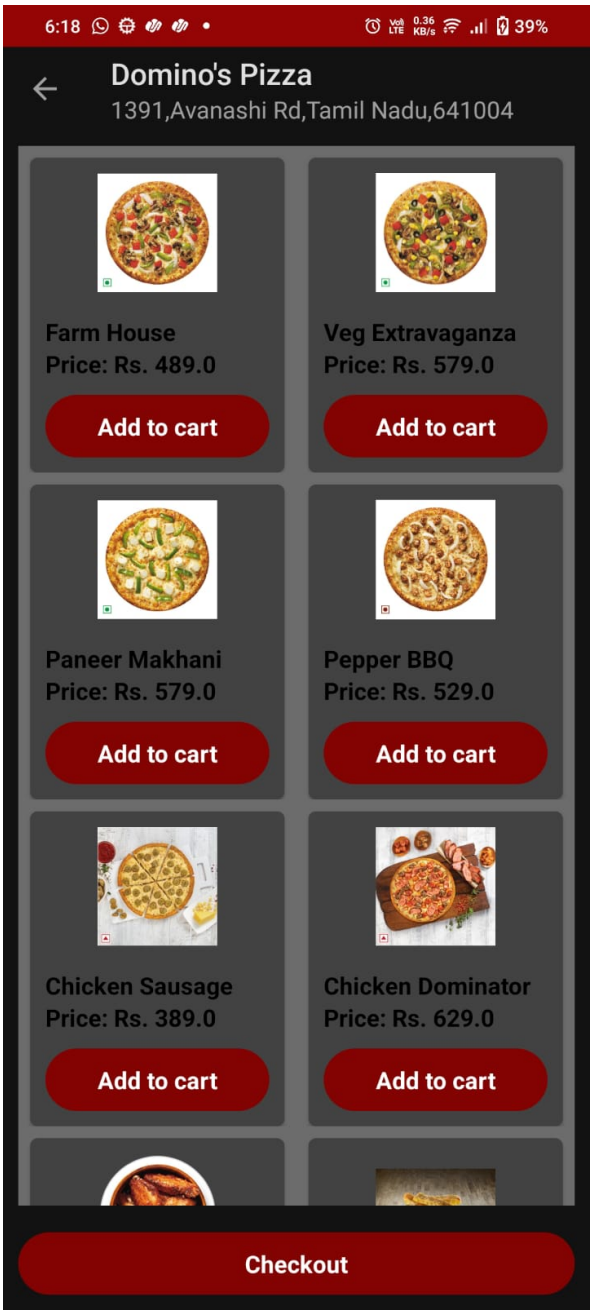
Outputs: P. T. O.



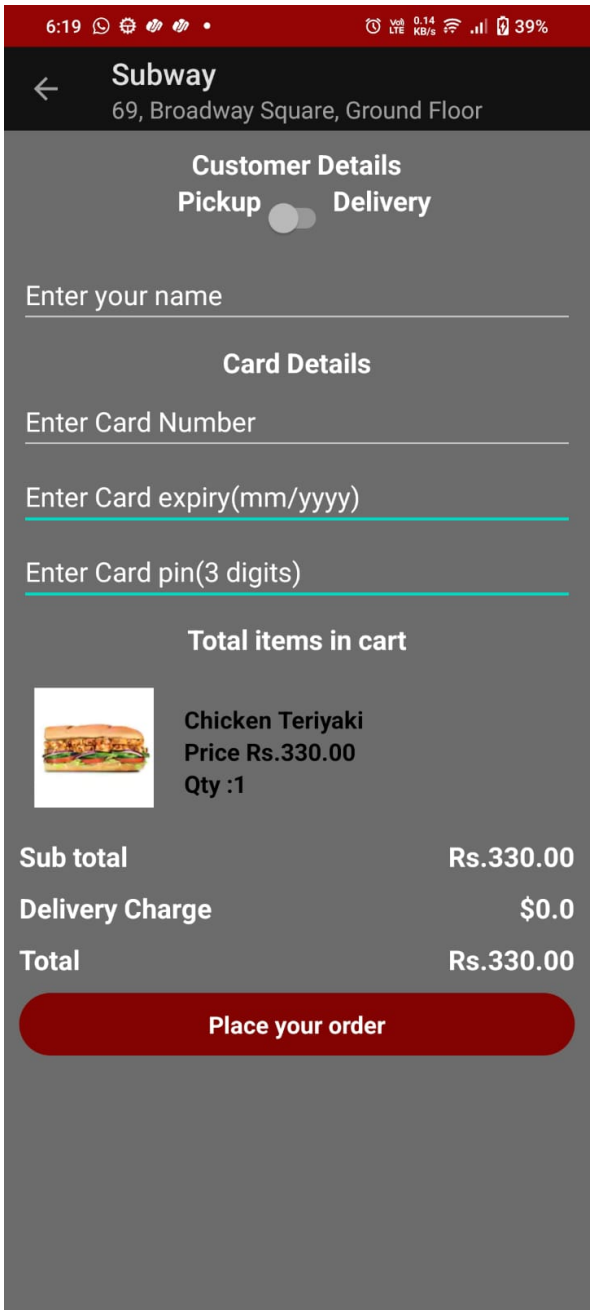
Launch Screen



Restaurant List:



Restaurant Menu



Cart Checkout:

10. Advantages and Disadvantages:

10.1 Advantages:

- Convenience: Food delivery apps make it easy to order food from your favorite restaurants without having to leave your home or office.

- Variety: Food delivery apps offer a wide variety of restaurants to choose from, so you can find something to your taste.
- Competitive pricing: Food delivery apps often offer competitive pricing on food, especially when compared to dining in at a restaurant.
- Time savings: Food delivery apps can save you time by delivering your food right to your door.
- Positive customer experience: Food delivery apps can provide a positive customer experience by offering features such as tracking your order status, real-time notifications, and the ability to rate and review restaurants.

10.2 Disadvantages:

- Cost: Food delivery apps can be expensive, especially when you factor in the cost of delivery fees and tip.
- Food quality: The quality of food delivered from a food delivery app may not be as good as the food you would get if you dined in at a restaurant.
- Delays: Food delivery orders can sometimes be delayed, especially during peak hours.
- Convenience fees: Some food delivery apps charge convenience fees for placing orders.
- Safety: There is some risk associated with food delivery, as you are trusting a stranger to deliver your food to your door

11. Conclusion:

We have created a food delivery android app which is working and convinent to use. The customer can order the food with the luxury of not travelling and save time in takeaways and waiting.

12. Appendix:

Full Project link: https://github.com/smartinternz02/SI-GuidedProject-605460-1697980993