```
using System;
using System.Text;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Threading.Tasks;

using Amazon.Lambda.Core;
using Amazon.Lambda.APIGatewayEvents;

using Newtonsoft.Json;
using Newtonsoft.Json.Serialization;
using ReVersionVCS_API_Lambdas.Models;
using ReVersionVCS_API_Lambdas.Response_Objects;
using ReVersionVCS_API_Lambdas.Request_Objects;
using System.Text.RegularExpressions;
using static ReVersionVCS_API_Lambdas.SQLOperations;


// Assembly attribute to enable the Lambda function's JSON input to be converted into a .NET class.
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]

namespace ReVersionVCS_API_Lambdas
{
    public partial class Functions
    {
        private readonly JsonSerializerSettings jsonSerializerSettings;

        private S3Operations s3Ops;


        const string S3_REGION_ENVIRONMENT_VARIABLE_LOOKUP = "S3_REGION";

        public const string ID_QUERY_STRING_NAME = "Id";

        /// <summary>
        /// Default constructor that Lambda will invoke.
        /// </summary>
        public Functions()
        {
            // set jsonSerializerSettings to properly handle Camel-Case Names
            //  i.e. the JSON property "fooBar" serializes to the C# property "FooBar"
            DefaultContractResolver contractResolver = new DefaultContractResolver
            {
                NamingStrategy = new CamelCaseNamingStrategy()
            };
            jsonSerializerSettings = new JsonSerializerSettings
            {
                ContractResolver = contractResolver
            };
        }


        // TODO: add authorization check and return 401 (HttpStatusCode.Unauthorized) for everything (unless this is handled by API Gatway)
        // TODO: same for 500 code (HttpStatusCode.InternalServerError)
        /// <summary>
        /// A Lambda function that gets a list of all of the repositories
        /// </summary>
        /// <param name="request"></param>
        /// <returns></returns>
        public async Task<APIGatewayProxyResponse> GetRepositoriesAsync(APIGatewayProxyRequest request, ILambdaContext context)
        {
            return
                await Task.Run(() =>
                    {
                        using (ReVersion_DatabaseContext db = new ReVersion_DatabaseContext())
                        {
                            List<RepositoryLookup> repositoryNames = QueryRepositories(db);
                            List<ResourceItem> repositoryResponse = repositoryNames.Select(x => new ResourceItem
                            {
                                DisplayData = x.Name,
                                Owner = x.Owner,
                                Href = request.Path + $"/{x.Name}"
```

```
                        }).ToList();

                        var response = new APIGatewayProxyResponse
                        {
                            StatusCode = (int)HttpStatusCode.OK,
                            Body = JsonConvert.SerializeObject(new { Resources = repositoryResponse }, jsonSerializerSettings),
                            Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
                        };

                        return response;
                    }
                });
        }

        /// <summary>
        /// A Lambda function that creates a new repository
        /// </summary>
        /// <param name="request"></param>
        /// <returns></returns>
        public async Task<APIGatewayProxyResponse> CreateRepositoryAsync(APIGatewayProxyRequest request, ILambdaContext context)
        {
            using (var db = new ReVersion_DatabaseContext())
            {
                s3Ops = new S3Operations(Environment.GetEnvironmentVariable(S3_REGION_ENVIRONMENT_VARIABLE_LOOKUP));

                RepositoryData requestBody;
                try
                {
                    requestBody = JsonConvert.DeserializeObject<RepositoryData>(request.Body, jsonSerializerSettings);
                }
                catch (Exception)
                {
                    return new APIGatewayProxyResponse
                    {
                        StatusCode = (int)HttpStatusCode.BadRequest
                    };
                }

                string repo = requestBody.RepositoryId;
                string user = requestBody.UserName;
                string bucketname = S3Operations.bucketPrefix + '.' + repo.ToLower() + ".main";
                if ( !BucketNameValid(db, bucketname) || !UsernameExists(db, user))
                {
                    return new APIGatewayProxyResponse
                    {
                        StatusCode = (int)HttpStatusCode.BadRequest
                    };
                }

                if (RepositoryExists(db, repo))
                {
                    return new APIGatewayProxyResponse
                    {
                        StatusCode = (int)HttpStatusCode.Conflict
                    };
                }

                InsertIntoRepoTable(db, repo, user);
                InsertIntoBranchTable(db, repo, "main");
                InsertIntoRepoPermissionsTable(db, repo, user);
                await s3Ops.CreateS3BucketAsync(repo, "main");
                InsertIntoEventLog(db, repo, "main", user, requestBody.Message, "create_repository");

                var responseObject = new ResourceItem
                {
                    DisplayData = repo,
                    Href = request.Path + $"/{repo}",
                    Owner = user
                };

                await db.SaveChangesAsync();
```

```
