backlog-generator

Agent Test Scenario: Backlog Generator

This document contains test scenarios for validating the Backlog Generator agent.

Scenario 1: Basic Feature Request

Input

Prompt:

```
Generate user stories for a task notification system. Users should receive
notifications when tasks are assigned to them and when tasks are approaching
their due dates.
```

Context: No additional files or context provided.

Expected Output

Format:

- Multiple user stories following "As a... I want... So that..." format
- Each story should have acceptance criteria
- Stories should be independent (INVEST principles)
- Should include estimates and priorities

- Should identify dependencies if any

Content Expectations:

- At least 3-4 user stories
- Stories cover: task assignment notifications, due date notifications
- May include: notification preferences, notification delivery methods
- Acceptance criteria are testable
- Estimates are reasonable (e.g., 1-5 story points)

Example Expected Story:

```
### Story 1: Receive Notification on Task Assignment

**As a** team member
**I want** to receive a notification when a task is assigned to me
**So that** I can be immediately aware of new responsibilities

**Acceptance Criteria:**
- [ ] Given a task is assigned to me, When the assignment is saved,
      Then I receive a notification within 1 minute
- [ ] Given I receive a notification, When I click it,
      Then I am taken to the task details page
- [ ] Given multiple tasks are assigned, When they are assigned in bulk,
      Then I receive a single summary notification

**Dependencies:** None
**Estimate:** 3 points
**Priority:** High
```

Success Criteria

- ☐ Output follows defined format
- ☐ All stories use "As a... I want... So that..." structure
- ☐ Acceptance criteria are clear and testable
- ☐ Stories are appropriately sized
- ☐ Priorities make sense
- ☐ No duplicate functionality across stories

Scenario 2: Epic Breakdown

Input

Prompt:

```
Break down the "Task Management Dashboard" epic into user stories.
The dashboard should allow users to:
- View all their assigned tasks in one place
- Filter and sort tasks by various criteria
- See task status at a glance
- Get insights into their productivity

Target users are individual contributors managing 10-50 tasks.
```

Context: TaskManager.Domain and TaskManager.Application code visible.

Expected Output

Format:

- Epic description at top
- 6-10 user stories covering all stated functionality
- Stories organized by theme/feature
- Dependencies between stories identified
- Story estimates provided

Content Expectations:

- Stories cover: task list view, filtering, sorting, status visualization, productivity metrics
- Stories are progressively built (e.g., basic view before advanced features)
- Acceptance criteria include UI/UX details where relevant
- Performance criteria for large task lists (10-50 tasks)
- Mobile/responsive considerations may be mentioned

Example Expected Stories:

```markdown
## Epic: Task Management Dashboard

### Theme: Core View

#### Story 1: View Task List
**As a** user
**I want** to see all my assigned tasks in a single list
**So that** I have a central place to manage my work

**Acceptance Criteria:**
- [ ] Given I have assigned tasks, When I open the dashboard,
      Then I see a list of all tasks assigned to me
- [ ] Given the list, When there are more than 20 tasks,
      Then the list is paginated
- [ ] Given each task, When displayed,
      Then I see title, due date, priority, and status
```

```
**Dependencies:** None
**Estimate:** 5 points
**Priority:** High

---

#### Story 2: Filter Tasks by Status
**As a** user
**I want** to filter tasks by status (Not Started, In Progress, Completed)
**So that** I can focus on active work

**Acceptance Criteria:**
- [ ] Given the task list, When I select a status filter,
      Then only tasks with that status are displayed
- [ ] Given filtered results, When I clear the filter,
      Then all tasks are displayed again
- [ ] Given multiple filters, When I apply them,
      Then the count of visible tasks updates immediately

**Dependencies:** Story 1 (View Task List)
**Estimate:** 3 points
**Priority:** High
```

Success Criteria

- ☐ Epic is properly introduced
- ☐ Stories cover all epic requirements
- ☐ Stories are logically ordered
- ☐ Dependencies correctly identified
- ☐ Realistic estimates (consider complexity)
- ☐ Balance between completeness and story size

Scenario 3: Refinement with Constraints

Input

Prompt:

```
Generate user stories for adding task comments and collaboration features.
Constraints:
- Must integrate with existing Task aggregate
- Comments should support @mentions
- Real-time updates preferred but not required for MVP
- Target completion: 2-week sprint

Keep stories small enough to fit in the sprint.
```

Context:

- TaskManager.Domain/Tasks/Task.cs open
- Current Sprint capacity: ~20 story points

Expected Output

Format:

- 4-6 small stories (2-5 points each)
- Total estimate ≤ 20 points
- Stories prioritized for MVP
- Real-time features marked as stretch goals or future stories
- Technical notes about integration with Task aggregate

Content Expectations:

- MVP stories: basic commenting, viewing comments
- Enhanced stories: @mentions, notifications
- Future/stretch: real-time updates
- Acceptance criteria consider existing domain model
- May reference specific domain patterns (DDD)

