Sprint Retrospective Report

Date Submitted:

**Team**: Group 7

**Report Created By**: Jacob Haas and Daniel Rivera

**For Sprint**: 3

**List the goals your team set for this sprint. It is best to name specific features or bug fixes; use the exact issue titles from the GitHub issue tracker**. (*Add more rows if you need*)

|  |  |  |  |
| --- | --- | --- | --- |
| **Issue Title** | **Assigned To** | **Status (Complete, Incomplete, Not Started)** | **Comments (e.g., reasons why feature is incomplete)** |
| Assign a task to myself | Jacob | Completed |  |
| View a task | Jacob | Completed |  |
| Create a task on the board for my group | Jacob | Completed |  |
| Update Stage of my assigned Task | Jacob | Completed |  |
| Assign a Task on a Board from a Project for my Group to an employee in my group | Jacob | Completed |  |
| Assign/Remove Group(s) to/from Stage of Project | Daniel | Incomplete | Correct employees do not populate for assigning ticket. |
| View a history of events for a Task | Daniel | Completed |  |
| Post a comment to a Task on the Board of a Project for my Group | Daniel | Completed |  |
| Edit Group | Daniel | Completed |  |
| Edit Project | Daniel | Completed |  |

**For each team member, please provide (via copy/paste) 2-3 examples of evidence they are actively contributing to the development of the team (beyond just coding, etc.). These could include**:

* A git commit message (please include the commit ID and date/time)
* A comment on an issue in the issue tracker (please include the issue name)
* A bug issue they opened (please include the issue name)
* Message(s) from Slack, Discord, etc.

The examples must be substantive, e.g.,

* explaining why a change was made
* providing detail as to how to reproduce a bug
* suggest options to the team for how to solve a particular problem

|  |  |
| --- | --- |
| **Team Member** | **Example Communication(s)** |
| Jacob | Implemented Ticket Assignment Logic  1) Once a ticket has been taken from the first column it will be assigned to that person unless it is admin or project manager. 2) When a ticket is moved back to the first column it will be unassigned again. 3)The ticket assignment can be changed in the edit ticket modal but only the employees on the project will be the options. |
| Daniel | CSS fix  1. fixed manage columns modal  2. removed random symbol |
| Daniel | Restricted Columns  1. can only assign ticket to member of your group  2. columns are restricted to groups  3. once moving a ticket out of your groups column it unassigns |
| Jacob | Updated Documentation  1) Added Sprint coverage report.  2) Added use case scenarios.  3) Updated Use Case Diagram.  4) Added img files for the added and updated documentation. |

**List your team’s goals for the next sprint.** (*Add more rows if you need*)

|  |  |
| --- | --- |
| **Priority** | **Issue Title** |
| 1 | Assign a task to myself |
| 1 | View Available Tasks |
| 1 | View my assigned Tasks |
| 1 | View Task |
| 1 | Create a Task on the Board of a Project for my Group |
| 1 | Update Stage of my assigned Task |
| 1 | Post a comment to a Task on the Board of a Project for my Group |

# Report on Generative AI Use

**For every session where you used generative AI, give the following:**

1. **Name(s) of team member(s) who created the prompts**
2. **A copy of your session including both prompts and responses provided in a separate document (one per session) with the name of the document included here**
3. **Purpose of the session (e.g., generate example code for connecting to SQL DB)**

**ID of the commit that includes the related source code or documentation changes (state “no changes made” if the session did not lead to any changes)  
  
  
Daniel -  
  
I used chat to help me with the Home controller methods.**/// <summary>

/// Adds the ticket comment.

/// </summary>

/// <param name="ticketId">The ticket identifier.</param>

/// <param name="commentText">The comment text.</param>

/// <returns>adds a ticket comments</returns>

[HttpPost]

public async Task<IActionResult> AddTicketComment(int ticketId, string commentText)

{

var ticket = await \_context.Tickets.FindAsync(ticketId);

if (ticket == null) return NotFound();

var comment = new TicketComment

{

TicketId = ticketId,

CommentText = commentText,

AuthorName = User.Identity.Name ?? "System"

};

\_context.TicketComments.Add(comment);

await \_context.SaveChangesAsync();

return Json(new { success = true });

}

/// <summary>

/// Gets the employees associated with a particular project.

