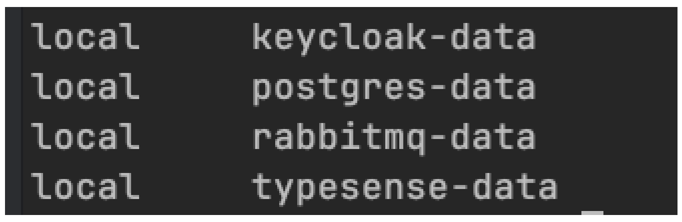
Answers Challenge - Solution

Getting started.

1. You may want to clean the DBs before beginning this so that you have clean data in the DB. To do this we can delete the volumes in docker and then restart the AppHost. Run the following command to check the persistent volumes in docker:

```
docker volume ls
```

2. You should see the following:



```
local     keycloak-data
local     postgres-data
local     rabbitmq-data
local     typesense-data
```

3. To delete the postgres-data and the typesense-data run the following:

```
docker volume rm typesense-data postgres-data
```

4. Now restart the AppHost and you will have clean databases. Create a few questions so you have something in both DBs

Question Service

1. Create a new entity called "Answer.cs" using the info from the specification. Ensure you configure this as a "fully defined relationship" with the Question entity.
2. Add another property to the Question.cs to track the AnswerCount.

```csharp
using System.ComponentModel.DataAnnotations;
using System.Text.Json.Serialization;

namespace QuestionService.Models;

public class Answer
{
    [MaxLength(36)]
    public string Id { get; set; } = Guid.NewGuid().ToString();
    [MaxLength(5000)]
    public required string Content { get; set; }
    [MaxLength(36)]
    public required string UserId { get; set; }
    [MaxLength(300)]
    public required string UserDisplayName { get; set; }
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    public DateTime? UpdatedAt { get; set; }
    public bool Accepted { get; set; }

    // nav properties
    [MaxLength(36)]
    public required string QuestionId { get; set; }
    [JsonIgnore]
    public Question Question { get; set; } = null!;
}
```

```csharp
using System.ComponentModel.DataAnnotations;

namespace QuestionService.Models;
```

```

public class Question
{
    [MaxLength(36)]
    public string Id { get; set; } = Guid.NewGuid().ToString();
    [MaxLength(300)]
    public required string Title { get; set; }
    [MaxLength(5000)]
    public required string Content { get; set; }
    [MaxLength(36)]
    public required string AskerId { get; set; }
    [MaxLength(300)]
    public required string AskerDisplayName { get; set; }
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    public DateTime? UpdatedAt { get; set; }
    public int ViewCount { get; set; }
    public List<string> TagSlugs { get; set; } = [];
    public bool HasAcceptedAnswer { get; set; }
    public int Votes { get; set; }
    public int AnswerCount { get; set; }

    // navigation properties
    public List<Answer> Answers { get; set; } = [];
}
```

3. Add the Answer as a new DbSet in the DbContext so we can query the answers table directly.

```
public class QuestionDbContext(DbContextOptions options) : DbContext(options)
{
    public DbSet<Question> Questions { get; set; }
    public DbSet<Tag> Tags { get; set; }
    public DbSet<Answer> Answers { get; set; }
```

4. Create a new migration for the QuestionDbContext. Check the migration and ensure that:
    1. The QuestionId column in the Answers table is not nullable
    2. The onDelete behavior in the constraints is ReferentialAction.Cascade

```
using System;
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace QuestionService.Data.Migrations
{
    /// <inheritdoc />
    public partial class AnswerAdded : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.AddColumn<int>(
                name: "AnswerCount",
                table: "Questions",
                type: "integer",
                nullable: false,
                defaultValue: 0);

            migrationBuilder.CreateTable(
                name: "Answer",
                columns: table => new
                {
                    Id = table.Column<string>(type: "character varying(36)", maxLength: 36,
nullable: false),
                    Content = table.Column<string>(type: "character varying(5000)", maxLength: 5000, nullable: false),
                    UserId = table.Column<string>(type: "character varying(36)", maxLength: 36, nullable: false),
                    UserDisplayName = table.Column<string>(type: "character varying(300)", maxLength: 300, nullable: false),
                    CreatedAt = table.Column<DateTime>(type: "timestamp with time zone", nullable: false),
                    UpdatedAt = table.Column<DateTime>(type: "timestamp with time zone", nullable: true),
                    Accepted = table.Column<bool>(type: "boolean", nullable: false),
                    QuestionId = table.Column<string>(type: "character varying(36)", maxLength: 36, nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Answer", x => x.Id);
                    table.ForeignKey(
                        name: "FK_Answer_Questions_QuestionId",
                        column: x => x.QuestionId,
                        principalTable: "Questions",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateIndex(
                name: "IX_Answer_QuestionId",
                table: "Answer",
                column: "QuestionId");
        }

        /// <inheritdoc />
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "Answer");

            migrationBuilder.DropColumn(
                name: "AnswerCount",
                table: "Questions");
        }
    }
}

