

2026-10-14

Table des matières

1	Project Requirements - Template	1
1.1	Requirements Terminology	1
1.2	Gherkin Overview and Applications	2
1.2.a	Core Gherkin Elements	2
1.2.b	Best Practices for Writing Gherkin	3
1.3	Using Gherkin Tags	3
1.3.a	Proposed Tag Classification System	4
1.4	Glossary template	4
1.5	Gherkin Specification Format and Numbering template	5
1.6	Example Requirements Structure	5
1.6.a	EPIC: E01 - [User Account Management]	5
2	Acceptance Tests	5
3	Acceptance Tests	6
3.1	Requirement Dependencies and Relationships	6
3.1.a	Dependency Types	6
3.1.b	Example Usage	7
3.1.c	Authoring and Validation	7
3.2	Requirement Traceability Framework	7
3.2.a	Traceability Matrix Template	7
3.2.b	Traceability Tags	7
3.2.c	Example with Traceability	8
3.2.d	Continuous Traceability	8

1 Project Requirements - Template

1.1 Requirements Terminology

Term	Definition
Epic	A large body of work that can be broken down into multiple features. Epics represent a significant area of functionality or a major user need.
Feature	A distinct piece of functionality that delivers value to users. Features are smaller units of work that make up an epic.
User Story	A short, simple description of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system.
As a [user type], I want [goal] so that [benefit/value].	A common format for writing user stories that includes the user type, the goal or desired functionality, and the benefit or value of the feature.
Gherkin	A Business Readable, Domain Specific Language that lets you describe software's behavior without detailing

Term	Definition
	how that behavior is implemented.
Given-When-Then	A structured approach used in Gherkin for writing test cases: «Given» sets up the test context, «When» specifies the action, and «Then» verifies the outcome.
Scenario	A concrete example that illustrates a business rule or requirement, consisting of a specific situation, action, and outcome.
Acceptance Test	A test that determines whether a feature or user story satisfies the acceptance criteria and works as intended.
Functional Requirements	Specifications of what the system should do - the specific behaviors, functions, or features the system must support.
Non-Functional Requirements	Specifications of how the system should be - the quality attributes, design constraints, and implementation constraints the system must meet.
Acceptance Criteria	The conditions that must be satisfied for a user story to be considered complete and working as expected.

1.2 Gherkin Overview and Applications

Gherkin is a domain-specific language designed to describe the behavior of software in a human-readable way while still being structu-

red enough to enable automated testing. While originally developed for Behavior-Driven Development (BDD), Gherkin's «Given-When-Then» format has proven valuable across multiple phases of software development :

1. **Requirements Gathering:** Gherkin helps stakeholders express requirements clearly and unambiguously.
2. **Design Documentation:** The scenarios serve as living documentation of how the system should behave.
3. **Test Case Definition:** Gherkin scenarios automatically become test cases for validation.
4. **User Acceptance Testing:** Business stakeholders can review Gherkin scenarios to verify the system meets their expectations.
5. **Automated Testing:** Scenarios can be directly connected to automated test code using frameworks like Cucumber.
6. **Communication Tool:** Gherkin serves as a common language between technical and non-technical team members.

1.2.a Core Gherkin Elements

- **Feature:** Describes a discrete piece of functionality in the system
- **Scenario:** An example demonstrating a specific aspect of the feature
- **Given:** Sets up the initial context/precondition
- **When:** Describes the action or event that occurs
- **Then:** Specifies the expected outcome
- **And/But:** Used to add more context, actions, or expected outcomes
- **Background:** Common steps for all scenarios in a feature
- **Scenario Outline:** Template for multiple similar scenarios with Examples table
- **Tags:** Metadata prefixed with @ to categorize and filter scenarios

1.2.b Best Practices for Writing Gherkin

1. **Focus on Behavior, Not Implementation:** Describe what the system should do, not how it does it.

```
# Good: When the user submits a
valid login form
# Bad: When the user clicks the
submit button which triggers the
login() function
```