/// </summary>

/// <param name="projectId">The project identifier.</param>

/// <returns>List of employees on a project</returns>

[HttpGet]

public async Task<IActionResult> GetProjectEmployees(int projectId)

{

var userId = GetLoggedInUserId();

var userGroupIds = await \_context.Groups

.Where(g => g.ManagerId == userId || g.EmployeeGroups.Any(eg => eg.EmployeeId == userId))

.Select(g => g.Id)

.ToListAsync();

var employees = await \_context.EmployeeGroups

.Where(eg => userGroupIds.Contains(eg.GroupId) &&

eg.Group.ProjectGroups.Any(pg => pg.ProjectId == projectId))

.Select(eg => new {

id = eg.Employee.Id,

name = eg.Employee.UserName

})

.Distinct()

.ToListAsync();

return Json(employees);

}

/// <summary>

/// Gets the column group.

/// </summary>

/// <param name="columnId">The column identifier.</param>

/// <returns> the column group access</returns>

[HttpGet]

public async Task<IActionResult> GetColumnGroup(int columnId)

{

var access = await \_context.ColumnGroupAccesses

.Include(c => c.Group)

.FirstOrDefaultAsync(c => c.KanbanColumnId == columnId);

if (access == null)

return Json(new { success = false });

return Json(new

{

success = true,

groupId = access.GroupId,

groupName = access.Group.Name

});

}

/// <summary>

/// Updates the column group access.

/// </summary>

/// <param name="columnId">The column identifier.</param>

/// <param name="groupId">The group identifier.</param>

/// <returns>the updated column group access</returns>

[HttpPost]

public async Task<IActionResult> UpdateColumnGroupAccess(int columnId, int groupId)

{

var existing = await \_context.ColumnGroupAccesses

.Where(a => a.KanbanColumnId == columnId)

.ToListAsync();

\_context.ColumnGroupAccesses.RemoveRange(existing);

\_context.ColumnGroupAccesses.Add(new ColumnGroupAccess

{

KanbanColumnId = columnId,

GroupId = groupId

});

await \_context.SaveChangesAsync();

return Json(new { success = true });

