SuperTech Innovations

Group members:

* Yash Mothilall (Group Leader)
* Adhir Sachin Harripersadh

GitHub Link:

* .

**1. Work Agreement:**

A work agreement is a set of guidelines and expectations that a team establishes to ensure effective collaboration and workflow. It typically includes:

* **Roles and Responsibilities:**
* Yash: Group leader. Developed the app.
* Adhir: Group member. Developed the website. Documentation. Testing both the website and app.
* **Communication Channels:**
* Discord for our video calls and sharing screens.
* WhatsApp for all our voice calls.
* **Work Hours and Availability:**
* We should meet at campus twice a week.
* We have discord calls frequently.
* **Decision-Making Process:**
* Since there are only 2 people in my group we can't put it to a vote. So, we do more research before we get to our final decision.
* **Conflict Resolution:**
* Keep Calm: Emotions can run high during conflicts. Encourage everyone to remain calm and focused on finding a solution.
* Remind team members to discuss the problem at hand rather than personal attacks.
* Establish Ground Rules: Consider developing team guidelines for resolving conflicts to provide a framework for discussions.
* Document Agreements: If the resolution requires specific actions or changes, document them to ensure accountability.
* **Project Timeline and Milestones:**
* Project Planning: Create a complete project plan outlining all tasks, dependencies, and deadlines. Divide the project into small segments and establish reasonable milestones.
* Prioritisation: Sort jobs according to their significance and interdependence. Concentrate on critical path tasks that have a direct influence on project timeframes.
* Task Assignment: Clearly describe each team member's tasks and responsibilities. Tasks should be assigned based on individual skills and expertise.
* Regular Check-Ins: Hold regular check-in sessions to evaluate progress. Use these sessions to address issues, offer updates, and, if required, reallocate resources.
* Agile approaches: Think about using agile approaches like Scrum or Kanban. Agile development enables flexibility, adaptation, and iterative development.
* 14. Communication with the NGO: Keep the NGO updated on progress and anticipated timeframe modifications. Set reasonable expectations and discuss any issues as soon as possible.

2) **Definition of Ready**:

* Clear and easy to understand: The user narrative or task is expressed clearly and simply.
* Acceptance Criteria: All acceptance criteria have been developed and approved.
* Dependencies are recognised, as are any dependencies on other jobs or external sources.
* Estimation: The team has calculated the amount of time needed to execute the work.
* Available Resources: Any essential resources, such as data or tools, are available.

**3. Definition of Done (DoD):**

The Definition of Done is a set of criteria that must be met for a user story or task to be considered complete. This could include:

* **Code Review:** The code has been reviewed by at least one other team member.
* **Unit Tests:** Unit tests have been written and passed.
* **Integration Tests:** Integration tests have been conducted and passed.
* **Documentation:** Any necessary documentation, such as user manuals or technical documentation, is complete.
* **User Acceptance Testing (UAT):** The feature has been tested and accepted by our NGO.

4. **Roadmap (High-level Plan):**

* Key Milestones: Identify significant milestones and deadlines. Keep to deadlines and have specific milestones for appropriate times.
* Sprint Duration:
* Period: A sprint is a fixed-length session with a set duration. It might vary based on the preferences of the team and the nature of the project. Sprints are typically two weeks, three weeks, or four weeks long. The time chosen is determined by factors such as project complexity, team velocity, and the type of the task.
* Consistency: It is typically suggested to maintain a constant sprint duration throughout the project. Consistency facilitates better planning and enables the team to adjust and develop over time.
* Meetings for Planning and Review: The frequency of major Scrum activities such as Sprint Planning, Daily Stand-ups, and Sprint Review is determined by the duration of the sprint. These events, for example, may occur more often in a two-week sprint than in a lengthier, four-week sprint.
* Adaptability: Shorter sprint durations provide the team more opportunity to inspect and change their procedures. It enables faster feedback loops and changes based on lessons learnt after each sprint.
* Delivery Period: Sprint duration has an impact on how frequently potentially shippable increments are delivered. Shorter sprints result in more frequent releases, which might be beneficial in projects that require frequent updates.
* Velocity Team: Teams frequently determine their velocity, or the amount of work they can consistently do in a sprint. Velocity aids in anticipating future sprint capacity and planning the quantity of work to be taken on.
* Management of Risk: Shorter sprints can aid in risk management by allowing teams to more often review priorities and adjust to changing requirements.
* Resource Planning: Ensure everyone's availability is considered during the project duration;
* identifying Resources: Determine the skills and knowledge necessary for the project in human resources. Determine whether team members or external expertise will be required for various tasks.
* Tools and Equipment: Identify the tools, software, and equipment needed for the project.
* Materials: If the project requires tangible items, list the materials required.
* Calculating Resource Needs:
* Time Estimates: Estimate the amount of time necessary for each task or activity.
* Cost Estimates: Calculate the costs of human resources, equipment, materials, and any other related charges.
* Managing Resources:
* Assigning assignments: Based on team members' talents and availability, assign particular assignments or activities to them.
* Planning: Create a project timeline that specifies when each resource will be used. Human and non-human resources are included.
* Workload Distribution:
* Avoid Overloading: Make sure that no team member is overburdened with work, since this can lead to burnout and lower productivity.
* Availability: Ensure that resources, particularly essential team members, are available when needed.
* Planning for the Unexpected:
* Anticipate possible hazards and obstacles that may affect resource availability or requirements.
* Creating Contingency Plans: Prepare for unforeseen situations by having backup resources or alternate ways.
* Monitoring and modifying:
* Monitor resource utilisation, project progress, and any deviations from the plan on a regular basis.
* Adjusting as Needed: Be prepared to make changes to the resource plan because of changing project needs, unforeseen challenges, or new knowledge.
* Communication:
* Maintain open communication with team members to ensure they are aware of their roles and responsibilities.
* Keep stakeholders informed about resource requirements, changes, and any potential consequences on the project timetable or budget.
* Making Use of Project Management Software:
* Use project management tools and software to help with resource planning, allocation, and tracking.
* Collaboration Platforms: Use collaboration platforms to improve team communication and coordination.

