

```

1 using System;
2 using System.Text;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Net;
6 using System.Threading.Tasks;
7
8 using Amazon.Lambda.Core;
9 using Amazon.Lambda.APIGatewayEvents;
10
11 using Newtonsoft.Json;
12 using Newtonsoft.Json.Serialization;
13 using ReVersionVCS_API_Lambdas.Models;
14 using ReVersionVCS_API_Lambdas.Response_Objects;
15 using ReVersionVCS_API_Lambdas.Request_Objects;
16 using System.Text.RegularExpressions;
17 using static ReVersionVCS_API_Lambdas.SQLOperations;
18
19
20 // Assembly attribute to enable the Lambda function's JSON input to be converted into a .NET class.
21 [assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]
22
23 namespace ReVersionVCS_API_Lambdas
24 {
25     public partial class Functions
26     {
27         private readonly JsonSerializerSettings jsonSerializerSettings;
28
29         private S3Operations s3Ops;
30
31
32         const string S3_REGION_ENVIRONMENT_VARIABLE_LOOKUP = "S3_REGION";
33
34         public const string ID_QUERY_STRING_NAME = "Id";
35
36         /// <summary>
37         /// Default constructor that Lambda will invoke.
38         /// </summary>
39         public Functions()
40         {
41             // set jsonSerializerSettings to properly handle Camel-Case Names
42             // i.e. the JSON property "fooBar" serializes to the C# property "FooBar"
43             DefaultContractResolver contractResolver = new DefaultContractResolver
44             {
45                 NamingStrategy = new CamelCaseNamingStrategy()
46             };
47             jsonSerializerSettings = new JsonSerializerSettings
48             {
49                 ContractResolver = contractResolver
50             };
51         }
52
53
54         // TODO: add authorization check and return 401 (HttpStatusCode.Unauthorized) for everything (
55         // unless this is handled by API Gateway)
56         // TODO: same for 500 code (HttpStatusCode.InternalServerError)
57         /// <summary>
58         /// A Lambda function that gets a list of all of the repositories
59         /// </summary>
60         /// <param name="request"></param>
61         /// <returns></returns>
62         public async Task<APIGatewayProxyResponse> GetRepositoriesAsync(APIGatewayProxyRequest request,
63             ILambdaContext context)
64         {
65             return
66                 await Task.Run(() =>
67                 {
68                     using (ReVersion.DatabaseContext db = new ReVersion.DatabaseContext())
69                     {
70                         List<RepositoryLookup> repositoryNames = QueryRepositories(db);
71                         List<ResourceItem> repositoryResponse = repositoryNames.Select(x => new
72
73 ResourceItem
74
75 {
76     DisplayData = x.Name,
77     Owner = x.Owner,
78     Href = request.Path + $"/{x.Name}"
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

74         }).ToList();
75
76         var response = new APIGatewayProxyResponse
77         {
78             StatusCode = (int)HttpStatusCode.OK,
79             Body = JsonConvert.SerializeObject(new { Resources = repositoryResponse },
            jsonSerializerSettings),
            Headers = new Dictionary<string, string> { { "Content-Type", "application/
            json" } }
            };
81
82
83         return response;
84     }
85
86 }
87
88 /// <summary>
89 /// A Lambda function that creates a new repository
90 /// </summary>
91 /// <param name="request"></param>
92 /// <returns></returns>
93 public async Task<APIGatewayProxyResponse> CreateRepositoryAsync(APIGatewayProxyRequest request,
    ILambdaContext context)
94 {
95     using (var db = new ReVersion.DatabaseContext())
96     {
97         s3Ops = new S3Operations(Environment.GetEnvironmentVariable(
            S3_REGION_ENVIRONMENT_VARIABLE_LOOKUP));
98
99         RepositoryData requestBody;
100         try
101         {
102             requestBody = JsonConvert.DeserializeObject<RepositoryData>(request.Body,
            jsonSerializerSettings);
103         }
104         catch (Exception)
105         {
106             return new APIGatewayProxyResponse
107             {
108                 StatusCode = (int)HttpStatusCode.BadRequest
109             };
110         }
111
112         string repo = requestBody.RepositoryId;
113         string user = requestBody.UserName;
114         string bucketname = S3Operations.bucketPrefix + '.' + repo.ToLower() + ".main";
115         if (!BucketNameValid(db, bucketname) || !UsernameExists(db, user))
116         {
117             return new APIGatewayProxyResponse
118             {
119                 StatusCode = (int)HttpStatusCode.BadRequest
120             };
121         }
122
123         if (RepositoryExists(db, repo))
124         {
125             return new APIGatewayProxyResponse
126             {
127                 StatusCode = (int)HttpStatusCode.Conflict
128             };
129         }
130
131         InsertIntoRepoTable(db, repo, user);
132         InsertIntoBranchTable(db, repo, "main");
133         InsertIntoRepoPermissionsTable(db, repo, user);
134         await s3Ops.CreateS3BucketAsync(repo, "main");
135         InsertIntoEventLog(db, repo, "main", user, requestBody.Message, "create_repository");
136
137         var responseObject = new ResourceItem
138         {
139             DisplayData = repo,
140             Href = request.Path + $"/{repo}",
141             Owner = user
142         };
143
144         await db.SaveChangesAsync();

```

```

145         return new APIGatewayProxyResponse
146         {
147             StatusCode = (int)HttpStatusCode.Created,
148             Body = JsonConvert.SerializeObject(responseObject),
149             Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
150         };
151     }
152 }
153
154 /// <summary>
155 /// A Lambda function that deletes an entire repository
156 /// </summary>
157 /// <param name="request"></param>
158 /// <returns></returns>
159 public async Task<APIGatewayProxyResponse> DeleteRepositoryAsync(APIGatewayProxyRequest request,
ILambdaContext context)
160 {
161     using (var db = new ReVersion.DatabaseContext())
162     {
163         s3Ops = new S3Operations(Environment.GetEnvironmentVariable(
164             S3.REGION_ENVIRONMENT_VARIABLE_LOOKUP));
165
166         if (request.PathParameters == null || !request.PathParameters.ContainsKey("repositoryId"))
167             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
168
169         string repo;
170         if (!request.PathParameters.TryGetValue("repositoryId", out repo))
171             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
172
173         if (!RepositoryExists(db, repo))
174             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NotFound };
175
176         List<BranchLookup> branches = QueryBranches(db, repo);
177
178         foreach (var branch in branches)
179         {
180             if (branch.Locked)
181                 return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.Conflict };
182         }
183
184         DeleteAllFromBranchesTable(db, repo);
185         DeleteAllFromVersionsTable(db, repo);
186         DeleteAllFromPermissionsTable(db, repo);
187         DeleteAllFromPermissionRequestsTable(db, repo);
188         DeleteFromRepositoryTable(db, repo);
189
190         await db.SaveChangesAsync();
191         return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NoContent };
192     }
193 }
194
195 /// <summary>
196 /// A Lambda function that gets a list of all of the branches in the repository
197 /// </summary>
198 /// <param name="request"></param>
199 /// <returns></returns>
200 public async Task<APIGatewayProxyResponse> GetBranchesAsync(APIGatewayProxyRequest request,
ILambdaContext context)
201 {
202     return
203         await Task.Run(() =>
204         {
205             using (var db = new ReVersion.DatabaseContext())
206             {
207                 if (request.PathParameters == null || !request.PathParameters.ContainsKey("
repositoryId"))
208                     return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
209
210                 string repo;
211                 if (!request.PathParameters.TryGetValue("repositoryId", out repo))
212                     return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };

```

```

213         if (!RepositoryExists(db, repo))
214             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
215 NotFound };
216
217         List<BranchLookup> branches = QueryBranches(db, repo);
218
219         List<ResourceItem> responseObject =
220             (from branch in branches
221              select new ResourceItem
222              {
223                  DisplayData = branch.BranchName,
224                  Href = request.Path + $"/{branch.BranchName}"
225              }).ToList();
226
227         string responseContent = JsonConvert.SerializeObject(new { Resources =
responseObject }, JsonSerializerSettings);
228
229         return new APIGatewayProxyResponse
230         {
231             StatusCode = (int)HttpStatusCode.OK,
232             Body = responseContent,
233             Headers = new Dictionary<string, string> { { "Content-Type", "application/
234 json" } }
235         };
236     }
237 }
238
239
240 /// <summary>
241 /// A Lambda function that creates a new branch within the repository
242 /// </summary>
243 /// <param name="request"></param>
244 /// <returns></returns>
245 public async Task<APIGatewayProxyResponse> CreateBranchAsync(APIGatewayProxyRequest request,
ILambdaContext context)
246 {
247     using (var db = new ReVersion.DatabaseContext())
248     {
249         s3Ops = new S3Operations(Environment.GetEnvironmentVariable(
S3_REGION_ENVIRONMENT_VARIABLE_LOOKUP));
250         if (request.PathParameters == null || !request.PathParameters.ContainsKey("repositoryId"))
251             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
252
253         string repo;
254         if (!request.PathParameters.TryGetValue("repositoryId", out repo))
255             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
256
257         if (!RepositoryExists(db, repo))
258             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NotFound };
259
260         NewBranchData requestBody;
261         try
262         {
263             requestBody = JsonConvert.DeserializeObject<NewBranchData>(request.Body);
264         }
265         catch (Exception)
266         {
267             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
268         }
269
270         if (!UsernameExists(db, requestBody.UserName))
271             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };
272
273         if (BranchExists(db, repo, requestBody.BranchId))
274             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.Conflict };
275
276         InsertIntoBranchTable(db, repo, requestBody.BranchId);
277         await s3Ops.CreateS3BucketAsync(repo, requestBody.BranchId);
278         InsertIntoEventLog(db, repo, requestBody.BranchId, requestBody.UserName, requestBody.
Message, "create_branch");
279

