ITCS 4102-5102 Term Project Report Fall 2024

Loan Management Application

Under Supervision of Prof. Ali Sever

Team Members

Varun Tadepalli -801365164
Dhanuj Reddy Goli - 801350433
Lokesh Chowdary Jasti - 801365011
Yashwanth Chowdary Kanaparthi - 801364617
Jayanth Sai BhogiReddy - 801393137
Krishna Teja Mantripragada - 801328356

1. **Introduction**

Currently, there are emerging customer expectations that are unique for the digital age, which include self-service and immediate access to services. The financial sector alongside with loan management, is not an exception. The previous methods of loan management may often mean the customer needs to physically go to the bank branches, or rely on the CSR for data and other actions, which is far from optimal. This project seeks to close this gap by proposing an effective Customer Self-Service Loan Portal App which will allow users to control their loans using the best device – their mobile phones.

The main purpose for undertaking this project is to improve the overall satisfaction of the customer due to simplicity, comprehensiveness and security of the platform. Some of these include; ability to register loan, tracking facility for loan, facility to pay a particular loan, extra features; EMIs, schedule for loan as per your preference etc. The use of new technologies in some aspects of the application such as Dart together with Flutter framework for the frontend, Spring Boot with SQL Server as the backend makes the app efficient, scalable and secure.

Thus, using it, customers receive unrestricted access to essential loan data around-the-clock, resulting from the virtual exclusion of dependance on paper-based procedures and, therefore, unclear financial activities. Banks benefit in the sense that they cut on their costs, especially in aspects of customer support, as well as creating good customer relations. This is a step towards the modernisation of loan management as a way of facilitating the over arching concept of the digitisation of the financial services sector.

1. **Language Specifications**

2.1. Paradigm of the Dart Programming Language

1. Object-Oriented Programming (OOP) Paradigm

First of all, Dart is an object oriented language at heart, which means that code in Dart is organized around objects and classes. This paradigm means reusability of code and that enhances scalability and modularity.

- **Classes and Objects:** As we have earlier pointed out, Dart defines blueprints for objects in the concept of classes where objects encompass data (fields) and behavior of the object – methods.
- **Inheritance:** Inheritance in Dart is single types, the ability of one class to be inherited from another, this increases the coding reusability, and helps with organization in a hierarchy.
- **Encapsulation:** The data encapsulation in Dart is achieved through using private variables such as underscore _ and then public accessibility.
- **Polymorphism:** It also supports method overriding mechanism, whereby different classes provide their own implementation for one method

2. Functional Programming (FP) Paradigm

Dart also bases on functional programming paradigms that allow sourcing clean and concise code from developers.

- **Higher-Order Functions:** Functions can be declared to a variable, can be given as an argument or can be called as a result of a function.
- **Anonymous Functions (Lambdas):** Dart also allows for anonymous functions, normally used on Collections and Streams.
- **Immutability:** Even though it is not required by default, Dart is supportive of methodologies for creating immutable objects with const and final modifiers, which make code constant and predictable .

3. Procedural Programming Paradigm

Dart is an Object-oriented, it is also support procedural programming the code is divided into a series of procedures or functions. This paradigm is most effective when working with script, Smaller applications still require full control over every aspect.

- **Top-Level Functions:** Functions are not restricted to being defined in classes which means that Dart is good for small scripts or util.
- **Control Structures:** Dart is equipped with frequently used procedural features such as cycles (for, while), conditional (if-else), and switch.

4. The Asynchronous Programming Paradigm

Otherwise known as event I/O programming, Dart thrives in operations such as I/O, network requests, and real-time data set processing without interfering with the main thread.

- **Async-Await:** async and await from Dart have largely helped in the addressing the challenges of generating neat asynchronous codes.
- **Streams and Futures:** Streams consumes sequences of asynchronous events, which is beneficial in processing real time data.

