COLLEGE CODE :9103

COLLEGE NAME :CHENDHURAN COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT:COMPUTER SCIENCE

STUDENT NM-ID : 7F42E92A638F36097B14325442645DEA

ROLL NO :23CS19

DATE :24.10.2025

Completed the project named as

Phase3 TECHNOLOGY PROJECT NAME

: IBM-NJ-EVENT SCHEDULER APP

SUBMITTED BY,

NAME:R.GURU
MOBILE NO:9487636133

IBM-NJ-EVENT SCHEDULER APP: Phase 3 — Minimum Viable Product Implementation Report

I. Executive Summary and Strategic Context

The following report details the strategic blueprint and technical specifications for the IBM-NJ Event Scheduler Application, focusing exclusively on Phase 3: Minimum Viable Product (MVP) implementation, scheduled for completion by Week 8. This phase is designed to deliver immediate, measurable business value while establishing a secure and scalable foundation for future expansion.

1.1. Summary of Project Phase 3: MVP Core Value and Deliverables

The primary purpose of the Event Scheduler MVP is to address a critical operational deficiency within the organization: reliance on manual scheduling processes which inherently lead to non-optimized resource allocation and high administrative overhead.[1] The MVP solution, therefore, is defined as the minimum feature set necessary to resolve this core problem, specifically focusing on reliable, conflict-free scheduling capabilities for internal teams.

The MVP solution emphasizes immediate value delivery.[1] The core functional objective is to provide event creation, attendee management, and essential conflict detection features within a time-bound framework. The key deliverable for this phase is the fully tested, production-ready MVP code base, finalized by the Week 8 deadline, accompanied by comprehensive supporting documentation.[1] Success will be quantitatively measured using Key Success Metrics (KSMs) beyond simple deployment metrics. For example, the system must demonstrate 99.9% availability during beta testing and target a verifiable 50% reduction in manual scheduling conflict reports post-pilot.[2] These metrics are crucial for providing evidence to support the claims of value made to stakeholders.

1.2. Architecture Decision Rationale (ADR) Overview and Non-Functional Requirements (NFRs)

Successfully executing a high-stakes MVP requires transparent decision-making that balances agility with enterprise standards. This implementation necessitates deliberate architectural trade-offs, which are formally captured using Architectural Decision Records (ADRs).[3] This formal process provides the rationale (the "why") behind the chosen patterns and technologies, fostering transparency and accountability within the development team.[3]

A critical aspect of the MVP implementation is managing the inherent tension between speed and enterprise rigor. The project prioritizes rapid development and time-to-market.[5] To meet the Week 8 deadline, certain non-critical complexities, such as advanced microservice decomposition, may be deferred to Phase 4. Such decisions constitute deliberate technical debt. The ADR process ensures these compromises are explicitly documented, listing their resolution as a high-priority requirement for subsequent phases, transforming a technical gamble into an auditable plan.[4]

The stability and acceptance of the application are directly linked to meeting baseline Non-Functional Requirements (NFRs). Although an MVP generally avoids excessive investment in non-core attributes, certain NFRs are critical and cannot be deferred.[6] For the Event Scheduler, critical NFRs include reliability for scheduled data integrity and baseline security compliance, which must adhere to strict IBM code documentation standards.[7] Specific, measurable, achievable, relevant, and time-bound (SMART) targets must be established for quality attributes such as performance (e.g., latency targets for core actions) and availability (e.g., 99.9% uptime).[6]

The rigor applied to documentation structure, visual aids, and consistent formatting [8] is a strategic investment in long-term knowledge transfer.[3] By mandating detailed documentation, the project reduces friction for Phase 4 adoption and simplifies the onboarding process for new development and maintenance teams.[10]

II. Core Implementation Blueprint: Project Setup and Technology Stack

This section provides the foundational blueprint for the technical environment, detailing the structure, technology rationale, and non-functional requirements that govern Phase 3 development.

2.1. Project Initialization and Environmental Setup

Project setup begins with defining the standardized repository structure. The repository is the centralized knowledge hub [9], requiring immediate discoverability of core artifacts.[11] This includes the mandatory creation of a README.md file for project overview, a CONTRIBUTING.md for guidelines, and a SECURITY.md file, which provides essential instructions to collaborators on how to report security vulnerabilities.[12]

Given the enterprise context, security protection begins immediately at the repository level. The project mandates the immediate activation of GitHub's available security features, including Dependabot alerts (for vulnerability notification in dependencies), Secret scanning (for identifying exposed API keys or tokens), and Push protection (to actively block attempts to introduce secrets into the repository).[12] Dependency management must specify core package managers and enforce a process for dependency review to mitigate supply chain risks.