Example Expected Output:

```
## Feature: Task Comments and Collaboration

**Sprint Capacity:** 20 points
**MVP Scope:** Basic commenting with @mentions

### Story 1: Add Comment to Task (MVP)
**As a** team member
**I want** to add comments to a task
**So that** I can provide updates or ask questions

**Acceptance Criteria:**
- [ ] Given a task, When I add a comment with text,
      Then the comment is saved and associated with the task
- [ ] Given a comment, When it's saved,
      Then it includes author, timestamp, and text
- [ ] Given a task with comments, When I view the task,
      Then I see all comments in chronological order

**Technical Notes:**
- Add Comment value object to Task aggregate
- Maintain aggregate boundaries per DDD
- Comments are part of Task aggregate, not separate entity
```

```markdown
**Dependencies:** None
**Estimate:** 5 points
**Priority:** High (MVP)

---

### Story 2: View Task Comments (MVP)
**As a** team member
**I want** to view all comments on a task
**So that** I can follow the conversation history

**Acceptance Criteria:**
- [ ] Given a task with comments, When I open task details,
      Then I see all comments in chronological order
- [ ] Given a comment, When displayed,
      Then I see author name, timestamp, and full text
- [ ] Given many comments (>10), When viewing,
      Then performance is acceptable (< 500ms load)

**Dependencies:** Story 1
**Estimate:** 3 points
**Priority:** High (MVP)

---

### Story 3: @Mention Users in Comments (MVP)
**As a** commenter
**I want** to @mention other users in comments
**So that** I can direct specific people's attention

**Acceptance Criteria:**
- [ ] Given a comment, When I type @username,
      Then the system recognizes it as a mention
- [ ] Given a mention, When the comment is saved,
      Then the mentioned user is recorded
- [ ] Given a comment with mentions, When displayed,
      Then @mentions are visually highlighted

**Technical Notes:**
- Parse mentions from comment text
- Store mentions as part of Comment value object
- Notification delivery is separate story

**Dependencies:** Story 1
**Estimate:** 5 points
**Priority:** High (MVP)

---
```

```
### Story 4: Notify Users of @Mentions (Stretch)
**As a** mentioned user
**I want** to receive a notification when I'm @mentioned
**So that** I can respond promptly

**Acceptance Criteria:**
- [ ] Given I'm @mentioned in a comment, When the comment is saved,
      Then I receive a notification
- [ ] Given the notification, When I click it,
      Then I'm taken to the task with the comment highlighted

**Dependencies:** Story 3
**Estimate:** 3 points
**Priority:** Medium (Stretch goal)

---

### Future Stories (Out of Sprint)
- Real-time comment updates
- Edit/delete comments
- Rich text formatting
- File attachments to comments

**Total MVP Estimate:** 13 points (leaves buffer for testing/refinement)
```

Success Criteria

- [ ] Total estimate fits sprint capacity
- [ ] Stories are appropriately sized (2-5 points)
- [ ] MVP vs stretch clearly differentiated
- [ ] Technical notes consider existing architecture
- [ ] Dependencies logical and minimal
- [ ] Future work identified but not detailed

Scenario 4: Clarification Request

Input

Prompt:

```
Generate user stories for improving task management.
```

Context: Minimal context provided.

Expected Output

Format:

- Agent should ask clarifying questions before generating stories
- Questions should help scope the feature
- Questions should uncover user needs and constraints

Expected Questions:

- What specific improvements are you looking for?
- Who are the target users?
- What pain points exist with current task management?
- Are there any technical constraints?
- What's the timeline or priority?
- How many stories are you expecting?

Success Criteria

- [ ] Agent recognizes vague input
- [ ] Asks relevant clarifying questions
- [ ] Doesn't make assumptions without input
- [ ] Professional and helpful tone
- [ ] Clear what information is needed

Scenario 5: Technical Story

Input

Prompt:

```
Generate user stories for implementing OpenTelemetry distributed tracing
in the TaskManager API. This is for observability and debugging, not a
user-facing feature.
```

Context: TaskManager.Api/Program.cs open.

Expected Output

Format:

- Stories framed from developer/operator perspective
- Technical acceptance criteria
- May use "As a developer/operator" instead of end user

- Include validation and testing criteria

Content Expectations:

- Stories cover: instrumentation, span creation, context propagation, export
- Acceptance criteria are technical and measurable
- May include performance impact criteria
- Testing approach included

Example Expected Story:

```markdown
### Story 1: Add OpenTelemetry Instrumentation to API

**As a** DevOps engineer
**I want** distributed tracing instrumentation in the API
**So that** I can diagnose performance issues and request flows

**Acceptance Criteria:**
- [ ] Given the API starts, When OpenTelemetry is configured,
      Then traces are exported to the console (development)
- [ ] Given an HTTP request, When it's processed,
      Then a root span is created with request details
- [ ] Given the instrumentation, When running,
      Then performance overhead is < 5% of request time

**Technical Details:**
- Use OpenTelemetry.Extensions.Hosting
- Configure ActivitySource in Program.cs
- Use console exporter for workshop demo

**Dependencies:** None
**Estimate:** 3 points
**Priority:** High
```

Success Criteria

- [ ] Stories appropriate for technical/infrastructure work
- [ ] Personas are developers/operators, not end users
- [ ] Acceptance criteria are technical and measurable
- [ ] Performance impact considered
- [ ] Testing strategy included

Testing Checklist

When testing the Backlog Generator agent with these scenarios:

- [ ] All scenarios produce output in expected format
- [ ] Stories follow "As a... I want... So that..." structure
- [ ] Acceptance criteria are testable
- [ ] Estimates are reasonable
- [ ] Dependencies are correctly identified
- [ ] Priorities make logical sense
- [ ] INVEST principles applied (Independent, Negotiable, Valuable, Estimable, Small, Testable)
- [ ] Agent adapts tone and detail to context provided
- [ ] Agent asks clarifying questions when input is vague
- [ ] Technical stories handled appropriately
- [ ] Sprint constraints respected
- [ ] Output is actionable and ready to use

See Also

- Backlog Generator Agent
- Custom Agent Catalog
- Lab 07: Workflow Agents