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.Created,
                Body = JsonConvert.SerializeObject(responseObject),
                Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
            };
        }
    }

    /// <summary>
    /// A Lambda function that deletes an entire repository
    /// </summary>
    /// <param name="request"></param>
    /// <returns></returns>
    public async Task<APIGatewayProxyResponse> DeleteRepositoryAsync(APIGatewayProxyRequest request, ILambdaContext context)
    {

        using (var db = new ReVersion_DatabaseContext())
        {
            s3Ops = new S3Operations(Environment.GetEnvironmentVariable(S3_REGION_ENVIRONMENT_VARIABLE_LOOKUP));

            if (request.PathParameters == null || !request.PathParameters.ContainsKey("repositoryId"))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError };

            string repo;
            if (!request.PathParameters.TryGetValue("repositoryId", out repo))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError };

            if (!RepositoryExists(db, repo))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NotFound };

            List<BranchLookup> branches = QueryBranches(db, repo);

            foreach (var branch in branches)
            {
                if (branch.Locked)
                    return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.Conflict };
            }

            DeleteAllFromBranchesTable(db, repo);
            DeleteAllFromVersionsTable(db, repo);
            DeleteAllFromPermissionsTable(db, repo);
            DeleteAllFromPermissionRequestsTable(db, repo);
            DeleteFromRepositoryTable(db, repo);

            await db.SaveChangesAsync();
            return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NoContent };
        }
    }

    /// <summary>
    /// A Lambda function that gets a list of all of the branches in the repository
    /// </summary>
    /// <param name="request"></param>
    /// <returns></returns>
    public async Task<APIGatewayProxyResponse> GetBranchesAsync(APIGatewayProxyRequest request, ILambdaContext context)
    {
        return
            await Task.Run(() =>
                {
                    using (var db = new ReVersion_DatabaseContext())
                    {
                        if (request.PathParameters == null || !request.PathParameters.ContainsKey("repositoryId"))
                            return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError };

                        string repo;
                        if (!request.PathParameters.TryGetValue("repositoryId", out repo))
                            return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError };
```

```
                        if (!RepositoryExists(db, repo))
                            return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
NotFound };

                        List<BranchLookup> branches = QueryBranches(db, repo);

                        List<ResourceItem> responseObject =
                            (from branch in branches
                             select new ResourceItem
                             {
                                 DisplayData = branch.BranchName,
                                 Href = request.Path + $"/{branch.BranchName}"
                             }).ToList();

                        string responseContent = JsonConvert.SerializeObject(new { Resources =
responseObject }, jsonSerializerSettings);

                        return new APIGatewayProxyResponse
                        {
                            StatusCode = (int)HttpStatusCode.OK,
                            Body = responseContent,
                            Headers = new Dictionary<string, string> { { "Content-Type", "application/
json" } }
                        };
                    }
                }
            );
    }

    /// <summary>
    /// A Lambda function that creates a new branch within the repository
    /// </summary>
    /// <param name="request"></param>
    /// <returns></returns>
    public async Task<APIGatewayProxyResponse> CreateBranchAsync(APIGatewayProxyRequest request,
ILambdaContext context)
    {
        using (var db = new ReVersion_DatabaseContext())
        {
            s3Ops = new S3Operations(Environment.GetEnvironmentVariable(
S3_REGION_ENVIRONMENT_VARIABLE_LOOKUP));
            if (request.PathParameters == null || !request.PathParameters.ContainsKey("repositoryId"))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };

            string repo;
            if (!request.PathParameters.TryGetValue("repositoryId", out repo))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };

            if (!RepositoryExists(db, repo))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NotFound };

            NewBranchData requestBody;
            try
            {
                requestBody = JsonConvert.DeserializeObject<NewBranchData>(request.Body);
            }
            catch (Exception)
            {
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
            }

            if (!UsernameExists(db, requestBody.UserName))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };

            if (BranchExists(db, repo, requestBody.BranchId))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.Conflict };

            InsertIntoBranchTable(db, repo, requestBody.BranchId);
            await s3Ops.CreateS3BucketAsync(repo, requestBody.BranchId);
            InsertIntoEventLog(db, repo, requestBody.BranchId, requestBody.UserName, requestBody.
Message, "create_branch");