2.2. Architectural Selection and Design Patterns

The selection of the technology stack and architectural patterns is critical for ensuring the MVP meets the Week 8 deadline while remaining scalable post-launch.[13] Architectural selections are formally integrated into the ADR process, prioritizing factors such as Time-to-Market, scalability headroom, developer availability, and inherent security features.[5] For instance, a decision to choose a popular frontend framework like React must be justified by its ability to facilitate rapid development and leverage a broad base of available developers.[5]

Application Architecture: Model-View-Presenter (MVP) Selection

The MVP pattern is mandated for the application architecture. This decision is based on the superior decoupling provided by MVP, which separates View logic from core business logic (Presenter).[14] This separation is crucial for two reasons:

1. **Testability:** MVP minimizes reliance on browser-dependent testing tools (like GWTTestCase), allowing the bulk of the business logic to be tested using faster, lightweight JRE tests.[15]
2. **Concurrency:** Decoupling creates compartmentalized areas of responsibility, which is essential for enabling multiple development teams to work simultaneously without significant code conflict.[15]

In this architecture, the Model handles data access and business entities; the View manages the UI/UX and delegates user interactions; and the Presenter contains the business logic, communicates with the Model, converts data into a UI-friendly format, and updates the View.[14]

Design Pattern Documentation

The implementation must utilize documented, repeatable solutions to common software design problems, which are captured as design patterns.[16] These patterns provide generalized solutions that improve robustness over ad-hoc designs and facilitate developer communication.[16] For example, the **Observer Pattern** is required for handling real-time conflict detection and instant user notifications, and a **Factory Pattern** may be employed for complex object creation related to different event types.

The decision to adopt the MVP pattern primarily for testability and decoupling represents a deliberate management of complexity. While introducing a specific pattern might slightly increase initial development time if the team is unfamiliar with it, the long-term gain in maintainability and reduced testing cost justifies the trade-off of Time-to-Market versus Maintainability.

The following table summarizes key architectural decisions for Phase 3:

Table: Architectural Decision Record Summary (Phase 3 MVP)

Decision ID	**Problem Addressed**	**Options Considered**	**Decision Outcome**	**Trade-Offs & Rationale**
ADR-001	High-Volume Event Processing	Polling, WebSockets, Message Queue	Message Queue (e.g., Kafka)	Higher initial setup complexity traded for superior scalability and decoupling.[3]
ADR-002	Frontend Framework Selection	React, Angular, Vue.js	React	Faster time-to-marke t and broader developer availability for MVP.[5]
ADR-003	State Management	Redux, Context API, Local State	Context API for MVP Core State	Simplifies boilerplate for initial launch, deferring Redux complexity to Phase 4.[4]

ADR-004	Persistent Storage Choice	File System, NoSQL Document Store, SQL Relational DB	SQL Relational Database	Guarantees transactional integrity and structured, relational data storage, critical for scheduling.[18]

2.3. Non-Functional Requirements (NFRs) for MVP

NFRs define the essential quality attributes that the MVP must satisfy for Phase 3 readiness.[20] For the Event Scheduler, strict adherence to enterprise standards makes security a foundational NFR. Adherence must be confirmed against specific internal security standards and implementation protocols.[21] The architecture decision must confirm that the clear separation between the Model/Presenter and View layers, enforced by the MVP pattern, contributes directly to system security by preventing business logic exploitation via the UI.[14]

The key NFR categories established for the MVP are:

- **Performance:** Latency targets are established for critical user actions. For instance, Event creation must complete in less than 500 milliseconds, and the conflict resolution query must complete within 1 second.