```

```

280         var responseObject = new ResourceItem
281         {
282             DisplayData = requestBody.BranchId,
283             Href = request.Path + $"/{requestBody.BranchId}"
284         };
285
286         var responseContent = JsonConvert.SerializeObject(responseObject);
287
288         await db.SaveChangesAsync();
289         return new APIGatewayProxyResponse
290         {
291             StatusCode = (int)HttpStatusCode.Created,
292             Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } },
293             Body = responseContent
294         };
295     }
296 }
297
298 /// <summary>
299 /// A Lambda function that gets a list of all of the objects within a branch
300 /// </summary>
301 /// <param name="request"></param>
302 /// <returns></returns>
303 public async Task<APIGatewayProxyResponse> GetBranchAsync(APIGatewayProxyRequest request,
ILambdaContext context)
304 {
305     return
306         await Task.Run(() =>
307         {
308             using (var db = new ReVersion.DatabaseContext())
309             {
310                 context.Logger.Log("start of GetBranchAsync function");
311                 if (request.PathParameters == null || !request.PathParameters.ContainsKey("
repositoryId")
312                     || !request.PathParameters.ContainsKey("branchId"))
313                     return new APIGatewayProxyResponse
314                     {
315                         StatusCode = (int)HttpStatusCode.InternalServerError,
316                         Body = "path parameters are empty or missing repositoryId or branchId"
317                     };
318
319                 context.Logger.Log("Passed the first check repositoryId and branchId exist");
320
321                 string repo;
322                 string branch;
323                 if (!request.PathParameters.TryGetValue("repositoryId", out repo)
324                     || !request.PathParameters.TryGetValue("branchId", out branch))
325                     return new APIGatewayProxyResponse
326                     {
327                         StatusCode = (int)HttpStatusCode.InternalServerError,
328                         Body = "missing repositoryId or branchId in the second check"
329                     };
330
331                 context.Logger.Log($"Passed the second check. repositoryId = {repo} and branchId =
{branch}");
332
333                 if (!BranchExists(db, repo, branch))
334                     return new APIGatewayProxyResponse
335                     {
336                         StatusCode = (int)HttpStatusCode.NotFound,
337                         Body = $"Branch does not exist for branch = {branch} and repository = {
repo}"
338                     };
339
340                 context.Logger.Log($"Determined that branch exists for repository {repo} and
branch {branch}");
341
342                 HierarchyNode branchFiles = QueryHierarchy(db, repo, branch);
343
344                 context.Logger.Log($"Successfully queried branch hierarchy from repository {repo}
and branch {branch}");
345
346                 return new APIGatewayProxyResponse
347                 {
348                     StatusCode = (int)HttpStatusCode.OK,
349                     Body = JsonConvert.SerializeObject(branchFiles),

```

```

350         Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
351     };
352     }
353 });
354 }
355
356 /// <summary>
357 /// A Lambda function that commits all local changes to a branch
358 /// </summary>
359 /// <param name="request"></param>
360 /// <returns></returns>
361 public async Task<APIGatewayProxyResponse> CommitChangesAsync(APIGatewayProxyRequest request,
ILambdaContext context)
362 {
363     return await Task.Run(() =>
364     {
365         s3Ops = new S3Operations(Environment.GetEnvironmentVariable(
S3_REGION_ENVIRONMENT_VARIABLE_LOOKUP));
366
367         return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError
368     });
369     // TODO
370     //throw new NotImplementedException();
371 }
372
373 /// <summary>
374 /// A Lambda function that gets a file within a branch
375 /// </summary>
376 /// <param name="request"></param>
377 /// <returns></returns>
378 public async Task<APIGatewayProxyResponse> GetObjectAsync(APIGatewayProxyRequest request,
ILambdaContext context)
379 {
380     using (var db = new ReVersion.DatabaseContext())
381     {
382         s3Ops = new S3Operations(Environment.GetEnvironmentVariable(
S3_REGION_ENVIRONMENT_VARIABLE_LOOKUP));
383
384         if (request.PathParameters == null || !request.PathParameters.ContainsKey("repositoryId")
385         || !request.PathParameters.ContainsKey("branchId")
386         || !request.QueryStringParameters.ContainsKey("awsObjectKey"))
387             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
388         string repo;
389         string branch;
390         string objectKey;
391         if (!request.PathParameters.TryGetValue("repositoryId", out repo)
392         || !request.PathParameters.TryGetValue("branchId", out branch)
393         || !request.PathParameters.TryGetValue("awsObjectKey", out objectKey))
394             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
395
396         if (!BranchExists(db, repo, branch))
397             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NotFound };
398
399         string responseContent = await s3Ops.GetTextFromObjectAsync(repo, branch, objectKey);
400
401         return new APIGatewayProxyResponse
402         {
403             StatusCode = (int)HttpStatusCode.OK,
404             Body = responseContent,
405             Headers = new Dictionary<string, string> { { "Content-Type", "text/plain" } }
406         };
407     }
408 }
409
410 /// <summary>
411 /// A Lambda function that gets a list of all of the versions within a branch
412 /// </summary>
413 /// <param name="request"></param>
414 /// <returns></returns>
415 public async Task<APIGatewayProxyResponse> GetVersionsAsync(APIGatewayProxyRequest request,
ILambdaContext context)
416 {

```

```

417         return await Task.Run(() =>
418         {
419             using (var db = new ReVersion_DatabaseContext())
420             {
421                 if (request.PathParameters == null || !request.PathParameters.ContainsKey("
repositoryId")
422                     || !request.PathParameters.ContainsKey("branchId")
423                     || !request.QueryStringParameters.ContainsKey("awsObjectKey"))
424                     return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
425                 string repo;
426                 string branch;
427                 if (!request.PathParameters.TryGetValue("repositoryId", out repo)
428                     || !request.PathParameters.TryGetValue("branchId", out branch))
429                     return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
430
431                 if (!BranchExists(db, repo, branch))
432                     return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NotFound };
433
434                 var versionList = QueryVersions(db, repo, branch);
435                 List<ResourceItem> responseContent =
436                     versionList.Select(x => new ResourceItem
437                     {
438                         DisplayData = $" {x} ",
439                         Href = $" {request.Path}/{x} "
440                     }).ToList();
441
442                 return new APIGatewayProxyResponse
443                 {
444                     StatusCode = (int)HttpStatusCode.OK,
445                     Body = JsonConvert.SerializeObject(new { Resources = responseContent }),
446                     Headers = new Dictionary<string, string> { { "Content-Type", "application/json" }
}
447                 };
448             }
449         });
450     }
451
452     /// <summary>
453     /// A Lambda function that gets a list of all of the files within a version of a branch
454     /// </summary>
455     /// <param name="request"></param>
456     /// <returns></returns>
457     public async Task<APIGatewayProxyResponse> GetVersionAsync(APIGatewayProxyRequest request,
ILambdaContext context)
458     {
459         return await Task.Run(() =>
460         {
461             using (var db = new ReVersion_DatabaseContext())
462             {
463                 if (request.PathParameters == null || !request.PathParameters.ContainsKey("
repositoryId")
464                     || !request.PathParameters.ContainsKey("branchId")
465                     || !request.QueryStringParameters.ContainsKey("versionId"))
466                     return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
467                 string repo;
468                 string branch;
469                 string version;
470                 if (!request.PathParameters.TryGetValue("repositoryId", out repo)
471                     || !request.PathParameters.TryGetValue("branchId", out branch)
472                     || !request.PathParameters.TryGetValue("versionId", out version))
473                     return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
474
475                 if (!VersionExists(db, repo, branch, version))
476                     return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NotFound };
477
478                 HierarchyNode versionFiles = QueryHierarchy(db, repo, branch, version);
479
480                 return new APIGatewayProxyResponse
481                 {
482                     StatusCode = (int)HttpStatusCode.OK,
483                     Body = JsonConvert.SerializeObject(versionFiles),
484                     Headers = new Dictionary<string, string> { { "Content-Type", "application/json" }

```

```

}
485     };
486     }
487     });
488 }
489
490     /// <summary>
491     /// A Lambda function that gets a file from a non-current version of a branch
492     /// </summary>
493     /// <param name="request"></param>
494     /// <returns></returns>
495     public async Task<APIGatewayProxyResponse> GetPastObjectAsync(APIGatewayProxyRequest request,
ILambdaContext context)
496     {
497         return await Task.Run(() =>
498         {
499             s3Ops = new S3Operations(Environment.GetEnvironmentVariable(
S3_REGION_ENVIRONMENT_VARIABLE_LOOKUP));
500             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError
501         });
502         // TODO
503         //throw new NotImplementedException();
504     }
505
506     /// <summary>
507     /// A Lambda function that merges a branch with its parent
508     /// </summary>
509     /// <param name="request"></param>
510     /// <returns></returns>
511     public async Task<APIGatewayProxyResponse> MergeBranchAsync(APIGatewayProxyRequest request,
ILambdaContext context)
512     {
513         return await Task.Run(() =>
514         {
515             s3Ops = new S3Operations(Environment.GetEnvironmentVariable(
S3_REGION_ENVIRONMENT_VARIABLE_LOOKUP));
516             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError
517         });
518         // TODO
519         //throw new NotImplementedException();
520     }
521
522     /// <summary>
523     /// A Lambda function that places a lock on a branch
524     /// </summary>
525     /// <param name="request"></param>
526     /// <returns></returns>
527     public async Task<APIGatewayProxyResponse> LockBranchAsync(APIGatewayProxyRequest request,
ILambdaContext context)
528     {
529         using (var db = new ReVersion.DatabaseContext())
530         {
531             if (request.PathParameters == null || !request.PathParameters.ContainsKey("repositoryId")
532             || !request.PathParameters.ContainsKey("branchId"))
533                 return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
534
535             string repo;
536             string branch;
537
538             if (!request.PathParameters.TryGetValue("repositoryId", out repo)
539             || !request.PathParameters.TryGetValue("branchId", out branch))
540                 return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
541
542             LogData requestBody;
543             try
544             {
545                 requestBody = JsonConvert.DeserializeObject<LogData>(request.Body);
546             }
547             catch (Exception)
548             {
549                 return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };

```



```

550     }
551
552     if (!UsernameExists(db, requestBody.UserName))
553         return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };
554
555     if (!BranchExists(db, repo, branch))
556         return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NotFound };
557
558     LockState lockState = AcquireLock(db, repo, branch);
559
560     if (lockState == LockState.AlreadyLockedConflict)
561         return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.Conflict };
562
563     if (lockState != LockState.SuccessfulLockOperation)
564         return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
565
566     InsertIntoEventLog(db, repo, branch, requestBody.UserName, requestBody.Message, "
place_lock");
567
568     LockData responseObject = QueryLockEventByBranch(db, repo, branch);
569
570     await db.SaveChangesAsync();
571     return new APIGatewayProxyResponse
572     {
573         StatusCode = (int)HttpStatusCode.OK,
574         Body = JsonConvert.SerializeObject(responseObject),
575         Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
576     };
577 }
578 }
579
580 /// <summary>
581 /// A Lambda function that "safely" removes a lock from a branch (only works if the same user set
the lock)
582 /// </summary>
583 /// <param name="request"></param>
584 /// <returns></returns>
585 public async Task<APIGatewayProxyResponse> UnlockBranchAsync(APIGatewayProxyRequest request,
ILambdaContext context)
586 {
587     using (var db = new ReVersion_DatabaseContext())
588     {
589         if (request.PathParameters == null || !request.PathParameters.ContainsKey("repositoryId")
|| !request.PathParameters.ContainsKey("branchId"))
590             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
591
592         string repo;
593         string branch;
594
595         if (!request.PathParameters.TryGetValue("repositoryId", out repo)
|| !request.PathParameters.TryGetValue("branchId", out branch))
596             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
597
598         LogData requestBody;
599         try
600         {
601             requestBody = JsonConvert.DeserializeObject<LogData>(request.Body);
602         }
603         catch (Exception)
604         {
605             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
606         }
607
608         if (!UsernameExists(db, requestBody.UserName))
609             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };
610
611         if (!BranchExists(db, repo, branch))
612             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NotFound };
613
614         LockState lockState = SafeReleaseLock(db, repo, branch, requestBody.UserName);
615
616         if (lockState == LockState.AlreadyUnlockedConflict || lockState == LockState.