```

```
            var responseObject = new ResourceItem
            {
                DisplayData = requestBody.BranchId,
                Href = request.Path + $"/{requestBody.BranchId}"
            };

            var responseContent = JsonConvert.SerializeObject(responseObject);

            await db.SaveChangesAsync();
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.Created,
                Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } },
                Body = responseContent
            };
        }
    }

    /// <summary>
    /// A Lambda function that gets a list of all of the objects within a branch
    /// </summary>
    /// <param name="request"></param>
    /// <returns></returns>
    public async Task<APIGatewayProxyResponse> GetBranchAsync(APIGatewayProxyRequest request, ILambdaContext context)
    {
        return
            await Task.Run(() =>
            {
                using (var db = new ReVersion_DatabaseContext())
                {
                    context.Logger.Log("start of GetBranchAsync function");
                    if (request.PathParameters == null || !request.PathParameters.ContainsKey("repositoryId")
                        || !request.PathParameters.ContainsKey("branchId"))
                        return new APIGatewayProxyResponse
                        {
                            StatusCode = (int)HttpStatusCode.InternalServerError,
                            Body = "path parameters are empty or missing repositoryId or branchId"
                        };

                    context.Logger.Log("Passed the first check repositoryId and branchId exist");

                    string repo;
                    string branch;
                    if (!request.PathParameters.TryGetValue("repositoryId", out repo)
                        || !request.PathParameters.TryGetValue("branchId", out branch))
                        return new APIGatewayProxyResponse
                        {
                            StatusCode = (int)HttpStatusCode.InternalServerError,
                            Body = "missing repositoryId or branchId in the second check"
                        };

                    context.Logger.Log($"Passed the second check. repositoryId = {repo} and branchId = {branch}");

                    if (!BranchExists(db, repo, branch))
                        return new APIGatewayProxyResponse
                        {
                            StatusCode = (int)HttpStatusCode.NotFound,
                            Body = $"Branch does not exist for branch = {branch} and repository = {repo}"
                        };

                    context.Logger.Log($"Determined that branch exists for repository {repo} and branch {branch}");

                    HierarchyNode branchFiles = QueryHierarchy(db, repo, branch);

                    context.Logger.Log($"Successfully queried branch hierarchy from repository {repo} and branch {branch}");

                    return new APIGatewayProxyResponse
                    {
                        StatusCode = (int)HttpStatusCode.OK,
                        Body = JsonConvert.SerializeObject(branchFiles),
```

```
                        Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
                    };
                }
            });
    }

    /// <summary>
    /// A Lambda function that commits all local changes to a branch
    /// </summary>
    /// <param name="request"></param>
    /// <returns></returns>
    public async Task<APIGatewayProxyResponse> CommitChangesAsync(APIGatewayProxyRequest request, ILambdaContext context)
    {
        return await Task.Run(() =>
        {
            s3Ops = new S3Operations(Environment.GetEnvironmentVariable(S3_REGION_ENVIRONMENT_VARIABLE_LOOKUP));

            return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError };
        });
        // TODO
        //throw new NotImplementedException();
    }

    /// <summary>
    /// A Lambda function that gets a file within a branch
    /// </summary>
    /// <param name="request"></param>
    /// <returns></returns>
    public async Task<APIGatewayProxyResponse> GetObjectAsync(APIGatewayProxyRequest request, ILambdaContext context)
    {
        using (var db = new ReVersion_DatabaseContext())
        {
            s3Ops = new S3Operations(Environment.GetEnvironmentVariable(S3_REGION_ENVIRONMENT_VARIABLE_LOOKUP));

            if (request.PathParameters == null || !request.PathParameters.ContainsKey("repositoryId")
                || !request.PathParameters.ContainsKey("branchId")
                || !request.QueryStringParameters.ContainsKey("awsObjectKey"))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError };
            string repo;
            string branch;
            string objectKey;
            if (!request.PathParameters.TryGetValue("repositoryId", out repo)
                || !request.PathParameters.TryGetValue("branchId", out branch)
                || !request.PathParameters.TryGetValue("awsObjectKey", out objectKey))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError };

            if (!BranchExists(db, repo, branch))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NotFound };

            string responseContent = await s3Ops.GetTextFromObjectAsync(repo, branch, objectKey);

            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.OK,
                Body = responseContent,
                Headers = new Dictionary<string, string> { { "Content-Type", "text/plain" } }
            };
        }
    }

    /// <summary>
    /// A Lambda function that gets a list of all of the versions within a branch
    /// </summary>
    /// <param name="request"></param>
