* I have used code-first approach and used EFCore with LINQ to connect and query to my database.
* I have included two videos, the first is a full test of the API. The second is a test with the hint functionality because I forgot to show it in the first video. Please note the HATEOAS responses in those videos are older and do not contain the new HATEOAS responses seen in the images below here. The videos are within the folder “System Tests Evidence”.
* Further down in the testing section, mentions of a type called “Result” is a type I created to fulfil HATEOAS responses.

# System Tests

## User hasn’t signed up test:

Graphical user interface, application

Description automatically generated

## Text Description automatically generatedGraphical user interface, text, application Description automatically generatedInitial details fails on validation (No Category, No Username, No number of questions typed):

## Graphical user interface, text Description automatically generatedGraphical user interface, text, application, email Description automatically generatedIntial details fails validation as “Choose one for me” isnt a category:

## Graphical user interface, application Description automatically generatedInitial details fails validation since there are not 5000 questions in the category:

## Username is taken error when attempting to start game:Graphical user interface, application Description automatically generated

## Game hasn’t been finished yet for player “TheIdiot”, calling endpoint “end” test:

Graphical user interface, application

Description automatically generated

## Submitting a wrong file type for picture endpoint error:

Graphical user interface, text, application

Description automatically generated

## Complex Question displayed to user:

Graphical user interface, text, application

Description automatically generated

## Basic Question displayed to user:

Graphical user interface, text, application

Description automatically generated

# ER-Diagram of Database:

Diagram

Description automatically generated

# Unit Tests:

Hyperlink to API level tests: [here](Assignment4WC/Assignment4WC.Tests/API)

Hyperlink to Application/Infrastructure level tests: [here](Assignment4WC/Assignment4WC.Tests/Logic)

## Summary of tests:

Text

Description automatically generated

## Summary of FourWeekChallengeController tests + naming convention:

Text

Description automatically generated

## A Few Basic Tests:Text Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

## A Few Complex Tests:Text Description automatically generated Text Description automatically generated Text Description automatically generated

## Sidenote - Assertion Package modification to test my custom type:

Text

Description automatically generated

## Code in AnswerValidator Tests:

public class AnswerValidatorTests : IDisposable

{

protected readonly AssignmentContext Context;

protected readonly IFixture Fixture;

public AnswerValidatorTests()

{

Context = new AssignmentContext(AssignmentContextOptions.GetDbOptions());

Fixture = new Fixture();

}

[Theory]

[InlineData(null)]

[InlineData("")]

[InlineData(" ")]

public void GivenAnswerIsNullOrEmptyOrWhitespace\_ReturnValidationErrorWithAnswerIsEmptyError(string answer)

{

var dtoModel = Fixture.Build<AnswerDto>()

.With(dto => dto.Answer, () => answer)

.Create();

new AnswerValidator().TestValidate(dtoModel)

.ShouldHaveValidationErrorFor(dto => dto.Answer)

.WithErrorMessage("'Answer' must not be empty.");

}

[Fact]

public void GivenAnswerIsValid\_ThenThereShouldBeNoValidationErrorFoAnswer()

{

var dtoModel = Fixture.Build<AnswerDto>()

.With(dto => dto.Answer, () => Fixture.Create<string>())

.Create();

new AnswerValidator().TestValidate(dtoModel)

.ShouldNotHaveValidationErrorFor(dto => dto.Answer);

}

public void Dispose()

{

Context.Dispose();

}

## Code in InitialDetailsValidator Tests:

public class InitialDetailsValidatorTests : IDisposable

{

protected readonly AssignmentContext Context;

protected readonly IFixture Fixture;

public InitialDetailsValidatorTests()

{

Context = new AssignmentContext(AssignmentContextOptions.GetDbOptions());

Fixture = new Fixture();

}

[Theory]

[InlineData(null)]

[InlineData("")]

[InlineData(" ")]

public void GivenUsernameIsNullOrEmptyOrWhitespace\_ReturnValidationErrorWithUsernameIsEmptyError(string username)

{

var dtoModel = Fixture.Build<InitialDetailsDto>()

.With(dto => dto.Username, () => username)

.Create();

new InitialDetailsValidator().TestValidate(dtoModel)

.ShouldHaveValidationErrorFor(dto => dto.Username)

.WithErrorMessage("'Username' must not be empty.");

}

[Fact]

public void GivenUsernameIsValid\_ThenThereShouldBeNoValidationErrorForUsername()

{

var dtoModel = Fixture.Build<InitialDetailsDto>()

.With(dto => dto.Username, () => Fixture.Create<string>())

.Create();

new InitialDetailsValidator().TestValidate(dtoModel)

.ShouldNotHaveValidationErrorFor(dto => dto.Username);

}

[Fact]

public void GivenCategoryNameIsNotAValidCategory\_ReturnValidationErrorWithCategoryIsNotValidError()

{

var dtoModel = Fixture.Build<InitialDetailsDto>()

.With(dto => dto.Category, () => Fixture.Create<string>())

.Create();

new InitialDetailsValidator().TestValidate(dtoModel)

.ShouldHaveValidationErrorFor(dto => dto.Category)

.WithErrorMessage($"'{dtoModel.Category}' is not a valid Category");

}

[Fact]

public void GivenCategoryNameIsAValidCategory\_ThenThereShouldBeNoValidationErrorForCategory()

{

var dtoModel = Fixture.Build<InitialDetailsDto>()

.With(dto => dto.Category, () => Fixture.Create<CategoryType>().ToString())

.Create();

new InitialDetailsValidator().TestValidate(dtoModel)

.ShouldNotHaveValidationErrorFor(dto => dto.Category);

}

[Theory]

[InlineData(0)]

[InlineData(-1)]

[InlineData(-24)]

[InlineData(-42465556)]

[InlineData(int.MinValue)]

public void GivenNumOfQuestionsIsLessThanOrEqualToZero\_ReturnValidationErrorWithNumOfQuestionsMustBeGreaterThanZero(int numOfQuestions)

{

var dtoModel = Fixture.Build<InitialDetailsDto>()

.With(dto => dto.NumOfQuestions, () => numOfQuestions)

.Create();

new InitialDetailsValidator().TestValidate(dtoModel)

.ShouldHaveValidationErrorFor(dto => dto.NumOfQuestions)

.WithErrorMessage($"'Num Of Questions' must be greater than '0'.");

}

[Fact]

public void GivenNumOfQuestionsIsGreaterThanZero\_ThenThereShouldBeNoValidationErrorForNumOfQuestions()

{

var dtoModel = Fixture.Build<InitialDetailsDto>()

.With(dto => dto.NumOfQuestions, () => Fixture.Create<int>())

.Create();

new InitialDetailsValidator().TestValidate(dtoModel)

.ShouldNotHaveValidationErrorFor(dto => dto.NumOfQuestions);

}

public void Dispose()

{

Context.Dispose();

}

}

## Code in LocationValidator Tests:

public class LocationValidatorTests : IDisposable

{

protected readonly AssignmentContext Context;

protected readonly IFixture Fixture;

public LocationValidatorTests()

{

Context = new AssignmentContext(AssignmentContextOptions.GetDbOptions());

Fixture = new Fixture();

}

[Fact]

public void GivenLatitudeIsNullOrEmpty\_ReturnValidationErrorWithLatitudeIsEmptyError()

{

var dtoModel = Fixture.Build<LocationDto>()

.With(dto => dto.Latitude, () => default)

.Create();

new LocationValidator().TestValidate(dtoModel)

.ShouldHaveValidationErrorFor(dto => dto.Latitude)

.WithErrorMessage("'Latitude' must not be empty.");

}

[Theory]

[InlineData(-91)]

[InlineData(-178)]

[InlineData(-1000)]

[InlineData(-43000)]

[InlineData(int.MinValue)]

public void GivenLatitudeIsBelowBounds\_ReturnValidationErrorWithLatitudeMustBeGreaterThanOrEqualToBoundary(decimal latitude)

{

var dtoModel = Fixture.Build<LocationDto>()

.With(dto => dto.Latitude, () => latitude)

.Create();

new LocationValidator().TestValidate(dtoModel)

.ShouldHaveValidationErrorFor(dto => dto.Latitude)

.WithErrorMessage("'Latitude' must be greater than or equal to '-90'.");

}

[Theory]

[InlineData(91)]

[InlineData(178)]

[InlineData(1000)]

[InlineData(43000)]

[InlineData(int.MaxValue)]

public void GivenLatitudeIsAboveBounds\_ReturnValidationErrorWithLatitudeMustBeLessThanOrEqualToBoundary(decimal latitude)

{

var dtoModel = Fixture.Build<LocationDto>()

.With(dto => dto.Latitude, () => latitude)

.Create();

new LocationValidator().TestValidate(dtoModel)

.ShouldHaveValidationErrorFor(dto => dto.Latitude)

.WithErrorMessage("'Latitude' must be less than or equal to '90'.");

}

[Theory]

[AutoData]

public void GivenLatitudeIsValid\_ThenThereShouldBeNoValidationErrorForLatitude([Range(-90, 90)] decimal latitude)

{

var dtoModel = Fixture.Build<LocationDto>()

.With(dto => dto.Latitude, () => latitude)

.Create();

new LocationValidator().TestValidate(dtoModel)

.ShouldNotHaveValidationErrorFor(dto => dto.Latitude);

}

[Theory]

[InlineData(-181)]

[InlineData(-200)]

[InlineData(-1000)]

[InlineData(-43000)]

[InlineData(int.MinValue)]

public void GivenLongitudeIsBelowBounds\_ReturnValidationErrorWithLongitudeMustBeGreaterThanOrEqualToBoundary(decimal longitude)

{

var dtoModel = Fixture.Build<LocationDto>()

.With(dto => dto.Longitude, () => longitude)

.Create();

new LocationValidator().TestValidate(dtoModel)

.ShouldHaveValidationErrorFor(dto => dto.Longitude)

.WithErrorMessage("'Longitude' must be greater than or equal to '-180'.");

}

[Theory]

[InlineData(181)]

[InlineData(200)]

[InlineData(1000)]

[InlineData(43000)]

[InlineData(int.MaxValue)]

public void GivenLongitudeIsAboveBounds\_ReturnValidationErrorWithLongitudeMustBeLessThanOrEqualToBoundary(decimal longitude)

{

var dtoModel = Fixture.Build<LocationDto>()

.With(dto => dto.Longitude, () => longitude)

.Create();

new LocationValidator().TestValidate(dtoModel)

.ShouldHaveValidationErrorFor(dto => dto.Longitude)

.WithErrorMessage("'Longitude' must be less than or equal to '180'.");

}

[Theory]

[AutoData]

public void GivenLongitudeIsValid\_ThenThereShouldBeNoValidationErrorForLongitude([Range(-90, 90)] decimal longitude)

{

var dtoModel = Fixture.Build<LocationDto>()

.With(dto => dto.Longitude, () => longitude)

.Create();

new LocationValidator().TestValidate(dtoModel)

.ShouldNotHaveValidationErrorFor(dto => dto.Longitude);

}

public void Dispose()

{

Context.Dispose();