```

```

619         LockedByDifferentUser)
620             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.Conflict };
621
622         if (lockState != LockState.SuccessfulLockOperation)
623             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
624 InternalServerError };
625
626         InsertIntoEventLog(db, repo, branch, requestBody.UserName, requestBody.Message, "
627 safely_remove_lock");
628
629         LockData responseObject = QueryLockEventByBranch(db, repo, branch);
630
631         await db.SaveChangesAsync();
632         return new APIGatewayProxyResponse
633         {
634             StatusCode = (int)HttpStatusCode.OK,
635             Body = JsonConvert.SerializeObject(responseObject),
636             Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
637         };
638     }
639
640     /// <summary>
641     /// A Lambda function that "unsafely" removes a lock from a branch (any user can do this)
642     /// </summary>
643     /// <param name="request"></param>
644     /// <returns></returns>
645     public async Task<APIGatewayProxyResponse> BreakLockAsync(APIGatewayProxyRequest request,
646 ILambdaContext context)
647     {
648         using (var db = new ReVersion.DatabaseContext())
649         {
650             if (request.PathParameters == null || !request.PathParameters.ContainsKey("repositoryId")
651 || !request.PathParameters.ContainsKey("branchId"))
652                 return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
653 InternalServerError };
654
655             string repo;
656             string branch;
657
658             if (!request.PathParameters.TryGetValue("repositoryId", out repo)
659 || !request.PathParameters.TryGetValue("branchId", out branch))
660                 return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
661 InternalServerError };
662
663             LogData requestBody;
664             try
665             {
666                 requestBody = JsonConvert.DeserializeObject<LogData>(request.Body);
667             }
668             catch (Exception)
669             {
670                 return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
671 InternalServerError };
672             }
673
674             if (!UsernameExists(db, requestBody.UserName))
675                 return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };
676
677             if (!BranchExists(db, repo, branch))
678                 return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NotFound };
679
680             LockState lockState = ReleaseLock(db, repo, branch);
681
682             if (lockState == LockState.AlreadyUnlockedConflict)
683                 return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.Conflict };
684
685             if (lockState != LockState.SuccessfulLockOperation)
686                 return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
687 InternalServerError };
688
689             InsertIntoEventLog(db, repo, branch, requestBody.UserName, requestBody.Message, "
690 force_remove_lock");

```

```

686         LockData responseObject = QueryLockEventByBranch(db, repo, branch);
687
688         await db.SaveChangesAsync();
689         return new APIGatewayProxyResponse
690         {
691             StatusCode = (int)HttpStatusCode.OK,
692             Body = JsonConvert.SerializeObject(responseObject),
693             Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } };
694         };
695     }
696 }
697
698 /// <summary>
699 /// A Lambda function that requests permission for a user to access a repository
700 /// </summary>
701 /// <param name="request"></param>
702 /// <returns></returns>
703 public async Task<APIGatewayProxyResponse> RequestPermissionAsync(APIGatewayProxyRequest request,
ILambdaContext context)
704 {
705     using (var db = new ReVersion.DatabaseContext())
706     {
707         if (request.PathParameters == null || !request.PathParameters.ContainsKey("repositoryId"))
708             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
709
710         string repo;
711
712         if (!request.PathParameters.TryGetValue("repositoryId", out repo))
713             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
714
715         LogData requestBody;
716         try
717         {
718             requestBody = JsonConvert.DeserializeObject<LogData>(request.Body);
719         }
720         catch (Exception)
721         {
722             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
723         }
724
725         if (!UsernameExists(db, requestBody.UserName))
726             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };
727
728         if (!RepositoryExists(db, repo))
729             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NotFound };
730
731         if (UserCanAccessRepository(db, requestBody.UserName, repo))
732             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.Conflict };
733
734         if (PermissionRequestIsLogged(db, repo, requestBody.UserName))
735             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NoContent };
736
737         InsertIntoEventLog(db, repo, "main", requestBody.UserName, requestBody.Message, "
request_permission");
738         int eventId = QueryLastEventIdByUser(db, requestBody.UserName);
739
740         InsertIntoPermissionRequestTable(db, repo, eventId);
741
742         await db.SaveChangesAsync();
743         return new APIGatewayProxyResponse
744         {
745             StatusCode = (int)HttpStatusCode.NoContent,
746             Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } };
747         };
748     }
749 }
750
751
752 // TODO: in the client-side, you need to remember to add /grant and /deny to all of the resource
URLs!!!
753
754 /// <summary>
755 /// A Lambda function that gets a list of all of the unanswered permission requests submitted

```

```

756     /// to access all of the repositories owned by a user
757     /// </summary>
758     /// <param name="request"></param>
759     /// <returns></returns>
760     public async Task<APIGatewayProxyResponse> GetPermissionRequestsAsync(APIGatewayProxyRequest
request, ILambdaContext context)
761     {
762         return await Task.Run(() =>
763         {
764             context.Logger.Log("entering function");
765             using (var db = new ReVersion.DatabaseContext())
766             {
767                 if (request.Headers == null || !request.Headers.ContainsKey("username"))
768                     return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
769
770                 context.Logger.Log("passed initial parameter check");
771
772                 string user;
773                 if (!request.Headers.TryGetValue("username", out user))
774                     return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest
};
775
776                 context.Logger.Log($"username is {user}");
777
778                 List<PermissionLookup> permissions = QueryPermissionRequests(db, user);
779
780                 context.Logger.Log("okay, i just made a database call");
781
782                 List<ResourceItem> responseObject =
783                     permissions.Select(x => new ResourceItem
784                     {
785                         DisplayData = $"{x.RequestingUser} requested access to {x.RepositoryName} on "
+ x.LogTimestamp.ToString(@"MM/dd/yy H:mm tt"),
786                         Href = $"repositories/{x.RequestId}"
787                     }).ToList();
788
789                 context.Logger.Log("i just made the response object");
790
791                 return new APIGatewayProxyResponse
792                 {
793                     StatusCode = (int)HttpStatusCode.OK,
794                     Body = JsonConvert.SerializeObject(new { Resources = responseObject }),
795                     Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
796                 };
797             }
798         });
799     }
800
801     /// <summary>
802     /// A Lambda function that grants a user's request to access a repository
803     /// </summary>
804     /// <param name="request"></param>
805     /// <returns></returns>
806     public async Task<APIGatewayProxyResponse> GrantPermissionAsync(APIGatewayProxyRequest request,
ILambdaContext context)
807     {
808         using (var db = new ReVersion.DatabaseContext())
809         {
810             if (request.PathParameters == null || !request.PathParameters.ContainsKey("requestId"))
811                 return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
812
813             if (!request.PathParameters.TryGetValue("requestId", out string permissionRequest))
814             {
815                 return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
816             }
817
818             int requestId;
819             try
820             {
821                 requestId = Convert.ToInt32(permissionRequest);
822             }
823             catch (Exception)

```

```

824         {
825             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };
826         }
827
828
829         LogData requestBody;
830         try
831         {
832             requestBody = JsonConvert.DeserializeObject<LogData>(request.Body);
833         }
834         catch (Exception)
835         {
836             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
837         }
838
839         if (!UsernameExists(db, requestBody.UserName))
840             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };
841
842         string repo = QueryRepositoryNameFromRequest(db, requestId);
843
844         if (!UserOwnsRepository(db, repo, requestBody.UserName))
845             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.Forbidden };
846
847         InsertIntoRepoPermissionsTable(db, repo, requestBody.UserName);
848         DeleteFromPermissionRequestTable(db, requestId);
849         InsertIntoEventLog(db, repo, "main", requestBody.UserName, requestBody.Message, "
grant_permission");
850
851         await db.SaveChangesAsync();
852         return new APIGatewayProxyResponse
853         {
854             StatusCode = (int)HttpStatusCode.NoContent,
855             Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
856         };
857     }
858 }
859
860 /// <summary>
861 /// A Lambda function that rejects a user's request to access a repository
862 /// </summary>
863 /// <param name="request"></param>
864 /// <returns></returns>
865 public async Task<APIGatewayProxyResponse> DenyPermissionAsync(APIGatewayProxyRequest request,
ILambdaContext context)
866 {
867     using (var db = new ReVersion.DatabaseContext())
868     {
869         if (request.PathParameters == null || !request.PathParameters.ContainsKey("requestId"))
870             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
871
872         if (!request.PathParameters.TryGetValue("requestId", out string permissionRequest))
873         {
874             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
875         }
876
877         int requestId;
878         try
879         {
880             requestId = Convert.ToInt32(permissionRequest);
881         }
882         catch (Exception)
883         {
884             return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };
885         }
886
887
888         LogData requestBody;
889         try
890         {
891             requestBody = JsonConvert.DeserializeObject<LogData>(request.Body);
892         }
893         catch (Exception)
894         {

```

```

895         return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
896     }
897
898     if (!UsernameExists(db, requestBody.UserName))
899         return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };
900
901     string repo = QueryRepositoryNameFromRequest(db, requestId);
902
903     if (!UserOwnsRepository(db, repo, requestBody.UserName))
904         return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.Forbidden };
905
906     DeleteFromPermissionRequestTable(db, requestId);
907     InsertIntoEventLog(db, repo, "main", requestBody.UserName, requestBody.Message, "
deny_permission");
908
909     await db.SaveChangesAsync();
910     return new APIGatewayProxyResponse
911     {
912         StatusCode = (int)HttpStatusCode.NoContent,
913         Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
914     };
915 }
916
917
918
919     /// <exception cref="ArgumentNullException">from call to Convert.FromBase64String</exception>
920     /// <exception cref="FormatException">from call to Convert.FromBase64String if not in base64
format</exception>
921     /// <exception cref="ArgumentException">from call to Encoding.GetString if byte array contains
invalid Unicode code points</exception>
922     /// <exception cref="DecoderFallbackException">if fallback occurred due to character encoding in
.NET
923     ///
DecoderExceptionFallback"</exception>
924     private string DecodeBase64String(string str)
925     => Encoding.UTF8.GetString(Convert.FromBase64String(str));
926
927     private string EncodeBase64String(string str) =>
928     Convert.ToBase64String(Encoding.UTF8.GetBytes(str));
929
930     private bool BucketNameValid(ReVersion_DatabaseContext db, string bucketName)
931     {
932         var nameArray = bucketName.Split('.');
933         if (nameArray.Length != 3) return false;
934         if (!nameArray[0].Equals(S3Operations.bucketPrefix)) return false;
935         /*
936         if (nameArray[2].ToLower().Equals("main") && RepositoryExists(db, nameArray[1])) return false;
937         if (!nameArray[2].ToLower().Equals("main") && BranchExists(db, nameArray[1], nameArray[2]))
return false;
938         */
939         foreach (string str in nameArray)
940         {
941             if (Regex.IsMatch(str, @"[\s._]")) return false;
942             if (str.Length > 25) return false;
943             if (!Regex.IsMatch(str, @"^[a-zA-Z0-9][a-zA-Z0-9-]*[a-zA-Z0-9]$")) return false;
944         }
945
946         return true;
947     }
948 }
949 }
950 }