5) Non-functional requirements:

* Performance:
* Response Time: Specify the system's maximum allowed response time to a user request.
* Throughput: Specify how many transactions or operations the system should process per unit of time.
* Reliability:
* Availability: Calculate the percentage of time the system should be usable.
* Fault Tolerance: Specify the system's capacity to continue running in the event of a hardware or software failure.
* Security:
* Authentication: Define the procedures and conditions for user authentication.
* Authorization: Define distinct users' or roles' access levels and permissions.
* Data Encryption: Specify the necessity for sensitive data to be encrypted during transmission and storage.
* Usability:
* Define the user interface criteria for usability, accessibility, and user pleasure.
* Training and documentation: Specify the level of end-user training necessary, as well as the necessity for user documentation.
* Maintainability:
* Maintainability of the code: Establish standards and practises to ensure that the code is readily maintained.
* Specify the degree of documentation necessary for future maintenance and improvements.
* Scalability:
* Horizontal Scalability: Determine how effectively the system can grow as more computers are added to the network.
* Vertical Scalability: This refers to how well the system can scale by adding more resources to a single computer.
* Compatibility:
* Hardware Compatibility: Specify the hardware with which the system should be compatible.
* Software Compatibility: Specify the software and versions with which the system must be compatible.
* Compliance with regulations:
* Legal criteria: List any legal or regulatory criteria that must be met by the system.
* Data Protection: Specify rules for the security and privacy of user data.
* Performance:
* demand Handling: Determine the expected demand on the system, particularly during peak usage.
* Interoperability:
* Integration: Specify if the system will need to integrate with other existing systems or third-party services.

6) Architecture artefacts:

* Microservices Architecture:

Why: Microservices include dividing a programme into tiny, autonomous services that interact via APIs. This paradigm was chosen for its scalability, maintainability, and independence in deploying and updating services.

* Layered Architecture:

Why is layered architecture used? Layered design divides an application into several levels (presentation, business logic, and data) to promote maintainability and scalability. It uses a modular architecture that allows you to change or replace specific layers without impacting others.

* Event-Driven Architecture:

Why is this design predicated on event creation, detection, consumption, and reaction? It was chosen for its ability to allow real-time processing, permit asynchronous communication, and decouple components.

* SOA (Service-Oriented Architecture):

Why: SOA requires the system to be designed as a collection of loosely connected, autonomous services. It was chosen for its modularity, reusability, and ability to independently scale components.

* Architecture for Serverless Computing:

Why: as serverless architecture, code is deployed as stateless functions that are triggered by events. It was chosen for its scalability, cost effectiveness (pay-per-execution basis), and the flexibility to concentrate on creating code rather than maintaining infrastructure.

* RESTful Architecture:

Why: RESTful design is built on Representational State Transfer principles, and it provides a standardised means of developing scalable and stateless online services. It was chosen because of its simplicity, scalability, and ease of integration.