2. **Write from the User's Perspective:** Express scenarios in terms of user actions and visible outcomes.
3. **Be Specific and Concrete:** Use clear, definitive language avoiding vague terms.

```
# Good: Then the shopping cart
displays 2 items
# Bad: Then the shopping cart is
updated
```

4. **Keep Scenarios Independent:** Each scenario should be self-contained and not depend on other scenarios.
5. **Use Consistent Language:** Establish domain terminology and use it consistently throughout specifications.
6. **Limit Scenario Length:** Aim for 3-5 steps per scenario; longer scenarios suggest a need for refactoring.
7. **Use Background for Common Setup:** Extract repetitive Given steps into a Background section.
8. **Leverage Scenario Outlines**:** Use templates with Examples tables for data-driven scenarios.

```
Scenario      Outline:      Adding
<quantity> product to cart
```

```
Given the user is on the product
page
When they add <quantity> items
to the cart
Then the cart total should be
<price>
```

Examples:

quantity	price
1	\$10
2	\$20

9. **Apply Tags Strategically:** Use tags to group related scenarios for better organization and test execution.
10. **Avoid Technical Details:** Keep scenarios business-focused, avoiding technical implementation details.

The consistent structure of Gherkin specifications allows for clear traceability from requirements to implementation to testing, making it an effective tool throughout the entire software development lifecycle.

1.3 Using Gherkin Tags

Tags are metadata markers that can be applied to features and scenarios in Gherkin specifications, prefixed with the @ symbol. They serve multiple valuable purposes in the requirements management process:

1. **Categorization:** Tags like @ui, @api, @security, or @performance categorize requirements by type.
2. **Filtering:** Test runners can execute specific subsets of scenarios based on tags, such as @smoke or @regression.
3. **Priority Marking:** Tags like @critical, @high, @medium, or @low indicate the importance of requirements.

4. **Release Management:** Tags such as `@sprint-5` or `@v2.1` associate requirements with specific development cycles.
5. **Status Tracking:** Tags like `@wip` (work in progress) or `@blocked` indicate the current state of implementation.
6. **Stakeholder Reference:** Tags can indicate which department or role a feature is primarily for, like `@marketing` or `@admin`.

1.3.a Proposed Tag Classification System

For consistent organization and filtering of requirements, we recommend the following tag classification scheme:

Priority Tags: - `@critical` - Must be implemented for core functionality - `@high` - Important feature needed soon - `@medium` - Standard priority feature - `@low` - Nice-to-have feature

Type Tags: - `@functional` - Core behavior functionality - `@non-functional` - Quality attributes like performance, security - `@ui` - User interface related - `@api` - API functionality - `@data` - Data handling and storage - `@integration` - Integration with external systems

Testing Tags: - `@smoke` - Basic functionality verification - `@regression` - Tests for regression detection - `@automated` - Scenarios automated in test suite - `@manual` - Scenarios requiring manual testing - `@performance` - Performance testing scenarios - `@security` - Security testing scenarios - `@accessibility` - Accessibility testing scenarios - `@usability` - Usability testing scenarios

Status Tags: - `@wip` - Work in progress - `@blocked` - Implementation blocked - `@implemented` - Feature implemented - `@verified` - Feature tested and verified

Release Tags: - `@sprint-X` - Associated with a specific sprint - `@release-X.Y.Z` - Associated

with a specific release - `@milestone-X` - Associated with a specific milestone

Stakeholder Tags: - `@user-role-X` - Feature relevant to specific user role - `@department-X` - Feature relevant to specific department

Including appropriate tags in your Gherkin specifications enhances traceability, filtering capabilities, and overall requirements management efficiency.

1.4 Glossary template

A glossary is necessary to ensure that all stakeholders have a common understanding of the terms used in the requirements documentation. The glossary should be updated as new terms are introduced or definitions are refined.