}

}

## Code in FourWeekChallengeController Tests:

1. public class FourWeekChallengeControllerTests : IDisposable
2. {
3. protected readonly AssignmentContext Context;
4. protected readonly IFixture Fixture;
5. protected FourWeekChallengeControllerTests()
6. {
7. Context = new AssignmentContext(AssignmentContextOptions.GetDbOptions());
8. Fixture = new Fixture();
9. Fixture.Behaviors.Remove(new ThrowingRecursionBehavior());
10. Fixture.Behaviors.Add(new OmitOnRecursionBehavior());
11. }
12. public class GetCategoriesTests : FourWeekChallengeControllerTests
13. {
14. [Fact]
15. public void GivenThereAreCategoriesWithQuestions\_ThenReturnCategoriesWithQuestionCount()
16. {
17. var managerMock = new Mock<IFourWeekChallengeManager>();
18. var expectedValue = new Result<List<CategoryWithQuestionCount>>(
19. new List<CategoryWithQuestionCount>
20. {
21. Fixture.Build<CategoryWithQuestionCount>()
22. .With(c => c.QuestionCount,
23. () => Fixture.CreateMany<int>(5)).Create()
24. });
25. managerMock
26. .Setup(manager => manager.GetCategoriesAndQuestionCount())
27. .Returns(() => expectedValue);
28. new FourWeekChallengeController(managerMock.Object)
29. .GetCategories()
30. .Should().BeOkObjectResult()
31. .WithValueEquivalentTo(expectedValue.GetValueAndLinks());
32. }
33. }
34. public class StartGameTests : FourWeekChallengeControllerTests
35. {
36. [Theory]
37. [InlineData(-1)]
38. [InlineData(-24)]
39. [InlineData(-55645)]
40. [InlineData(-42465556)]
41. [InlineData(int.MinValue)]
42. public void GivenAppIdIsLessThanZero\_ThenReturnBadRequestErrorResult(int appId)
43. {
44. var initialDetailsDto = GetValidInitialDetailsDto();
45. var expectedValue = $"'{nameof(appId)}' cannot be less than 0.";
46. new FourWeekChallengeController(new Mock<IFourWeekChallengeManager>().Object)
47. .StartGame(appId, initialDetailsDto)
48. .Should().BeBadRequestObjectResult()
49. .WithValueEquivalentTo(expectedValue);
50. }
51. [Theory]
52. [AutoData]
53. public void GivenAddNewPlayerFailed\_ThenReturnBadRequestErrorResultWithLinks(int appId, string errorMessage)
54. {
55. var managerMock = new Mock<IFourWeekChallengeManager>();
56. var initialDetailsDto = GetValidInitialDetailsDto();
57. var expectedValue = new Result(new ErrorMessage(HttpStatusCode.BadRequest, errorMessage));
58. SetupManagerMockAddNewPlayer(managerMock, appId, initialDetailsDto, expectedValue);
59. new FourWeekChallengeController(managerMock.Object)
60. .StartGame(appId, initialDetailsDto)
61. .Should().BeBadRequestObjectResult()
62. .WithValueEquivalentTo(expectedValue.GetErrorAndLinks());
63. }
64. [Theory]
65. [AutoData]
66. public void GivenAddNewPlayerIsSuccessful\_ThenReturnSuccessWithLinks(int appId)
67. {
68. var managerMock = new Mock<IFourWeekChallengeManager>();
69. var httpContextMock = new Mock<HttpContext>();
70. var initialDetailsDto = GetValidInitialDetailsDto();
71. var expectedValue = new LinkReferencer()
72. .AddLink(FourWeekChallengeEndpoint.GetQuestionRouteWith(initialDetailsDto.Username))
73. .GetLinks();
74. SetupManagerMockAddNewPlayer(managerMock, appId, initialDetailsDto, new Result().Ok());
75. httpContextMock.SetupGet(context => context.Request.Path)
76. .Returns(() => $"/{Fixture.Create<string>()}");
77. var controller = new FourWeekChallengeController(managerMock.Object);
78. controller.ControllerContext = new ControllerContext(
79. new ActionContext(httpContextMock.Object, new RouteData(), new ControllerActionDescriptor()));
80. controller
81. .StartGame(appId, initialDetailsDto)
82. .Should().BeCreatedResult()
83. .WithValueEquivalentTo(expectedValue);
84. }
85. private static void SetupManagerMockAddNewPlayer(Mock<IFourWeekChallengeManager> managerMock,
86. int appId, InitialDetailsDto initialDetailsDto, Result returnVal)
87. {
88. managerMock
89. .Setup(manager => manager.AddNewPlayer(
90. appId,
91. initialDetailsDto.Username,
92. Enum.Parse<CategoryType>(initialDetailsDto.Category),
93. initialDetailsDto.NumOfQuestions))
94. .Returns(() => returnVal);
95. }
96. private InitialDetailsDto GetValidInitialDetailsDto()
97. {
98. return Fixture.Build<InitialDetailsDto>()
99. .With(dto => dto.Category, () => It.IsAny<CategoryType>().ToString())
100. .Create();
101. }
102. }
103. public class GetQuestionTests : FourWeekChallengeControllerTests
104. {
105. [Fact]
106. public void GivenGetCurrentQuestionDataReturnsErrorResult\_ThenReturnErrorResultWithLinks()
107. {
108. var managerMock = new Mock<IFourWeekChallengeManager>();
109. var username = Fixture.Create<string>();
110. var errorMessage = Fixture.Create<string>();
111. var expectedValue = new Result<Questions>(new ErrorMessage(HttpStatusCode.BadRequest, errorMessage));
113. SetupManagerMockGetCurrentQuestionData(managerMock, username, expectedValue);
114. new FourWeekChallengeController(managerMock.Object)
115. .GetQuestion(username)
116. .Should().BeBadRequestObjectResult()
117. .WithValueEquivalentTo(expectedValue.GetErrorAndLinks());
118. }
119. [Fact]
120. public void GivenCurrentQuestionDataIsNull\_ThenReturnOkResultWithLinks()
121. {
122. var managerMock = new Mock<IFourWeekChallengeManager>();
123. var username = Fixture.Create<string>();
124. var expectedValue = new LinkReferencer()
125. .AddLink(FourWeekChallengeEndpoint.EndGameRouteWith(username))
126. .GetLinks();
127. SetupManagerMockGetCurrentQuestionData(managerMock, username, new Result<Questions>((Questions)null!));
128. new FourWeekChallengeController(managerMock.Object)
129. .GetQuestion(username)
130. .Should().BeOkObjectResult()
131. .WithValueEquivalentTo(expectedValue);
132. }
133. [Theory]
134. [AutoData]
135. public void GivenCurrentQuestionTypeIsMultipleChoice\_ThenReturnOkResultWithLinks(string username)
136. {
137. var managerMock = new Mock<IFourWeekChallengeManager>();
138. var result = new Result<Questions>(
139. Fixture.Build<Questions>()
140. .With(questions => questions.QuestionType, () => QuestionType.MultipleChoice)
141. .Create());
142. var resultData = result.Unwrap();
143. var expectedValue = new Result<QuestionAndAnswersDto>(new QuestionAndAnswersDto(
144. resultData.Question,
145. QuestionsExtensions.GetAnswersFromQuestionData(resultData)))
146. .AddLink("submit", FourWeekChallengeEndpoint.SubmitAnswerRouteWith(username));
147. SetupManagerMockGetCurrentQuestionData(managerMock, username, result);
148. new FourWeekChallengeController(managerMock.Object)
149. .GetQuestion(username)
150. .Should().BeOkObjectResult()
151. .WithValueEquivalentTo(expectedValue.GetValueAndLinks());
152. }
153. [Theory]
154. [InlineAutoData(QuestionType.Text)]
155. public void GivenCurrentQuestionTypeIsNotMultipleChoiceAndNotPicture\_ThenReturnOkResultWithLinks(QuestionType questionType, string username)
156. {
157. var managerMock = new Mock<IFourWeekChallengeManager>();
158. var result = new Result<Questions>(
159. Fixture.Build<Questions>()
160. .With(questions => questions.QuestionType, () => questionType)
161. .Create());
162. var expectedValue = new Result<QuestionDto>(new QuestionDto(
163. result.Unwrap().Question))
164. .AddLink("submit", FourWeekChallengeEndpoint.SubmitAnswerRouteWith(username));
165. SetupManagerMockGetCurrentQuestionData(managerMock, username, result);
166. new FourWeekChallengeController(managerMock.Object)
167. .GetQuestion(username)
168. .Should().BeOkObjectResult()
169. .WithValueEquivalentTo(expectedValue.GetValueAndLinks());
170. }
171. [Theory]
172. [AutoData]
173. public void GivenCurrentQuestionTypeIsPicture\_ThenReturnOkResultWithLinks(string username)
174. {
175. var managerMock = new Mock<IFourWeekChallengeManager>();
176. var result = new Result<Questions>(
177. Fixture.Build<Questions>()
178. .With(questions => questions.QuestionType, () => QuestionType.Picture)
179. .Create());
180. var expectedValue = new Result<QuestionDto>(new QuestionDto(
181. result.Unwrap().Question))
182. .AddLink("submit", FourWeekChallengeEndpoint.SubmitPictureAnswerRouteWith(username));
183. SetupManagerMockGetCurrentQuestionData(managerMock, username, result);
184. new FourWeekChallengeController(managerMock.Object)
185. .GetQuestion(username)
186. .Should().BeOkObjectResult()
187. .WithValueEquivalentTo(expectedValue.GetValueAndLinks());
188. }
189. private static void SetupManagerMockGetCurrentQuestionData(
190. Mock<IFourWeekChallengeManager> managerMock, string username, Result<Questions> expectedValue)
191. {
192. managerMock.Setup(manager => manager.GetCurrentQuestionData(username))
193. .Returns(() => expectedValue);
194. }
195. }
196. public class SubmitAnswerTests : FourWeekChallengeControllerTests
197. {
198. [Theory]
199. [AutoData]
200. public void GivenSubmitAnswerReturnsErrorResult\_ThenReturnBadRequestResultWithErrorAndLinks
201. (string username, string answer, string errorMessage)
202. {
203. var managerMock = new Mock<IFourWeekChallengeManager>();
204. var expectedValue = new Result<bool>(new ErrorMessage(HttpStatusCode.BadRequest, errorMessage));
205. SetupManagerMockSubmitAnswer(username, answer, managerMock, expectedValue);
206. new FourWeekChallengeController(managerMock.Object)
207. .SubmitAnswer(username, new AnswerDto(answer))
208. .Should().BeBadRequestObjectResult()
209. .WithValueEquivalentTo(expectedValue.GetErrorAndLinks());
210. }
211. [Theory]
212. [AutoData]
213. public void GivenSubmitAnswerReturnsSuccessResult\_ThenReturnOkRequestResultWithValueAndLinks(
214. string username, string answer, bool resultData)
215. {
216. var managerMock = new Mock<IFourWeekChallengeManager>();
217. var expectedValue = new Result<bool>(resultData);
218. SetupManagerMockSubmitAnswer(username, answer, managerMock, expectedValue);
219. new FourWeekChallengeController(managerMock.Object)
220. .SubmitAnswer(username, new AnswerDto(answer))
221. .Should().BeOkObjectResult()
222. .WithValueEquivalentTo(expectedValue.GetValueAndLinks());
223. }
224. private static void SetupManagerMockSubmitAnswer(string username, string answer,
225. Mock<IFourWeekChallengeManager> managerMock, Result<bool> expectedValue)
226. {
227. managerMock.Setup(manager => manager.SubmitAnswer(username, answer))
228. .Returns(() => expectedValue);
229. }
230. }
231. public class SubmitPictureAnswerTests : FourWeekChallengeControllerTests
232. {
233. [Theory]
234. [AutoData]
235. public void GivenSubmitPictureAnswerReturnsErrorResult\_ThenReturnBadRequestResultWithErrorAndLinks
236. (string username, string errorMessage)
237. {
238. var managerMock = new Mock<IFourWeekChallengeManager>();
239. var expectedValue = new Result<bool>(new ErrorMessage(HttpStatusCode.BadRequest, errorMessage));