```

```

1 {
2   "AWSTemplateFormatVersion" : "2010-09-09",
3   "Transform" : "AWS::Serverless-2016-10-31",
4   "Description" : "AWS Serverless API that exposes the add, remove and get operations for a blogging
5     platform using Amazon DynamoDB.",
6   "Parameters" : {
7     "DBName" : {
8       "Type" : "String",
9       "Description" : "The name for the PostgreSQL database",
10      "MinLength" : "1",
11      "MaxLength" : "63"
12    },
13    "DBUsername" : {
14      "Type" : "String",
15      "Description" : "Master username for PostgreSQL database",
16      "MinLength" : "1",
17      "MaxLength" : "41",
18      "AllowedPattern" : "^[a-zA-Z0-9]+$"
19    },
20    "DBPassword" : {
21      "Type" : "String",
22      "Description" : "Master password for PostgreSQL database",
23      "NoEcho" : "true",
24      "MinLength" : "1",
25      "MaxLength" : "41"
26    },
27    "DBPort" : {
28      "Type" : "Number",
29      "Description" : "TCP/IP port for the database",
30      "MinValue" : "1150",
31      "MaxValue" : "65535",
32      "Default" : "5432"
33    },
34    "DBHostname" : {
35      "Type" : "String",
36      "Description" : "Hostname (endpoint) for PostgreSQL database",
37      "MinLength" : "1"
38    }
39  },
40  "Resources" : {
41    "GetRepositories" : {
42      "Type" : "AWS::Serverless::Function",
43      "Properties" : {
44        "Handler" : "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GetRepositoriesAsync",
45        "Runtime" : "dotnetcore2.1",
46        "CodeUri" : "",
47        "Description" : "Function to get a list of all repositories",
48        "MemorySize" : 256,
49        "Timeout" : 30,
50        "Role" : "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
51        "Policies" : [ "AWSLambdaFullAccess" ],
52        "Environment" : {
53          "Variables" : {
54            "RDS.DB.HOSTNAME" : { "Ref" : "DBHostname" },
55            "RDS.DB.PORT" : { "Ref" : "DBPort" },
56            "RDS.DB.NAME" : { "Ref" : "DBName" },
57            "RDS.DB.USERNAME" : { "Ref" : "DBUsername" },
58            "RDS.DB.PASSWORD" : { "Ref" : "DBPassword" }
59          }
60        },
61        "Events" : {
62          "PutResource" : {
63            "Type" : "Api",
64            "Properties" : {
65              "Path" : "/repositories",
66              "Method" : "GET"
67            }
68          }
69        }
70      }
71    },
72    "CreateRepository" : {
73      "Type" : "AWS::Serverless::Function",

```

```

76     "Properties": {
77         "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::CreateRepositoryAsync",
78         "Runtime": "dotnetcore2.1",
79         "CodeUri": "",
80         "Description": "Function to create a repository",
81         "MemorySize": 256,
82         "Timeout": 30,
83         "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
84         "Policies": [ "AWSLambdaFullAccess" ],
85         "Environment": {
86             "Variables": {
87                 "RDS_DB_HOSTNAME": { "Ref": "DBHostname" },
88                 "RDS_DB_PORT": { "Ref": "DBPort" },
89                 "RDS_DB_NAME": { "Ref": "DBName" },
90                 "RDS_DB_USERNAME": { "Ref": "DBUsername" },
91                 "RDS_DB_PASSWORD": { "Ref": "DBPassword" },
92                 "S3_REGION": { "Ref": "AWS::Region" }
93             }
94         },
95         "Events": {
96             "PutResource": {
97                 "Type": "Api",
98                 "Properties": {
99                     "Path": "/repositories",
100                     "Method": "POST"
101                 }
102             }
103         }
104     },
105 },
106
107 "DeleteRepository" : {
108     "Type" : "AWS::Serverless::Function",
109     "Properties": {
110         "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::DeleteRepositoryAsync",
111         "Runtime": "dotnetcore2.1",
112         "CodeUri": "",
113         "Description": "Function to delete an entire repository",
114         "MemorySize": 256,
115         "Timeout": 30,
116         "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
117         "Policies": [ "AWSLambdaFullAccess" ],
118         "Environment": {
119             "Variables": {
120                 "RDS_DB_HOSTNAME": { "Ref": "DBHostname" },
121                 "RDS_DB_PORT": { "Ref": "DBPort" },
122                 "RDS_DB_NAME": { "Ref": "DBName" },
123                 "RDS_DB_USERNAME": { "Ref": "DBUsername" },
124                 "RDS_DB_PASSWORD": { "Ref": "DBPassword" },
125                 "S3_REGION": { "Ref": "AWS::Region" }
126             }
127         },
128         "Events": {
129             "PutResource": {
130                 "Type": "Api",
131                 "Properties": {
132                     "Path": "/repositories/{repositoryId}",
133                     "Method": "DELETE"
134                 }
135             }
136         }
137     },
138 },
139
140 "GetBranches" : {
141     "Type" : "AWS::Serverless::Function",
142     "Properties": {
143         "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GetBranchesAsync",
144         "Runtime": "dotnetcore2.1",
145         "CodeUri": "",
146         "Description": "Function to get a list of all branches in the repository",
147         "MemorySize": 256,
148         "Timeout": 30,
149         "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
150         "Policies": [ "AWSLambdaFullAccess" ],
151         "Environment" : {

```



```

152     "Variables" : {
153     "RDS.DB.HOSTNAME" : { "Ref" : "DBHostname" },
154     "RDS.DB.PORT" : { "Ref" : "DBPort" },
155     "RDS.DB.NAME" : { "Ref" : "DBName" },
156     "RDS.DB.USERNAME" : { "Ref" : "DBUsername" },
157     "RDS.DB.PASSWORD" : { "Ref" : "DBPassword" }
158     },
159     },
160     "Events": {
161     "PutResource": {
162     "Type": "Api",
163     "Properties": {
164     "Path": "/repositories/{repositoryId}/branches",
165     "Method": "GET"
166     }
167     }
168     }
169     },
170     },
171
172     "CreateBranch" : {
173     "Type" : "AWS::Serverless::Function",
174     "Properties": {
175     "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::CreateBranchAsync",
176     "Runtime": "dotnetcore2.1",
177     "CodeUri": "",
178     "Description": "Function to create a new branch within the repository",
179     "MemorySize": 256,
180     "Timeout": 30,
181     "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
182     "Policies": [ "AWSLambdaFullAccess" ],
183     "Environment" : {
184     "Variables" : {
185     "RDS.DB.HOSTNAME" : { "Ref" : "DBHostname" },
186     "RDS.DB.PORT" : { "Ref" : "DBPort" },
187     "RDS.DB.NAME" : { "Ref" : "DBName" },
188     "RDS.DB.USERNAME" : { "Ref" : "DBUsername" },
189     "RDS.DB.PASSWORD" : { "Ref" : "DBPassword" },
190     "S3-REGION" : { "Ref" : "AWS::Region" }
191     }
192     },
193     "Events": {
194     "PutResource": {
195     "Type": "Api",
196     "Properties": {
197     "Path": "/repositories/{repositoryId}/branches",
198     "Method": "POST"
199     }
200     }
201     }
202     },
203     },
204
205     "GetBranch" : {
206     "Type" : "AWS::Serverless::Function",
207     "Properties": {
208     "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GetBranchAsync",
209     "Runtime": "dotnetcore2.1",
210     "CodeUri": "",
211     "Description": "Function to get a list of all objects within a branch",
212     "MemorySize": 256,
213     "Timeout": 30,
214     "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
215     "Policies": [ "AWSLambdaFullAccess" ],
216     "Environment" : {
217     "Variables" : {
218     "RDS.DB.HOSTNAME" : { "Ref" : "DBHostname" },
219     "RDS.DB.PORT" : { "Ref" : "DBPort" },
220     "RDS.DB.NAME" : { "Ref" : "DBName" },
221     "RDS.DB.USERNAME" : { "Ref" : "DBUsername" },
222     "RDS.DB.PASSWORD" : { "Ref" : "DBPassword" }
223     }
224     },
225     "Events": {
226     "PutResource": {
227     "Type": "Api",

```

```

228         "Properties": {
229             "Path": "/repositories/{repositoryId}/branches/{branchId}",
230             "Method": "GET"
231         }
232     }
233 }
234 },
235 },
236
237 "CommitChanges" : {
238     "Type" : "AWS::Serverless::Function",
239     "Properties": {
240         "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::CommitChangesAsync",
241         "Runtime": "dotnetcore2.1",
242         "CodeUri": "",
243         "Description": "Function to commit changes to branch",
244         "MemorySize": 256,
245         "Timeout": 30,
246         "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
247         "Policies": [ "AWSLambdaFullAccess" ],
248         "Environment" : {
249             "Variables" : {
250                 "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
251                 "RDS_DB_PORT" : { "Ref" : "DBPort" },
252                 "RDS_DB_NAME" : { "Ref" : "DBName" },
253                 "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
254                 "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" },
255                 "S3_REGION" : { "Ref" : "AWS::Region" }
256             }
257         },
258         "Events": {
259             "PutResource": {
260                 "Type": "Api",
261                 "Properties": {
262                     "Path": "/repositories/{repositoryId}/branches/{branchId}",
263                     "Method": "PUT"
264                 }
265             }
266         }
267     },
268 },
269
270 "GetObject" : {
271     "Type" : "AWS::Serverless::Function",
272     "Properties": {
273         "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GetObjectAsync",
274         "Runtime": "dotnetcore2.1",
275         "CodeUri": "",
276         "Description": "Function to get a file within the branch",
277         "MemorySize": 256,
278         "Timeout": 30,
279         "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
280         "Policies": [ "AWSLambdaFullAccess" ],
281         "Environment" : {
282             "Variables" : {
283                 "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
284                 "RDS_DB_PORT" : { "Ref" : "DBPort" },
285                 "RDS_DB_NAME" : { "Ref" : "DBName" },
286                 "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
287                 "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" },
288                 "S3_REGION" : { "Ref" : "AWS::Region" }
289             }
290         },
291         "Events": {
292             "PutResource": {
293                 "Type": "Api",
294                 "Properties": {
295                     "Path": "/repositories/{repositoryId}/branches/{branchId}/object",
296                     "Method": "GET"
297                 }
298             }
299         }
300     },
301 },
302
303 "GetVersions" : {