3. History Evaluation

Dart the language developed by Google came up in 2011 as a solution to these problems of developing complex large scale web applications. Google intended to fix typical problems of JavaScript including performance issues and challenges when working with large sets of code. First introduced as a rival to JavaScript, Dart had qualities such as optional static type, a concise syntax, and it was faster. The early versions were launched with platforms such as Dartium which is a chromium version supporting Dart and the Dart Editor. However, the language was met with much criticism and could not quite take off due to the overwhelming JavaScript framework as well as the other major browsers not coming to supports of the language.

Things began to turn for Dart when Google release its open source framework for building multiplatform mobile apps called Flutter in May 2017. flutter adopted Dart as its main language relying on its strong typing system and capabilities to support the asynchronous programming necessary to achieve this. This was a big year for Dart, and helped refocus the language from web to mobile. The next step in Dart's evolution came in 2018 when Arkitekt released Dart 2, which has increased the rigidity of the language's typing system and made it null safe, which helped improve the stability of the code and reduce the frequency of errors at runtime. : By being a successful company in the mobile development spectrum, Flutter started bringing Dart into the global developer audience's attention.

Recently Dart has evolved from supporting just mobile environment only to web, desktop as well as supporting server-side systems. The language remains active, and the most recent addition is Dart 3 that brings new aspects such as pattern matching and others that improve the tool's performance. Google's devotion to Dart together with Flutter's acceptance have made it a flexible, multi-Platform language. The process which Dart faced and overcomes is the perfect example of how it is becoming more suitable and useful in modern application development and provide developers with a powerful and efficient system to create modern and high-performance applications for almost all platforms.

4. Elements of the Dart Programming Language

Dart is a contemporary statically typed language that is equipped with diverse elements like: reserved words, primitive data types and structured types. These elements dictate how the language works, or behaves, and how the code it produces is laid out, allowing developers to write more coherent, coherent and uncomplicated code by using the language to it's best.

Reserved Words
A few words in Dart language have fixed meaning and cannot be used for variables naming, functions naming etc. These keywords are the elementary blocks within the structure of the language and it manages the flow of the language. Dart has a small number of restricted words that are used to define classes, functions and control structures. Some of the common reserved words in Dart include:
Control Flow: If, else, switch case, for, while, do while, break, continue, return
Class and Object-Oriented Concepts: class, extend, implement, super, this, static, new
Type System: integer, double, String, boolean, no return, dynamic, var
Access Control: privatre(prefix _), public (by implication)
Other keywords: null, true, false, try, catch, finally, await, async, const, final, enum
Apart from basic control structures whilst defining a Dart program keywords allow Dart to handle Object oriented programming constructs like classes and inheritance, Asynchronous operations and memory control through final and const.

5. Dart Language Syntax Overview

The Dart's syntax is concise, obvious, and readable, using the references to other C-style languages such as Java, C++, and JavaScript. In Dart variables maybe declared with the use of Var, final, or specific type of int, String, and double. The other property of a variable is the data type, which can be declared, or determined from the variable's value. For example, a variable can exist like int age = 25;, or like var name = "Alice;, however both a similar in nature. Dart also enables using of const and final for working with constant values: const for the compile-time constant, final for one-time variables, for example, final pi = 3.14;.

The Dart programming language implements the control structures such as if, else, for, while and switch. The if-else statement evaluates conditions to decide between alternative actions, such as:

```
if (age >= 18) {
  print("Adult");
} else {
  print("Minor");
}
```

Similarly, loops like for and while allow repetition of code blocks, such as:

```
for (var i = 0; i < 5; i++) {

  print(i);

}
```

Additionally, Dart supports both positional and named optional parameters in functions. For example:

```
void greet(String name, {int age}) {

  print("Hello $name, Age: $age");

}
```

This kind of flexibility of the syntax is good to manipulate all the paradigms of the programming, from the object paradigm to the functional features.