- **Availability:** The required percentage uptime during scheduled operational hours must be 99.9%, excluding planned maintenance windows.[6]
- **Security:** This includes mandatory adherence to IBM code documentation standards, robust input validation on all user inputs, and implementation of specific data encryption protocols for sensitive data both at rest and in transit.[7]
- **Scalability:** While full scalability is deferred, the MVP must support an initial operational load, defined as 500 concurrent users and 10,000 scheduled events per week. This establishes a baseline without excessive, non-MVP investment.[6]

III. Core Features Implementation Specification

This section serves as the Software Design Document (SDD), detailing the structure, interfaces, and specific component interactions necessary for implementing the core MVP features.[22]

3.1. Functional Requirements Mapping (User Stories to Components)

Development efforts in Phase 3 are strictly mapped to defined user stories, ensuring that every implemented feature delivers quantifiable user value.[23] All requirements are articulated using the user-centric template: "As a [type of user], I want to [action] so that [benefit]".[24] This clarity prevents ambiguity and provides a focused framework for the development team.23

Table: MVP Core Feature Mapping and Prioritization

User Story ID	**User Type**	**Goal/Action**	**Benefit**	**Technical Component**	**MVP Scope**
				t Owner	**(Core/Nice-to-Have)**
US-001	Registered User	View a calendar of their current week's events	Efficiently manage personal schedule	Frontend/Presenter	Core
US-002	Registered User	Create a new event with start/end times and attendees	Formalize new appointments	Frontend/API Layer	Core
US-003	System Administrator	Generate a report of monthly system	Track organizational	Backend/Reporting Module	Nice-to-Have (Phase 4) 1

		utilization	resource usage		
US-004	Registered User	Receive a notification if a new event conflicts with an existing one	Avoid double-booking	Presenter/Notification Service	Core

3.2. Detailed Feature Design Documentation

The SDD format is used to document the detailed design, including System Architecture, Data Design, Interface Design, and Component Design.[22]

3.2.1. Event Creation and Management Module Blueprint

This blueprint specifies the necessary API endpoints (e.g., POST /events/create, GET /events/{id}) including detailed requirements for data payload validation, expected data schemas, and success/error responses. The data flow dictates that the View delegates the request to the Presenter. The Presenter then orchestrates interaction with the Model/Database, handling data conversion before returning control to the View.[14]

3.2.2. Scheduling and Conflict Resolution Logic (UML Sequence Diagram)

The core value proposition of the application—conflict detection—requires complex inter-component interactions. These interactions must be visualized using a UML Sequence Diagram.[25] This operationalizes the design and minimizes miscommunication, which is essential for mitigating technical bottlenecks and dependencies.[22]

The Sequence Diagram for the Conflict Check Process outlines the following steps: (1) User submits a New Event request via the View; (2) The Presenter validates the input; (3) The Presenter calls the Model's conflict checking function (check_conflict(event_time_range, attendee_list)); (4) The Model queries the Database for intersecting schedules; (5) The Model returns the conflict status (Boolean flag and list of conflicting events); (6) The Presenter applies conditional logic (IF conflict = TRUE) to display an error message (preventing persistence) [25]; and (7) The Presenter initiates the Notification Service if the booking is successful or if a conflict requires administrative review.

3.3. Acceptance Criteria Definition

Every Core Feature must be paired with clear, concise, measurable, and testable Acceptance Criteria (AC).[28] These ACs define the defensive perimeter of the MVP, ensuring strict adherence to the core value proposition and protecting the Week 8 deadline from scope creep.[1] Any proposed functionality falling outside these defined ACs is immediately categorized as "Nice-to-Have" and deferred to Phase 4.

ACs are structured to bridge the gap between user needs and technical implementation.[28] For instance, criteria for the conflict detection feature (US-004) ensure that the system not only detects overlap but also communicates the specific details of the conflict to the user. Table: Acceptance Criteria Examples (Derived from US-004)

User Story	**Functional Requirement**	**Acceptance Criteria (Specific, Measurable, Testable)**
US-004: Avoid double-booking	Conflict Detection Logic	1. The system must query all existing events for all new attendees within the proposed StartTime and EndTime range.

		2. If overlap (even a single minute) is detected, the system SHALL display a blocking error message before persistence.
		3. The error message must list the EventName and StartTime of up to three conflicting events.
		4. The user can opt to override the conflict only if they are the sole mandatory attendee of the conflicting event (Admin Override).

IV. Data Storage and Persistence Strategy

Establishing a robust data foundation is critical for an application centered on transactional integrity like an event scheduler.[29]

4.1. Data Storage Rationale and Selection

The Event Scheduler requires the management of highly structured, relational data, including users, events, locations, and time slots.[19] The formal rationale (ADR-004) mandates the choice of a **Relational Database (SQL)** over NoSQL or file storage options. This selection is based on the need for transactional integrity (ACID properties), which is essential for reliable scheduling and conflict resolution.[19]

The use of file systems for storing data, even document attachments, is strongly advised against for any non-trivial number of documents due to complications related to backup synchronization, directory scheme management, and performance degradation when directories contain thousands of files.[18]

Local state (e.g., frontend caching or session management) is permitted only for temporary, non-critical data. No permanent, transactional data may be managed outside of the persistent storage layer.