    /// <returns></returns>
    public async Task<APIGatewayProxyResponse> GetVersionsAsync(APIGatewayProxyRequest request, ILambdaContext context)
    {
```

```
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" }
```

```
    }
                    };
                }
            });
        }

        /// <summary>
        /// A Lambda function that gets a file from a non-current version of a branch
        /// </summary>
        /// <param name="request"></param>
        /// <returns></returns>
        public async Task<APIGatewayProxyResponse> GetPastObjectAsync(APIGatewayProxyRequest request,
    ILambdaContext context)
        {
            return await Task.Run(() =>
            {
                s3Ops = new S3Operations(Environment.GetEnvironmentVariable(
    S3_REGION_ENVIRONMENT_VARIABLE_LOOKUP));
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError
    };
            });
            // TODO
            //throw new NotImplementedException();
        }

        /// <summary>
        /// A Lambda function that merges a branch with its parent
        /// </summary>
        /// <param name="request"></param>
        /// <returns></returns>
        public async Task<APIGatewayProxyResponse> MergeBranchAsync(APIGatewayProxyRequest request,
    ILambdaContext context)
        {
            return await Task.Run(() =>
            {
                s3Ops = new S3Operations(Environment.GetEnvironmentVariable(
    S3_REGION_ENVIRONMENT_VARIABLE_LOOKUP));
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError
    };
            });
            // TODO
            //throw new NotImplementedException();
        }

        /// <summary>
        /// A Lambda function that places a lock on a branch
        /// </summary>
        /// <param name="request"></param>
        /// <returns></returns>
        public async Task<APIGatewayProxyResponse> LockBranchAsync(APIGatewayProxyRequest request,
    ILambdaContext context)
        {
            using (var db = new ReVersion_DatabaseContext())
            {
                if (request.PathParameters == null || !request.PathParameters.ContainsKey("repositoryId")
                    || !request.PathParameters.ContainsKey("branchId"))
                    return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
    InternalServerError };

                string repo;
                string branch;

                if (!request.PathParameters.TryGetValue("repositoryId", out repo)
                    || !request.PathParameters.TryGetValue("branchId", out branch))
                    return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
    InternalServerError };

                LogData requestBody;
                try
                {
                    requestBody = JsonConvert.DeserializeObject<LogData>(request.Body);
                }
                catch (Exception)
                {
                    return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
    InternalServerError };
```

```
            }

            if (!UsernameExists(db, requestBody.UserName))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };

            if (!BranchExists(db, repo, branch))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NotFound };

            LockState lockState = AcquireLock(db, repo, branch);

            if (lockState == LockState.AlreadyLockedConflict)
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.Conflict };

            if (lockState != LockState.SuccessfulLockOperation)
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
    InternalServerError };

            InsertIntoEventLog(db, repo, branch, requestBody.UserName, requestBody.Message, "
    place_lock");

            LockData responseObject = QueryLockEventByBranch(db, repo, branch);

            await db.SaveChangesAsync();
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.OK,
                Body = JsonConvert.SerializeObject(responseObject),
                Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
            };
        }
    }

    /// <summary>
    /// A Lambda function that "safely" removes a lock from a branch (only works if the same user set
    the lock)
    /// </summary>
    /// <param name="request"></param>
    /// <returns></returns>
    public async Task<APIGatewayProxyResponse> UnlockBranchAsync(APIGatewayProxyRequest request,
    ILambdaContext context)
    {
        using (var db = new ReVersion_DatabaseContext())
        {
            if (request.PathParameters == null || !request.PathParameters.ContainsKey("repositoryId")
                || !request.PathParameters.ContainsKey("branchId"))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
    InternalServerError };

            string repo;
            string branch;

            if (!request.PathParameters.TryGetValue("repositoryId", out repo)
                || !request.PathParameters.TryGetValue("branchId", out branch))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
    InternalServerError };

            LogData requestBody;
            try
            {
                requestBody = JsonConvert.DeserializeObject<LogData>(request.Body);
            }
            catch (Exception)
            {
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
    InternalServerError };
            }

            if (!UsernameExists(db, requestBody.UserName))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };

            if (!BranchExists(db, repo, branch))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NotFound };

            LockState lockState = SafeReleaseLock(db, repo, branch, requestBody.UserName);

            if (lockState == LockState.AlreadyUnlockedConflict || lockState == LockState.
```