Dart also has a great endorsement for object-oriented-programming (OOP) and asynchronous programming. Essentials of Dart include Classes and objects; a class is created using the class keyword and objects are created using new or directly. For example, a simple class definition might look like:

```
class Loan {
  double amount;
  double interestRate;

  Loan(this.amount, this.interestRate);

  double calculateInterest() {
    return amount * (interestRate / 100);
  }
}
```

In addition to object-oriented constructs, Dart provides powerful tools for asynchronous programming using async and await for non-blocking code execution. For example:

```
Future<String> fetchData() async {
  await Future.delayed(Duration(seconds: 2));
  return 'Data loaded';
}
```

This asynchronous model helps improve performance by allowing Dart applications to handle time-consuming tasks, such as I/O operations, without blocking the main thread.

2.5. Control Abstraction

It gives basic control handling structures such as conditional structures, loops, and control structures that.control program flow. After all, if, else and switch are the main conditional structures which enable the developer to run particular portions of code depending on a set of conditions. For instance, there is an if-else statement to make decisions including ones where one asks, whether a person is either an adult or a minor?, while there is the switch statement that is valued when one wants to compare a variable with several different values. Dart's for and while loops are used for tasks that need to be repeated time and again; The for loop is for the a priori looping a definitive number of times, while the while loop loops endlessly until the condition is no longer met. Dart also recognises do-while, which ensures that a loop will run at least once.

Also the control flow tools like break and continue that are used for controlling the flow inside a loop are also included in Dart. A loop can be controlled using commands such as the break statement, which terminates a loop in the middle, as opposed to the continue statement, which enables skipping the current iteration of the loop. For each of the checks, Dart has assertions to check assumptions at runtime thus making the code more robust. These basic control abstractions are the components that are fundamental in order to obtain efficiency along with the simplicity and readability of code, as well as to build more elegant and interactive applications.

2.6. Handling Abstraction

This approach to abstraction is sound and is made possible by Dart through constructs such as functions, procedures, objects and modules to name just but a few. It is an extended version of C and supports both, functional as well as the object oriented mode of programming in different developed applications requirements.

Functions and Procedures

In Dart, the core method of abstraction is a function as a way of creating a reusable block of code. Functions are declared using void or return type, name of the function and the parameters. Dart supports both the positioned and named parameter meaning there is flexibility when calling a particular function. This means one function can take other functions as arguments this is per higher-order functions. Dart also supports anonymous functions (lambdas), making it easier to work with callbacks, for example:

```
void greet(String name) {
  print('Hello, $name!');
}
```

Objects and Classes

Dart extensively uses OO code style and offers two ways of abstraction: classes and objects. In keeping with classes in Dart, they determine the data and behavior of objects, field and functions respectively. Methods, on the other hand are stored in the class and describe what can be done on an example of an object from the particular class. OOP concepts such as encapsulation, Inheritance, and polymorphism are supported by Dart – a fact that means it is easy to make proper models of real-life objects. A class might be defined like this:

```
class Loan {
  double amount;
  double interestRate;

  Loan(this.amount, this.interestRate);

  double calculateInterest() {
    return amount * (interestRate / 100);
  }
}
```

This abstraction also enables the creator of a class to unveil only interesting operations to the outside world while keeping the nasty details out of sight.

Modules and Libraries

As for the code arranging and breaking the large efforts into parts, Dart employs libraries and packages. A library is a set of related class, functions and variables for reuse in different parts of an application. Originally, Dart language offered the import keyword, which enables a particular library or script to be inserted into a program; it means that a program can be divided into different files. Moreover, the support for packages – set of libraries – also helps in modularity in addition to the aforementioned features. Dart has a rich packages catalogue located at pub.dev where applications can be imported from other sources if needed. A typical import statement might look like this:

```
import 'dart:math';  // Imports Dart's core math library
```

2.7. Language Evaluation

We can compare and contrast how well or how ill writability, readability, and reliability are met in Dart through assessment of features and design of the language as outlined in chapter two of most programming language books. These three are important when selecting a language for development since they determine the levels of fluency of developers when writing, compilation, and even tracing their language code.