240. SetupManagerMockSubmitPictureAnswer(username, managerMock, expectedValue);
241. new FourWeekChallengeController(managerMock.Object)
242. .SubmitPictureAnswer(username, new Mock<IFormFile>().Object)
243. .Should().BeBadRequestObjectResult()
244. .WithValueEquivalentTo(expectedValue.GetErrorAndLinks());
245. }
246. [Theory]
247. [AutoData]
248. public void GivenSubmitPictureAnswerReturnsSuccessResult\_ThenReturnOkRequestResultWithValueAndLinks(
249. string username, bool resultData)
250. {
251. var managerMock = new Mock<IFourWeekChallengeManager>();
252. var expectedValue = new Result<bool>(resultData);
253. SetupManagerMockSubmitPictureAnswer(username, managerMock, expectedValue);
254. new FourWeekChallengeController(managerMock.Object)
255. .SubmitPictureAnswer(username, new Mock<IFormFile>().Object)
256. .Should().BeOkObjectResult()
257. .WithValueEquivalentTo(expectedValue.GetValueAndLinks());
258. }
259. private static void SetupManagerMockSubmitPictureAnswer(string username,
260. Mock<IFourWeekChallengeManager> managerMock, Result<bool> expectedValue)
261. {
262. managerMock.Setup(manager => manager.SubmitPictureAnswer(username, It.IsAny<IFormFile>()))
263. .Returns(() => expectedValue);
264. }
265. }
266. public class SetUserLocationTests : FourWeekChallengeControllerTests
267. {
268. [Theory]
269. [AutoData]
270. public void GivenSetUserLocationReturnsErrorResult\_ThenReturnBadRequestResultWithErrorAndLinks(
271. string username, decimal latitude, decimal longitude, string errorMessage)
272. {
273. var managerMock = new Mock<IFourWeekChallengeManager>();
274. var expectedValue = new Result(new ErrorMessage(HttpStatusCode.BadRequest, errorMessage));
275. SetupManagerMockUpdateUserLocation(username, latitude, longitude, managerMock, expectedValue);
276. new FourWeekChallengeController(managerMock.Object)
277. .SetUserLocation(username, new LocationDto(latitude, longitude))
278. .Should().BeBadRequestObjectResult()
279. .WithValueEquivalentTo(expectedValue.GetErrorAndLinks());
280. }
281. [Theory]
282. [AutoData]
283. public void GivenSetUserLocationReturnsSuccessResult\_ThenReturnNoContent(
284. string username, decimal latitude, decimal longitude)
285. {
286. var managerMock = new Mock<IFourWeekChallengeManager>();
287. var expectedValue = new Result().Ok();
288. SetupManagerMockUpdateUserLocation(username, latitude, longitude, managerMock, expectedValue);
289. new FourWeekChallengeController(managerMock.Object)
290. .SetUserLocation(username, new LocationDto(latitude, longitude))
291. .Should().BeNoContentResult();
292. }
293. private static void SetupManagerMockUpdateUserLocation(
294. string username, decimal latitude, decimal longitude, Mock<IFourWeekChallengeManager> managerMock, Result expectedValue)
295. {
296. managerMock.Setup(manager => manager.UpdateUserLocation(username, latitude, longitude))
297. .Returns(() => expectedValue);
298. }
299. }
300. public class GetHintTests : FourWeekChallengeControllerTests
301. {
302. [Theory]
303. [AutoData]
304. public void GivenGetHintReturnsErrorResult\_ThenReturnBadRequestResultWithErrorAndLinks(string username, string errorMessage)
305. {
306. var managerMock = new Mock<IFourWeekChallengeManager>();
307. var expectedValue = new Result<string>(new ErrorMessage(HttpStatusCode.BadRequest, errorMessage));
308. SetupManagerMockGetHint(username, managerMock, expectedValue);
309. new FourWeekChallengeController(managerMock.Object)
310. .GetHint(username)
311. .Should().BeBadRequestObjectResult()
312. .WithValueEquivalentTo(expectedValue.GetErrorAndLinks());
313. }
314. [Theory]
315. [AutoData]
316. public void GivenGetHintReturnsSuccessResult\_ThenReturnOkRequestResultWithValueAndLinks(string username, string resultData)
317. {
318. var managerMock = new Mock<IFourWeekChallengeManager>();
319. var expectedValue = new Result<string>(resultData);
320. SetupManagerMockGetHint(username, managerMock, expectedValue);
321. new FourWeekChallengeController(managerMock.Object)
322. .GetHint(username)
323. .Should().BeOkObjectResult()
324. .WithValueEquivalentTo(expectedValue.GetValueAndLinks());
325. }
326. private static void SetupManagerMockGetHint(string username, Mock<IFourWeekChallengeManager> managerMock, Result<string> expectedValue)
327. {
328. managerMock.Setup(manager => manager.GetHintFromQuestion(username))
329. .Returns(() => expectedValue);
330. }
331. }
332. public class GetLocationHintTests : FourWeekChallengeControllerTests
333. {
334. [Theory]
335. [AutoData]
336. public void GivenGetLocationHintReturnsErrorResult\_ThenReturnBadRequestResultWithErrorAndLinks(string username, string errorMessage)
337. {
338. var managerMock = new Mock<IFourWeekChallengeManager>();
339. var expectedValue = new Result<string>(new ErrorMessage(HttpStatusCode.BadRequest, errorMessage));
340. SetupManagerMockGetLocationHint(username, managerMock, expectedValue);
341. new FourWeekChallengeController(managerMock.Object)
342. .GetLocationHint(username)
343. .Should().BeBadRequestObjectResult()
344. .WithValueEquivalentTo(expectedValue.GetErrorAndLinks());
345. }
346. [Theory]
347. [AutoData]
348. public void GivenGetLocationHintReturnsSuccessResult\_ThenReturnOkRequestResultWithValueAndLinks(string username, string resultData)
349. {
350. var managerMock = new Mock<IFourWeekChallengeManager>();
351. var expectedValue = new Result<string>(resultData);
352. SetupManagerMockGetLocationHint(username, managerMock, expectedValue);
353. new FourWeekChallengeController(managerMock.Object)
354. .GetLocationHint(username)
355. .Should().BeOkObjectResult()
356. .WithValueEquivalentTo(expectedValue.GetValueAndLinks());
357. }
358. private static void SetupManagerMockGetLocationHint(string username, Mock<IFourWeekChallengeManager> managerMock, Result<string> expectedValue)
359. {
360. managerMock.Setup(manager => manager.GetLocationHintFromQuestion(username))
361. .Returns(() => expectedValue);
362. }
363. }
364. public class EndGameTests : FourWeekChallengeControllerTests
365. {
366. [Theory]
367. [AutoData]
368. public void GivenEndGameReturnsErrorResult\_ThenReturnBadRequestResultWithErrorAndLinks(string username, string errorMessage)
369. {
370. var managerMock = new Mock<IFourWeekChallengeManager>();
371. var expectedValue = new Result(new ErrorMessage(HttpStatusCode.BadRequest, errorMessage));
372. SetupManagerMockEndGame(username, managerMock, expectedValue);
373. new FourWeekChallengeController(managerMock.Object)
374. .EndGame(username)
375. .Should().BeBadRequestObjectResult()
376. .WithValueEquivalentTo(expectedValue.GetErrorAndLinks());
377. }
378. [Theory]
379. [AutoData]
380. public void GivenEndGameReturnsSuccessResult\_ThenReturnNoContent(string username)
381. {
382. var managerMock = new Mock<IFourWeekChallengeManager>();
383. var expectedValue = new Result().Ok();
384. SetupManagerMockEndGame(username, managerMock, expectedValue);
385. new FourWeekChallengeController(managerMock.Object)
386. .EndGame(username)
387. .Should().BeOkObjectResult()
388. .WithValueEquivalentTo(new LinkReferencer()
389. .AddLink("categories", FourWeekChallengeEndpoint.GetCategories)
390. .AddLink("score", FourWeekChallengeEndpoint.GetUserScoreRouteWith(username))
391. .AddLink("highScores", FourWeekChallengeEndpoint.GetHighScoresRoute)
392. .GetLinks());
393. }
394. private static void SetupManagerMockEndGame(string username, Mock<IFourWeekChallengeManager> managerMock, Result expectedValue)
395. {
396. managerMock.Setup(manager => manager.EndGame(username))
397. .Returns(() => expectedValue);
398. }
399. }
400. public class GetUserScoreTests : FourWeekChallengeControllerTests
401. {
402. [Theory]
403. [AutoData]
404. public void GivenGetUserScoreReturnsErrorResult\_ThenReturnBadRequestResultWithErrorAndLinks(string username, string errorMessage)
405. {
406. var managerMock = new Mock<IFourWeekChallengeManager>();
407. var expectedValue = new Result<int>(new ErrorMessage(HttpStatusCode.BadRequest, errorMessage));
408. SetupManagerMockGetUserScore(username, managerMock, expectedValue);
409. new FourWeekChallengeController(managerMock.Object)
410. .GetUserScore(username)
411. .Should().BeBadRequestObjectResult()
412. .WithValueEquivalentTo(expectedValue.GetErrorAndLinks());
413. }
414. [Theory]
415. [AutoData]
416. public void GivenGetUserScoreReturnsSuccessResult\_ThenReturnOkRequestResultWithValueAndLinks(string username, int resultData)
417. {
418. var managerMock = new Mock<IFourWeekChallengeManager>();
419. var expectedValue = new Result<int>(resultData);
420. SetupManagerMockGetUserScore(username, managerMock, expectedValue);
421. new FourWeekChallengeController(managerMock.Object)
422. .GetUserScore(username)
423. .Should().BeOkObjectResult()
424. .WithValueEquivalentTo(expectedValue.GetValueAndLinks());
425. }
426. private static void SetupManagerMockGetUserScore(string username, Mock<IFourWeekChallengeManager> managerMock, Result<int> expectedValue)
427. {
428. managerMock.Setup(manager => manager.GetUserScore(username))
429. .Returns(() => expectedValue);
430. }
431. }
432. public class GetHighScoresTests : FourWeekChallengeControllerTests
433. {
434. [Theory]
435. [AutoData]
436. public void GivenGetHighScoresReturnsErrorResult\_ThenReturnBadRequestResultWithErrorAndLinks(string errorMessage)
437. {
438. var managerMock = new Mock<IFourWeekChallengeManager>();
439. var expectedValue = new Result<List<UserScore>>(new ErrorMessage(HttpStatusCode.BadRequest, errorMessage));
440. SetupManagerMockGetHighScores(managerMock, expectedValue);
441. new FourWeekChallengeController(managerMock.Object)
442. .GetHighScores()
443. .Should().BeBadRequestObjectResult()
444. .WithValueEquivalentTo(expectedValue.GetErrorAndLinks());
445. }
446. [Theory]
447. [AutoData]
448. public void GivenGetHighScoresReturnsSuccessResult\_ThenReturnOkRequestResultWithValueAndLinks(List<UserScore> resultData)
449. {
450. var managerMock = new Mock<IFourWeekChallengeManager>();
451. var expectedValue = new Result<List<UserScore>>(resultData);
452. SetupManagerMockGetHighScores(managerMock, expectedValue);
453. new FourWeekChallengeController(managerMock.Object)
454. .GetHighScores()
455. .Should().BeOkObjectResult()
456. .WithValueEquivalentTo(expectedValue.GetValueAndLinks());
457. }
458. private static void SetupManagerMockGetHighScores(Mock<IFourWeekChallengeManager> managerMock, Result<List<UserScore>> expectedValue)
459. {
460. managerMock.Setup(manager => manager.GetHighScores())
461. .Returns(() => expectedValue);
462. }
463. }
464. public void Dispose()
465. {
466. Context.Dispose();
467. }
468. }