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

```
            LockData responseObject = QueryLockEventByBranch(db, repo, branch);

            await db.SaveChangesAsync();
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.OK,
                Body = JsonConvert.SerializeObject(responseObject),
                Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
            };
        }
    }

    /// <summary>
    /// A Lambda function that requests permission for a user to access a repository
    /// </summary>
    /// <param name="request"></param>
    /// <returns></returns>
    public async Task<APIGatewayProxyResponse> RequestPermissionAsync(APIGatewayProxyRequest request, ILambdaContext context)
    {
        using (var db = new ReVersion_DatabaseContext())
        {
            if (request.PathParameters == null || !request.PathParameters.ContainsKey("repositoryId"))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError };

            string repo;

            if (!request.PathParameters.TryGetValue("repositoryId", out repo))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError };

            LogData requestBody;
            try
            {
                requestBody = JsonConvert.DeserializeObject<LogData>(request.Body);
            }
            catch (Exception)
            {
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError };
            }

            if (!UsernameExists(db, requestBody.UserName))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };

            if (!RepositoryExists(db, repo))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NotFound };

            if (UserCanAccessRepository(db, requestBody.UserName, repo))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.Conflict };

            if (PermissionRequestIsLogged(db, repo, requestBody.UserName))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.NoContent };

            InsertIntoEventLog(db, repo, "main", requestBody.UserName, requestBody.Message, "request_permission");
            int eventId = QueryLastEventIdByUser(db, requestBody.UserName);

            InsertIntoPermissionRequestTable(db, repo, eventId);

            await db.SaveChangesAsync();
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.NoContent,
                Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
            };
        }
    }


    // TODO: in the client-side, you need to remember to add /grant and /deny to all of the resource URLs!!!

    /// <summary>
    /// A Lambda function that gets a list of all of the unanswered permission requests submitted
```

```
        /// to access all of the repositories owned by a user
        /// </summary>
        /// <param name="request"></param>
        /// <returns></returns>
        public async Task<APIGatewayProxyResponse> GetPermissionRequestsAsync(APIGatewayProxyRequest request, ILambdaContext context)
        {
            return await Task.Run(() =>
            {
                context.Logger.Log("entering function");
                using (var db = new ReVersion_DatabaseContext())
                {
                    if (request.Headers == null || !request.Headers.ContainsKey("username"))
                        return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError };

                    context.Logger.Log("passed initial parameter check");

                    string user;
                    if (!request.Headers.TryGetValue("username", out user))
                        return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };

                    context.Logger.Log($"username is {user}");

                    List<PermissionLookup> permissions = QueryPermissionRequests(db, user);

                    context.Logger.Log("okay, i just made a database call");

                    List<ResourceItem> responseObject =
                        permissions.Select(x => new ResourceItem
                        {
                            DisplayData = $"{x.RequestingUser} requested access to {x.RepositoryName} on " + x.LogTimestamp.ToString(@"MM/dd/yy H:mm tt"),
                            Href = $"repositories/{x.RequestId}"
                        }).ToList();

                    context.Logger.Log("i just made the response object");

                    return new APIGatewayProxyResponse
                    {
                        StatusCode = (int)HttpStatusCode.OK,
                        Body = JsonConvert.SerializeObject(new { Resources = responseObject }),
                        Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
                    };
                }
            });
        }

        /// <summary>
        /// A Lambda function that grants a user's request to access a repository
        /// </summary>
        /// <param name="request"></param>
        /// <returns></returns>
        public async Task<APIGatewayProxyResponse> GrantPermissionAsync(APIGatewayProxyRequest request, ILambdaContext context)
        {
            using (var db = new ReVersion_DatabaseContext())
            {
                if (request.PathParameters == null || !request.PathParameters.ContainsKey("requestId"))
                    return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError };

                if (!request.PathParameters.TryGetValue("requestId", out string permissionRequest))
                {
                    return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.InternalServerError };
                }

                int requestId;
                try
                {
                    requestId = Convert.ToInt32(permissionRequest);
                }
                catch (Exception)
```

```
            {
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };
            }


            LogData requestBody;
            try
            {
                requestBody = JsonConvert.DeserializeObject<LogData>(request.Body);
            }
            catch (Exception)
            {
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
            }

            if (!UsernameExists(db, requestBody.UserName))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };

            string repo = QueryRepositoryNameFromRequest(db, requestId);

            if (!UserOwnsRepository(db, repo, requestBody.UserName))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.Forbidden };

            InsertIntoRepoPermissionsTable(db, repo, requestBody.UserName);
            DeleteFromPermissionRequestTable(db, requestId);
            InsertIntoEventLog(db, repo, "main", requestBody.UserName, requestBody.Message, "
grant_permission");

            await db.SaveChangesAsync();
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.NoContent,
                Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
            };
        }
    }

    /// <summary>
    /// A Lambda function that rejects a user's request to access a repository
    /// </summary>
    /// <param name="request"></param>
    /// <returns></returns>
    public async Task<APIGatewayProxyResponse> DenyPermissionAsync(APIGatewayProxyRequest request,
ILambdaContext context)
    {
        using (var db = new ReVersion_DatabaseContext())
        {
            if (request.PathParameters == null || !request.PathParameters.ContainsKey("requestId"))
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };

            if (!request.PathParameters.TryGetValue("requestId", out string permissionRequest))
            {
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
InternalServerError };
            }

            int requestId;
            try
            {
                requestId = Convert.ToInt32(permissionRequest);
            }
            catch (Exception)
            {
                return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };
            }


            LogData requestBody;
            try
            {
                requestBody = JsonConvert.DeserializeObject<LogData>(request.Body);
            }
            catch (Exception)
            {
```

```
                    return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.
    InternalServerError };
                }

                if (!UsernameExists(db, requestBody.UserName))
                    return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.BadRequest };

                string repo = QueryRepositoryNameFromRequest(db, requestId);

                if (!UserOwnsRepository(db, repo, requestBody.UserName))
                    return new APIGatewayProxyResponse { StatusCode = (int)HttpStatusCode.Forbidden };

                DeleteFromPermissionRequestTable(db, requestId);
                InsertIntoEventLog(db, repo, "main", requestBody.UserName, requestBody.Message, "
    deny_permission");

                await db.SaveChangesAsync();
                return new APIGatewayProxyResponse
                {
                    StatusCode = (int)HttpStatusCode.NoContent,
                    Headers = new Dictionary<string, string> { { "Content-Type", "application/json" } }
                };
            }
        }


        /// <exception cref="ArgumentNullException">from call to Convert.FromBase64String</exception>
        /// <exception cref="FormatException">from call to Convert.FromBase64String if not in base64
    format</exception>
        /// <exception cref="ArgumentException">from call to Encoding.GetString if byte array contains
    invalid Unicode code points</exception>
        /// <exception cref="DecoderFallbackException">if fallback occurred due to character encodinng in
    .NET
        ///                                         AND "DecoderFallback" is set to "
    DecoderExceptionFallback"</exception>
        private string DecodeBase64String(string str)
            => Encoding.UTF8.GetString(Convert.FromBase64String(str));

        private string EncodeBase64String(string str) =>
            Convert.ToBase64String(Encoding.UTF8.GetBytes(str));

        private bool BucketNameValid(ReVersion_DatabaseContext db, string bucketName)
        {
            var nameArray = bucketName.Split('.');
            if (nameArray.Length != 3) return false;
            if (!nameArray[0].Equals(S3Operations.bucketPrefix)) return false;
            /*
            if (nameArray[2].ToLower().Equals("main") && RepositoryExists(db, nameArray[1])) return false;
            if (!nameArray[2].ToLower().Equals("main") && BranchExists(db, nameArray[1], nameArray[2]))
    return false;
            */
            foreach (string str in nameArray)
            {
                if (Regex.IsMatch(str, @"[\s._]")) return false;
                if (str.Length > 25) return false;
                if (!Regex.IsMatch(str, @"^[a-zA-Z0-9][a-zA-Z0-9-]*[a-zA-Z0-9]$")) return false;
            }

            return true;
        }

    }
}
```