Writability

Writability is the ease with which a programmer can write a code in the language. Dart is made to be consummate to write, particularly for developers whom already know other C-style languages such as JavaScript and Java among others. Its syntax is clean and minimal and uses familiar concepts and constructs on the base of which the more complex ones are built. Another extraordinary feature is that Dart supports both object-oriented and functional paradigm that allows gaining.Proxy newProxyInstance Class A proxy is an interface to its real object and is used to control the access to the real object. The language's support for high-level abstractions such as, class, function and lambdas, facilitate efficient development of software applications. Also, Dart has a good standard library and has tools like flutter for cross mobile application development which makes it high on writability as most of the classes and appliances come already developed. But the language itself could pose problems for people who are just starting out because aside from being object-oriented, there are concepts like asynchronous programming that is still involved here even if it is 'hidden' and which are well-documented but for which examples are lacking.

Readability

Maintainability deal with the ease of readability of codes written in the language. Again, Dart stands out due to the relatively low level of change in syntax between queries and results. Its variable and method naming convention, combined with a good number control structures make the language easy to follow. There are no real surprises here for a developer who has worked in JavaScript or Java, although the syntax is not overly complicated. Furthermore, Dart came with modern features such as null safety, especially in the newer versions, which makes it easier to read and manage code, especially in asynchronous programming using async and await; It means that something that takes time, such as network, is also easy to handle. This language is also very effective in good coding practices such as coercions that compel high typing to overcome the problem of ambiguity. Dart's precise function and class definitions also help to make it a bit easier to read. However, it most certainly is easy to read for developers already familiar with object oriented languages, although it may take fresh air to get over some of the syntactical peculiarities.

Reliability

Each of the concepts described above is related and contains some idea of reliability that assesses the extent to which the language assists programmers in correctly and failure-free coding. The recommended typing system in Dart together with the new feature of null safety is just perfect to advance the reliability of the code. The compiler can detect various problems during the compile-process; for example, the discrepancy of using the wrong type can be detected early. Another positive thing that Dart has is scalable built-in libraries which are reliable and minimal chances of faulty or insecure code. Moreover, async, await and Future provided in Dart also facilitate concurrent programming in a way that is safer than, for example, managing threads and avoiding deadlocks. The application of assertions in development also helps check assumptions, that is, some forms of logical this mistake might go unnoticed. On the downside Dart productivity may be affected by left dependencies which seems more stable at the runtime, while using external libraries or packages can lead to some untrusted packages to impact on the overall productivity if not well managed or if they are not well updated.

2.8. Strengths and Weakness

Major Strengths of Dart

Cross-Platform Development with Flutter: Dart is moderately coupled with the Flutter framework and thus can be considered perfect for mobile and web applications developments. Flutter gave developers the ability to produce the same program of both IOS and Android saving development time and money. Dart outperforms other languages mainly due to its AOT compilation to guarantee that the Flutter-developed applications smoothly drive both platforms.

Rich Standard Library and Ecosystem: In Dart, there is a standard library, which may provide almost all necessary functionalities such as collections, I/O and networking. Further, the Dart ecosystem is progressing, and at the moment, there are lots of open-source packages in the pub.dev catalogue. This makes third-party libraries more accessible for developers; for example, they no longer have to build everything for authentication, data storage and the likes from scratch.

Strong Typing and Null Safety: Dart also has a quite good typing system that helps also define the errors during the compile time rather than during run time. In Dart 2.12, null safety has been made mandatory, and developers have to set up their actions to address or flag null, which greatly enhances the stability and renders null reference exceptions redundant—one of the most recurrent and potent error factors in numerous programming languages.

Asynchronous Programming Support: Dart fulvously supports off-thread I/O operations and other asynchronous patterns of development via async, await, and Future constructs. This makes it possible for developers to write synchronous computation of code that is powerful and easy to comprehend, most frequently in applications that use network requests, user interfaces, or other time-consuming tasks.

High Performance: Dart is engineered for speed and is compiled both, JIT (Just-In-Time) for development and for AOT (Ahead-Of-Time) for production. This arrangement enables quick turn arounds in application build and deployment while giving optimal application performance in runtime environments. It also scales nicely to be used in mobile applications as it is capable of running applications quickly despite limited capabilities.