## Code for Logic:

1. public class FourWeekChallengeManagerTests
2. {
3. protected readonly IFixture Fixture;
4. protected FourWeekChallengeManagerTests()
5. {
6. Fixture = new Fixture();
7. Fixture.Behaviors.Remove(new ThrowingRecursionBehavior());
8. Fixture.Behaviors.Add(new OmitOnRecursionBehavior());
9. }
10. public class GetCategoriesAndQuestionCountTests : FourWeekChallengeManagerTests
11. {
12. [Theory]
13. [AutoData]
14. public void ThenReturnsCategoriesWithQuestionCount(int questionCount)
15. {
16. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
17. SetupMockCountQuestionsFromCategory(repositoryMock, questionCount);
18. var expectedValue = new Result<List<CategoryWithQuestionCount>>(
19. Enum.GetNames(typeof(CategoryType))
20. .Select(categoryString => (CategoryType) Enum.Parse(typeof(CategoryType), categoryString))
21. .Select(category => new CategoryWithQuestionCount(
22. category.ToString(),
23. GetQuestionCountInIncrements(questionCount, 5)))
24. .ToList()).AddLink(FourWeekChallengeEndpoint.StartRoute);
25. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
26. .GetCategoriesAndQuestionCount()
27. .Should().BeResultEquivalentTo(expectedValue);
28. }
29. private static void SetupMockCountQuestionsFromCategory(Mock<IGlobalRepository> repositoryMock,
30. int questionCount)
31. {
32. repositoryMock.Setup(repository =>
33. repository.Questions.CountQuestionsFromCategory(It.IsAny<CategoryType>()))
34. .Returns(questionCount);
35. }
36. private IEnumerable<int> GetQuestionCountInIncrements(int questionCount, int increments)
37. {
38. for (var i = 0; i < questionCount / increments; i++)
39. {
40. yield return increments \* (i + 1);
41. }
42. }
43. }
44. public class UpdateUserLocationTests : FourWeekChallengeManagerTests
45. {
46. [Theory]
47. [AutoData]
48. public void GivenUsernameDoesNotExist\_ThenReturnMemberDoesNotExistErrorResult(string username,
49. decimal latitude, decimal longitude)
50. {
51. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
52. SetupMockGetMemberOrNull(repositoryMock, username, null);
53. var expectedValue = new Result(new ErrorMessage(HttpStatusCode.NotFound,
54. $"Member with username '{username}' does not exist."))
55. .AddLink(FourWeekChallengeEndpoint.GetCategories);
56. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
57. .UpdateUserLocation(username, latitude, longitude)
58. .Should().BeResultEquivalentTo(expectedValue);
59. }
60. [Theory]
61. [AutoData]
62. public void GivenMemberDoesNotHaveAnInitialLocationSet\_ThenSetMemberLocationAndReturnSuccessResult(
63. int appId, string username, decimal latitude, decimal longitude, int locationId)
64. {
65. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
66. var fakeMember = new Members()
67. {
68. AppId = appId,
69. Username = username,
70. };
71. var newLocationWithId = Fixture.Build<Locations>()
72. .With(locations => locations.LocationId, locationId)
73. .Create();
74. SetupMockGetMemberOrNull(repositoryMock, username, fakeMember);
75. SetUpMockAddLocation(repositoryMock);
76. SetupMockGetLocationByLocation(repositoryMock, newLocationWithId);
77. SetupMockSaveChanges(repositoryMock);
78. var expectedValue = new Result().Ok();
79. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
80. .UpdateUserLocation(username, latitude, longitude)
81. .Should().BeResultEquivalentTo(expectedValue);
82. repositoryMock.Verify(repository => repository.SaveChanges(), Times.Exactly(2));
83. }
84. [Theory]
85. [AutoData]
86. public void GivenMemberHasAnInitialLocationSet\_ThenUpdateMemberLocationAndReturnSuccessResult(
87. int appId, string username, decimal latitude, decimal longitude, int locationId)
88. {
89. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
90. var fakeMember = new Members()
91. {
92. AppId = appId,
93. Username = username,
94. LocationId = locationId
95. };
96. var newLocationWithId = Fixture.Build<Locations>()
97. .With(locations => locations.LocationId, locationId)
98. .Create();
99. SetupMockGetMemberOrNull(repositoryMock, username, fakeMember);
100. SetUpMockGetLocationByLocationId(repositoryMock, locationId, newLocationWithId);
101. SetupMockSaveChanges(repositoryMock);
102. var expectedValue = new Result().Ok();
103. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
104. .UpdateUserLocation(username, latitude, longitude)
105. .Should().BeResultEquivalentTo(expectedValue);
106. repositoryMock.Verify(repository => repository.SaveChanges(), Times.Once);
107. }
108. private static void SetupMockGetLocationByLocation(Mock<IGlobalRepository> repositoryMock, Locations newLocationWithId)
109. {
110. repositoryMock.Setup(repository => repository.Locations.GetLocationByLocation(
111. It.IsAny<Locations>()))
112. .Returns(newLocationWithId);
113. }
114. private static void SetUpMockGetLocationByLocationId(Mock<IGlobalRepository> repositoryMock, int locationId,
115. Locations newLocationWithId)
116. {
117. repositoryMock.Setup(repository => repository.Locations.GetLocationByLocationId(
118. locationId))
119. .Returns(newLocationWithId);
120. }
121. private static void SetUpMockAddLocation(Mock<IGlobalRepository> repositoryMock)
122. {
123. repositoryMock.Setup(repository => repository.Locations.Add(It.IsAny<Locations>()));
124. }
125. }
126. public class GetHintFromQuestionTests : FourWeekChallengeManagerTests
127. {
128. [Theory]
129. [AutoData]
130. public void GivenUsernameDoesNotExist\_ThenReturnMemberDoesNotExistErrorResult(string username)
131. {
132. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
133. SetupMockGetMemberOrNull(repositoryMock, username, null);
134. var expectedValue = new Result(new ErrorMessage(HttpStatusCode.NotFound,
135. $"Member with username '{username}' does not exist."))
136. .AddLink(FourWeekChallengeEndpoint.GetCategories);
137. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
138. .GetHintFromQuestion(username)
139. .Should().BeResultEquivalentTo(expectedValue);
140. }
141. [Fact]
142. public void GivenCurrentQuestionNumberIsOutsideTheRangeOfIndexesInQuestionIds\_ThenReturnIndexOutOfRangeErrorResult()
143. {
144. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
145. var fakeMember = Fixture.Build<Members>()
146. .With(members => members.QuestionIds, () => BuildQuestionIds(10))
147. .Create();
148. fakeMember.CurrentQuestionNumber = fakeMember.QuestionIds.Split(",").Length + 1;
149. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
150. var expectedValue = new Result<string>(new ErrorMessage(HttpStatusCode.InternalServerError,
151. $"Index '{nameof(fakeMember.CurrentQuestionNumber)}' was outside the range for the number of questionIds the member has."));
152. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
153. .GetHintFromQuestion(fakeMember.Username)
154. .Should().BeResultEquivalentTo(expectedValue);
155. }
156. [Fact]
157. public void GivenCurrentQuestionNumberIsEqualToQuestionIdsLength\_ThenReturnGameHasEndedErrorResult()
158. {
159. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
160. var fakeMember = Fixture.Build<Members>()
161. .With(members => members.QuestionIds, BuildQuestionIds(10))
162. .Create();
163. fakeMember.CurrentQuestionNumber = fakeMember.QuestionIds.Split(",").Length;
164. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
165. var expectedValue = new Result<string>(new ErrorMessage(HttpStatusCode.NotFound, $"Game has ended for member with name '{fakeMember.Username}'."))
166. .AddLink(FourWeekChallengeEndpoint.EndGameRouteWith(fakeMember.Username));
167. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
168. .GetHintFromQuestion(fakeMember.Username)
169. .Should().BeResultEquivalentTo(expectedValue);
170. }
171. [Fact]
172. public void GivenComplexQuestionDataIsNullAndQuestionWithQuestionIdDoesNotExist\_ThenReturnQuestionDoesNotExistErrorResult()
173. {
174. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
175. var questionIds = BuildQuestionIds(10);
176. var fakeMember = Fixture.Build<Members>()
177. .With(members => members.QuestionIds, questionIds)
178. .With(members => members.CurrentQuestionNumber, 0)
179. .Create();
180. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMember.CurrentQuestionNumber]);
181. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
182. SetupMockGetComplexQuestion(repositoryMock, currentQuestionId, null);
183. SetupMockDoesQuestionExist(repositoryMock, currentQuestionId, false);
184. var expectedValue = new Result<string>(new ErrorMessage(HttpStatusCode.NotFound,
185. $"Question with ID '{currentQuestionId}' does not exist in database."))
186. .AddLink(FourWeekChallengeEndpoint.GetQuestionRouteWith(fakeMember.Username));
187. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
188. .GetHintFromQuestion(fakeMember.Username)
189. .Should().BeResultEquivalentTo(expectedValue);
190. }
191. [Fact]
192. public void GivenComplexQuestionDataIsNullAndQuestionWithQuestionIdDoesExist\_ThenReturnThisQuestionIsNotComplexErrorResult()
193. {
194. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
195. var questionIds = BuildQuestionIds(10);
196. var fakeMember = Fixture.Build<Members>()
197. .With(members => members.QuestionIds, questionIds)
198. .With(members => members.CurrentQuestionNumber, 0)
199. .Create();
200. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMember.CurrentQuestionNumber]);
201. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
202. SetupMockGetComplexQuestion(repositoryMock, currentQuestionId, null);
203. SetupMockDoesQuestionExist(repositoryMock, currentQuestionId, true);
204. var expectedValue = new Result<string>(new ErrorMessage(HttpStatusCode.BadRequest,
205. $"This question is not a complex question. Cannot provide additional details for this question."))
206. .AddLink(FourWeekChallengeEndpoint.GetQuestionRouteWith(fakeMember.Username));
207. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
208. .GetHintFromQuestion(fakeMember.Username)
209. .Should().BeResultEquivalentTo(expectedValue);
210. }
211. [Theory]
212. [AutoData]
213. public void GivenComplexQuestionData\_ThenReturnComplexQuestionHintInSuccessResult(bool questionExists)
214. {
215. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
216. var questionIds = BuildQuestionIds(10);
217. var fakeMember = Fixture.Build<Members>()
218. .With(members => members.QuestionIds, questionIds)
219. .With(members => members.CurrentQuestionNumber, 0)
220. .Create();
221. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMember.CurrentQuestionNumber]);
222. var fakeComplexQuestion = Fixture.Create<ComplexQuestions>();
223. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
224. SetupMockGetComplexQuestion(repositoryMock, currentQuestionId, fakeComplexQuestion);
225. SetupMockDoesQuestionExist(repositoryMock, currentQuestionId, questionExists);
226. SetupMockSaveChanges(repositoryMock);
227. var expectedValue = new Result<string>(fakeComplexQuestion.Hint);
228. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
229. .GetHintFromQuestion(fakeMember.Username)
230. .Should().BeResultEquivalentTo(expectedValue);
231. repositoryMock.Verify(repository => repository.SaveChanges(), Times.Once);
232. }
233. }
234. public class AddNewPlayerTests : FourWeekChallengeManagerTests
235. {
236. [Theory]
237. [AutoData]
238. public void GivenUsernameAlreadyExists\_ThenReturnMemberWithUsernameExistsErrorResult(
239. int appId, string username, CategoryType category, int numOfQuestions)
240. {
241. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
242. SetupMockDoesUsernameExist(repositoryMock, username, true);
243. var expectedValue = new Result(new ErrorMessage(HttpStatusCode.BadRequest,
244. $"The username '{username}' already exists, try another."))
245. .AddLink(FourWeekChallengeEndpoint.StartRouteWith(appId.ToString()));
246. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
247. .AddNewPlayer(appId, username, category, numOfQuestions)
248. .Should().BeResultEquivalentTo(expectedValue);
249. }
250. [Theory]
251. [AutoData]
252. public void GivenCategoryDoesNotExist\_ThenReturnCategoryDoesNotExistErrorResult(
253. int appId, string username, int numOfQuestions)
254. {
255. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
256. var category = (CategoryType)int.MaxValue;
257. var expectedValue = new Result(new ErrorMessage(HttpStatusCode.NotFound,
258. $"Category '{category}' does not exist in the database."))
259. .AddLink(FourWeekChallengeEndpoint.StartRouteWith(appId.ToString()));
260. SetupMockDoesUsernameExist(repositoryMock, username, false);
261. SetupMockGetCategoryIdFromCategory(repositoryMock, category, null);
262. new FourWeekChallengeManager(repositoryMock.Object, new QuestionRandomiser(repositoryMock.Object))
263. .AddNewPlayer(appId, username, category, numOfQuestions)
264. .Should().BeResultEquivalentTo(expectedValue);
265. }
266. [Theory]
267. [AutoData]
268. public void GivenNumOfQuestionsRequestedIsLargerThanActualNumOfQuestions\_ThenReturnCategoryDoesNotExistErrorResult(
269. int appId, string username, CategoryType category, int categoryId, List<int> allQuestionIdsForCategory)
270. {
271. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
272. var requestedNumOfQuestions = allQuestionIdsForCategory.Count + 1;
273. var expectedValue = new Result(new ErrorMessage(HttpStatusCode.BadRequest,
274. $"{requestedNumOfQuestions} question(s) from category '{category}' were requested but this category only contains {allQuestionIdsForCategory.Count} question(s)."))
275. .AddLink(FourWeekChallengeEndpoint.StartRouteWith(appId.ToString()));
276. SetupMockDoesUsernameExist(repositoryMock, username, false);
277. SetupMockGetCategoryIdFromCategory(repositoryMock, category, categoryId);
278. SetupMockGetAllQuestionsByCategoryId(repositoryMock, categoryId, allQuestionIdsForCategory);
279. new FourWeekChallengeManager(repositoryMock.Object, new QuestionRandomiser(repositoryMock.Object))
280. .AddNewPlayer(appId, username, category, requestedNumOfQuestions)
281. .Should().BeResultEquivalentTo(expectedValue);
282. }
283. [Theory]
284. [AutoData]
285. public void GivenRandomisedQuestionIds\_ThenReturnSuccessResult(
286. int appId, string username, CategoryType category, int categoryId, List<int> allQuestionIdsForCategory)
287. {
288. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
289. var requestedNumOfQuestions = allQuestionIdsForCategory.Count;
290. var expectedValue = new Result().Ok();
291. SetupMockDoesUsernameExist(repositoryMock, username, false);
292. SetupMockGetCategoryIdFromCategory(repositoryMock, category, categoryId);
293. SetupMockGetAllQuestionsByCategoryId(repositoryMock, categoryId, allQuestionIdsForCategory);
294. SetupMockAddMember(repositoryMock);
295. SetupMockSaveChanges(repositoryMock);
296. new FourWeekChallengeManager(repositoryMock.Object, new QuestionRandomiser(repositoryMock.Object))
297. .AddNewPlayer(appId, username, category, requestedNumOfQuestions)
298. .Should().BeResultEquivalentTo(expectedValue);
299. repositoryMock.Verify(repository => repository.Members.Add(It.IsAny<Members>()), Times.Once);
300. repositoryMock.Verify(repository => repository.SaveChanges(), Times.Once);
301. }
302. private static void SetupMockAddMember(Mock<IGlobalRepository> repositoryMock)
303. {
304. repositoryMock.Setup(repository => repository.Members.Add(It.IsAny<Members>()));
305. }
306. private static void SetupMockGetAllQuestionsByCategoryId(Mock<IGlobalRepository> repositoryMock,
307. int categoryId,
308. List<int> allQuestionIdsForCategory)
309. {
310. repositoryMock.Setup(repository => repository.Questions.GetAllQuestionsByCategoryId(categoryId))
311. .Returns(allQuestionIdsForCategory);
312. }
313. private static void SetupMockGetCategoryIdFromCategory(Mock<IGlobalRepository> repositoryMock,
314. CategoryType category, int? expected)
315. {
316. repositoryMock.Setup(repository => repository.Categories.GetCategoryIdFromCategory(category))
317. .Returns(() => expected);
318. }
319. private static void SetupMockDoesUsernameExist(Mock<IGlobalRepository> repositoryMock, string username,
320. bool expected)
321. {
322. repositoryMock.Setup(repository => repository.Members.DoesUsernameExist(username))
323. .Returns(expected);
324. }
325. }
326. public class GetCurrentQuestionDataTests : FourWeekChallengeManagerTests
327. {
328. [Theory]
329. [AutoData]
330. public void GivenUsernameDoesNotExist\_ThenReturnMemberDoesNotExistErrorResult(string username)
331. {
332. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
333. SetupMockGetMemberOrNull(repositoryMock, username, null);
334. var expectedValue = new Result(new ErrorMessage(HttpStatusCode.NotFound,
335. $"Member with username '{username}' does not exist."))
336. .AddLink(FourWeekChallengeEndpoint.GetCategories);
337. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
338. .GetCurrentQuestionData(username)
339. .Should().BeResultEquivalentTo(expectedValue);
340. }
341. [Fact]
342. public void GivenCurrentQuestionNumberIsOutsideTheRangeOfIndexesInQuestionIds\_ThenReturnIndexOutOfRangeErrorResult()
343. {
344. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
345. var fakeMember = Fixture.Build<Members>()
346. .With(members => members.QuestionIds, () => BuildQuestionIds(10))
347. .Create();
348. ;
349. fakeMember.CurrentQuestionNumber = fakeMember.QuestionIds.Split(",").Length + 1;
350. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
351. var expectedValue = new Result<Questions>(new ErrorMessage(HttpStatusCode.InternalServerError,
352. $"Index '{nameof(fakeMember.CurrentQuestionNumber)}' was outside the range for the number of questionIds the member has."));
353. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
354. .GetCurrentQuestionData(fakeMember.Username)
355. .Should().BeResultEquivalentTo(expectedValue);
356. }
357. [Fact]
358. public void GivenCurrentQuestionNumberIsEqualToQuestionIdsLength\_ThenReturnGameHasEndedErrorResult()
359. {
360. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
361. var fakeMember = Fixture.Build<Members>()
362. .With(members => members.QuestionIds, BuildQuestionIds(10))
363. .Create();
364. fakeMember.CurrentQuestionNumber = fakeMember.QuestionIds.Split(",").Length;
365. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
366. var expectedValue = new Result<Questions>(new ErrorMessage(HttpStatusCode.NotFound, $"Game has ended for member with name '{fakeMember.Username}'."))
367. .AddLink(FourWeekChallengeEndpoint.EndGameRouteWith(fakeMember.Username));
368. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
369. .GetCurrentQuestionData(fakeMember.Username)
370. .Should().BeResultEquivalentTo(expectedValue);
371. }
372. [Fact]
373. public void GivenQuestionIdDoesNotMatchAQuestion\_ThenReturnQuestionWithQuestionIdDoesNotExistErrorResult()
374. {
375. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
376. var questionIds = BuildQuestionIds(10);
377. var fakeMember = Fixture.Build<Members>()
378. .With(members => members.QuestionIds, questionIds)
379. .With(members => members.CurrentQuestionNumber, 0)
380. .Create();
381. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMember.CurrentQuestionNumber]);
382. var expectedValue = new Result<Questions>(
383. new ErrorMessage(
384. HttpStatusCode.NotFound,
385. $"Question with ID '{currentQuestionId}' does not exist in database."))
386. .AddLink(FourWeekChallengeEndpoint.GetQuestionRouteWith(fakeMember.Username));
387. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
388. SetupMockGetQuestionOrNull(repositoryMock, currentQuestionId, null);
389. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
390. .GetCurrentQuestionData(fakeMember.Username)
391. .Should().BeResultEquivalentTo(expectedValue);
392. }
393. [Fact]
394. public void GivenQuestionComplexityIsSimple\_ThenReturnQuestionSuccessResult()
395. {
396. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
397. var questionIds = BuildQuestionIds(10);
398. var fakeMember = Fixture.Build<Members>()
399. .With(members => members.QuestionIds, questionIds)
400. .With(members => members.CurrentQuestionNumber, 0)
401. .Create();
402. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMember.CurrentQuestionNumber]);
403. var question = Fixture.Build<Questions>()
404. .With(question => question.Discriminator, QuestionComplexity.Simple.ToString)
405. .Create();
406. var expectedValue = new Result<Questions>(question);
407. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
408. SetupMockGetQuestionOrNull(repositoryMock, currentQuestionId, question);
409. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
410. .GetCurrentQuestionData(fakeMember.Username)
411. .Should().BeResultEquivalentTo(expectedValue);
412. }
413. [Fact]
414. public void GivenQuestionComplexityIsComplex\_ThenReturnQuestionSuccessResultWithLinks()
415. {
416. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
417. var questionIds = BuildQuestionIds(10);
418. var fakeMember = Fixture.Build<Members>()
419. .With(members => members.QuestionIds, questionIds)
420. .With(members => members.CurrentQuestionNumber, 0)
421. .Create();
422. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMember.CurrentQuestionNumber]);
423. var question = Fixture.Build<Questions>()
424. .With(question => question.Discriminator, QuestionComplexity.Complex.ToString)
425. .Create();
426. var expectedValue = new Result<Questions>(question)
427. .AddLink("hint", FourWeekChallengeEndpoint.GetHintRouteWith(fakeMember.Username))
428. .AddLink("setLocation", FourWeekChallengeEndpoint.SetUserLocationRouteWith(fakeMember.Username))
429. .AddLink("locationHint", FourWeekChallengeEndpoint.GetLocationHintRouteWith(fakeMember.Username));
430. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
431. SetupMockGetQuestionOrNull(repositoryMock, currentQuestionId, question);
432. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
433. .GetCurrentQuestionData(fakeMember.Username)
434. .Should().BeResultEquivalentTo(expectedValue);
435. }
436. }
437. public class GetLocationHintFromQuestionTests : FourWeekChallengeManagerTests
438. {
439. [Theory]
440. [AutoData]
441. public void GivenUsernameDoesNotExist\_ThenReturnMemberDoesNotExistErrorResult(string username)
442. {
443. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
444. SetupMockGetMemberOrNull(repositoryMock, username, null);
445. var expectedValue = new Result(new ErrorMessage(HttpStatusCode.NotFound,
446. $"Member with username '{username}' does not exist."))
447. .AddLink(FourWeekChallengeEndpoint.GetCategories);
448. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
449. .GetLocationHintFromQuestion(username)
450. .Should().BeResultEquivalentTo(expectedValue);
451. }
452. [Fact]
453. public void GivenCurrentQuestionNumberIsOutsideTheRangeOfIndexesInQuestionIds\_ThenReturnIndexOutOfRangeErrorResult()
454. {
455. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
456. var fakeMember = Fixture.Build<Members>()
457. .With(members => members.QuestionIds, () => BuildQuestionIds(10))
458. .Create();
459. fakeMember.CurrentQuestionNumber = fakeMember.QuestionIds.Split(",").Length + 1;
460. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
461. var expectedValue = new Result<string>(new ErrorMessage(HttpStatusCode.InternalServerError,
462. $"Index '{nameof(fakeMember.CurrentQuestionNumber)}' was outside the range for the number of questionIds the member has."));
463. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
464. .GetLocationHintFromQuestion(fakeMember.Username)
465. .Should().BeResultEquivalentTo(expectedValue);
466. }
467. [Fact]