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Transform" : "AWS::Serverless-2016-10-31",
  "Description" : "AWS Serverless API that exposes the add, remove and get operations for a blogging
    platform using Amazon DynamoDB.",
  "Parameters" : {
  "DBName" : {
    "Type" : "String",
    "Description" : "The name for the PostgreSQL database",
    "MinLength" : "1",
    "MaxLength" : "63"
  },
  "DBUsername" : {
    "Type" : "String",
    "Description" : "Master username for PostgreSQL database",
    "MinLength" : "1",
    "MaxLength" : "41",
    "AllowedPattern" : "^[a-zA-Z0-9]+$"
  },
  "DBPassword" : {
    "Type" : "String",
    "Description" : "Master password for PostgreSQL database",
    "NoEcho" : "true",
    "MinLength" : "1",
    "MaxLength" : "41"
  },
  "DBPort" : {
    "Type" : "Number",
    "Description" : "TCP/IP port for the database",
    "MinValue" : "1150",
    "MaxValue" : "65535",
    "Default" : "5432"
  },
  "DBHostname" : {
    "Type" : "String",
    "Description" : "Hostname (endpoint) for PostgreSQL database",
    "MinLength" : "1"
  }
  },

  "Resources" : {

    "GetRepositories" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GetRepositoriesAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to get a list of all repositories",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/repositories",
              "Method": "GET"
            }
          }
        }
      }
    },

    "CreateRepository" : {
      "Type" : "AWS::Serverless::Function",
```

```
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::CreateRepositoryAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to create a repository",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" },
      "S3_REGION" : { "Ref" : "AWS::Region" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/repositories",
              "Method": "POST"
            }
          }
        }
      }
    },

    "DeleteRepository" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::DeleteRepositoryAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to delete an entire repository",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" },
      "S3_REGION" : { "Ref" : "AWS::Region" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/repositories/{repositoryId}",
              "Method": "DELETE"
            }
          }
        }
      }
    },

    "GetBranches" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GetBranchesAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to get a list of all branches in the repository",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
```

```
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/repositories/{repositoryId}/branches",
              "Method": "GET"
            }
          }
        }
      }
    },

  "CreateBranch" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::CreateBranchAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to create a new branch within the repository",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" },
      "S3_REGION" : { "Ref" : "AWS::Region" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/repositories/{repositoryId}/branches",
              "Method": "POST"
            }
          }
        }
      }
    },

  "GetBranch" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GetBranchAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to get a list of all objects within a branch",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
```

```
            "Properties": {
              "Path": "/repositories/{repositoryId}/branches/{branchId}",
              "Method": "GET"
            }
          }
        }
      }
    },

  "CommitChanges" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::CommitChangesAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to commit changes to branch",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" },
      "S3_REGION" : { "Ref" : "AWS::Region" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/repositories/{repositoryId}/branches/{branchId}",
              "Method": "PUT"
            }
          }
        }
      }
    },

  "GetObject" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GetObjectAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to get a file within the branch",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" },
      "S3_REGION" : { "Ref" : "AWS::Region" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/repositories/{repositoryId}/branches/{branchId}/object",
              "Method": "GET"
            }
          }
        }
      }
    },

  "GetVersions" : {
```

```
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GetVersionsAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to get a list of versions within a branch",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/repositories/{repositoryId}/branches/{branchId}/versions",
              "Method": "GET"
            }
          }
        }
      }
    },

  "GetVersion" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GetVersionAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to get a list of files within a version of a branch",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/repositories/{repositoryId}/branches/{branchId}/versions/{versionId}",
              "Method": "GET"
            }
          }
        }
      }
    },

  "GetPastObject" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GetPastObjectAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to get a file from a non-current version of a branch",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
```

```
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" },
      "S3_REGION" : { "Ref" : "AWS::Region" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/repositories/{repositoryId}/branches/{branchId}/versions/{versionId}/object",
              "Method": "GET"
            }
          }
        }
      }
    },

  "MergeBranch" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::MergeBranchAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to merge a branch with its parent",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" },
      "S3_REGION" : { "Ref" : "AWS::Region" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/repositories/{repositoryId}/branches/{branchId}/merge",
              "Method": "PUT"
            }
          }
        }
      }
    },

  "LockBranch" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::LockBranchAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to put a lock on a branch",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
```

```
            "Properties": {
              "Path": "/repositories/{repositoryId}/branches/{branchId}/lock",
              "Method": "PUT"
            }
          }
        }
      }
    },

  "UnlockBranch" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::UnlockBranchAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to remove the lock from a branch (only works if the same user set the
    lock)",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/repositories/{repositoryId}/branches/{branchId}/unlock",
              "Method": "PUT"
            }
          }
        }
      }
    },

  "BreakLock" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::BreakLockAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to force-remove a lock from a branch (anyone can do this)",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/repositories/{repositoryId}/branches/{branchId}/force-unlock",
              "Method": "PUT"
            }
          }
        }
      }
    },

  "RequestPermission" : {
      "Type" : "AWS::Serverless::Function",
```

```
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::RequestPermissionAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to request permission to access this repository",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/repositories/{repositoryId}/request-permission",
              "Method": "PUT"
            }
          }
        }
      }
    },

  "GetPermissionRequests" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::
    GetPermissionRequestsAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to get a list of pending requests to access repositories owner by the
    user submitting this request",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/permission-requests",
              "Method": "GET"
            }
          }
        }
      }
    },

  "GrantPermission" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::GrantPermissionAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to grant the requested access",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
```

```
            "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/permission-requests/{requestId}/grant",
              "Method": "PUT"
            }
          }
        }
      },