4.2. Data Model Specification

Data modeling begins with a **Conceptual Data Model**—a high-level diagram of entities and relationships (User, Event, Location) that facilitates shared understanding among technical and non-technical stakeholders.[30]

A **Physical Data Model (Schema)** provides the detailed blueprint, outlining all tables, primary keys, foreign keys, indices, and data types in the chosen SQL database. Normalization principles must be enforced to ensure data consistency, minimize redundancy, and optimize efficiency.[29] For instance, repeating data, such as a supplier’s address, must be placed in a separate, linked table.[31]

4.3. Data Dictionary

The Data Dictionary is mandated as the centralized repository of metadata, serving as a critical tool for database administrators and application programmers.[32] It ensures data consistency and proper interpretation of data assets.[33] By strictly defining formats, constraints, and relationships within the dictionary, the project enforces underlying business rules (e.g., required time zone standards, constraints on unique event names), thus mitigating the risk of "Schema Drift" and improving long-term application stability.[29]

The Data Dictionary must document every table and column, including: Object Name, Data
Type, Size, Nullability, Data Classification (e.g., PII status), Description, and explicit Relationships.[33] Consistent adherence to enterprise-specific naming conventions (e.g., snake_case for columns) is required.[29]

Table: Data Dictionary Snapshot: Event Entity (Excerpt)

Field Name	**Data Type**	**Nullable**	**Description**	**Constraints / Relationships**
EventID	INT (PK)	NO	Unique identifier for the event.	Primary Key. Auto-incremented.
EventName	VARCHAR(255)	NO	Title of the scheduled event.	Must be unique within the OrganizerID scope.
StartTime	DATETIME	NO	Scheduled start time (UTC).	Must be before EndTime. Index for conflict queries.
EndTime	DATETIME	NO	Scheduled end time (UTC).	Must be after StartTime.
OrganizerID	INT (FK)	NO	Foreign Key linking to the User table.	Defines ownership.[31]

4.4. Migration Strategy for Phase 3 Launch

The MVP data migration scope is limited to initial data seeding (lookup tables, administrative users).[34] The plan must define the execution model and clear ownership.[35] Critically, this migration process must be viewed as a dry run. The process necessitates rigorous post-migration testing and auditing for data integrity, along with a mandatory rollback plan defined in case of schema errors or validation failures.[35] Documenting lessons learned regarding tool efficiency, skill gaps (e.g., specific SQL dialects or data transformation requirements), and validation effort will refine the plan for the eventual Phase 4 migration of larger, production datasets.[34]

V. Quality Assurance and Testing Core Features

This section outlines the Test Plan, designed to assure stakeholders that the MVP delivery meets all defined quality standards by the Week 8 deadline.[36] Skipping a clear testing strategy is a significant oversight that leads to bugs in production.[37]

5.1. Testing Strategy and Environment

The mandated testing approach adheres to the Testing Pyramid: a layered methodology prioritizing Unit testing at the base, followed by Integration testing, and concluding with End-to-End (E2E) testing.[36] The strategy prioritizes testing smaller system components over exhaustive testing of the whole product.[6]

A dedicated, production-like test server (staging environment) is required for execution, equipped with network simulation and security tools, to minimize the risk of deployment discrepancies.[36] The resource allocation specifies clear roles, assigning developers responsibility for unit tests and the QA team responsibility for integration and UAT procedures.[36]

5.2. Test Case Generation and Coverage

Test cases are explicitly derived from the Acceptance Criteria defined in Section 3.3, ensuring that every measurable condition and user value is validated.[28] Furthermore, quantitative coverage metrics are implemented to manage development effort effectively and mitigate severe bugs.[38]

The distribution of test effort is weighted toward the cheaper, faster, and more essential tests that target core business logic.[39] By assigning Unit testing the highest weight (50%) and E2E testing the lowest (20%), the team invests resources heavily in insulating the complex scheduling and conflict resolution logic (residing in the Presenter/Model layers) from bugs, minimizing the resource-intensive UI validation required by E2E tests.[6]