Major Weaknesses of Dart

Smaller Community and Ecosystem: However, as observed from this graph, Dart has an ever growing community even if it is relatively small compared to other languages such as JavaScript, Python or Java. Therefore, while developing applications, applications may find less number of resources, tutorials, and third-party libraries than in more recognized languages. This can be restrictive especially in some cases where a strongly backed up, mature library or tool is required.

Limited Adoption Outside of Flutter: Previously, Dart was mostly bound to the management of HTML's UI, but today, Flutter is the driving force behind Dart's evolution. Nevertheless, there are very few applications of Dart outside of the Flutter framework in the rest of the programming world. It may be less attractive for developers seeking to apply it in environments other than mobile or web, such as non-mobile or non-web applications.

Steep Learning Curve for Beginners: While Dart resembles C-style languages to some degree, developers will find that Dart's strong typing, OOP concepts, and asynchronous programming paradigms might be a barrier to entry for learners still new to object-oriented design and/or asynchronous programming. The fact that these concepts would be necessary to comprehend could overrun new learners to the language.

Lack of Widespread Enterprise Support: Although Dart is actively developing applications for smartphones and web, it has not found a place in the enterprise as, for example, Java, C #, or Python. This can lead to a fewer number of job opportunities for Dart developers especially in non-mobile and non-web areas which would not be appealing to some people.

Tooling and IDE Support: Dart has advanced its tooling in the last years, contributors supported vscode but in general you don't have that many tools as for other more mature languages. Some may include a weak module, less optimized debugging, profiling, and other code improving features which may not be up to par with languages with more populations.

2.9. Overview of the Programs Included

In the mobile application designed for a customer self-service portal for loans, following programs were created to showcase the Dart language features & the potential of this language. These programs were centered on features including registration and user login, loan information views, loan payment, generation of loan schedules and computation of EMIs. Data modeling was one of the areas where Dart had the most implement of OOP, where classes like, User class and Loan class incorporated attributes and or methods associated with them. For example, the Loan class is used to hold details of loan including principal sum and interest rate and possesses functions to determine an interest on the loan or EMI. Therefore, the utilization of the classes and objects made the handling of complex data easy, and the code neatly, modularly organized as well as can be reused.

This application also benefited from Dart's asynchronous programming characteristics, as we shall soon discover. Asynchronous programming was used for other system tasks, such as getting data on loans from a server, accepting payments from users, and so on, that do not run on the main thread. Using the async and await keywords, Dart was able to complete these heavy operations without slowing down the app and reacting to user's actions. This non-blocking execution model helped in freeing up the control to the users thereby avoiding the disruption the app's flow. Dart's Future and Stream also played a role of taming asynchronous operations and making transitions between stages of the app's operations seamless.

The modularity and structure provided by Dart's rich standard libraries and strong typing were key factors in ensuring the reliability of the app. Dart's standard libraries, such as dart:, limiting the need to locally construct novel solutions to integrate into convert for JSON parsing as well as dart:math for the mathematical computations that cut time in development. Also, Dart was set to benefit strongly from typing as well as null safety to be certain that all the data types required were well governed so that the running of errors could be well prevented. The null safety enforced means that Dart made it simple to handle cases where the value could be null or the reference undefined; null reference exceptions were avoided. Developing with Dart was relatively easy thanks to the sweetness of its syntax and the availability of several language features, but to get the most of what Dart can offer, object-oriented design and asynchronous programming are required and not every team member was familiar with these concepts at the time of the development.

Problem Definition:

Our goal is to create a customer self-service portal App for Loans where customers can access various details of their loans on their mobile device Following features are supported planned as part of the app

- Customers will be able to login as well as register themselves if they are a new user
- Customers will be able to see the list of active loans they currently have
- Customers will have the access to see the detailed information of each of the active loans as in like interest rate, principal amount, instalment amount etc.
- Customers will have an option to pay off the loan amount completely or partially just the instalment amount for that term
- Upon successful payment the records in the database for each term will be updated based on the payment amount.
- Customers will have an option to create a loan schedule based on the custom values provided for the parameters like loan amount, interest rate, payment frequency etc.
- Customers are provided with an option to calculate the Emi based on the various inputs from them

Intuition - why should it be better than the state of the art?

