**Functional Scenarios**

**1) Food Delivery (Zomato, Swiggy, Eatsure)**

**a) User registration and login**

* User should be able to create login account (Jira board)

**b) Restaurant search**

* User can search for restaurant by name, ratings and location

**c) Restaurant Menu browsing and item selection**

* Each restaurant should have menu with items, users can add to cart

**d) Order placement and Payment**

* User can place an order and pay via UPI, COD and Card

**e) Real-time order tracking**

* Users can track their order

**2) School Management System (Admins, teachers, students)**

**a) Role based login (student, teachers and parents)**

* Admin, teachers and students should login at different level

**b) Student attendance management**

* Teachers can mark and view daily student attendance

**c) Timetable and class schedule**

* System should allow creation and class schedule

**d) Grade and report card generation**

* Teachers can enter marks and report card at one click

**e) Fees section**

* Parents should be able to make easy transactions

**Functional Documentations**

***(Stakeholders: Director, Architect (Infra, Application), Enterprise, Pre-sales, Technical Project Manager)***

**1. Functional Analysis and Architecture**

* Analysis of Functionality
* Crunch and break into HLD (Decide Monolithic or Microservices) (15 pages PPT) (Draw.io and Visio)
* Decided to go with Monolithic application

**2. Use Case and LLD (Low-Level Design)**

* **LLD is created with detailed information**
* **Application Stack:**
  + Frontend: Angular, React
  + Backend: Java, .NET
  + Database: MySQL, Postgres

**3. Agile Project Setup**

* PI Planning (Quarterly)
* User stories are created (Azure Board, Jira, Rally – PM tools)
* Sprints are created (15 or 21 days)
* Tasks are assigned on user stories
* Project begins with a Scrum call to discuss sprint (15-30 minutes call duration)
* Retrospective call is scheduled to discuss scrum calls (Backlog, spill over, completion)
* Development is completed

**Infrastructure Setup Using Terraform (IaC Tool)**

* **Parallelly we have created infrastructure using IaC tool Terraform with Generic Module using:**
  + for\_each map of object
  + dynamic block
  + optional attribute
  + conditional literature

**1. Onboarding/Offboarding Solution**

* Management group
* Subscription
* Association of subscription to MG
* Group creation
* RBAC
* User creation
* Addition to group
* Pro assignment

**2. Environment Setup**

Environments: Dev, Test, QA, Prod

**2.a - Core Infrastructure**

* RG, VNet, Subnet, VM (Frontend and Backend – Linux or Windows)
* VMSS
* Database (Cosmos, Postgres, MySQL – Depends on customer requirement)
* Firewall, NSG, ASG
* Load Balancer
* Application Gateway
* Front Door
* Region Services
* Traffic Manager

**2.b - Monitoring**

* Azure Monitor
* Log Analytics Workspace
* Prometheus
* Grafana
* Nagios

**2.c - Backup and DR**

* Backup halt
* Highly availability (DR)

**2.d - Security, Governance and Compliance**

* Defender
* Azure Policy
* Security Center
* Key Vault
* NSG
* ASG
* Bastion

**2.e - Cost Optimisation**

**2.f - Network Connectivity**

* VNet Peering
* VPN Gateway
* ExpressRoute

**Module Management and Repository Strategy**

* Each child module is pushed to the Azure repository with proper tag and versioning
* Call the child module, create a parent module, and inside the parent module, we create infra – sandbox, preprod, and prod
* We call the module based on the child module, put the right dependency, and generate tfvars for the input

**Example: Creating a VM (Real Ticket Scenario)**

* Suppose we have created sandbox
* We will push the codes to trunk-based strategy on Azure repository

**Steps:**

1. Clone parent module from Azure repository
2. Create a feature branch – e.g., vm-creation-ticket2035
3. Message – add one VM in tfvars in map of object
4. Post addition – push the code to feature branch (remote repo)

**CI/CD Workflow (YAML Pipeline)**

* As soon as we push the code to the remote feature branch, the pipeline will run

Before plan generation, the pipeline performs validation/scans using:

* tfsec (vulnerabilities)
* trufflehog (secret management)
* tflint (linting process)
* Chef, BlackAqua, CheckOps

**Approval and Apply Stage**

* If everything looks good and the plan is successful, then only the reviewer can approve and merge the PR (Pull Request) to the main branch
* As soon as it gets merged, automatically the pipeline will run from the main branch
* It will again run all security, testing, and linting checks using:
  + tfsec, trufflehog, tflint, Chef, BlackAqua, CheckOps
* If all checks are passed:
  + It goes to plan stage
  + Then enters a manual validation stage
  + Once the lead approves, apply will happen
  + VM will be added