6.1) Design Patterns:

* Pattern of a singleton:
* Explanation: This property is used to ensure that a class only has one instance and to offer a global point of access to that instance.
* Use Case: When a single point of control or coordination is required, such as with a configuration manager or logging service.
* Factory Method Design Pattern:
* Explanation: Defines an interface for constructing an object but leaves the type selection to subclasses.
* Use Case: When a class cannot predict the type of objects it must produce or when the classes instantiate vary depending on the circumstances.
* Builder Design:
* Explanation: Distinguishes the production of a complex item from its representation, allowing the same building method to provide several representations.
* Use Case: When an item must be built with multiple configuration choices.
* Pattern of Strategy:
* Explanation: A family of algorithms is defined, encapsulated, and interchangeable. It enables the client to select the best algorithm during runtime. When you wish to define a family of algorithms, encapsulate them all, and make them interchangeable.
* Pattern of Observation:
* Explanation: Defines a one-to-many dependence between objects such that when one object changes state, all dependents are immediately alerted and changed
* When an object, such as in a publish-subscribe system, needs to notify other objects about changes in its state.
* Decorator Style:
* Explanation: Dynamically assigns new responsibilities to an item. Decorators offer a more flexible option for expanding functionality than subclassing.
* Use Case: When you wish to change or enhance the behaviour of items without changing their code
* Adapter Design:
* Explanation: Allows an existing class's interface to be utilised as another interface. It is frequently used to make existing classes compatible with others without updating their source code.
* Use Case: When adding new features or components to an existing system without modifying the code.
* Pattern of Command:
* Explanation: Encapsulates a request as an object, allowing for parameterization of clients with varied requests, request queuing, and recording of request parameters.
* When you wish to parameterize objects with operations, queue requests, or log actions, here is the place to go.

6.2) Architecture Patterns:

* Architecture of Microservices:
* Explanation: A microservices architecture divides an application into separate, autonomous services that interact via APIs. Each service is in charge of a distinct business capability.

Selection criteria:

* Scalability: The capacity to independently scale services based on demand.
* Maintainability: Individual services are easier to change and deploy without affecting the overall application.
* Flexibility: Allows for the use of various technologies for various services.
* Monolithic Structure:
* Explanation: A monolithic architecture develops and deploys the complete application as a single unit.

Selection criteria:

* Simplicity: Smaller projects are easier to build and implement.
* Singular Codebase: Can be easier to maintain for some sorts of lower-complexity applications.
* Architecture for Serverless Computing:
* Explanation: In serverless architecture, specific functions are deployed that are triggered by events and operate in stateless containers.

Selection criteria:

* Cost Effectiveness: For some workloads, the pay-per-execution strategy can be cost-effective.
* Scalability: Scales automatically based on demand.
* Architecture with Layers:
* Explanation: A layered architecture divides an application into logical layers (display, business logic, and data) to improve modularity and maintainability.

Selection criteria: Because of the explicit separation of concerns, modularity makes it easier to maintain and comprehend.

* Reuse: Each layer's components can be reused in other portions of the application.
* SOA (Service-Oriented Architecture):
* SOA entails building the application as a set of loosely connected services that communicate via messages.

Selection criteria:

* Interoperability: This is useful when integrating with various systems and services.
* Scalability: Services can be scaled individually.

7) Security

* Authentication and Authorization: MFA (Multi-Factor Authentication):
* To add an extra layer of protection, utilise MFA for user authentication.
* Implement stringent password policies.
* RBAC (Role-Based Access Control):
* Define and enforce role-based access controls.
* Limit rights to the absolute minimum required for each user or system component.
* Data Protection: Rest-of-the-World Encryption:
* Securely store sensitive data in databases or file systems.
* To securely handle encryption keys, use Key Management Services (KMS).
* Masking and redaction of data:
* Use data masking to hide portions of sensitive information in user interfaces.To conceal sensitive information in logs or error messages, use redaction.

8) DevOps

* Continuous Integration (CI): CI/CD Pipeline Implementation:
* CI/CD pipelines may be used to automate the building, testing, and deployment procedures.
* Builds should be triggered automatically on code commits to enable continuous integration of new modifications.
* Testing Automation: Integrate automated testing into the CI pipeline to detect and resolve problems early in the development process. Unit tests, integration tests, and other applicable test suites should all be included.
* IaC (Infrastructure as Code):
* To define and provide infrastructure, use IaC tools (e.g., Terraform, AWS CloudFormation).
* Reduce the risk of deployment errors by ensuring consistency across several settings.
* Infrastructure that cannot be changed: Implement an immutable infrastructure strategy, in which infrastructure components are replaced rather than updated. This ensures consistency and prevents configuration drift.
* Automated Deployment: Continuous Deployment (CD):
* Automate the deployment process to save deployment time and eliminate human mistakes.
* CI/CD pipeline-triggered deployments should be repeatable and dependable.
* Toggle Features: During deployment, use feature toggles to enable or disable individual capabilities. This provides for quick reversal in the event of a problem and allows for feature exploration.
* Compliance and security: Automation of security:
* Integrate security checks within the CI/CD process to detect vulnerabilities as early as possible.
* Implement security scanning technologies that are automated.
* Code of Conduct: Define compliance requirements as code to ensure that deployments comply with regulatory requirements. Audit and update compliance policies on a regular basis.