468. public void GivenCurrentQuestionNumberIsEqualToQuestionIdsLength\_ThenReturnGameHasEndedErrorResult()
469. {
470. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
471. var fakeMember = Fixture.Build<Members>()
472. .With(members => members.QuestionIds, BuildQuestionIds(10))
473. .Create();
474. fakeMember.CurrentQuestionNumber = fakeMember.QuestionIds.Split(",").Length;
475. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
476. var expectedValue = new Result<string>(new ErrorMessage(HttpStatusCode.NotFound, $"Game has ended for member with name '{fakeMember.Username}'."))
477. .AddLink(FourWeekChallengeEndpoint.EndGameRouteWith(fakeMember.Username));
478. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
479. .GetLocationHintFromQuestion(fakeMember.Username)
480. .Should().BeResultEquivalentTo(expectedValue);
481. }
482. [Fact]
483. public void GivenComplexQuestionDataIsNullAndQuestionWithQuestionIdDoesNotExist\_ThenReturnQuestionDoesNotExistErrorResult()
484. {
485. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
486. var questionIds = BuildQuestionIds(10);
487. var fakeMember = Fixture.Build<Members>()
488. .With(members => members.QuestionIds, questionIds)
489. .With(members => members.CurrentQuestionNumber, 0)
490. .Create();
491. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMember.CurrentQuestionNumber]);
492. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
493. SetupMockGetComplexQuestion(repositoryMock, currentQuestionId, null);
494. SetupMockDoesQuestionExist(repositoryMock, currentQuestionId, false);
495. var expectedValue = new Result<string>(new ErrorMessage(HttpStatusCode.NotFound,
496. $"Question with ID '{currentQuestionId}' does not exist in database."))
497. .AddLink(FourWeekChallengeEndpoint.GetQuestionRouteWith(fakeMember.Username));
498. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
499. .GetLocationHintFromQuestion(fakeMember.Username)
500. .Should().BeResultEquivalentTo(expectedValue);
501. }
502. [Fact]
503. public void GivenComplexQuestionDataIsNullAndQuestionWithQuestionIdDoesExist\_ThenReturnThisQuestionIsNotComplexErrorResult()
504. {
505. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
506. var questionIds = BuildQuestionIds(10);
507. var fakeMember = Fixture.Build<Members>()
508. .With(members => members.QuestionIds, questionIds)
509. .With(members => members.CurrentQuestionNumber, 0)
510. .Create();
511. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMember.CurrentQuestionNumber]);
512. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
513. SetupMockGetComplexQuestion(repositoryMock, currentQuestionId, null);
514. SetupMockDoesQuestionExist(repositoryMock, currentQuestionId, true);
515. var expectedValue = new Result<string>(new ErrorMessage(HttpStatusCode.BadRequest,
516. $"This question is not a complex question. Cannot provide additional details for this question."))
517. .AddLink(FourWeekChallengeEndpoint.GetQuestionRouteWith(fakeMember.Username));
518. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
519. .GetLocationHintFromQuestion(fakeMember.Username)
520. .Should().BeResultEquivalentTo(expectedValue);
521. }
522. [Theory]
523. [AutoData]
524. public void GivenComplexQuestionData\_ThenReturnComplexQuestionHintInSuccessResult(bool questionExists)
525. {
526. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
527. var questionIds = BuildQuestionIds(10);
528. var fakeMember = Fixture.Build<Members>()
529. .With(members => members.QuestionIds, questionIds)
530. .With(members => members.CurrentQuestionNumber, 0)
531. .Create();
532. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMember.CurrentQuestionNumber]);
533. var fakeComplexQuestion = Fixture.Create<ComplexQuestions>();
534. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
535. SetupMockGetComplexQuestion(repositoryMock, currentQuestionId, fakeComplexQuestion);
536. SetupMockDoesQuestionExist(repositoryMock, currentQuestionId, questionExists);
537. SetupMockSaveChanges(repositoryMock);
538. var expectedValue = new Result<string>(fakeComplexQuestion.LocationHint);
539. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
540. .GetLocationHintFromQuestion(fakeMember.Username)
541. .Should().BeResultEquivalentTo(expectedValue);
542. repositoryMock.Verify(repository => repository.SaveChanges(), Times.Once);
543. }
544. }
545. public class EndGameTests : FourWeekChallengeManagerTests
546. {
547. [Theory]
548. [AutoData]
549. public void GivenUsernameDoesNotExist\_ThenReturnMemberDoesNotExistErrorResult(string username)
550. {
551. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
552. SetupMockGetMemberOrNull(repositoryMock, username, null);
553. var expectedValue = new Result(new ErrorMessage(HttpStatusCode.NotFound,
554. $"Member with username '{username}' does not exist."))
555. .AddLink(FourWeekChallengeEndpoint.GetCategories);
556. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
557. .EndGame(username)
558. .Should().BeResultEquivalentTo(expectedValue);
559. }
560. [Fact]
561. public void GivenGameHasNotEnded\_ThenReturnGameHasNotEnededErrorResult()
562. {
563. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
564. var fakeMember = Fixture.Build<Members>()
565. .With(members => members.QuestionIds, BuildQuestionIds(10))
566. .With(members => members.CurrentQuestionNumber, 0)
567. .Create();
568. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
569. var expectedValue = new Result(new ErrorMessage(HttpStatusCode.BadRequest,
570. $"Game has not ended for member with name '{fakeMember.Username}.'"))
571. .AddLink(FourWeekChallengeEndpoint.GetQuestionRouteWith(fakeMember.Username));
572. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
573. .EndGame(fakeMember.Username)
574. .Should().BeResultEquivalentTo(expectedValue);
575. }
576. [Fact]
577. public void GivenGameHaEnded\_ThenReturnSuccessResult()
578. {
579. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
580. var fakeMember = Fixture.Build<Members>()
581. .With(members => members.QuestionIds, BuildQuestionIds(10))
582. .With(members => members.CurrentQuestionNumber, 10)
583. .Create();
584. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
585. var expectedValue = new Result().Ok();
586. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
587. .EndGame(fakeMember.Username)
588. .Should().BeResultEquivalentTo(expectedValue);
589. }
590. }
591. public class GetUserScoreTests : FourWeekChallengeManagerTests
592. {
593. [Theory]
594. [AutoData]
595. public void GivenUsernameDoesNotExist\_ThenReturnMemberDoesNotExistErrorResult(string username)
596. {
597. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
598. SetupMockGetMemberOrNull(repositoryMock, username, It.IsAny<Members>());
599. var expectedValue = new Result<int>(new ErrorMessage(HttpStatusCode.NotFound,
600. $"Member with username '{username}' does not exist."))
601. .AddLink(FourWeekChallengeEndpoint.GetCategories);
602. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
603. .GetUserScore(username)
604. .Should().BeResultEquivalentTo(expectedValue);
605. }
606. [Fact]
607. public void GivenMemberExists\_ThenReturnUserScoreInSuccessResult()
608. {
609. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
610. var fakeMember = Fixture.Create<Members>();
611. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
612. var expectedValue = new Result<int>(fakeMember.UserScore)
613. .AddLink("category", FourWeekChallengeEndpoint.GetCategories)
614. .AddLink("highScore", FourWeekChallengeEndpoint.GetHighScoresRoute);
615. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
616. .GetUserScore(fakeMember.Username)
617. .Should().BeResultEquivalentTo(expectedValue);
618. }
619. }
620. public class GetHighScoresTests : FourWeekChallengeManagerTests
621. {
622. [Fact]
623. public void GivenThereAreNoMembersThatCurrentlyExist\_ThenReturnNoMembersExistErrorResult()
624. {
625. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
626. SetupMockAnyMembersExist(repositoryMock, false);
627. var expectedValue = new Result<List<UserScore>>(
628. new ErrorMessage(HttpStatusCode.BadRequest, "No members currently exist."))
629. .AddLink("categories", FourWeekChallengeEndpoint.GetCategories);
630. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
631. .GetHighScores()
632. .Should().BeResultEquivalentTo(expectedValue);
633. }
634. [Fact]
635. public void GivenMemberDoExist\_ThenReturnUserScoreInSuccessResult()
636. {
637. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
638. var fakeMember = Fixture.Create<Members>();
639. var userScores = Fixture.CreateMany<UserScore>().ToList();
640. SetupMockAnyMembersExist(repositoryMock, true);
641. SetupMockGetUserScoreInDescendingOrder(repositoryMock, userScores);
642. var expectedValue = new Result<List<UserScore>>(userScores)
643. .AddLink("categories", FourWeekChallengeEndpoint.GetCategories);
644. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
645. .GetHighScores()
646. .Should().BeResultEquivalentTo(expectedValue);
647. }
648. private static void SetupMockGetUserScoreInDescendingOrder(Mock<IGlobalRepository> repositoryMock,
649. List<UserScore> userScores)
650. {
651. repositoryMock.Setup(repository => repository.Members.GetUserScoreInDescendingOrder())
652. .Returns(userScores);
653. }
654. private static void SetupMockAnyMembersExist(Mock<IGlobalRepository> repositoryMock, bool expected)
655. {
656. repositoryMock.Setup(repository => repository.Members.Any())
657. .Returns(expected);
658. }
659. }
660. public class SubmitPictureAnswerTests : FourWeekChallengeManagerTests
661. {
662. [Theory]
663. [AutoData]
664. public void GivenFileTypeIsNotAccepted\_ThenReturnFileTypeNotAcceptedErrorResult(string username,
665. string fileName, string fileExtension)
666. {
667. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
668. var fileMock = Fixture.Create<Mock<IFormFile>>();
669. fileMock.SetupGet(file => file.FileName)
670. .Returns($"{fileName}.{fileExtension}");
671. var expectedValue = new Result<bool>(new ErrorMessage(HttpStatusCode.UnprocessableEntity,
672. $"File types are limited to '.jpg','.jpeg' and '.png'. The uploaded file type was .{fileExtension}"))
673. .AddLink(FourWeekChallengeEndpoint.SubmitPictureAnswerRouteWith(username));
674. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
675. .SubmitPictureAnswer(username, fileMock.Object)
676. .Should().BeResultEquivalentTo(expectedValue);
677. }
678. [Theory]
679. [InlineAutoData("jpg")]
680. [InlineAutoData("jpeg")]
681. [InlineAutoData("png")]
682. public void GivenFileTypeIsAccepted\_WhenSubmitAnswerReturnsSuccessResult\_ThenReturnSuccessResult(string fileExtension,
683. string username, string fileName)
684. {
685. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
686. var fileMock = Fixture.Create<Mock<IFormFile>>();
687. var expectedValue = new Result<bool>(false)
688. .AddLink(FourWeekChallengeEndpoint.GetQuestionRouteWith(username));
689. SetupGetMockOnFileName(fileExtension, fileName, fileMock);
690. SetupMockForceSubmitAnswerForSuccess(repositoryMock, username);
691. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
692. .SubmitPictureAnswer(username, fileMock.Object)
693. .Should().BeResultEquivalentTo(expectedValue);
694. }
695. //Purely here to result in any form of success in the submit answer method since it is not possible to mock it.
696. //Unit tests in this class is for SubmitPictureAnswer method, not SubmitAnswer method.
697. private void SetupMockForceSubmitAnswerForSuccess(Mock<IGlobalRepository> repositoryMock, string username)
698. {
699. var member = Fixture.Build<Members>()
700. .With(members => members.CurrentQuestionNumber, 0)
701. .With(members => members.QuestionIds, BuildQuestionIds(10))
702. .Create();
703. var question = Fixture.Build<Questions>()