```

```

304     "Type" : "AWS::Serverless::Function" ,
305     "Properties": {
306         "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GetVersionsAsync" ,
307         "Runtime": "dotnetcore2.1" ,
308         "CodeUri": "" ,
309         "Description": "Function to get a list of versions within a branch" ,
310         "MemorySize": 256 ,
311         "Timeout": 30 ,
312         "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role" ,
313         "Policies": [ "AWSLambdaFullAccess" ] ,
314         "Environment" : {
315             "Variables" : {
316                 "RDS.DB.HOSTNAME" : { "Ref" : "DBHostname" } ,
317                 "RDS.DB.PORT" : { "Ref" : "DBPort" } ,
318                 "RDS.DB.NAME" : { "Ref" : "DBName" } ,
319                 "RDS.DB.USERNAME" : { "Ref" : "DBUsername" } ,
320                 "RDS.DB.PASSWORD" : { "Ref" : "DBPassword" }
321             }
322         } ,
323         "Events": {
324             "PutResource": {
325                 "Type": "Api" ,
326                 "Properties": {
327                     "Path": "/repositories/{repositoryId}/branches/{branchId}/versions" ,
328                     "Method": "GET"
329                 }
330             }
331         }
332     }
333 },
334
335 "GetVersion" : {
336     "Type" : "AWS::Serverless::Function" ,
337     "Properties": {
338         "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GetVersionAsync" ,
339         "Runtime": "dotnetcore2.1" ,
340         "CodeUri": "" ,
341         "Description": "Function to get a list of files within a version of a branch" ,
342         "MemorySize": 256 ,
343         "Timeout": 30 ,
344         "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role" ,
345         "Policies": [ "AWSLambdaFullAccess" ] ,
346         "Environment" : {
347             "Variables" : {
348                 "RDS.DB.HOSTNAME" : { "Ref" : "DBHostname" } ,
349                 "RDS.DB.PORT" : { "Ref" : "DBPort" } ,
350                 "RDS.DB.NAME" : { "Ref" : "DBName" } ,
351                 "RDS.DB.USERNAME" : { "Ref" : "DBUsername" } ,
352                 "RDS.DB.PASSWORD" : { "Ref" : "DBPassword" }
353             }
354         } ,
355         "Events": {
356             "PutResource": {
357                 "Type": "Api" ,
358                 "Properties": {
359                     "Path": "/repositories/{repositoryId}/branches/{branchId}/versions/{versionId}" ,
360                     "Method": "GET"
361                 }
362             }
363         }
364     }
365 },
366
367 "GetPastObject" : {
368     "Type" : "AWS::Serverless::Function" ,
369     "Properties": {
370         "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GetPastObjectAsync" ,
371         "Runtime": "dotnetcore2.1" ,
372         "CodeUri": "" ,
373         "Description": "Function to get a file from a non-current version of a branch" ,
374         "MemorySize": 256 ,
375         "Timeout": 30 ,
376         "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role" ,
377         "Policies": [ "AWSLambdaFullAccess" ] ,
378         "Environment" : {
379             "Variables" : {

```

```

380     "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
381     "RDS_DB_PORT" : { "Ref" : "DBPort" },
382     "RDS_DB_NAME" : { "Ref" : "DBName" },
383     "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
384     "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" },
385     "S3_REGION" : { "Ref" : "AWS::Region" }
386   },
387   },
388   "Events": {
389     "PutResource": {
390       "Type": "Api",
391       "Properties": {
392         "Path": "/repositories/{repositoryId}/branches/{branchId}/versions/{versionId}/object",
393         "Method": "GET"
394       }
395     }
396   }
397 },
398 },
399
400 "MergeBranch" : {
401   "Type" : "AWS::Serverless::Function",
402   "Properties": {
403     "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::MergeBranchAsync",
404     "Runtime": "dotnetcore2.1",
405     "CodeUri": "",
406     "Description": "Function to merge a branch with its parent",
407     "MemorySize": 256,
408     "Timeout": 30,
409     "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
410     "Policies": [ "AWSLambdaFullAccess" ],
411     "Environment" : {
412       "Variables" : {
413         "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
414         "RDS_DB_PORT" : { "Ref" : "DBPort" },
415         "RDS_DB_NAME" : { "Ref" : "DBName" },
416         "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
417         "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" },
418         "S3_REGION" : { "Ref" : "AWS::Region" }
419       }
420     },
421     "Events": {
422       "PutResource": {
423         "Type": "Api",
424         "Properties": {
425           "Path": "/repositories/{repositoryId}/branches/{branchId}/merge",
426           "Method": "PUT"
427         }
428       }
429     }
430   }
431 },
432
433 "LockBranch" : {
434   "Type" : "AWS::Serverless::Function",
435   "Properties": {
436     "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::LockBranchAsync",
437     "Runtime": "dotnetcore2.1",
438     "CodeUri": "",
439     "Description": "Function to put a lock on a branch",
440     "MemorySize": 256,
441     "Timeout": 30,
442     "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
443     "Policies": [ "AWSLambdaFullAccess" ],
444     "Environment" : {
445       "Variables" : {
446         "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
447         "RDS_DB_PORT" : { "Ref" : "DBPort" },
448         "RDS_DB_NAME" : { "Ref" : "DBName" },
449         "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
450         "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" }
451       }
452     },
453     "Events": {
454       "PutResource": {
455         "Type": "Api",

```

```

456         "Properties": {
457             "Path": "/repositories/{repositoryId}/branches/{branchId}/lock",
458             "Method": "PUT"
459         }
460     }
461 }
462 }
463 },
464
465 "UnlockBranch" : {
466     "Type" : "AWS::Serverless::Function",
467     "Properties": {
468         "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::UnlockBranchAsync",
469         "Runtime": "dotnetcore2.1",
470         "CodeUri": "",
471         "Description": "Function to remove the lock from a branch (only works if the same user set the
lock)",
472         "MemorySize": 256,
473         "Timeout": 30,
474         "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
475         "Policies": [ "AWSLambdaFullAccess" ],
476         "Environment" : {
477             "Variables" : {
478 "RDS.DB.HOSTNAME" : { "Ref" : "DBHostname" },
479 "RDS.DB.PORT" : { "Ref" : "DBPort" },
480 "RDS.DB.NAME" : { "Ref" : "DBName" },
481 "RDS.DB.USERNAME" : { "Ref" : "DBUsername" },
482 "RDS.DB.PASSWORD" : { "Ref" : "DBPassword" }
483             }
484         },
485         "Events": {
486             "PutResource": {
487                 "Type": "Api",
488                 "Properties": {
489                     "Path": "/repositories/{repositoryId}/branches/{branchId}/unlock",
490                     "Method": "PUT"
491                 }
492             }
493         }
494     }
495 },
496
497 "BreakLock" : {
498     "Type" : "AWS::Serverless::Function",
499     "Properties": {
500         "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::BreakLockAsync",
501         "Runtime": "dotnetcore2.1",
502         "CodeUri": "",
503         "Description": "Function to force-remove a lock from a branch (anyone can do this)",
504         "MemorySize": 256,
505         "Timeout": 30,
506         "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
507         "Policies": [ "AWSLambdaFullAccess" ],
508         "Environment" : {
509             "Variables" : {
510 "RDS.DB.HOSTNAME" : { "Ref" : "DBHostname" },
511 "RDS.DB.PORT" : { "Ref" : "DBPort" },
512 "RDS.DB.NAME" : { "Ref" : "DBName" },
513 "RDS.DB.USERNAME" : { "Ref" : "DBUsername" },
514 "RDS.DB.PASSWORD" : { "Ref" : "DBPassword" }
515             }
516         },
517         "Events": {
518             "PutResource": {
519                 "Type": "Api",
520                 "Properties": {
521                     "Path": "/repositories/{repositoryId}/branches/{branchId}/force-unlock",
522                     "Method": "PUT"
523                 }
524             }
525         }
526     }
527 },
528
529 "RequestPermission" : {
530     "Type" : "AWS::Serverless::Function",

```

```

531     "Properties": {
532         "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::RequestPermissionAsync",
533         "Runtime": "dotnetcore2.1",
534         "CodeUri": "",
535         "Description": "Function to request permission to access this repository",
536         "MemorySize": 256,
537         "Timeout": 30,
538         "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
539         "Policies": [ "AWSLambdaFullAccess" ],
540         "Environment": {
541             "Variables": {
542                 "RDS_DB_HOSTNAME": { "Ref": "DBHostname" },
543                 "RDS_DB_PORT": { "Ref": "DBPort" },
544                 "RDS_DB_NAME": { "Ref": "DBName" },
545                 "RDS_DB_USERNAME": { "Ref": "DBUsername" },
546                 "RDS_DB_PASSWORD": { "Ref": "DBPassword" }
547             }
548         },
549         "Events": {
550             "PutResource": {
551                 "Type": "Api",
552                 "Properties": {
553                     "Path": "/repositories/{repositoryId}/request-permission",
554                     "Method": "PUT"
555                 }
556             }
557         }
558     },
559 },
560
561 "GetPermissionRequests" : {
562     "Type" : "AWS::Serverless::Function",
563     "Properties": {
564         "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::
565         GetPermissionRequestsAsync",
566         "Runtime": "dotnetcore2.1",
567         "CodeUri": "",
568         "Description": "Function to get a list of pending requests to access repositories owner by the
569         user submitting this request",
570         "MemorySize": 256,
571         "Timeout": 30,
572         "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
573         "Policies": [ "AWSLambdaFullAccess" ],
574         "Environment": {
575             "Variables": {
576                 "RDS_DB_HOSTNAME": { "Ref": "DBHostname" },
577                 "RDS_DB_PORT": { "Ref": "DBPort" },
578                 "RDS_DB_NAME": { "Ref": "DBName" },
579                 "RDS_DB_USERNAME": { "Ref": "DBUsername" },
580                 "RDS_DB_PASSWORD": { "Ref": "DBPassword" }
581             }
582         },
583         "Events": {
584             "PutResource": {
585                 "Type": "Api",
586                 "Properties": {
587                     "Path": "/permission-requests",
588                     "Method": "GET"
589                 }
590             }
591         }
592     },
593 },
594
595 "GrantPermission" : {
596     "Type" : "AWS::Serverless::Function",
597     "Properties": {
598         "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GrantPermissionAsync",
599         "Runtime": "dotnetcore2.1",
600         "CodeUri": "",
601         "Description": "Function to grant the requested access",
602         "MemorySize": 256,
603         "Timeout": 30,
604         "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
605         "Policies": [ "AWSLambdaFullAccess" ],
606         "Environment": {

