Vision	**DONE**
Supplementary specifications	**DONE**
FURPS+	**DONE**
Features list	**DONE**
Glossary	DONE
Domain Rules	**DONE**
Risk List	**DONE**
Feasibility	**DONE**
Use cases	**DONE**
Draw package diagram	DONE
Activity Diagram	DONE
activity diagram with swim lane	DONE
Draw	
Business level use cases diagram	DONE
Analysis level use cases diagram	DONE
Statechart diagram	DONE

- [x] ~~DOMAIN MODEL~~
- [ ] USE CASE CONTRACTS
- [x] ~~DESIGN CLASS DIAGRAM~~
- [ ] SEQUENCE DIAGRAM
- [x] ~~SYSTEM SEQUENCE DIAGRAM~~
- [x] ~~COLLABORATION DIAGRAM~~
- [x] ~~PROTOTYPE~~
- [x] ~~ARCHITECTURE~~

https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter
https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

USE CASE CONTRACTS

Freelancing Marketplace:

Operation: SearchBy()
Cross Reference: Use Cases: Freelance Marketplace: Search Services
Pre-Condition: User searches for a narrator
Post-Condition: Relative searches will be shown

Operation: PayBill(Bill_Id: Bill_Id, Amount: integer)
Cross Reference: Use Cases:Payment Management System
Pre-Condition:
Post-Condition: PayBill instance p was created. p is associated with DisplayBilling().p.Bill_Id becomes Bill_Id.Based on the Bill_Id, the payment is made.

Operation:
Cross Reference:
Pre-Condition:
Post-Condition:

Operation:
Cross Reference:
Pre-Condition:
Post-Condition:

- Operation:Login(Username:string,Password:string)
- Operation:SignUp()
- Operation:Logout()
- Operation:8
- Operation:
- Operation:
- Operation:
- Operation:
- Operation:https://homepage.cs.uiowa.edu/~tinelli/classes/022/Spring15/Notes/chap11.pdf
- https://www.eidk.org/diffrence-between-mvp-mvc-mvvm.html

Architecture:

It is, of course, very early and impossible to anticipate all the things when we create our web application. Especially, in these turbulent times, when the market, user needs, and demands, business goals, and technology can change very abruptly. The same goes for our decision for choosing an architecture for our application.

What architecture are we using?

We are using MVC as our architecture pattern.

WHY MVC?

MVC has been broadly adopted as a design pattern for WWW applications in major programming languages, Even though initially, it was developed for desktop computing. A lot of web frameworks have been created that enforce the pattern.

During the traditional approach of programming, there was a lack of maintainability, testability as well as scalability in the application development because the UI coding, business logic, and applications data domain were written into a single file.

With the arrival of the MVC approach, an application can be created where different aspects of the application (input logic, business logic, and UI logic) are divided into separate layers while providing a loose coupling between these elements. The pattern specifies the location of each kind of logic within the application. The UI logic, input logic, and business logic belong in the view, controller, and model respectively. This segregation helps in managing complexity by focusing on one aspect of the implementation at a time. For example, without depending on the input logic, you can focus on the view.

Following are some advantages and disadvantages of the MVC pattern:

Advantages:

- **Parallel development**: There can be parallel working on the model, controller, and view by multiple developers.
- **Loose coupling:** This feature makes the MVC framework flexible thus simplifying testing, maintenance, and troubleshooting procedures.
- **Easy modification:** As the responsibilities are separated from each other, future modification or development is easier i.e. product's scalability is increased.
- **A model with multiple views:** There can be Models having multiple views.
- **High Cohesion:** MVC permits logical grouping of related actions together on a controller and grouping of views for a specific model together. This feature is associated with traits like robustness, reliability, and reusability.

Disadvantages:

- **Code navigability** — The framework navigation can be complex because new layers of abstraction are introduced and users are required to adapt to the decomposition criteria of MVC.
- **Consistency problems:** The data is scattered, due to the breaking down of a feature into three artifacts. Thus, developers are required to maintain the consistency of multiple representations at once.

Conclusion:

After the above discussion, it can be deduced that in comparison to the traditional approach, using MVC architecture software development becomes smoother & robust due to its advantages of code reuse, parallel application development, etc. Nowadays, all the popular platforms of development like .NET, PHP, Java, have facilitated the use of MVC architecture.

Loose coupling

The goal of a loose coupling architecture is to reduce the risk that a change made within one element will create unanticipated changes within other elements. Limiting interconnections can help isolate problems when things go wrong and simplify testing, maintenance and troubleshooting procedures

(<<Smjnhy k liye hai baad main hata dyna>>Loose coupling means that the system is designed using components that communicate with each other via interfaces rather than having dependencies hard coded into them (or tightly coupled to them).)

High cohesion:

In computer programming, cohesion refers to the *degree to which the elements inside a module belong together*.[1] In one sense, it is a measure of the strength of relationship between the methods and data of a class and some unifying purpose or concept served by that class. In another sense, it is a measure of the strength of relationship between the class's methods and data themselves. Modules with high cohesion tend to be preferable, because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability.

Why not mvvm?

John Gossman has criticized the MVVM pattern and its application in specific uses, stating that MVVM can be "overkill" when creating simple user interfaces. For larger applications, he believes that generalizing the viewmodel upfront can be difficult, and that large-scale data binding can lead to lower performance.

https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel

Why mvp not mvc?

MVP pattern overcomes the challenges of MVC and provides an easy way to structure the project codes. The reason why MVP is widely accepted is that **it provides modularity, testability, and a more clean and maintainable codebase**.

1. **Challenges in MVC**
   A few years ago, Android developers started using MVC architectural pattern in the process of creating an Android app. But the app was far from being robust due to using MVC pattern. Robust was not only the hurdle faced by developers, but the code was not cleared.
   We found two reasons for this problem –
   - The continuous changes in the User Interface (UI).
   - The lack of an architecture, which could provide flexibility as what developers needed.

In MVC, both the ‘Controller’ and the ‘View’ depend on the ‘Model’. The ‘Controller’ updates the data, and the ‘View’ gets the data.
All in whole, MVC is not meant to suffice developers’ needs. There are some reasons

because of which developers abandoned the MVC:
– MVC makes the code more complicated for Android application developers.
– For developers, it is impossible to perform the unit test on their own.
It is, of course, impossible to foresee everything when you create your web application. The technology, market, user needs, and business goals of your company can change very quickly in these turbulent times.

These software frameworks vary in their interpretations, mainly in the way that the MVC responsibilities are divided between the client and server.[16]

Some web MVC frameworks take a thin client approach that places almost the entire model, view and controller logic on the server. In this approach, the client sends either hyperlink requests or form submissions to the controller and then receives a complete and updated web page (or other document) from the view; the model exists entirely on the server.[16]

As with other software patterns, MVC expresses the "core of the solution" to a problem while allowing it to be adapted for each system

Traditionally used for desktop graphical user interfaces (GUIs), this pattern became popular for designing web applications.[4] Popular programming languages have MVC frameworks that facilitate implementation of the pattern.

What are the basic purposes of all architectures?

The goal of all architectures is to divide the responsibilities of data viewing, data processing, and data management for UI based applications and to increase the following:

- Modularity
- Maintainability
- Flexibility
- Testability