704. .With(questions => questions.Discriminator, QuestionComplexity.Simple.ToString)
705. .Create();
706. var currentQuestionId = int.Parse(member.QuestionIds.Split(",")[member.CurrentQuestionNumber]);
707. SetupMockGetMemberOrNull(repositoryMock, username, member);
708. SetupMockGetQuestionOrNull(repositoryMock, currentQuestionId, question);
709. SetupMockGetComplexQuestion(repositoryMock, currentQuestionId, null);
710. }
711. private static void SetupGetMockOnFileName(string fileExtension, string fileName, Mock<IFormFile> fileMock)
712. {
713. fileMock.SetupGet(file => file.FileName)
714. .Returns($"{fileName}.{fileExtension}");
715. }
716. }
717. public class SubmitAnswerTests : FourWeekChallengeManagerTests
718. {
719. [Theory]
720. [AutoData]
721. public void GivenUsernameDoesNotExist\_ThenReturnMemberDoesNotExistErrorResult(string username,
722. string answer)
723. {
724. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
725. SetupMockGetMemberOrNull(repositoryMock, username, null);
726. var expectedValue = new Result(new ErrorMessage(HttpStatusCode.NotFound,
727. $"Member with username '{username}' does not exist."))
728. .AddLink(FourWeekChallengeEndpoint.GetCategories);
729. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
730. .SubmitAnswer(username, answer)
731. .Should().BeResultEquivalentTo(expectedValue);
732. }
733. [Theory]
734. [AutoData]
735. public void GivenCurrentQuestionNumberIsOutsideTheRangeOfIndexesInQuestionIds\_ThenReturnIndexOutOfRangeErrorResult(string answer)
736. {
737. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
738. var fakeMember = Fixture.Build<Members>()
739. .With(members => members.QuestionIds, () => BuildQuestionIds(10))
740. .Create();
741. ;
742. fakeMember.CurrentQuestionNumber = fakeMember.QuestionIds.Split(",").Length + 1;
743. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
744. var expectedValue = new Result<bool>(new ErrorMessage(HttpStatusCode.InternalServerError,
745. $"Index '{nameof(fakeMember.CurrentQuestionNumber)}' was outside the range for the number of questionIds the member has."));
746. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
747. .SubmitAnswer(fakeMember.Username, answer)
748. .Should().BeResultEquivalentTo(expectedValue);
749. }
750. [Theory]
751. [AutoData]
752. public void GivenCurrentQuestionNumberIsEqualToQuestionIdsLength\_ThenReturnGameHasEndedErrorResult(string answer)
753. {
754. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
755. var fakeMember = Fixture.Build<Members>()
756. .With(members => members.QuestionIds, BuildQuestionIds(10))
757. .Create();
758. fakeMember.CurrentQuestionNumber = fakeMember.QuestionIds.Split(",").Length;
759. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
760. var expectedValue = new Result<bool>(new ErrorMessage(HttpStatusCode.NotFound, $"Game has ended for member with name '{fakeMember.Username}'."))
761. .AddLink(FourWeekChallengeEndpoint.EndGameRouteWith(fakeMember.Username));
762. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
763. .SubmitAnswer(fakeMember.Username, answer)
764. .Should().BeResultEquivalentTo(expectedValue);
765. }
766. [Theory]
767. [AutoData]
768. public void GivenQuestionIdDoesNotMatchAQuestion\_ThenReturnQuestionWithQuestionIdDoesNotExistErrorResult(string answer)
769. {
770. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
771. var questionIds = BuildQuestionIds(10);
772. var fakeMember = Fixture.Build<Members>()
773. .With(members => members.QuestionIds, questionIds)
774. .With(members => members.CurrentQuestionNumber, 0)
775. .Create();
776. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMember.CurrentQuestionNumber]);
777. var expectedValue = new Result<bool>(
778. new ErrorMessage(
779. HttpStatusCode.NotFound,
780. $"Question with ID '{currentQuestionId}' does not exist in database."))
781. .AddLink(FourWeekChallengeEndpoint.GetQuestionRouteWith(fakeMember.Username));
782. SetupMockGetMemberOrNull(repositoryMock, fakeMember.Username, fakeMember);
783. SetupMockGetQuestionOrNull(repositoryMock, currentQuestionId, null);
784. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
785. .SubmitAnswer(fakeMember.Username, answer)
786. .Should().BeResultEquivalentTo(expectedValue);
787. }
788. [Theory]
789. [AutoData]
790. public void GivenAnswerIsIncorrect\_ThenReturnSuccessResultWithoutIncrementingQuestionNumberOrUserScore(
791. string username, string answer)
792. {
793. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
794. var fakeQuestion = Fixture.Build<Questions>()
795. .With(question => question.Discriminator, QuestionComplexity.Simple.ToString)
796. .Create();
797. var questionIds = BuildQuestionIds(10);
798. var fakeMemberMock = Fixture.Create<Mock<Members>>();
799. var fakeMemberObj = fakeMemberMock.Object;
800. fakeMemberObj.Username = username;
801. fakeMemberObj.QuestionIds = questionIds;
802. fakeMemberObj.CurrentQuestionNumber = 0;
803. var oldQuestionNumber = fakeMemberObj.CurrentQuestionNumber;
804. var oldUserScore = fakeMemberObj.UserScore;
805. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMemberObj.CurrentQuestionNumber]);
806. SetupMockGetMemberOrNull(repositoryMock, fakeMemberObj.Username, fakeMemberObj);
807. SetupMockGetQuestionOrNull(repositoryMock, currentQuestionId, fakeQuestion);
808. var expectedValue = new Result<bool>(false)
809. .AddLink(FourWeekChallengeEndpoint.GetQuestionRouteWith(fakeMemberObj.Username));
810. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
811. .SubmitAnswer(fakeMemberObj.Username, answer)
812. .Should().BeResultEquivalentTo(expectedValue);
813. repositoryMock.Verify(repository => repository.SaveChanges(), Times.Never);
814. Assert.True(oldQuestionNumber == fakeMemberObj.CurrentQuestionNumber);
815. Assert.True(oldUserScore == fakeMemberObj.UserScore);
816. }
817. [Theory]
818. [InlineAutoData("pass", QuestionComplexity.Simple)]
819. [InlineAutoData("PaSs", QuestionComplexity.Simple)]
820. [InlineAutoData("PASS", QuestionComplexity.Simple)]
821. [InlineAutoData("pass", QuestionComplexity.Complex)]
822. [InlineAutoData("PaSs", QuestionComplexity.Complex)]
823. [InlineAutoData("PASS", QuestionComplexity.Complex)]
824. public void GivenAnswerIsPass\_ThenReturnSuccessResultAndIncrementCurrentQuestionNumber(
825. string answer, QuestionComplexity complexity, string username)
826. {
827. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
828. var fakeQuestion = Fixture.Build<Questions>()
829. .With(question => question.Discriminator, complexity.ToString)
830. .Create();
831. var fakeComplexQuestion = Fixture.Build<ComplexQuestions>()
832. .With(complexQuestion => complexQuestion.Discriminator, complexity.ToString)
833. .Create();
834. var questionIds = BuildQuestionIds(10);
835. var fakeMemberMock = Fixture.Create<Mock<Members>>();
836. var fakeMemberObj = fakeMemberMock.Object;
837. fakeMemberObj.Username = username;
838. fakeMemberObj.QuestionIds = questionIds;
839. fakeMemberObj.CurrentQuestionNumber = 0;
840. var oldQuestionNumber = fakeMemberObj.CurrentQuestionNumber;
841. var oldUserScore = fakeMemberObj.UserScore;
842. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMemberObj.CurrentQuestionNumber]);
843. SetupMockGetMemberOrNull(repositoryMock, fakeMemberObj.Username, fakeMemberObj);
844. SetupMockGetQuestionOrNull(repositoryMock, currentQuestionId, fakeQuestion);
845. if(complexity == QuestionComplexity.Complex) SetupMockGetComplexQuestion(repositoryMock, fakeQuestion.QuestionId, fakeComplexQuestion);
846. SetupMockSaveChanges(repositoryMock);
847. var expectedValue = new Result<bool>(true)
848. .AddLink(FourWeekChallengeEndpoint.GetQuestionRouteWith(fakeMemberObj.Username));
849. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
850. .SubmitAnswer(fakeMemberObj.Username, answer)
851. .Should().BeResultEquivalentTo(expectedValue);
852. repositoryMock.Verify(repository => repository.SaveChanges(), Times.Once);
853. Assert.True(oldQuestionNumber + 1 == fakeMemberObj.CurrentQuestionNumber);
854. Assert.True(oldUserScore == fakeMemberObj.UserScore);
855. }
856. [Theory]
857. [AutoData]
858. public void GivenQuestionComplexityIsComplexAndAnswerIsCorrectLocationIsNull\_ThenReturnErrorResultLocationNotSet(
859. string username, string answer)
860. {
861. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
862. var fakeQuestion = Fixture.Build<Questions>()
863. .With(question => question.Discriminator, QuestionComplexity.Complex.ToString)
864. .With(question => question.CorrectAnswer, answer)
865. .Create();
866. var fakeComplexQuestion = Fixture.Build<ComplexQuestions>()
867. .With(complexQuestion => complexQuestion.Discriminator, QuestionComplexity.Complex.ToString)
868. .Create();
869. var fakeLocation = (Locations)null;
870. var questionIds = BuildQuestionIds(10);
871. var fakeMemberMock = Fixture.Create<Mock<Members>>();
872. var fakeMemberObj = fakeMemberMock.Object;
873. fakeMemberObj.Username = username;
874. fakeMemberObj.QuestionIds = questionIds;
875. fakeMemberObj.CurrentQuestionNumber = 0;
876. var oldQuestionNumber = fakeMemberObj.CurrentQuestionNumber;
877. var oldUserScore = fakeMemberObj.UserScore;
878. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMemberObj.CurrentQuestionNumber]);
879. SetupMockGetMemberOrNull(repositoryMock, fakeMemberObj.Username, fakeMemberObj);
880. SetupMockGetQuestionOrNull(repositoryMock, currentQuestionId, fakeQuestion);
881. SetupMockGetComplexQuestion(repositoryMock, fakeQuestion.QuestionId, fakeComplexQuestion);
882. SetupMockGetLocationByLocationIdOrNull(repositoryMock, fakeMemberObj, fakeLocation);
883. var expectedValue = new Result<bool>(
884. new ErrorMessage(HttpStatusCode.NotFound,
885. $"Member with username '{fakeMemberObj.Username}' does not have a location set."))
886. .AddLink(FourWeekChallengeEndpoint.SetUserLocationRouteWith(fakeMemberObj.Username));
887. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
888. .SubmitAnswer(fakeMemberObj.Username, answer)
889. .Should().BeResultEquivalentTo(expectedValue);
890. repositoryMock.Verify(repository => repository.SaveChanges(), Times.Never);
891. Assert.True(oldQuestionNumber == fakeMemberObj.CurrentQuestionNumber);
892. Assert.True(oldUserScore == fakeMemberObj.UserScore);
893. }
894. [Theory]
895. [AutoData]
896. public void GivenQuestionComplexityIsComplexAndAnswerIsCorrectAndLocationIsIncorrect\_ThenReturnSuccessResultWithoutIncrementingQuestionNumberOrUserScore(
897. string username, string answer)
898. {
899. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
900. var fakeUserLocation = CreateFakeLocation();
901. var fakeQuestion = Fixture.Build<Questions>()
902. .With(question => question.Discriminator, QuestionComplexity.Complex.ToString)
903. .With(question => question.CorrectAnswer, answer)
904. .Create();
905. var fakeComplexQuestion = Fixture.Build<ComplexQuestions>()
906. .With(complexQuestion => complexQuestion.Discriminator, QuestionComplexity.Complex.ToString)
907. .With(questions => questions.Location, CreateFakeLocation())
908. .Create();
909. var questionIds = BuildQuestionIds(10);
910. var fakeMemberMock = Fixture.Create<Mock<Members>>();
911. var fakeMemberObj = fakeMemberMock.Object;
912. fakeMemberObj.Username = username;
913. fakeMemberObj.QuestionIds = questionIds;
914. fakeMemberObj.CurrentQuestionNumber = 0;
915. var oldQuestionNumber = fakeMemberObj.CurrentQuestionNumber;
916. var oldUserScore = fakeMemberObj.UserScore;
917. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMemberObj.CurrentQuestionNumber]);
918. SetupMockGetMemberOrNull(repositoryMock, fakeMemberObj.Username, fakeMemberObj);
919. SetupMockGetQuestionOrNull(repositoryMock, currentQuestionId, fakeQuestion);
920. SetupMockGetComplexQuestion(repositoryMock, fakeQuestion.QuestionId, fakeComplexQuestion);