The team must measure and report coverage across combined test suites, taking care to analyze the metrics of independent test suites and combine them accurately without double-counting code covered by both unit and E2E tests.[38]

Table: Phase 3 MVP Test Coverage Matrix

Test Type	**Focus Area**	**Target Coverage (%)**	**Rationale / Weighting**	**Tooling**
Unit Testing	Business Logic (Presenter/Model)	> 85%	Essential for core business logic stability and future refactoring (50% weight).[14]	JUnit/Jest
Integration Testing	API Endpoints & DB Access	> 70%	Validates successful interaction between core system components (30%	Postman/Cypress

			weight).[38]	
E2E Testing	Critical User Flows (UAT)	> 50%	Ensures the MVP can deliver core value	Selenium/Playwright
			end-to-end (20% weight).[37]	

5.3. User Acceptance Testing (UAT) Plan

UAT serves as the formal final gate before launch.[1] The UAT plan mandates detailed step-by-step procedures covering critical user scenarios (e.g., "Scenario: User attempts to book a conflicting meeting").[36] The success criteria for UAT must directly link back to the product vision and core goals established in the executive summary.[1]

Crucially, the team must ensure that all documentation, including user stories and system diagrams, is kept up-to-date.[8] This prevents outdated information from hindering the testing team’s ability to define and execute accurate test cases.[36] By linking test cases directly to the approved acceptance criteria, the team ensures alignment between requirements and delivered implementation.

VI. Version Control and Collaboration (GitHub)

Strict adherence to professional version control standards is necessary to maintain code integrity, traceability, and secure collaboration within the enterprise framework.[41]

6.1. Repository Structure and Documentation Standards

The GitHub repository is established as the central knowledge hub.[9] This requires all documentation, including API specifications, to be consistently structured, easily discoverable, and frequently updated.[8] Adherence to IBM's standards for consistent formatting, indentation, and use of templates for high-level documentation is mandatory.[7] Collaboration requires defined processes for Pull Requests (PRs) and mandatory code reviews before merging into sensitive branches.[9]

6.2. Git Branching Model (Gitflow Adaptation)

A modified Gitflow model is mandated.[42] This model, which embraces long-lived branches for different stability stages, is appropriate because the Event Scheduler is a software product that will be explicitly versioned and may require support for multiple versions or hotfixes in the operational environment.[42]

The model relies on dedicated branches: main (for stable, production-ready code), develop (for integrating new features), feature/ branches (for development), release/ branches (for stabilizing the MVP), and hotfix/ branches.[43] The main and develop branches are protected, requiring mandatory review and approval before merging.[41] The release branch will be created near Week 8 for stabilization tasks (documentation generation, final bug fixes). Once ready, it is merged into main (and tagged with the version number) and merged back into develop.[43]

Developers must strive for atomic commits—each commit should address only one change or fix.[45] When merging feature branches into develop or main, the option to squash multiple commits into a single, clean commit may be used to maintain a cleaner, high-level history on the main line.[41]

6.3. Professional Commit Message Standards

Strict enforcement of the "Seven Rules of a Great Git Commit Message" is required to ensure the commit history serves as a high-fidelity audit trail and reliable historical documentation.[46]

The mandatory rules are:

1. Separate the subject from the body with a blank line.
2. Limit the subject line to 50 characters (72 characters maximum).[46] This forces focused, concise summaries of the change.[46]
3. Capitalize the subject line.
4. Do not end the subject line with a period.
5. Use the imperative mood in the subject line (e.g., "Fix bug," not "Fixed bug").[45]
6. Wrap the body text at 72 characters.[46]
7. Use the body to explain *what* problem the commit is solving and *why* the change was made, focusing on the context and consequences, rather than *how* the change was implemented.[45]

All commits must include references to associated issue tracker tickets (e.g., JIRA) at the bottom of the body (e.g., Resolves: #123), ensuring clear traceability between the code change and the functional requirement.[46]

VII. Risk Management and Project Dependencies

This section proactively identifies threats to the Phase 3 delivery and defines mitigation strategies to ensure the Week 8 deadline is met.[47]

7.1. Enterprise Risk Assessment (Phase 3 Scope)

The objective is to proactively identify potential risks impacting the schedule, scope, and cost.[47] The assessment takes a holistic approach, identifying financial, operational, technical, and schedule risks.[48] Technical risks include potential integration issues or technical debt introduced during rapid development; schedule risks center on the rigidity of the Week 8 deadline.[47]