```

5. Create a CreateAnswerDto.cs as per the specification.

```
namespace QuestionService.DTOs;

public record CreateAnswerDto(string Content);
```

6. Create the remaining API endpoints as defined below in the QuestionsController.

Method	Route	Purpose	Auth?	Returns
POST	/questions/{questionId}/answers	Create answer for question.	Auth	201 with **Answer**
PUT	/questions/{questionId}/answers/{answerId}	Update answer	Auth	204 no content
DELETE	/questions/{questionId}/answers/{answerId}	Delete answer. Do not allow accepted answers to be deleted.	Auth	204 no content

Method	Route	Purpose	Auth?	Returns
POST	/questions/{questionId}/answers/{answerId}/accept	Set answer as accepted answer. Ensure that the question does not already have an accepted answer.	Auth	204 no content

POST Answer

```
[Authorize]
    [HttpPost("{questionId}/answers")]
    public async Task<ActionResult> PostAnswer(string questionId, CreateAnswerDto dto)
    {
        var question = await db.Questions.FindAsync(questionId);

        if (question is null) return NotFound();

        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
        var name = User.FindFirstValue("name");

        if (userId is null || name is null) return BadRequest("Cannot get user details");

        var answer = new Answer
        {
            Content = dto.Content,
            UserId = userId,
            UserDisplayName = name,
            QuestionId = questionId
        };

        question.Answers.Add(answer);
        question.AnswerCount++;

        await db.SaveChangesAsync();

        return Created($"/questions/{questionId}", answer);
    }
```

PUT Answer

```
    [Authorize]
    [HttpPut("{questionId}/answers/{answerId}")]
    public async Task<ActionResult> UpdateAnswer(string questionId, string answerId,
CreateAnswerDto dto)
    {
        var answer = await db.Answers.FindAsync(answerId);
        if (answer is null) return NotFound();
        if (answer.QuestionId != questionId) return BadRequest("Cannot update answer
details");

        answer.Content = dto.Content;
        answer.UpdatedAt = DateTime.UtcNow;

        await db.SaveChangesAsync();

        return NoContent();
    }
```

DELETE Answer

```
    [Authorize]
    [HttpDelete("{questionId}/answers/{answerId}")]
    public async Task<ActionResult> DeleteAnswer(string questionId, string answerId)
    {
```

```csharp
        var answer = await db.Answers.FindAsync(answerId);
        var question = await db.Questions.FindAsync(questionId);
        if (answer is null || question is null) return NotFound();
        if (answer.QuestionId != questionId || answer.Accepted) return BadRequest("Cannot delete this answer");

        db.Answers.Remove(answer);
        question.AnswerCount--;

        await db.SaveChangesAsync();

        await bus.PublishAsync(new AnswerCountUpdated(questionId, question.AnswerCount));

        return NoContent();
    }
```

POST AcceptAnswer

```csharp
    [Authorize]
    [HttpPost("questions/{questionId}/answer/{answerId}/accept")]
    public async Task<ActionResult> AcceptAnswer(string questionId, string answerId)
    {
        var answer = await db.Answers.FindAsync(answerId);
        var question = await db.Questions.FindAsync(questionId);
        if (answer is null || question is null) return NotFound();
        if (answer.QuestionId != questionId || question.HasAcceptedAnswer) return BadRequest("Cannot accept answer");

        answer.Accepted = true;
        question.HasAcceptedAnswer = true;

        await db.SaveChangesAsync();

        return NoContent();
    }
```