  "DenyPermission" : {
      "Type" : "AWS::Serverless::Function",
      "Properties": {
        "Handler": "ReVersionVCS_API_Lambdas::ReVersionVCS_API_Lambdas.Functions::DenyPermissionAsync",
        "Runtime": "dotnetcore2.1",
        "CodeUri": "",
        "Description": "Function to deny the requested access",
        "MemorySize": 256,
        "Timeout": 30,
        "Role": "arn:aws:iam::761575818457:role/Lambda-ReVersion-VPC-And-S3-Role",
        "Policies": [ "AWSLambdaFullAccess" ],
        "Environment" : {
          "Variables" : {
      "RDS_DB_HOSTNAME" : { "Ref" : "DBHostname" },
            "RDS_DB_PORT" : { "Ref" : "DBPort" },
      "RDS_DB_NAME" : { "Ref" : "DBName" },
      "RDS_DB_USERNAME" : { "Ref" : "DBUsername" },
      "RDS_DB_PASSWORD" : { "Ref" : "DBPassword" }
          }
        },
        "Events": {
          "PutResource": {
            "Type": "Api",
            "Properties": {
              "Path": "/permission-requests/{requestId}/deny",
              "Method": "PUT"
            }
          }
        }
      }
  }
}
```

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Linq;
using System.Threading.Tasks;
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.S3.Util;
using System.IO;

namespace ReVersionVCS_API_Lambdas
{
    public class S3Operations
    {

        private IAmazonS3 s3Client;

        public const string bucketPrefix = "ReVersionVCS";

        private RegionEndpoint bucketRegion;


        public S3Operations(string regionName)
        {
            bucketRegion = RegionEndpoint.GetBySystemName(regionName);
        }

        public async Task<PutBucketResponse> CreateS3BucketAsync(string repoName, string branchName)
        {
            string bucketName = GetBucketName(repoName, branchName);
            using (s3Client = new AmazonS3Client(bucketRegion))
            {
                var responseTask = await CreateBucketAsync(bucketName);
                return responseTask;
            }
        }

        public async Task<HttpStatusCode> DeleteBranchBucketsByRepoAsync(string repoName, List<string>
    branchNames)
        {
            using (s3Client = new AmazonS3Client(bucketRegion))
            {
                try
                {
                    foreach (var branch in branchNames)
                    {
                        string bucketName = GetBucketName(repoName, branch);
                        List<string> objects = await GetObjectKeysAsync(bucketName);
                        foreach (var item in objects)
                        {
                            await s3Client.DeleteObjectAsync(bucketName, item);
                        }
                        await s3Client.DeleteBucketAsync(bucketName);
                    }
                    return HttpStatusCode.OK;
                }
                catch(Exception)
                {
                    return HttpStatusCode.InternalServerError;
                }
            }
        }

        public async Task<HttpStatusCode> UpdateS3ObjectsAsync(List<FileData> files)
        {
            using (s3Client = new AmazonS3Client(bucketRegion))
            {
                try
                {
                    foreach (var item in files)
                    {
                        var response = await UpdateObjectAsync(item);
                    }
                    return HttpStatusCode.OK;
                }
```

```
                catch (AmazonS3Exception e)
                {
                    throw new AmazonS3Exception("Error writing an object", e);
                }
                catch (Exception e)
                {
                    throw new Exception("Unknown error encountered on server", e);
                    //return HttpStatusCode.InternalServerError;
                }
            }
        }

        public async Task<string> GetTextFromObjectAsync(string repoName, string branchName, string objectKey)
        {
            using (s3Client = new AmazonS3Client(bucketRegion))
            {
                var bucketName = GetBucketName(repoName, branchName);
                try
                {
                    return await GetObjectContentAsync(repoName, branchName, objectKey);
                }
                catch (Exception)
                {
                    return null;
                }
            }
        }
        /*
        public async Task MergeIntoBucketAsync(string repoName, string branchName, string parentBranchName)
        {
            using (s3Client = new AmazonS3Client(bucketRegion))
            {
                // TODO TODO TODO TODO




                throw new NotImplementedException();
            }
        }
        */



        private async Task<PutBucketResponse> CreateBucketAsync(string bucketName)
        {
            try
            {
                if (!(await AmazonS3Util.DoesS3BucketExistAsync(s3Client, bucketName)))
                {
                    var putBucketRequest = new PutBucketRequest
                    {
                        BucketName = bucketName,
                        UseClientRegion = true
                    };

                    PutBucketResponse putBucketResponse = await s3Client.PutBucketAsync(putBucketRequest);
                    return putBucketResponse;
                }
                else
                {
                    return new PutBucketResponse { HttpStatusCode = HttpStatusCode.Conflict };
                }
            }
            catch (Exception)
            {
                return new PutBucketResponse { HttpStatusCode = HttpStatusCode.InternalServerError };
            }
        }

        private async Task<string> FindBucketLocationAsync(string repositoryName, string branchName)
        {
            var request = new GetBucketLocationRequest()
```

```
            {
                BucketName = GetBucketName(repositoryName, branchName)
            };
            GetBucketLocationResponse response = await s3Client.GetBucketLocationAsync(request);
            return response.Location.ToString();
        }

        private async Task<List<string>> GetObjectKeysAsync(string bucketName, string prefix = null)
        {
            ListObjectsRequest request =
                (string.IsNullOrEmpty(prefix)) ?