Term	Definition
Term 1	Definition of a key concept in the project
Term 2	Definition of another important concept
Term 3	Definition of a technical term
Term 4	Definition of a domain-specific term
Term 5	Definition of a process-related term
Term 6	Definition of a user-facing feature
Term 7	Definition of a technical component
Term 8	Definition of a metric or measurement
Term 9	Definition of a project-specific methodology
Term 10	Definition of a key performance indicator

1.5 Gherkin Specification Format and Numbering template

The numbering will follow the hierarchical format below using two-digit numbers: - EPIC: E[01-99] (e.g., E01) - Feature: E[01-99]-F[01-99] (e.g., E01-F01) - User Story: E[01-99]-F[01-99]-US[01-99] (e.g., E01-F01-US01) - Scenario: E[01-99]-F[01-99]-S[01-99] (e.g., E01-F01-S01) - Acceptance Test: E[01-99]-F[01-99]-TA[01-99] (e.g., E01-F01-TA01)

```
## EPIC: E[01-99] - [EPIC NAME]

### Feature: E[01-99]-F[01-99] - [FEATURE NAME]

@priority @type @sprint-X @user-role-X
**User Story:** E[01-99]-F[01-99]-US[01-99] - As a [user type], I want [goal] so that [benefit/value].

@tag1 @tag2 @tag3
Scenario: E[01-99]-F[01-99]-S[01-99] - [Main scenario title]
  Given [initial context]
  When [triggered action]
  Then [expected result]
  And [additional result]

@tag1 @tag2 @tag3
Scenario: E[01-99]-F[01-99]-S[01-99] - [Another main scenario title]
  Given [initial context]
  When [triggered action]
  Then [expected result]
  And [additional result]

# Acceptance Test

@tag1 @tag2 @tag3
Scenario: E[01-99]-F[01-99]-TA[01-99] - [Acceptance test title]
```

```
Given [specific initial context]
When [particular action]
Then [precise acceptance criteria]
    And [additional acceptance criteria]
```

1.6 Example Requirements Structure

1.6.a EPIC: E01 - [User Account Management]

1.6.a.a Feature: E01-F01 - [User Registration]

@functional @critical @sprint-1 @user-role-all **User Story:** E01-F01-US01 - As a new user, I want to create an account so that I can access the application's features.

@happy-path @smoke @automated **Scenario:** E01-F01-S01 - Successful Registration Given the user is on the registration page When the user enters valid registration information And submits the registration form Then a new account should be created And the user should be redirected to the welcome page

@negative @security @automated **Scenario:** E01-F01-S02 - Registration with Invalid Data Given the user is on the registration page When the user enters invalid registration information Then appropriate error messages should be displayed And the registration should not be completed

2 Acceptance Tests

@automated @regression @sprint-1 **Scenario:** E01-F01-TA01 - Verify User Registration with Valid Data Given a new user with valid information When the registration process is completed Then the user should be able to log in with the new credentials And the user profile should contain the correct information

@security @automated @regression Scenario: E01-F01-TA02 - Verify Email Verification Process Given a user has registered but not verified their email When they click on the verification link sent to their email Then their account should be marked as verified And they should get access to all verified user features

2.0.0.a Feature: E01-F02 - [User Authentication]

@functional @critical @sprint-1 @security **User Story:** E01-F02-US01 - As a registered user, I want to log in to my account so that I can access my personal data.

@happy-path @smoke @automated Scenario: E01-F02-S01 - Successful Login Given the user is on the login page When the user enters valid credentials And submits the login form Then the user should be authenticated And redirected to their dashboard

@non-functional @security @automated Scenario: E01-F02-S02 - Password Security Requirements Given the system security configuration When a password is evaluated Then it must contain at least 8 characters And include at least one uppercase letter, one lowercase letter, and one number And reject commonly used passwords

@non-functional @performance @manual Scenario: E01-F02-S03 - Login Performance Given the system is under normal load When 100 concurrent users attempt to log in Then all authentication requests should complete within 2 seconds And the system maintains normal responsiveness