921. SetupMockGetLocationByLocationIdOrNull(repositoryMock, fakeMemberObj, fakeUserLocation);
922. SetupMockSaveChanges(repositoryMock);
923. var expectedValue = new Result<bool>(false)
924. .AddLink(FourWeekChallengeEndpoint.GetQuestionRouteWith(fakeMemberObj.Username));
925. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
926. .SubmitAnswer(fakeMemberObj.Username, answer)
927. .Should().BeResultEquivalentTo(expectedValue);
928. repositoryMock.Verify(repository => repository.SaveChanges(), Times.Once);
929. Assert.True(oldQuestionNumber == fakeMemberObj.CurrentQuestionNumber);
930. Assert.True(oldUserScore == fakeMemberObj.UserScore);
931. }
932. [Theory]
933. [AutoData]
934. public void GivenQuestionComplexityIsComplexAndAnswerIsCorrectAndHintAsked\_ThenReturnSuccessResultWithIncrementingQuestionNumberOrUserScoreByOne(
935. string username, string answer)
936. {
937. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
938. var fakeLocation = CreateFakeLocation();
939. var fakeQuestion = Fixture.Build<Questions>()
940. .With(question => question.Discriminator, QuestionComplexity.Complex.ToString)
941. .With(question => question.CorrectAnswer, answer)
942. .Create();
943. var fakeComplexQuestion = Fixture.Build<ComplexQuestions>()
944. .With(complexQuestion => complexQuestion.Discriminator, QuestionComplexity.Complex.ToString)
945. .With(complexQuestion => complexQuestion.Location, fakeLocation)
946. .Create();
947. var questionIds = BuildQuestionIds(10);
948. var fakeMemberMock = Fixture.Create<Mock<Members>>();
949. var fakeMemberObj = fakeMemberMock.Object;
950. fakeMemberObj.Username = username;
951. fakeMemberObj.QuestionIds = questionIds;
952. fakeMemberObj.CurrentQuestionNumber = 0;
953. fakeMemberObj.HintAsked = true;
954. var oldQuestionNumber = fakeMemberObj.CurrentQuestionNumber;
955. var oldUserScore = fakeMemberObj.UserScore;
956. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMemberObj.CurrentQuestionNumber]);
957. SetupMockGetMemberOrNull(repositoryMock, fakeMemberObj.Username, fakeMemberObj);
958. SetupMockGetQuestionOrNull(repositoryMock, currentQuestionId, fakeQuestion);
959. SetupMockGetComplexQuestion(repositoryMock, fakeQuestion.QuestionId, fakeComplexQuestion);
960. SetupMockGetLocationByLocationIdOrNull(repositoryMock, fakeMemberObj, fakeLocation);
961. SetupMockSaveChanges(repositoryMock);
962. var expectedValue = new Result<bool>(true)
963. .AddLink(FourWeekChallengeEndpoint.GetQuestionRouteWith(fakeMemberObj.Username));
964. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
965. .SubmitAnswer(fakeMemberObj.Username, answer)
966. .Should().BeResultEquivalentTo(expectedValue);
967. repositoryMock.Verify(repository => repository.SaveChanges(), Times.Once);
968. Assert.True(oldQuestionNumber + 1 == fakeMemberObj.CurrentQuestionNumber);
969. Assert.True(oldUserScore + 1 == fakeMemberObj.UserScore);
970. }
971. [Theory]
972. [AutoData]
973. public void GivenQuestionComplexityIsComplexAndAnswerIsCorrectAndLocationHintAsked\_ThenReturnSuccessResultWithIncrementingQuestionNumberOrUserScoreByOne(
974. string username, string answer)
975. {
976. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
977. var fakeLocation = CreateFakeLocation();
978. var fakeQuestion = Fixture.Build<Questions>()
979. .With(question => question.Discriminator, QuestionComplexity.Complex.ToString)
980. .With(question => question.CorrectAnswer, answer)
981. .Create();
982. var fakeComplexQuestion = Fixture.Build<ComplexQuestions>()
983. .With(complexQuestion => complexQuestion.Discriminator, QuestionComplexity.Complex.ToString)
984. .With(complexQuestion => complexQuestion.Location, fakeLocation)
985. .Create();
986. var questionIds = BuildQuestionIds(10);
987. var fakeMemberMock = Fixture.Create<Mock<Members>>();
988. var fakeMemberObj = fakeMemberMock.Object;
989. fakeMemberObj.Username = username;
990. fakeMemberObj.QuestionIds = questionIds;
991. fakeMemberObj.CurrentQuestionNumber = 0;
992. fakeMemberObj.LocationHintAsked = true;
993. var oldQuestionNumber = fakeMemberObj.CurrentQuestionNumber;
994. var oldUserScore = fakeMemberObj.UserScore;
995. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMemberObj.CurrentQuestionNumber]);
996. SetupMockGetMemberOrNull(repositoryMock, fakeMemberObj.Username, fakeMemberObj);
997. SetupMockGetQuestionOrNull(repositoryMock, currentQuestionId, fakeQuestion);
998. SetupMockGetComplexQuestion(repositoryMock, fakeQuestion.QuestionId, fakeComplexQuestion);
999. SetupMockGetLocationByLocationIdOrNull(repositoryMock, fakeMemberObj, fakeLocation);
1000. SetupMockSaveChanges(repositoryMock);
1001. var expectedValue = new Result<bool>(true)
1002. .AddLink(FourWeekChallengeEndpoint.GetQuestionRouteWith(fakeMemberObj.Username));
1003. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
1004. .SubmitAnswer(fakeMemberObj.Username, answer)
1005. .Should().BeResultEquivalentTo(expectedValue);
1006. repositoryMock.Verify(repository => repository.SaveChanges(), Times.Once);
1007. Assert.True(oldQuestionNumber + 1 == fakeMemberObj.CurrentQuestionNumber);
1008. Assert.True(oldUserScore + 1 == fakeMemberObj.UserScore);
1009. }
1010. [Theory]
1011. [AutoData]
1012. public void GivenQuestionComplexityIsComplexAndAnswerIsCorrectAndNoHintAsked\_ThenReturnSuccessResultWithIncrementingQuestionNumberOrUserScoreByTwo(
1013. string username, string answer)
1014. {
1015. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
1016. var fakeLocation = CreateFakeLocation();
1017. var fakeQuestion = Fixture.Build<Questions>()
1018. .With(question => question.Discriminator, QuestionComplexity.Complex.ToString)
1019. .With(question => question.CorrectAnswer, answer)
1020. .Create();
1021. var fakeComplexQuestion = Fixture.Build<ComplexQuestions>()
1022. .With(complexQuestion => complexQuestion.Discriminator, QuestionComplexity.Complex.ToString)
1023. .With(complexQuestion => complexQuestion.Location, fakeLocation)
1024. .Create();
1025. var questionIds = BuildQuestionIds(10);
1026. var fakeMemberMock = Fixture.Create<Mock<Members>>();
1027. var fakeMemberObj = fakeMemberMock.Object;
1028. fakeMemberObj.Username = username;
1029. fakeMemberObj.QuestionIds = questionIds;
1030. fakeMemberObj.CurrentQuestionNumber = 0;
1031. var oldQuestionNumber = fakeMemberObj.CurrentQuestionNumber;
1032. var oldUserScore = fakeMemberObj.UserScore;
1033. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMemberObj.CurrentQuestionNumber]);
1034. SetupMockGetMemberOrNull(repositoryMock, fakeMemberObj.Username, fakeMemberObj);
1035. SetupMockGetQuestionOrNull(repositoryMock, currentQuestionId, fakeQuestion);
1036. SetupMockGetComplexQuestion(repositoryMock, fakeQuestion.QuestionId, fakeComplexQuestion);
1037. SetupMockGetLocationByLocationIdOrNull(repositoryMock, fakeMemberObj, fakeLocation);
1038. SetupMockSaveChanges(repositoryMock);
1039. var expectedValue = new Result<bool>(true)
1040. .AddLink(FourWeekChallengeEndpoint.GetQuestionRouteWith(fakeMemberObj.Username));
1041. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
1042. .SubmitAnswer(fakeMemberObj.Username, answer)
1043. .Should().BeResultEquivalentTo(expectedValue);
1044. repositoryMock.Verify(repository => repository.SaveChanges(), Times.Once);
1045. Assert.True(oldQuestionNumber + 1 == fakeMemberObj.CurrentQuestionNumber);
1046. Assert.True(oldUserScore + 2 == fakeMemberObj.UserScore);
1047. }
1048. [Theory]
1049. [AutoData]
1050. public void GivenGameHasEnded\_ThenReturnSuccessResultWithEndGameRouteLink(
1051. string username, string answer)
1052. {
1053. var repositoryMock = new Mock<IGlobalRepository>(MockBehavior.Strict);
1054. var fakeLocation = CreateFakeLocation();
1055. var fakeQuestion = Fixture.Build<Questions>()
1056. .With(question => question.Discriminator, QuestionComplexity.Complex.ToString)
1057. .With(question => question.CorrectAnswer, answer)
1058. .Create();
1059. var fakeComplexQuestion = Fixture.Build<ComplexQuestions>()
1060. .With(complexQuestion => complexQuestion.Discriminator, QuestionComplexity.Complex.ToString)
1061. .With(complexQuestion => complexQuestion.Location, fakeLocation)
1062. .Create();
1063. var questionIds = BuildQuestionIds(10);
1064. var fakeMemberMock = Fixture.Create<Mock<Members>>();
1065. var fakeMemberObj = fakeMemberMock.Object;
1066. fakeMemberObj.Username = username;
1067. fakeMemberObj.QuestionIds = questionIds;
1068. fakeMemberObj.CurrentQuestionNumber = questionIds.Split(",").Length - 1;
1069. var oldQuestionNumber = fakeMemberObj.CurrentQuestionNumber;
1070. var oldUserScore = fakeMemberObj.UserScore;
1071. var currentQuestionId = int.Parse(questionIds.Split(",")[fakeMemberObj.CurrentQuestionNumber]);
1072. SetupMockGetMemberOrNull(repositoryMock, fakeMemberObj.Username, fakeMemberObj);
1073. SetupMockGetQuestionOrNull(repositoryMock, currentQuestionId, fakeQuestion);
1074. SetupMockGetComplexQuestion(repositoryMock, fakeQuestion.QuestionId, fakeComplexQuestion);
1075. SetupMockGetLocationByLocationIdOrNull(repositoryMock, fakeMemberObj, fakeLocation);
1076. SetupMockSaveChanges(repositoryMock);
1077. var expectedValue = new Result<bool>(true)
1078. .AddLink(FourWeekChallengeEndpoint.EndGameRouteWith(fakeMemberObj.Username));
1079. new FourWeekChallengeManager(repositoryMock.Object, new Mock<IQuestionRandomiser>().Object)
1080. .SubmitAnswer(fakeMemberObj.Username, answer)
1081. .Should().BeResultEquivalentTo(expectedValue);
1082. repositoryMock.Verify(repository => repository.SaveChanges(), Times.Once);
1083. Assert.True(oldQuestionNumber + 1 == fakeMemberObj.CurrentQuestionNumber);
1084. Assert.True(oldUserScore + 2 == fakeMemberObj.UserScore);
1085. }
1086. private static void SetupMockGetLocationByLocationIdOrNull(Mock<IGlobalRepository> repositoryMock, Members fakeMemberObj,
1087. Locations location)
1088. {
1089. repositoryMock.Setup(repository =>
1090. repository.Locations.GetLocationByLocationIdOrNull(fakeMemberObj.LocationId))
1091. .Returns(location);
1092. }
1093. private Locations CreateFakeLocation()
1094. {
1095. var random = new Random();
1096. return Fixture.Build<Locations>()
1097. .With(locations => locations.Latitude, random.Next(90))
1098. .With(locations => locations.Longitude, random.Next(180))
1099. .Create();
1100. }
1101. }
1102. protected static void SetupMockGetMemberOrNull(Mock<IGlobalRepository> repositoryMock,
1103. string username,
1104. Members expected)
1105. {
1106. repositoryMock.Setup(repository => repository.Members.GetMemberOrNull(username))
1107. .Returns(() => expected);
1108. }
1109. protected static void SetupMockDoesQuestionExist(Mock<IGlobalRepository> repositoryMock, int currentQuestionId,
1110. bool expected)
1111. {
1112. repositoryMock.Setup(repository => repository.Questions.DoesQuestionExist(currentQuestionId))
1113. .Returns(expected);
1114. }
1115. protected static void SetupMockGetComplexQuestion(Mock<IGlobalRepository> repositoryMock, int currentQuestionId,
1116. ComplexQuestions expected)
1117. {
1118. repositoryMock.Setup(repository => repository.ComplexQuestions.GetComplexQuestion(currentQuestionId))
1119. .Returns(() => expected);
1120. }
1121. protected static void SetupMockGetQuestionOrNull(Mock<IGlobalRepository> repositoryMock, int currentQuestionId, Questions? expected)
1122. {
1123. repositoryMock.Setup(repository => repository.Questions.GetQuestionOrNull(currentQuestionId))
1124. .Returns(() => expected);
1125. }
1126. protected static void SetupMockSaveChanges(Mock<IGlobalRepository> repositoryMock)
1127. {
1128. repositoryMock.Setup(repository => repository.SaveChanges());
1129. }
1130. protected string BuildQuestionIds(int count)
1131. {
1132. return string.Join(",", Fixture.CreateMany<int>(count));
1133. }
1134. }