}

**Jacob -   
I used chatGPT to help me with ticket movement logic and writing tests for the Home Controller.  
  
/// <summary>**

**/// Moves a ticket between Kanban columns.**

**/// </summary>**

**/// <param name="ticketId">The ticket ID.</param>**

**/// <param name="columnId">The target column ID.</param>**

**/// <returns>Returns success if the operation is successful.</returns>**

**[HttpPost]**

**public async Task<IActionResult> MoveTicket(int ticketId, int columnId)**

**{**

**var ticket = await \_context.Tickets.FindAsync(ticketId);**

**var column = await \_context.KanbanColumns**

**.Include(c => c.GroupAccess)**

**.FirstOrDefaultAsync(c => c.Id == columnId);**

**if (ticket == null || column == null) return NotFound();**

**var project = await \_context.Projects**

**.Include(p => p.KanbanBoard)**

**.ThenInclude(b => b.Columns)**

**.FirstOrDefaultAsync(p => p.Id == ticket.ProjectId);**

**if (project == null) return NotFound();**

**var kanbanBoard = project.KanbanBoard;**

**var firstColumn = kanbanBoard?.Columns.OrderBy(c => c.Order).FirstOrDefault();**

**var userId = GetLoggedInUserId();**

**var isAdmin = User.IsInRole("admin");**

**var isManager = project.ProjectManagerId == userId;**

**var isPrivileged = isAdmin || isManager;**

**if (!isPrivileged)**

**{**

**var userGroupIds = await \_context.Groups**

**.Where(g => g.ManagerId == userId || g.EmployeeGroups.Any(eg => eg.EmployeeId == userId))**

**.Select(g => g.Id)**

**.ToListAsync();**

**var hasAccess = column.GroupAccess.Any(ga => userGroupIds.Contains(ga.GroupId));**

**if (!hasAccess)**

**{**

**ticket.AssignedToId = null;**

**}**

**else if (column.Id == firstColumn?.Id && string.IsNullOrEmpty(ticket.AssignedToId))**

**{**

**ticket.AssignedToId = userId;**

**}**

**}**

**else**

**{**

**if (column.Id == firstColumn?.Id && string.IsNullOrEmpty(ticket.AssignedToId))**

**{**

**ticket.AssignedToId = userId;**

**}**

**}**

**ticket.Status = column.Name;**

**var history = new TicketHistory**

**{**

**TicketId = ticketId,**

**Action = $"Moved to column '{column.Name}'",**

**PerformedBy = User.Identity.Name ?? "System"**

**};**

**\_context.TicketHistories.Add(history);**

**await \_context.SaveChangesAsync();**

**return Json(new { success = true });**

**}  
  
  
[Test]**

**public async Task Index\_UserHasGroupColumnAccess\_PopulatesAccessibleColumnIds()**

**{**

**var group = new Group { Id = 1, Name = "Group A", ManagerId = \_testUserId };**

**var column = new KanbanColumn { Id = 1, Name = "Col", Order = 1, GroupAccess = new List<ColumnGroupAccess>() };**

**var access = new ColumnGroupAccess { GroupId = 1, KanbanColumn = column, KanbanColumnId = 1, Group = group };**

**column.GroupAccess.Add(access);**

**var board = new KanbanBoard { Id = 1, ProjectName = "Board", Columns = new List<KanbanColumn> { column } };**

**var project = new Project { Id = 1, Title = "Test Project", ProjectManagerId = \_testUserId, KanbanBoard = board };**

**\_context.Groups.Add(group);**

**\_context.ColumnGroupAccesses.Add(access);**

**\_context.Projects.Add(project);**

**\_context.KanbanBoards.Add(board);**

**\_context.KanbanColumns.Add(column);**

**await \_context.SaveChangesAsync();**

**var result = await \_controller.Index(1) as ViewResult;**

**var accessible = result?.ViewData["AccessibleColumnIds"] as List<int>;**

**Assert.That(accessible, Is.Not.Null);**

**Assert.That(accessible.Contains(1), Is.True);**

**}**

**[Test]**

**public async Task MoveTicket\_ToFirstColumn\_Unassigned\_AssignsToUser()**

**{**

**var column = new KanbanColumn { Id = 1, Name = "To Do", Order = 1 };**

**var board = new KanbanBoard { Id = 1, ProjectName = "Board", Columns = new List<KanbanColumn> { column } };**

**column.KanbanBoard = board;**

**var project = new Project**

**{**

**Id = 1,**

**Title = "Project",**

**KanbanBoard = board,**

**ProjectManagerId = "manager123"**

**};**

**var ticket = new Ticket**

**{**

**TicketId = 1,**

**ProjectId = 1,**

**Status = "Backlog",**

**CreatedBy = "TestUser",**

**CreatedAt = DateTime.UtcNow,**

**Title = "Move Me",**

**Description = "Move ticket into first column"**

**};**

**\_context.Projects.Add(project);**

**\_context.KanbanBoards.Add(board);**

**\_context.KanbanColumns.Add(column);**

**\_context.Tickets.Add(ticket);**

**await \_context.SaveChangesAsync();**

**\_userMock.Setup(u => u.IsInRole(It.IsAny<string>())).Returns(true); // admin or project manager**

**\_userMock.Setup(u => u.FindFirst(ClaimTypes.NameIdentifier)).Returns(new Claim(ClaimTypes.NameIdentifier, \_testUserId));**

**\_controller.ControllerContext = new ControllerContext { HttpContext = new DefaultHttpContext { User = \_userMock.Object } };**

