**Client Requirements (Tapp Engine to be serviced by API):**

1. Onboarding
2. Asset Allocation
3. Rebalancing
4. Reporting
5. Integration with Tapp Engine Platform (Functional Requirements)
   1. KYC.
   2. Bank Account link.
   3. Fund Transfer payment integration.
   4. Broker integration (Order Management).
   5. Integration with data provider.
6. Integration with Tapp Engine Platform (Non - functional Requirements)
7. Dependencies/Information we require from client of Tapp Engine:
   1. Possible Risk Profiles/Scores of the user?
   2. What are the market data sources we will be getting and how will we get it?
   3. Possible goals of the users i.e., either custom or generic?
   4. Possible subscription plans and offerings in each plan?
   5. Goal based or Risk Profile based portfolios.

**Robo4Me Commercial Requirements:**

1. Ability to track user per month.
2. Ability to track AUM at the level of end user.
3. Ability to track application support requirements.
   1. Application module support.
   2. Infra support.
   3. Quant support.
   4. Training or usage support.
   5. Integration support.
4. Ability to track downstream post Robo recommendation.
   1. Order generation on trade platform.
   2. Order execution status.
   3. Brokerage detail at the level of end user.
   4. Any other chargeable service being provided to the end user.
5. Ability to track API calls.
   1. Calls coming in from third party platform.
   2. Successful execution of the API at our end.

**API Dependencies:**

* 1. Authentication type.
  2. Specification regarding API gateway security.
  3. Specification regarding standard error response.
  4. Total application firewall specification if any?

**Server sizing:**

* 1. Dev
  2. UAT
  3. Production-
     1. Expected Request Per Minute (RPM)

**Deployment approaches:**

First get the requirements of the modules from the client.

There are two approaches we can use for deployment:

1. Setup a cloud account on our end with all the required servers deployed on it and give its access to client.

2. Deploy our APIs on client’s server. This will include:

* Layer 1: Infrastructure based on requirements
* Layer 2: Application
* Layer 3: Database

We need to create an application flow control diagram which will be used by the clients to deploy the APIs on their server.

Also, we need to create a “Monitoring Service” or use “Grafana” to track the use of the APIs on the client side.

**Other Considerations:**

1. Internal billing and cost allocation
2. Audit and risk management
3. Information security, data protection and compliance
4. Change management
5. Development and operations
6. Adding to the scope regarding processes:
   1. Managing software licenses
   2. Change management
   3. Information Security
   4. QA testing
   5. This scope discussion may lead to provide insights on:
      1. Security
      2. Privacy
      3. Chargeback
7. Evaluate Cloud Service Agreement as pricing, SLA (Service Level Agreement) and other aspect can vary between different capabilities of the platform.
8. Examine costs and charges and whether limits can be placed on usage and expenditure. How charges are reported is also important.
9. Assess software licensing of the cloud service provider and of the software provider.
10. Consider compliance requirements.
11. Ability to port the application and data from one cloud service provider to another so as to avoid lock-in.
12. Change management with respect to software levels and patching.
13. Security, including availability, confidentiality and integrity of the applications and services in the environment.
14. Processing locations and where data is stored while also considering the regulatory.
15. System billing.
16. Exit process as to if client decide to stop using our offerings.
17. Factors to consider for application architecture and development:
    1. **Codebase:** One codebase in a code repository, tracked in revision control, many deploys (e.g. to developer, staging, production).
    2. **Dependencies:** Explicitly declare and isolate dependencies.
    3. **Configuration:** Store application configuration in the environment, completely separated from the codebase.
    4. **Backing services:** Treat services used by the application as loosely coupled attached resources.
    5. **Separate build, release, run:** Strictly separate build and run stages.
    6. **Processes:** Execute the app as one or more stateless, share-nothing processes.
    7. **Port binding:** Export all application services via port binding.
    8. **Concurrency:** Scale out via the process model – handle greater load by running multiple parallel processes all running the same code.
    9. **Disposability:** Maximize robustness with fast startup and graceful shutdown.
    10. **Development/production parity:** Keep development, staging, and production as similar as possible – use continuous deployment.
    11. **Logs:** Treat logs as event streams – not as files.
    12. **Admin processes:** Run admin/management tasks as one-off processes.
18. Security Components:
    1. Firewalls
    2. Secure communications handling
    3. API management
    4. Infrastructure and network protection capabilities, enabling monitoring and alerts for the communications between application components and with underlying services.
    5. Identity and Access Management capabilities
    6. Encryption of data at rest and in motion
    7. Data activity monitoring, tracking queries and updates to stored data
    8. Resilience capabilities, including replication/redundancy of data and applications, backup capabilities and high availability techniques.
    9. Data management capabilities including data location control and secure deletion.
    10. Security information event management, including analytics of events and log data to reveal activity with security implications and to help with remedial actions.