```

```

605     "Variables" : {
606         "RDS.DB.HOSTNAME" : { "Ref" : "DBHostname" },
607         "RDS.DB.PORT" : { "Ref" : "DBPort" },
608         "RDS.DB.NAME" : { "Ref" : "DBName" },
609         "RDS.DB.USERNAME" : { "Ref" : "DBUsername" },
610         "RDS.DB.PASSWORD" : { "Ref" : "DBPassword" }
611     },
612     "Events" : {
613         "PutResource" : {
614             "Type" : "Api",
615             "Properties" : {
616                 "Path" : "/permission-requests/{requestId}/grant",
617                 "Method" : "PUT"
618             }
619         }
620     }
621 },
622 },
623 },
624
625 "DenyPermission" : {
626     "Type" : "AWS::Serverless::Function",
627     "Properties" : {
628         "Handler" : "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::DenyPermissionAsync",
629         "Runtime" : "dotnetcore2.1",
630         "CodeUri" : "",
631         "Description" : "Function to deny the requested access",
632         "MemorySize" : 256,
633         "Timeout" : 30,
634         "Role" : "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
635         "Policies" : [ "AWSLambdaFullAccess" ],
636         "Environment" : {
637             "Variables" : {
638                 "RDS.DB.HOSTNAME" : { "Ref" : "DBHostname" },
639                 "RDS.DB.PORT" : { "Ref" : "DBPort" },
640                 "RDS.DB.NAME" : { "Ref" : "DBName" },
641                 "RDS.DB.USERNAME" : { "Ref" : "DBUsername" },
642                 "RDS.DB.PASSWORD" : { "Ref" : "DBPassword" }
643             }
644         },
645         "Events" : {
646             "PutResource" : {
647                 "Type" : "Api",
648                 "Properties" : {
649                     "Path" : "/permission-requests/{requestId}/deny",
650                     "Method" : "PUT"
651                 }
652             }
653         }
654     }
655 },
656 },
657 }

```

```

1 using System;
2 using System.Collections.Generic;
3 using System.Net;
4 using System.Linq;
5 using System.Threading.Tasks;
6 using Amazon;
7 using Amazon.S3;
8 using Amazon.S3.Model;
9 using Amazon.S3.Util;
10 using System.IO;
11
12 namespace ReVersionVCS_API_Lambdas
13 {
14     public class S3Operations
15     {
16
17         private IAmazonS3 s3Client;
18
19         public const string bucketPrefix = "ReVersionVCS";
20
21         private RegionEndpoint bucketRegion;
22
23
24         public S3Operations(string regionName)
25         {
26             bucketRegion = RegionEndpoint.GetBySystemName(regionName);
27         }
28
29         public async Task<PutBucketResponse> CreateS3BucketAsync(string repoName, string branchName)
30         {
31             string bucketName = GetBucketName(repoName, branchName);
32             using (s3Client = new AmazonS3Client(bucketRegion))
33             {
34                 var responseTask = await CreateBucketAsync(bucketName);
35                 return responseTask;
36             }
37         }
38
39         public async Task<HttpStatusCode> DeleteBranchBucketsByRepoAsync(string repoName, List<string>
branchNames)
40         {
41             using (s3Client = new AmazonS3Client(bucketRegion))
42             {
43                 try
44                 {
45                     foreach (var branch in branchNames)
46                     {
47                         string bucketName = GetBucketName(repoName, branch);
48                         List<string> objects = await GetObjectKeysAsync(bucketName);
49                         foreach (var item in objects)
50                         {
51                             await s3Client.DeleteObjectAsync(bucketName, item);
52                         }
53                         await s3Client.DeleteBucketAsync(bucketName);
54                     }
55                     return HttpStatusCode.OK;
56                 }
57                 catch (Exception)
58                 {
59                     return HttpStatusCode.InternalServerError;
60                 }
61             }
62         }
63
64         public async Task<HttpStatusCode> UpdateS3ObjectsAsync(List<FileData> files)
65         {
66             using (s3Client = new AmazonS3Client(bucketRegion))
67             {
68                 try
69                 {
70                     foreach (var item in files)
71                     {
72                         var response = await UpdateObjectAsync(item);
73                     }
74                     return HttpStatusCode.OK;
75                 }
76             }
77         }
78     }
79 }

```



```

76         catch (AmazonS3Exception e)
77         {
78             throw new AmazonS3Exception("Error writing an object", e);
79         }
80         catch (Exception e)
81         {
82             throw new Exception("Unknown error encountered on server", e);
83             //return HttpStatusCode.InternalServerError;
84         }
85     }
86 }
87
88 public async Task<string> GetTextFromObjectAsync(string repoName, string branchName, string
objectKey)
89 {
90     using (s3Client = new AmazonS3Client(bucketRegion))
91     {
92         var bucketName = GetBucketName(repoName, branchName);
93         try
94         {
95             return await GetObjectContentAsync(repoName, branchName, objectKey);
96         }
97         catch (Exception)
98         {
99             return null;
100         }
101     }
102 }
103 /*
104 public async Task MergeIntoBucketAsync(string repoName, string branchName, string parentBranchName
)
105 {
106     using (s3Client = new AmazonS3Client(bucketRegion))
107     {
108         // TODO TODO TODO TODO
109
110
111
112
113         throw new NotImplementedException();
114     }
115 }
116 */
117
118
119
120 private async Task<PutBucketResponse> CreateBucketAsync(string bucketName)
121 {
122     try
123     {
124         if (!(await AmazonS3Util.DoesS3BucketExistAsync(s3Client, bucketName)))
125         {
126             var putBucketRequest = new PutBucketRequest
127             {
128                 BucketName = bucketName,
129                 UseClientRegion = true
130             };
131
132             PutBucketResponse putBucketResponse = await s3Client.PutBucketAsync(putBucketRequest);
133             return putBucketResponse;
134         }
135         else
136         {
137             return new PutBucketResponse { HttpStatusCode = HttpStatusCode.Conflict };
138         }
139     }
140     catch (Exception)
141     {
142         return new PutBucketResponse { HttpStatusCode = HttpStatusCode.InternalServerError };
143     }
144 }
145
146 private async Task<string> FindBucketLocationAsync(string repositoryName, string branchName)
147 {
148     var request = new GetBucketLocationRequest()
149
```

```

150     {
151         BucketName = GetBucketName(repositoryName, branchName)
152     };
153     GetBucketLocationResponse response = await s3Client.GetBucketLocationAsync(request);
154     return response.Location.ToString();
155 }
156
157 private async Task<List<string>> GetObjectKeysAsync(string bucketName, string prefix = null)
158 {
159     ListObjectsRequest request =
160         (string.IsNullOrEmpty(prefix)) ?
161         new ListObjectsRequest { BucketName = bucketName } :
162         new ListObjectsRequest { BucketName = bucketName, Prefix = prefix };
163
164     List<string> bucketList = new List<string>();
165     ListObjectsResponse response;
166     do
167     {
168         response = await s3Client.ListObjectsAsync(request);
169         bucketList.AddRange(response.S3Objects.Select(x => x.Key));
170         request.Marker = response.NextMarker;
171     } while (response.IsTruncated);
172
173     return bucketList;
174 }
175
176 private async Task<PutObjectResponse> UpdateObjectAsync(FileData file)
177 {
178     try
179     {
180         var findResponse = await FindBucketLocationAsync(file.RepositoryName, file.BranchName);
181     }
182     catch (Exception e)
183     {
184         throw new Exception($"did not find bucket: {GetBucketName(file.RepositoryName, file.BranchName)}", e);
185     }
186     var request = new PutObjectRequest
187     {
188         BucketName = GetBucketName(file.RepositoryName, file.BranchName),
189         Key = file.ObjectKey,
190         ContentBody = file.Content
191     };
192
193     PutObjectResponse response = await s3Client.PutObjectAsync(request);
194     return response;
195 }
196
197 private async Task<string> GetObjectContentAsync(string repoName, string branchName, string
objectKey)
198 {
199     GetObjectRequest request = new GetObjectRequest
200     {
201         BucketName = GetBucketName(repoName, branchName),
202         Key = objectKey
203     };
204     using (GetObjectResponse response = await s3Client.GetObjectAsync(request))
205     using (Stream responseStream = response.ResponseStream)
206     using (StreamReader reader = new StreamReader(responseStream))
207     {
208         string responseBody = await reader.ReadToEndAsync();
209
210         return responseBody;
211     }
212 }
213
214 private string GetBucketName(string repoName, string branchName)
215     => (bucketPrefix + '.' + repoName + '.' + branchName).ToLower();
216
217 }
218
219 }

```

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5 using ReVersionVCS_API.Lambdas.Models;
6
7 namespace ReVersionVCS_API.Lambdas
8 {
9     public static partial class SQLOperations
10    {
11        public static List<RepositoryLookup> QueryRepositories(ReVersion.DatabaseContext db) =>
12            (from repository in db.Repositories
13             join user in db.Users on repository.Owner equals user.Id
14             select new RepositoryLookup { Name = repository.Name, Owner = user.UserName })
15            .ToList();
16
17        // NOT FULLY TESTED
18        // updated version not yet tested, should work fine
19        // (except maybe the case with the "versionId" parameter specified
20        public static HierarchyNode QueryHierarchy(ReVersion.DatabaseContext db, string repoName, string
branchName, string versionId = null)
21        {
22            var latestHierarchyJson =
23                from repository in db.Repositories
24                join branch in db.Branches on repository.Id equals branch.RepositoryId
25                where repository.Name == repoName
26                && branch.Name == branchName
27                select branch.LatestFileHierarchy;
28
29            var versionHierarchyJson =
30                from repository in db.Repositories
31                join branch in db.Branches on repository.Id equals branch.RepositoryId
32                join version in db.Versions on branch.Id equals version.BranchId
33                where repository.Name.Equals(repoName)
34                && branch.Name.Equals(branchName)
35                && version.VersionNumber.Equals(versionId ?? "-1")
36                select version.FileHierarchy;
37
38            string hierarchyJson = string.IsNullOrEmpty(versionId) ?
39                latestHierarchyJson.Single() :
40                versionHierarchyJson.Single();
41
42            FileHierarchyData hierarchyData = new FileHierarchyData(hierarchyJson);
43
44            return hierarchyData.GetHierarchyList();
45        }
46
47        public static List<BranchLookup> QueryBranches(ReVersion.DatabaseContext db, string repoName) =>
48            (from repository in db.Repositories
49             join branch in db.Branches
50             on repository.Id equals branch.RepositoryId
51             where repository.Name == repoName
52             select new BranchLookup { Locked = branch.Locked, BranchName = branch.Name })
53            .ToList();
54
55        public static int QueryBranchId(ReVersion.DatabaseContext db, string repoName, string branchName)
=>
56            (from repository in db.Repositories
57             join branch in db.Branches
58             on repository.Id equals branch.RepositoryId
59             where repository.Name == repoName
60             && branch.Name == branchName
61             select branch.Id)
62            .Single();
63
64        public static List<int> QueryVersions(ReVersion.DatabaseContext db, string repoName, string
branchName) =>
65            (from repository in db.Repositories
66             join branch in db.Branches on repository.Id equals branch.RepositoryId
67             where repository.Name == repoName && branch.Name == branchName
68             join version in db.Versions on branch.Id equals version.BranchId
69             select version.VersionNumber)
70            .ToList();
71
72        public static bool UserCanAccessRepository(ReVersion.DatabaseContext db, string username, string
repoName) =>

```