**var result = await \_controller.MoveTicket(1, 1) as JsonResult;**

**var updatedTicket = await \_context.Tickets.FindAsync(1);**

**Assert.That(result, Is.Not.Null);**

**Assert.That(updatedTicket.AssignedToId, Is.EqualTo(\_testUserId));**

**}**

**[Test]**

**public async Task EditTicket\_AssigneeChanged\_AddsHistory()**

**{**

**var manager = new Employee { Id = "manager1", UserName = "Manager" };**

**var newEmployee = new Employee { Id = "newEmp", UserName = "NewEmployee" };**

**var group = new Group { Id = 1, ManagerId = "manager1" };**

**var project = new Project { Id = 1, Title = "Project" };**

**var ticket = new Ticket**

**{**

**TicketId = 1,**

**Title = "Task",**

**Description = "Desc",**

**ProjectId = 1,**

**AssignedToId = "someone",**

**Status = "To Do",**

**CreatedBy = "Tester",**

**CreatedAt = DateTime.UtcNow**

**};**

**\_context.Users.Add(newEmployee);**

**\_context.Employees.Add(manager);**

**\_context.Groups.Add(group);**

**\_context.ProjectGroups.Add(new ProjectGroup { Project = project, Group = group });**

**\_context.EmployeeGroups.Add(new EmployeeGroup { Group = group, Employee = newEmployee });**

**\_context.Projects.Add(project);**

**\_context.Tickets.Add(ticket);**

**await \_context.SaveChangesAsync();**

**var result = await \_controller.EditTicket(1, "Updated", "Updated Desc", "newEmp") as JsonResult;**

**var updated = await \_context.Tickets.FindAsync(1);**

**var history = await \_context.TicketHistories.FirstOrDefaultAsync(h => h.TicketId == 1);**

**Assert.That(result, Is.Not.Null);**

**Assert.That(updated.AssignedToId, Is.EqualTo("newEmp"));**

**Assert.That(history?.Action, Does.Contain("Assigned to"));**

**}**

**[Test]**

**public async Task GetColumnGroup\_NoAccess\_ReturnsFalse()**

**{**

**var result = await \_controller.GetColumnGroup(999) as JsonResult;**

**Assert.That(result, Is.Not.Null);**

**var json = JObject.FromObject(result.Value!);**

**Assert.That((bool)json["success"]!, Is.False);**

**}**

**[Test]**

**public async Task GetColumnGroup\_Valid\_ReturnsGroup()**

**{**

**var group = new Group { Id = 1, Name = "Team A" };**

**var column = new KanbanColumn { Id = 10, Name = "Col" };**

**var access = new ColumnGroupAccess { KanbanColumnId = 10, GroupId = 1, Group = group };**

**\_context.Groups.Add(group);**

**\_context.KanbanColumns.Add(column);**

**\_context.ColumnGroupAccesses.Add(access);**

**await \_context.SaveChangesAsync();**

**var result = await \_controller.GetColumnGroup(10) as JsonResult;**

**Assert.That(result, Is.Not.Null);**

**var json = JObject.FromObject(result.Value!);**

**Assert.That((bool)json["success"]!, Is.True);**

**Assert.That((int)json["groupId"]!, Is.EqualTo(1));**

**Assert.That((string)json["groupName"]!, Is.EqualTo("Team A"));**

**}**

**[Test]**

**public async Task UpdateColumnGroupAccess\_ReplacesExisting\_ReturnsSuccess()**

**{**

**var groupOld = new Group { Id = 1, Name = "Old" };**

**var groupNew = new Group { Id = 2, Name = "New" };**

**var column = new KanbanColumn { Id = 1, Name = "Col" };**

**var oldAccess = new ColumnGroupAccess { KanbanColumnId = 1, GroupId = 1 };**

**\_context.Groups.AddRange(groupOld, groupNew);**

**\_context.KanbanColumns.Add(column);**

**\_context.ColumnGroupAccesses.Add(oldAccess);**

**await \_context.SaveChangesAsync();**

**var result = await \_controller.UpdateColumnGroupAccess(1, 2) as JsonResult;**

**var updated = await \_context.ColumnGroupAccesses.FirstOrDefaultAsync(a => a.KanbanColumnId == 1);**

**Assert.That(result, Is.Not.Null);**

**Assert.That(updated.GroupId, Is.EqualTo(2));**

**}**

**[Test]**

**public void ColumnGroupAccess\_Id\_CanBeSetAndRetrieved()**

**{**

**var access = new ColumnGroupAccess { Id = 42 };**

**Assert.That(access.Id, Is.EqualTo(42));**

**}**

**[Test]**

**public void KanbanBoard\_Project\_CanBeAssigned()**

**{**

**var project = new Project { Id = 7, Title = "Board's Project" };**

**var board = new KanbanBoard { Project = project };**

**Assert.That(board.Project, Is.Not.Null);**

**Assert.That(board.Project.Id, Is.EqualTo(7));**

**Assert.That(board.Project.Title, Is.EqualTo("Board's Project"));**

**}**

**[Test]**

**public void TicketComment\_Id\_CanBeSet()**

**{**

**var comment = new TicketComment { Id = 99 };**

**Assert.That(comment.Id, Is.EqualTo(99));**

**}**

**[Test]**

**public void TicketHistory\_Id\_CanBeSetAndRetrieved()**

**{**

**var history = new TicketHistory { Id = 101 };**

**Assert.That(history.Id, Is.EqualTo(101));**

**}**

**[Test]**

**public void TicketComment\_TicketReference\_CanBeAssigned()**

**{**

**var ticket = new Ticket { TicketId = 3, Title = "Parent Ticket" };**

**var comment = new TicketComment { Ticket = ticket };**

**Assert.That(comment.Ticket, Is.Not.Null);**

**Assert.That(comment.Ticket.TicketId, Is.EqualTo(3));**

**}**

**[Test]**

**public async Task MoveTicket\_UnprivilegedUserWithAccess\_AssignsToSelf()**

**{**

**var userId = \_testUserId;**

**// Create group and give user access**

**var group = new Group { Id = 1 };**

**var employee = new Employee { Id = userId };**

**var access = new ColumnGroupAccess { KanbanColumnId = 1, GroupId = 1 };**

**var column = new KanbanColumn**

**{**

**Id = 1,**

**Name = "To Do",**

**Order = 1,**

**GroupAccess = new List<ColumnGroupAccess> { access }**

**};**

**var board = new KanbanBoard**

**{**

**Id = 1,**

**ProjectName = "Board",**

**Columns = new List<KanbanColumn> { column }**

**};**

**var project = new Project**

**{**

**Id = 1,**

**KanbanBoard = board,**

**Title = "Test Project",**

**ProjectManagerId = "manager123"**

**};**

**var ticket = new Ticket**

**{**

**TicketId = 1,**

**ProjectId = project.Id,**

**Title = "Ticket",**

**Description = "Unassigned",**

**Status = "Backlog",**

**CreatedBy = "Tester"**

**};**

**\_context.Groups.Add(group);**

**\_context.Employees.Add(employee);**

**\_context.KanbanBoards.Add(board);**

**\_context.KanbanColumns.Add(column);**

**\_context.Projects.Add(project);**

**\_context.Tickets.Add(ticket);**

**\_context.ColumnGroupAccesses.Add(access);**

**\_context.EmployeeGroups.Add(new EmployeeGroup**

**{**

**Group = group,**

**Employee = employee**

**});**

**\_context.SaveChanges();**

**\_userMock.Setup(u => u.IsInRole(It.IsAny<string>())).Returns(false);**

**\_userMock.Setup(u => u.FindFirst(ClaimTypes.NameIdentifier))**

**.Returns(new Claim(ClaimTypes.NameIdentifier, userId));**

**\_controller.ControllerContext = new ControllerContext**

**{**

**HttpContext = new DefaultHttpContext { User = \_userMock.Object }**

**};**

**var result = await \_controller.MoveTicket(ticket.TicketId, column.Id) as JsonResult;**

**var updatedTicket = await \_context.Tickets.FindAsync(ticket.TicketId);**

**Assert.That(result, Is.Not.Null);**

**Assert.That(updatedTicket.AssignedToId, Is.EqualTo(userId));**

**}**