7.2. Quantitative Risk Prioritization and Mitigation Plan

Risks are prioritized using a quantitative matrix (Likelihood 1-5 multiplied by Impact 1-5), and every high-priority risk is assigned an owner and a specific control implementation plan.[49] This process is integrated into regular project reviews to serve as a continuous feedback loop [47], ensuring mitigation strategies are updated if the risk landscape changes.[49] Table: Quantitative Risk Assessment and Mitigation Plan

Risk ID	Description	Likelihood (1-5)	Impact (1-5)	Priority (L x I)	Mitigation Strategy	Owner
R-T1	Database Schema Drift / Data Integrity Loss	4	5	20 (High)	Implement automated schema migration tool (e.g., Flyway) and strict code review gates. Test recovery plans.[35]	Data Architect
R-O2	Failure to meet Week 8 deadline	3	5	15 (High)	Enforce MVP scope control; eliminate all "Nice-toHave" features.[1] Re-evaluate	Project Manager

					resource allocation daily.	
R-P3	Key Developer Skill Gap (New Stack)	3	4	12 (Medium)	Mandate paired programming; centralize knowledge documentation; invest in just-in-time training.[34]	Development Lead

R-C4	Security Vulnerability Exposure	2	5	10 (Medium)	Implement push protection, secret scanning, and mandatory security review using Code Scanning.[12]	Security Officer

7.3. Dependency Mapping

Dependency mapping is a proactive technique to identify internal and external factors that could impede progress.[27] In the context of the accelerated deadline, external dependencies (e.g., prerequisite cloud service access, integration points with existing IBM directory services) pose the greatest threat to schedule integrity.[47] A Dependency Matrix visualization is required to illustrate interdependencies between Phase 3 deliverables, highlighting external resource dependencies.[51] Effective resource allocation relies on understanding the timing and dependencies of these project phases.[51] Clear escalation paths and real-time communication channels must be established for status updates on these critical external handoffs.[35]

VIII. Conclusion and Future Scope (Phase 4 Planning)

The successful completion of Phase 3 confirms the deployment of a functioning, tested MVP that addresses the core need for reliable, conflict-free scheduling. The system meets the minimum defined NFRs and is ready for pilot deployment and initial stakeholder adoption.

8.1. Achievements of Phase 3 MVP

The MVP has achieved its primary goal: delivering validated core scheduling features based on the defined acceptance criteria (V. Quality Assurance) within the constraints of the Week 8 deadline. The architecture is sound, leveraging the MVP pattern for testability and a relational database for data integrity (II. Core Implementation Blueprint). The next immediate steps include scheduling a compliance audit and securing final stakeholder sign-off prior to the pilot launch.

8.2. Next Steps and Roadmap (Phase 4: Scaling and Adoption)

The transition to Phase 4, "Scaling and Adoption," pivots the focus from feature implementation to stabilization, adoption support, and future growth.[10]

Key Phase 4 Deliverables on the roadmap include:

- **Security Hardening:** A mandatory, full-scale security requirements review must be conducted.[10] As the system moves from pilot to adoption, its security exposure increases, requiring a pivot from functional focus to production robustness and compliance.
- **Technical Debt Resolution:** Known technical debt items documented via ADRs (e.g., scaling up state management from Context API to Redux, if necessary) must be prioritized and resolved.
- **Adoption and Training:** Comprehensive training and onboarding plans for the first few pioneer user teams are required to facilitate widespread organizational adoption.[10]
- **Feature Scaling:** Implementation of "Nice-to-Have" features deferred from Phase 3, such as the System Administrator Reporting (US-003), will proceed.[1]

Phase 4 planning must include an ROI calculation, based on the measured KSM impact metrics from the pilot, to justify continued investment in the overall application roadmap.[10] By transitioning the carefully managed list of deferred features [1] directly into the Phase 4 backlog, the project ensures that managed stakeholder expectations are met and that project momentum is maintained, providing a clear roadmap for long-term product evolution.[52]

Works cited

1. How to Create a Product Specification Document for MVP Development | Zartis, accessed October 24, 2025, https://www.zartis.com/how-to-create-a-product-specification-document-formvp-development/