@non-functional @accessibility @manual Scenario: E01-F02-S04 - Login Accessibility Given a user with screen reader technology When navigating the login form Then all form elements should be properly labeled And keyboard navigation should function correctly

And form validation errors should be announced by the screen reader

3 Acceptance Tests

@automated @smoke @sprint-1 Scenario: E01-F02-TA01 - Verify Login Session Management Given a user has logged in successfully When the user is inactive for more than 30 minutes Then the session should expire And the user should be prompted to login again

@security @automated @regression Scenario: E01-F02-TA02 - Verify Account Lockout Policy Given a user attempts to login with incorrect credentials When they fail 5 consecutive login attempts Then the account should be temporarily locked for 15 minutes And an email notification should be sent to the account owner

3.1 Requirement Dependencies and Relationships

Understanding and documenting relationships between requirements is crucial for effective project planning, implementation, and testing. Use the following conventions to document these relationships:

3.1.a Dependency Types

Relationship	Syntax	Description
Depends on	@depends-on:E01-F02	The current requirement depends on another requirement being implemented first
Extends	@extends:E01-F02	The current requirement extends or enhances functionality of another requirement

Relationship	Syntax	Description
Conflicts with	<code>@conflict-with:E01-F02</code>	The current requirement may conflict with another requirement, needing resolution
Replaces	<code>@replaces:F02</code>	The current requirement replaces or supersedes an earlier requirement
Related to	<code>@related-to:E01-F02</code>	The current requirement has a general relationship with another requirement

3.1.b Example Usage

```
@functional    @sprint-2    @depends-on:E01-F01
Scenario: E01-F03-S01 - User Profile Update
    Given the user is logged in
    When the user updates their profile information
    Then the changes should be saved
    And the updated information should be displayed
```

3.1.c Authoring and Validation

Tool	Purpose	Key Features
Visual Studio Code with	Authoring and syntax highlighting	Syntax highlighting, autocompletion, and validation

Tool	Purpose	Key Features
Cucumber plugin		
IntelliJ IDEA with	Gherkin editing and validation	Gherkin syntax highlighting, formatting, and error checking
Gherkin plugin		

3.2 Requirement Traceability Framework

Traceability ensures that requirements are linked to their sources, related artifacts, implementation components, and verification tests. This helps maintain alignment between stakeholder needs and delivered software.

3.2.a Traceability Matrix Template

Requirement ID	Source	User Story	Design Document	Implementation	Test Case	Status
E01-F01-S01	Client Meeting #2	E01-F01-US01	DB Schema v1.2	User Confirmation	Full Met F01-TA01	

3.2.b Traceability Tags

Use these special tags to establish traceability links in your Gherkin specifications:

Tag	Purpose	Example
<code>@source</code>	Links to requirement source	<code>@source:PRD-v1.2</code>
<code>@implements</code>	Links to design document	<code>@implements:design-doc-login</code>

Tag	Purpose	Example	4. Testing Phase: Link tests to requirements and implementation
@code	Links to implementation	@code:com.example.UserController	5. Delivery Phase: Verify complete traceability chains for delivered features
@tests	Links to test implementation	@tests:UserRegistrationTest	

3.2.c Example with Traceability

```
@functional @critical @sprint-1
@user-role-all
@source:client-workshop-2023-01-15
@implements:auth-system-design-v1
**User Story:** E01-F01-US01 - As a
new user, I want to create an account
so that I can access the application's
features.

@happy-path @smoke @automated
@code:com.example.auth.RegistrationController
@tests:com.example.tests.RegistrationTest
Scenario: E01-F01-S01 - Successful
Registration
    Given the user is on the
    registration page
    When the user enters valid
    registration information
    And submits the registration form
    Then a new account should be created
    And the user should be redirected
    to the welcome page
```

3.2.d Continuous Traceability

Maintain traceability throughout the project lifecycle:

1. **Requirements Phase:** Link requirements to their sources and other requirements
2. **Design Phase:** Link design artifacts to requirements
3. **Implementation Phase:** Link code to requirements and design