7. Use eager loading on the GetQuestionById endpoint to return the Answers along with the detailed question.

```csharp
    [HttpGet("{id}")]
    public async Task<ActionResult<Question>> GetQuestion(string id)
    {
        var question = await db.Questions
            .Include(x => x.Answers)
            .FirstOrDefaultAsync(x => x.Id == id);

        if (question is null) return NotFound();

        await db.Questions.Where(x => x.Id == id)
            .ExecuteUpdateAsync(setters => setters.SetProperty(x => x.ViewCount,
                x => x.ViewCount + 1));

        return question;
    }
```

8. Create the contracts for the events.
    1. UpdatedAnswerCount
    2. AnswerAccepted

```csharp
namespace Contracts;

public record UpdatedAnswerCount(string QuestionId, int AnswerCount);
```

```csharp
namespace Contracts;
```

```

public record AnswerAccepted(string QuestionId);
```

8. Publish the events from the respective endpoints:
    1. Post answer - UpdatedAnswerCount
    2. Delete answer - UpdatedAnswerCount
    3. Accept answer - AnswerAccepted

Note: We cannot "increment/decrement" in Typesense without risk of race conditions so we have to publish the updated AnswerCount of the questions.

```
    [Authorize]
    [HttpPost("{questionId}/answers")]
    public async Task<ActionResult> PostAnswer(string questionId, CreateAnswerDto dto)
    {
        var question = await db.Questions.FindAsync(questionId);

        if (question is null) return NotFound();

        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
        var name = User.FindFirstValue("name");

        if (userId is null || name is null) return BadRequest("Cannot get user details");

        var answer = new Answer
        {
            Content = dto.Content,
            UserId = userId,
            UserDisplayName = name,
            QuestionId = questionId
        };

        question.Answers.Add(answer);
        question.AnswerCount++;

        await db.SaveChangesAsync();

        await bus.PublishAsync(new AnswerCountUpdated(questionId, question.AnswerCount));

        return Created($"/questions/{questionId}", answer);
    }
```

```
    [Authorize]
    [HttpDelete("questions/{questionId}/answers/{answerId}")]
    public async Task<ActionResult> DeleteAnswer(string questionId, string answerId)
    {
        var answer = await db.Answers.FindAsync(answerId);
        var question = await db.Questions.FindAsync(questionId);
        if (answer is null || question is null) return NotFound();
        if (answer.QuestionId != questionId) return BadRequest("Cannot delete answer");

        db.Answers.Remove(answer);
        question.AnswerCount--;

        await db.SaveChangesAsync();

        await bus.PublishAsync(new AnswerCountUpdated(questionId, question.AnswerCount));

        return NoContent();
    }
```

```
    [Authorize]
    [HttpPost("questions/{questionId}/answer/{answerId}/accept")]
```

```csharp
    public async Task<ActionResult> AcceptAnswer(string questionId, string answerId)
    {
        var answer = await db.Answers.FindAsync(answerId);
        var question = await db.Questions.FindAsync(questionId);
        if (answer is null || question is null) return NotFound();
        if (answer.QuestionId != questionId || question.HasAcceptedAnswer) return 
BadRequest("Cannot accept answer");

        answer.Accepted = true;
        question.HasAcceptedAnswer = true;

        await db.SaveChangesAsync();

        await bus.PublishAsync(new AnswerAccepted(questionId));

        return NoContent();
    }
```

Search Service

1. Add the AnswerCountUpdatedHandler

```csharp
using Contracts;
using Typesense;

namespace SearchService.MessageHandlers;

public class AnswerCountUpdatedHandler(ITypesenseClient client)
{
    public async Task HandleAsync(AnswerCountUpdated message)
    {
        await client.UpdateDocument("questions", message.QuestionId,
            new { message.AnswerCount }
        );
    }
}
```

2. Add the AcceptAnswerHandler

```csharp
using Contracts;
using Typesense;

namespace SearchService.MessageHandlers;

public class AcceptAnswerHandler(ITypesenseClient client)
{
    public async Task HandleAsync(AnswerAccepted message)
    {
        await client.UpdateDocument("questions", message.QuestionId,
            new {HasAcceptedAnswer = true});
    }
}
```

Restart the AppHost and test the new functionality in Postman.