1. User – Authentication:

    A .Features:

•Registration for new users

•Login securely for existing users

B. Value:

• Secure and personalized access for the users.

• Enables data security by hashing passwords to ensure they are a safe.

2. View Active Loans:

a. Features:

• A List of all active loans of that customer can be viewed

b. Value:

• The users are provided a comprehensive overview of their loan status at one place.

• It simplifies loan tracking and management.

3. View Loans Details:

A .Features:

• All the essential information about the active loans is displayed like:

- Interest rate
- Principal amount
- Instalment Amount

B. Value:

• The transparency is increased. No details are hidden from the user.

• It simulates the users to make informed financial decisions based on the detailed information.

4. Payment Options:

a. Features:

•The database records are automatically updated when a payment is made.

•Payments options include using cards, upi payments using razorpay api. The payment options are included based on the Indian apps and banks.

b. Value:

•The manual intervention is reduced in tracking payments.

• Allows users to make payments in the apps itself to clear of loans.

5. View Loan Schedule:

a. Features:

• The active loans have a details schedule for each term included in the app to inform the user about how the loan is being paid off.

b. Value:

•It provides clarity based on the schedule for repayment timelines.

6. Emi Calculation:

a. Features:

•Emi can be calculated based on the loan amount, the interest rate, the frequency of payments etc which helps the user to know what would he be paying if he applied for the loan.

b. Value:

•It provides clarity based on the schedule for repayment timelines.

Proposed Methods:

1. Database Management and storage: SQL Server has been integrated into the application to allow for data storage facilitation. It provides effective handling of user and loan data. It provides secure storage of sensitive data. Triggers and auto updates the data whenever a payment is made. It integrates seamlessly with modern app frameworks allowing for smooth interaction between front end and back end.

2. Spring boot framework: It provides pre-configured setup and starter dependencies. It is used for handling complex dependency versions and conflicts. It provides seamless integration with spring MVC for building REST api's.

3. Flutter framework: Flutter provides cross platform development providing a single code base for running on both android and iOS devices. It provides an extensive collection of customizable widgets for UI elements. Instantly updates the app without having to restart the entire application. This is known as hot reloads. Also, responsive layouts can be built on top of this.

Testing:

The testing is done on the emulator and the physical android device to see if the application is working as expected.

Experimental Questions:

1. Is the loan schedule being calculated correctly?

2. Is the data being stored accurately and is there any data loss?

3. Is the data being updated correctly on the event triggers?

4. Is the layout and other widgets being displayed responsively?

All of the above questions have been answered and verified that the application is running as expected and is accurate.

Conclusion:

Nowadays, it is possible to create a customer self-service portal app for loans that is functional and equipped with a set of features that help users to manage loans easily. Some of the functionalities include; secure sign on of users and ability to access current active loan information, real time payments, ability to set up new loan schedules among others. The payment options such as Razorpay for loans and the EMI calculator help them repay their loans and effectively fixed prices respectively. As SQL Server is used as the backend and Spring Boot used for API layer the architecture of the app helps to manage data and do secure transactions. Test efforts have confirmed that second requirement of functional fitness and ensures that the Android and iOS applications are as responsive as possible in terms of perceived usability by end users.

Future Work:

Altogether, though the application corresponds to the current tasks and meets the basic aims, there are some possibilities for future development. The only area that a potential added value could be cited is the ability to integrate complex metrics into the application so that the users could monitor progress in regard to their repayments and visualize their effective loan management in the longer run. Otherwise, increasing the number of the regional payment gateways and adding the notifications/reminders for due payments can be useful. Further possible improvements can be also made on the interface design to enhance the usability of the app, and to be friendly used by all kinds of people. In addition, identifying more loan types which may be supported by the app or including an intelligent recommendation of loans will add even more value to the financial services.