2. Executive summary template | Confluence - Atlassian, accessed October 24, 2025, https://www.atlassian.com/software/confluence/templates/executive-summary
3. Master architecture decision records (ADRs): Best practices for effective decision-making, accessed October 24, 2025, https://aws.amazon.com/blogs/architecture/master-architecture-decision-record s-adrs-best-practices-for-effective-decision-making/
4. Architecture decision record - Microsoft Azure Well-Architected Framework, accessed October 24, 2025, https://learn.microsoft.com/en-us/azure/well-architected/architect-role/architectu re-decision-record
5. How Do I Choose the Right Technology Stack for My MVP? - Mobile app developers, accessed October 24, 2025, https://thisisglance.com/learning-centre/how-do-i-choose-the-right-technologystack-for-my-mvp
6. Nonfunctional Requirements: Examples, Types and Approaches - AltexSoft, accessed October 24, 2025, https://www.altexsoft.com/blog/non-functional-requirements/
7. What Is Code Documentation? - IBM, accessed October 24, 2025, https://www.ibm.com/think/topics/code-documentation
8. Knowledge Software Documentation Best Practices [With Examples] - Helpjuice, accessed October 24, 2025, https://helpjuice.com/blog/software-documentation
9. Project documentation best practices: 12 essential strategies for 2025 - MeisterTask, accessed October 24, 2025, https://www.meistertask.com/blog/project-documentation-best-practices-12-es sential-strategies-for-2025
10. The four phases to Minimum Viable Platform (MVP) - Humanitec, accessed October 24, 2025, https://humanitec.com/blog/the-four-phases-to-minimum-viable-platform-mvp

11. A Standard Project Structure for Documentation - OpenDevise, accessed October 24, 2025, https://opendevise.com/blog/standard-project-structure-for-docs/
12. Best practices for repositories - GitHub Docs, accessed October 24, 2025, https://docs.github.com/en/repositories/creating-and-managing-repositories/bes t-practices-for-repositories
13. MVP Development: How to Choose the Right Tech Stack - Mobisoft Infotech, accessed October 24, 2025, https://mobisoftinfotech.com/resources/blog/mvp-development-tech-stack-guid e
14. Architecture Design Patterns: MVP | by Ashley Ng - Medium, accessed October 24, 2025, https://medium.com/@shley_ng/architecture-design-patterns-mvp-da44690a8d6
9
15. Building MVP apps: MVP Part I - [GWT] Project, accessed October 24, 2025, https://www.gwtproject.org/articles/mvp-architecture.html
16. Design Patterns - SourceMaking, accessed October 24, 2025, https://sourcemaking.com/design_patterns
17. Software design pattern - Wikipedia, accessed October 24, 2025, https://en.wikipedia.org/wiki/Software_design_pattern
18. Recommended location for document storage - in database or elsewhere? - Stack Overflow, accessed October 24, 2025, https://stackoverflow.com/questions/512262/recommended-location-for-docume nt-storage-in-database-or-elsewhere
19. What Is the Difference Between Data Store and Database - Hypermode, accessed October 24, 2025, https://hypermode.com/blog/data-store-vs-database
20. How to Define Functional and Non-Functional Requirements - Clockwise Software, accessed October 24, 2025, https://clockwise.software/blog/functional-and-nonfunctional-requirements/

21. Standards - IBM, accessed October 24, 2025, https://www.ibm.com/docs/en/zos/3.1.0?topic=functions-standards
22. Software Design Document [Tips & Best Practices] | The Workstream - Atlassian, accessed October 24, 2025, https://www.atlassian.com/work-management/knowledge-sharing/documentatio n/software-design-document
23. User Stories | Examples and Template - Atlassian, accessed October 24, 2025, https://www.atlassian.com/agile/project-management/user-stories
24. User story mapping: A step-by-step guide with templates and examples - Aha.io, accessed October 24, 2025, https://www.aha.io/roadmapping/guide/release-management/what-is-user-storymapping
25. How to Generate Sequence Diagram from User Story? - Visual Paradigm, accessed October 24, 2025, https://www.visual-paradigm.com/tutorials/user-story-to-sequence-diagram.jsp
26. From user stories to sequence diagram - uml - Stack Overflow, accessed October 24, 2025, https://stackoverflow.com/questions/24040528/from-user-stories-to-sequencediagram
27. Dependency mapping template | Confluence - Atlassian, accessed October 24, 2025, https://www.atlassian.com/software/confluence/templates/dependency-mapping
28. Acceptance Criteria Explained [+ Examples & Tips] | The Workstream - Atlassian, accessed October 24, 2025, https://www.atlassian.com/work-management/project-management/acceptance
-criteria
29. Complete Guide to Database Schema Design | Integrate.io, accessed October 24, 2025,