```

73     QueryPermissions(db, username).Contains(repoName);
74
75     public static List<string> QueryPermissions(ReVersion.DatabaseContext db, string username) =>
76     (from user in db.Users
77      where user.UserName == username
78      join permission in db.RepositoryPermissions on user.Id equals permission.PermittedUser
79      join repository in db.Repositories on permission.RepositoryId equals repository.Id
80      select repository.Name)
81     .ToList();
82
83     public static List<PermissionLookup> QueryPermissionRequests(ReVersion.DatabaseContext db, string
username)
84     {
85         var query =
86             from user in db.Users
87             where user.UserName == username
88             join repository in db.Repositories on user.Id equals repository.Owner
89             join request in db.PermissionRequests on repository.Id equals request.RepositoryId
90             join log in db.EventLogs on request.EventId equals log.Id
91             join requestUser in db.Users on log.UserId equals requestUser.Id
92             select new PermissionLookup
93             {
94                 RequestId = request.Id,
95                 RequestingUser = requestUser.UserName,
96                 RepositoryName = repository.Name,
97                 Message = log.Message,
98                 LogTimestamp = log.LoggedAt
99             };
100
101         return query.ToList();
102     }
103
104     public static string QueryRepositoryNameFromRequest(ReVersion.DatabaseContext db, int requestId)
105     {
106         int trunkId = QueryMainTrunkFromRequest(db, requestId);
107         return
108             (from repository in db.Repositories
109              join request in db.PermissionRequests on repository.Id equals request.RepositoryId
110              where request.Id.Equals(requestId)
111              select repository.Name)
112             .Single();
113     }
114
115     public static int QueryMainTrunkFromRequest(ReVersion.DatabaseContext db, int requestId) =>
116     (from request in db.PermissionRequests
117      where request.Id == requestId
118      join log in db.EventLogs on request.EventId equals log.Id
119      join branch in db.Branches on log.BranchId equals branch.Id
120      select branch.Id)
121     .Single();
122
123     public static int QueryLastEventIdByUser(ReVersion.DatabaseContext db, string username) =>
124     (from log in db.EventLogs
125      join user in db.Users on log.UserId equals user.Id
126      where user.UserName == username
127      orderby log.LoggedAt descending, log.Id descending
128      select log.Id)
129     .First();
130
131     public static int QueryUserFromRequest(ReVersion.DatabaseContext db, int requestId) =>
132     (from request in db.PermissionRequests
133      where request.Id == requestId
134      join log in db.EventLogs on request.EventId equals log.Id
135      select log.UserId)
136     .Single();
137
138     public static bool BranchIsLocked(ReVersion.DatabaseContext db, string repoName, string branchName
) =>
139     (from repository in db.Repositories
140      join branch in db.Branches on repository.Id equals branch.RepositoryId
141      where repository.Name == repoName
142      && branch.Name == branchName
143      select branch.Locked)
144     .Single();
145
146     // Untested

```

```

147     public static bool PermissionRequestIsLogged(ReVersion.DatabaseContext db, string repoName, string
username) =>
148         (from request in db.PermissionRequests
149         join repository in db.Repositories on request.RepositoryId equals repository.Id
150         join branch in db.Branches on repository.Id equals branch.RepositoryId
151         join log in db.EventLogs on branch.Id equals log.BranchId
152         join user in db.Users on log.UserId equals user.Id
153         where repository.Name.Equals(repoName)
154         && branch.Name.Equals("main")
155         && user.UserName.Equals(username)
156         && log.Type.Equals("request_permission")
157         select log.Id)
158         .Any();
159
160     public static bool UserOwnsRepository(ReVersion.DatabaseContext db, string repo, string username)
=>
161         (from repository in db.Repositories
162         join user in db.Users on repository.Owner equals user.Id
163         where repository.Name.Equals(repo)
164         && user.UserName.Equals(username)
165         select user.Id).Any();
166
167     public static LockData QueryLockEventByBranch(ReVersion.DatabaseContext db, string repoName,
string branchName) =>
168         (from repository in db.Repositories
169         join branch in db.Branches on repository.Id equals branch.RepositoryId
170         join log in db.EventLogs on branch.Id equals log.BranchId
171         join user in db.Users on log.UserId equals user.Id
172         where repository.Name.Equals(repoName)
173         && branch.Name.Equals(branchName)
174         orderby log.LoggedAt descending, log.Id descending
175         select new LockData
176         {
177             UserName = user.UserName,
178             Message = log.Message,
179             Timestamp = log.LoggedAt,
180             LockedBranchId = branch.Name
181         }).First();
182
183     public static bool UserCanAccessBranch(ReVersion.DatabaseContext db, string repoName, string
branchName, string username) =>
184         (from repository in db.Repositories
185         join branch in db.Branches on repository.Id equals branch.RepositoryId
186         join permission in db.RepositoryPermissions on repository.Id equals permission.RepositoryId
187         join user in db.Users on permission.PermittedUser equals user.Id
188         where repository.Name.Equals(repoName)
189         && branch.Name.Equals(branchName)
190         && user.UserName.Equals(username)
191         select user.Id).Any();
192
193     public async static Task<bool> RequestExistsAsync(ReVersion.DatabaseContext db, int requestId) =>
194         await db.PermissionRequests.FindAsync(requestId) != null;
195
196     public static bool RepositoryExists(ReVersion.DatabaseContext db, string repoName) =>
197         (from repository in db.Repositories
198         where repository.Name.Equals(repoName)
199         select repository.Id).Any();
200
201     public static bool BranchExists(ReVersion.DatabaseContext db, string repoName, string branchName)
=>
202         (from repository in db.Repositories
203         join branch in db.Branches on repository.Id equals branch.RepositoryId
204         where repository.Name == repoName
205         && branch.Name == branchName
206         select branch.Id).Any();
207
208     public static bool UsernameExists(ReVersion.DatabaseContext db, string username) =>
209         (from user in db.Users
210         where user.UserName == username
211         select user.Id).Any();
212
213     public static bool VersionExists(ReVersion.DatabaseContext db, string repoName, string branchName,
string versionName) =>
214         QueryVersions(db, repoName, branchName).Where(x => x.Equals(versionName)).Any();
215 }
216 }

```

```

1 using System.Collections.Generic;
2 using System.Linq;
3 using ReVersionVCS_API.Lambdas.Models;
4
5 namespace ReVersionVCS_API.Lambdas
6 {
7     public static partial class SQLOperations
8     {
9
10         public enum LockState
11         {
12             AlreadyLockedConflict,
13             AlreadyUnlockedConflict,
14             SuccessfulLockOperation,
15             LockedByDifferentUser
16         }
17
18         public static void InsertIntoRepoTable(ReVersion_DatabaseContext db, string repoName, string
username) =>
19             db.Repositories.AddAsync(new Repository
20             {
21                 Name = repoName,
22                 Owner = (from user in db.Users
23                         where user.UserName == username
24                         select user.Id)
25                         .Single()
26             });
27
28         public static void InsertIntoBranchTable(ReVersion_DatabaseContext db, string repoName, string
branchName) =>
29             db.Branches.Add(
30                 (from repository in db.Repositories
31                  where repository.Name == repoName
32                  select new Branch
33                  {
34                      RepositoryId = repository.Id,
35                      Name = branchName
36                  })
37                 .Single()
38             );
39
40         public static void InsertIntoRepoPermissionsTable(ReVersion_DatabaseContext db, string repoName,
string username) =>
41             db.RepositoryPermissions.Add(
42                 (from repository in db.Repositories
43                  where repository.Name == repoName
44                  from user in db.Users
45                  where user.UserName == username
46                  select new RepositoryPermission
47                  {
48                      PermittedUser = user.Id,
49                      RepositoryId = repository.Id
50                  })
51                 .Single()
52             );
53
54         public static void InsertIntoEventLog(ReVersion_DatabaseContext db, string repoName, string
branchName, string username, string message, string type)
55         {
56             db.EventLogs.Add(
57                 (from user in db.Users
58                  from branch in db.Branches
59                  join repository in db.Repositories on branch.RepositoryId equals repository.Id
60                  where user.UserName == username
61                  && branch.Name == branchName
62                  && repository.Name == repoName
63                  select new EventLog
64                  {
65                      Type = type,
66                      BranchId = branch.Id,
67                      UserId = user.Id,
68                      Message = message
69                  })
70                 .Single()
71             );
72         }

```

```

73     public static void DeleteAllFromBranchesTable(ReVersion.DatabaseContext db, string repoName) =>
74         db.Branches.RemoveRange(
75             from branch in db.Branches
76             join repository in db.Repositories on branch.RepositoryId equals repository.Id
77             where repository.Name == repoName
78             select branch);
79
80     public static void DeleteAllFromVersionsTable(ReVersion.DatabaseContext db, string repoName) =>
81         db.Versions.RemoveRange(
82             from repository in db.Repositories
83             where repository.Name == repoName
84             join branch in db.Branches on repository.Id equals branch.RepositoryId
85             join version in db.Versions on branch.Id equals version.BranchId
86             select version);
87
88     public static void DeleteAllFromPermissionsTable(ReVersion.DatabaseContext db, string repoName) =>
89         db.RepositoryPermissions.RemoveRange(
90             from repository in db.Repositories
91             where repository.Name == repoName
92             join permission in db.RepositoryPermissions on repository.Id equals permission.
93             RepositoryId
94             select permission);
95
96     public static void DeleteAllFromPermissionRequestsTable(ReVersion.DatabaseContext db, string
97     repoName) =>
98         db.PermissionRequests.RemoveRange(
99             from repository in db.Repositories
100             where repository.Name == repoName
101             join request in db.PermissionRequests on repository.Id equals request.RepositoryId
102             select request);
103
104     public static void DeleteFromRepositoryTable(ReVersion.DatabaseContext db, string repoName) =>
105         db.Repositories.RemoveRange(
106             from repository in db.Repositories
107             where repository.Name == repoName
108             select repository);
109
110     public static LockState AcquireLock(ReVersion.DatabaseContext db, string repoName, string
111     branchName)
112     {
113         Branch branchEntity =
114             (from repository in db.Repositories
115              where repository.Name == repoName
116              join branch in db.Branches on repository.Id equals branch.RepositoryId
117              where branch.Name == branchName
118              select branch)
119             .Single();
120
121         if (branchEntity.Locked)
122         {
123             return LockState.AlreadyLockedConflict;
124         }
125
126         branchEntity.Locked = true;
127         return LockState.SuccessfulLockOperation;
128     }
129
130     public static void InsertIntoVersionsTable(ReVersion.DatabaseContext db, VersionData data) =>
131         db.Versions.Add((from thisBranch in db.Branches
132                          from parentBranch in db.Branches
133                          join repository in db.Repositories on thisBranch.RepositoryId equals
134                          repository.Id
135                          where thisBranch.Name == data.BranchName
136                          && parentBranch.Name == (data.NewBranch ? data.ParentBranchName : data.
137                          BranchName)
138                          && repository.Name == data.RepositoryName
139                          select new Version
140                          {
141                              VersionNumber = thisBranch.VersionNumber,
142                              BranchId = thisBranch.Id,
143                              ParentBranch = parentBranch.Id,
144                              RollbackDelta = data.DeltaContent,
145                              FileHierarchy = data.FileHierarchyString(),
146                              UpdateEventId = data.EventId
147                          }));

```

```

144         .Single());
145
146     public static void UpdateHierarchyDatumInBranchesTable(ReVersion.DatabaseContext db, string
147     repoName, string branchName, string hierarchyDatum)
148     {
149         Branch branchEntity =
150             (from repository in db.Repositories
151              where repository.Name == repoName
152              join branch in db.Branches on repository.Id equals branch.RepositoryId
153              where branch.Name == branchName
154              select branch)
155             .Single();
156
157         branchEntity.VersionNumber += 1;
158         branchEntity.LatestFileHierarchy = hierarchyDatum;
159     }
160
161     public static LockState ReleaseLock(ReVersion.DatabaseContext db, string repoName, string
162     branchName)
163     {
164         Branch branchEntity =
165             (from repository in db.Repositories
166              where repository.Name == repoName
167              join branch in db.Branches on repository.Id equals branch.RepositoryId
168              where branch.Name == branchName
169              select branch)
170             .Single();
171
172         if (!branchEntity.Locked)
173         {
174             return LockState.AlreadyUnlockedConflict;
175         }
176
177         branchEntity.Locked = false;
178         return LockState.SuccessfulLockOperation;
179     }
180
181     public static LockState SafeReleaseLock(ReVersion.DatabaseContext db, string repoName, string
182     branchName, string username)
183     {
184         Branch branchEntity =
185             (from repository in db.Repositories
186              join branch in db.Branches on repository.Id equals branch.RepositoryId
187              where repository.Name == repoName
188              && branch.Name == branchName
189              select branch)
190             .Single();
191
192         if (!branchEntity.Locked)
193         {
194             return LockState.AlreadyUnlockedConflict;
195         }
196
197         string usernameQuery =
198             (from log in db.EventLogs
199              where branchEntity.Id == log.BranchId
200              && log.Type == "place_lock"
201              join user in db.Users on log.UserId equals user.Id
202              orderby log.LoggedAt descending, log.Id descending
203              select user.UserName)
204             .First();
205
206         if (username != usernameQuery)
207         {
208             return LockState.LockedByDifferentUser;
209         }
210
211         branchEntity.Locked = false;
212         return LockState.SuccessfulLockOperation;
213     }
214
215     public static void InsertIntoPermissionRequestTable(ReVersion.DatabaseContext db, string repoName,
216     int eventId) =>
217     {
218         db.PermissionRequests.Add(new PermissionRequest
219         {
220             EventId = eventId,
221             RepositoryId = (from repository in db.Repositories
222                            where repository.Name == repoName
223                            select repository.Id).Single()
224         });
225     }

```



```

216
217     public static void DeleteFromPermissionRequestTable(ReVersion_DatabaseContext db, int requestId)
218     =>
219         db.PermissionRequests.Remove(db.PermissionRequests.Find(requestId));
220
221     public static void InsertIntoUsersTable(ReVersion_DatabaseContext db, string username) =>
222         db.Users.Add(new User
223         {
224             UserName = username
225         });
226 }

```

```

1 <!DOCTYPE html>
2
3 <html lang="en" xmlns="http://www.w3.org/1999/xhtml">
4 <head>
5   <meta charset="utf-8" />
6   <meta name="viewport" content="width=device-width, initial-scale=1" />
7   <title>ReVersion VCS</title>
8   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/normalize/8.0.1/normalize.css" />
9   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.7.4/css/bulma.css" />
10  <link rel="stylesheet" href="index.css" />
11
12  <script src="https://cdn.jsdelivr.net/npm/vue"></script>
13 </head>
14 <body>
15   <section class="section">
16     <div class="container has-background-primary column is-8 is-offset-2"
17       style="text-align:center">
18       <h1 class="title">
19         ReVersion VCS
20       </h1>
21       <p class="subtitle">
22         Version Control <i>as a</i><strong>service</strong>!
23       </p>
24     </div>
25   </section>
26
27   <section id="userData" class="section columns" v-if:"show">
28     <div class="field column">
29       <div class="column is-three-fifths is-offset-2">
30         <div class="control">
31           <input class="input is-primary" type="text" placeholder="username" />
32         </div>
33         <div class="control">
34           <input class="input is-info" type="password" placeholder="password" />
35         </div>
36       </div>
37       <div class="column">
38         <div class="columns">
39           <div class="column is-3 is-offset-3">
40             <input class="button is-primary" type="submit" value="Login" />
41           </div>
42           <div class="column is-3">
43             <a class="button is-info" href="#">Sign Up</a>
44           </div>
45         </div>
46       </div>
47     </div>
48   </section>
49
50   <section id="requestTable" class="section" v-if="show">
51     <div class="container">
52       <h1 class="title" style="text-align:center">Repository Permission Requests</h1>
53       <table class="table is-hoverable is-fullwidth">
54         <thead>
55           <tr>
56             <th scope="col">Repository Name</th>
57             <th scope="col">Requesting User</th>
58             <th scope="col">Timestamp</th>
59           </tr>
60         </thead>
61         <tbody>
62           <tr v-for="perReq in permissionRequests">
63             <td v-for="value in perReq">
64               {{value}}
65             </td>
66           </tr>
67         </tbody>
68       </table>
69     </div>
70   </section>
71
72   <section id="hierarchy" class="section">
73     <div class="container">
74       <div class="menu" id="menu">
75         <ul>

```

```

77     <input class="menuInput" type="radio" name="menu" id="archive" checked>
78     <li class="menuListItem">
79         <label for="archive" class="menuTitle"><i class="fa fa-folder"></i>Archive</label>
80         <a class="menuItem" href="#">New File</a>
81         <a class="menuItem" href="#">Open File</a>
82         <a class="menuItem" href="#">Save As...</a>
83         <a class="menuItem" href="#">Exit</a>
84     </li>
85     <input class="menuInput" type="radio" name="menu" id="edit">
86     <li class="menuListItem">
87         <label for="edit" class="menuTitle"><i class="fa fa-edit"></i>Edit</label>
88         <a class="menuItem" href="#">Copy</a>
89         <a class="menuItem" href="#">Cut</a>
90         <a class="menuItem" href="#">Paste</a>
91         <a class="menuItem" href="#">Undo</a>
92     </li>
93     <input class="menuInput" type="radio" name="menu" id="tools">
94     <li class="menuListItem">
95         <label for="tools" class="menuTitle"><i class="fa fa-gavel"></i>Tools</label>
96         <a class="menuItem" href="#">Build</a>
97         <a class="menuItem" href="#">Macros</a>
98         <a class="menuItem" href="#">Command</a>
99         <a class="menuItem" href="#">Snippets</a>
100    </li>
101    <input class="menuInput" type="radio" name="menu" id="preferences">
102    <li class="menuListItem">
103        <label for="preferences" class="menuTitle"><i class="fa fa-gears"></i>Preferences<
104    /label>
105        <a class="menuItem" href="#">Browser</a>
106        <a class="menuItem" href="#">Settings</a>
107        <a class="menuItem" href="#">Packages</a>
108        <a class="menuItem" href="#">Theme</a>
109    </li>
110    </ul>
111    </div>
112    </div>
113    </section>
114    <script src="index.js"></script>
115 </body>
116 </html>

```

```

1 @import url(https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css);
2 @import url(https://fonts.googleapis.com/css?family=Montserrat:400,700|Open+Sans:400,300);
3
4
5 body {
6     font-family: 'Montserrat', sans-serif;
7 }
8
9 #menu {
10     position: fixed;
11     display: flex;
12     align-items: flex-start;
13     width: 80%;
14     height: 100%;
15     background-color: #111;
16 }
17
18
19 li.menuListItem, ul label.menuTitle, ul, a {
20     width: 100%;
21     color: #FFF;
22     font-family: 'Montserrat', 'Open Sans', sans-serif;
23     display: block;
24     font-weight: bold;
25 }
26
27     ul label {
28         height: 35px;
29     }
30
31     ul li.menuListItem {
32         height: 35px;
33         overflow: hidden;
34         transition: all .3s;
35     }
36
37 li.menuListItem {
38     display: block;
39     background-color: #363636;
40 }
41
42 label.menuTitle {
43     font-size: 14px;
44     background: linear-gradient(#111, #2f2f2f);
45     padding: 10px 15px;
46     cursor: pointer;
47     transition: all .25s;
48 }
49
50 a.menuItem {
51     font-size: 12px;
52     text-decoration: none;
53     color: #FFF;
54     display: block;
55     padding: 10px 25px;
56     transition: all .25s;
57 }
58
59     a.menuItem:hover {
60         background-color: #444;
61         box-shadow: inset 5px 0px 0px 0px #fff;
62     }
63
64 label.menuTitle:hover {
65     text-shadow: 0px 0px 10px #fff;
66 }
67
68 input.menuInput[type="radio"] {
69     display: none;
70 }
71
72 #edit:checked + li.menuListItem, #archive:checked + li.menuListItem, #tools:checked + li.menuListItem, #
73     preferences:checked + li.menuListItem {
74     height: 179px;
75 }

```

```
76 i {  
77     margin-right: 12px;  
78 }  
79  
80 @media screen and (max-width: 600px) {  
81     #menu {  
82         width: 100%;  
83         position: relative;  
84     }  
85  
86     main {  
87         width: 100%;  
88         position: relative;  
89     }  
90 }
```

```

1 var requests = new Vue({
2   el: '#requestTable',
3   data: {
4     permissionRequests: [
5       { name: 'repo1', user: 'user1', timestamp: 'right now!' },
6       { name: 'repo2', user: 'user2', timestamp: 'idk awhile ago' },
7       { name: 'repo3', user: 'user3', timestamp: 'like forever ago!' },
8       { name: 'repo4', user: 'user4', timestamp: 'holy crap idk!' },
9       { name: 'repo5', user: 'user5', timestamp: 'ugh... yesterday?' }
10    ],
11    show: true
12  }
13 })
14
15 var login = new Vue({
16   el: '#userData',
17   data: {
18     show: true
19   }
20 })

```