https://www.integrate.io/blog/complete-guide-to-database-schema-design-guid e/
30. What is Data Modeling? - AWS, accessed October 24, 2025, https://aws.amazon.com/what-is/data-modeling/
31. Database design basics - Microsoft Support, accessed October 24, 2025, https://support.microsoft.com/en-us/office/database-design-basics-eb2159cf-1e 30-401a-8084-bd4f9c9ca1f5
32. Data Dictionary: Examples, Templates, & Best practices - Atlan, accessed October
24, 2025, https://atlan.com/what-is-a-data-dictionary/
33. Data Dictionary Blank Template - ScholarlyCommons, accessed October 24, 2025, https://repository.upenn.edu/entities/publication/0430ccdd-cbd8-4404-9f54-11c b81d5b3b1
34. Snowflake Migration Series — Lesson 1: The Power of a Strategic MVP: Your First Slice of Value | by Augusto Rosa - Medium, accessed October 24, 2025, https://medium.com/snowflake/snowflake-migration-series-lesson-1-the-powerof-a-strategic-mvp-your-first-slice-of-value-3ee99ee0f8ac
35. An MVP Approach to Data Migration Best Practices - Gable.ai, accessed October
24, 2025, https://www.gable.ai/blog/data-migration-best-practices
36. Free Test Plan Template | Confluence - Atlassian, accessed October 24, 2025, https://www.atlassian.com/software/confluence/resources/guides/how-to/test-pl an
37. MVP Specification Document 2025: Complete Software Requirement Specification, accessed October 24, 2025, https://www.f22labs.com/blogs/mvp-specification-document-2025-complete-so ftware-requirement-specification/
38. How to measure backend code coverage from E2E tests? (Yes, I know it's a questionable idea) : r/dotnet - Reddit, accessed October

24, 2025, https://www.reddit.com/r/dotnet/comments/1isbdk3/how_to_measure_backend_c ode_coverage_from_e2e/
39. Test Coverage Demystified: A Complete Introductory Guide | LinearB Blog, accessed October 24, 2025, https://linearb.io/blog/test-coverage-demystified
40. 9 Software Documentation Best Practices + Real Examples - Atlassian, accessed October 24, 2025, https://www.atlassian.com/blog/loom/software-documentation-best-practices
41. The Git branching model for mainframe development - IBM, accessed October 24, 2025, https://www.ibm.com/docs/en/z-devops-guide?topic=use-git-branching-modelmainframe-development
42. A successful Git branching model - nvie.com, accessed October 24, 2025, https://nvie.com/posts/a-successful-git-branching-model/
43. Gitflow Workflow | Atlassian Git Tutorial, accessed October 24, 2025, https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
44. Branching Workflows - Git, accessed October 24, 2025, https://git-scm.com/book/ms/v2/Git-Branching-Branching-Workflows
45. Commit message guidelines - GitHub Gist, accessed October 24, 2025, https://gist.github.com/robertpainsi/b632364184e70900af4ab688decf6f53
46. How to Write a Git Commit Message - cbea.ms, accessed October 24, 2025, https://cbea.ms/git-commit/
47. How to Make a Risk Assessment Report (Templates Included) - ProjectManager, accessed October 24, 2025, https://www.projectmanager.com/blog/risk-assessment-report
48. Tips for Conducting an Impactful Enterprise Risk Assessment - Moss Adams, accessed October 24, 2025,

https://www.mossadams.com/articles/2025/04/enterprise-risk-assessments
49. Risk Assessment Matrix: Overview and Guide - AuditBoard, accessed October
24, 2025, https://auditboard.com/blog/what-is-a-risk-assessment-matrix
50. Risk Assessment Matrix: How to Calculate & Use a Risk Matrix Effectively - Vector Solutions, accessed October 24, 2025, https://www.vectorsolutions.com/resources/blogs/risk-matrix-calculations-severit y-probability-risk-assessment/
51. Top 10 Dependency Matrix Templates with Examples and Samples Product Links - SlideTeam, accessed October 24, 2025, https://www.slideteam.net/blog/top-10-dependency-matrix-templates-with-exa mples-and-samples-product-links
52. Scoping Your MVP Development: The 4 Key Steps - Teravision Technologies, accessed October 24, 2025, https://www.teravisiontech.com/blog/scoping-your-mvp-development-the-4-key -steps