                new ListObjectsRequest { BucketName = bucketName } :
                new ListObjectsRequest { BucketName = bucketName, Prefix = prefix };

            List<string> bucketList = new List<string>();
            ListObjectsResponse response;
            do
            {
                response = await s3Client.ListObjectsAsync(request);
                bucketList.AddRange( response.S3Objects.Select(x => x.Key) );
                request.Marker = response.NextMarker;
            } while (response.IsTruncated);

            return bucketList;
        }

        private async Task<PutObjectResponse> UpdateObjectAsync(FileData file)
        {
            try
            {
                var findResponse = await FindBucketLocationAsync(file.RepositoryName, file.BranchName);
            }
            catch(Exception e)
            {
                throw new Exception($"did not find bucket: {GetBucketName(file.RepositoryName, file.
    BranchName)}", e);
            }
            var request = new PutObjectRequest
            {
                BucketName = GetBucketName(file.RepositoryName, file.BranchName),
                Key = file.ObjectKey,
                ContentBody = file.Content
            };

            PutObjectResponse response = await s3Client.PutObjectAsync(request);
            return response;
        }

        private async Task<string> GetObjectContentAsync(string repoName, string branchName, string
    objectKey)
        {
            GetObjectRequest request = new GetObjectRequest
            {
                BucketName = GetBucketName(repoName, branchName),
                Key = objectKey
            };
            using (GetObjectResponse response = await s3Client.GetObjectAsync(request))
            using (Stream responseStream = response.ResponseStream)
            using (StreamReader reader = new StreamReader(responseStream))
            {
                string responseBody = await reader.ReadToEndAsync();

                return responseBody;
            }
        }

        private string GetBucketName(string repoName, string branchName)
            => (bucketPrefix + '.' + repoName + '.' + branchName).ToLower();


    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using ReVersionVCS_API_Lambdas.Models;

namespace ReVersionVCS_API_Lambdas
{
    public static partial class SQLOperations
    {
        public static List<RepositoryLookup> QueryRepositories(ReVersion_DatabaseContext db) =>
            (from repository in db.Repositories
             join user in db.Users on repository.Owner equals user.Id
             select new RepositoryLookup { Name = repository.Name, Owner = user.UserName })
            .ToList();

        // NOT FULLY TESTED
        // updated version not yet tested, should work fine
        // (except maybe the case with the "versionId" parameter specified
        public static HierarchyNode QueryHierarchy(ReVersion_DatabaseContext db, string repoName, string branchName, string versionId = null)
        {
            var latestHierarchyJson =
                from repository in db.Repositories
                join branch in db.Branches on repository.Id equals branch.RepositoryId
                where repository.Name == repoName
                && branch.Name == branchName
                select branch.LatestFileHierarchy;

            var versionHierarchyJson =
                from repository in db.Repositories
                join branch in db.Branches on repository.Id equals branch.RepositoryId
                join version in db.Versions on branch.Id equals version.BranchId
                where repository.Name.Equals(repoName)
                && branch.Name.Equals(branchName)
                && version.VersionNumber.Equals(versionId ?? "-1")
                select version.FileHierarchy;

            string hierarchyJson = string.IsNullOrEmpty(versionId) ?
                                        latestHierarchyJson.Single() :
                                        versionHierarchyJson.Single();

            FileHierarchyData hierarchyData = new FileHierarchyData(hierarchyJson);

            return hierarchyData.GetHierarchyList();
        }

        public static List<BranchLookup> QueryBranches(ReVersion_DatabaseContext db, string repoName) =>
                (from repository in db.Repositories
                 join branch in db.Branches
                 on repository.Id equals branch.RepositoryId
                 where repository.Name == repoName
                 select new BranchLookup { Locked = branch.Locked, BranchName = branch.Name })
                    .ToList();

        public static int QueryBranchId(ReVersion_DatabaseContext db, string repoName, string branchName) =>
            (from repository in db.Repositories
             join branch in db.Branches
             on repository.Id equals branch.RepositoryId
             where repository.Name == repoName
                 && branch.Name == branchName
             select branch.Id)
                 .Single();

        public static List<int> QueryVersions(ReVersion_DatabaseContext db, string repoName, string branchName) =>
            (from repository in db.Repositories
             join branch in db.Branches on repository.Id equals branch.RepositoryId
             where repository.Name == repoName && branch.Name == branchName
             join version in db.Versions on branch.Id equals version.BranchId
             select version.VersionNumber)
                .ToList();

        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 }
```

```
using System.Collections.Generic;
using System.Linq;
using ReVersionVCS_API_Lambdas.Models;

namespace ReVersionVCS_API_Lambdas
{
    public static partial class SQLOperations
    {

        public enum LockState
        {
            AlreadyLockedConflict,
            AlreadyUnlockedConflict,
            SuccessfulLockOperation,
            LockedByDifferentUser
        }

        public static void InsertIntoRepoTable(ReVersion_DatabaseContext db, string repoName, string
    username) =>
            db.Repositories.AddAsync(new Repository
            {
                Name = repoName,
                Owner = (from user in db.Users
                         where user.UserName == username
                         select user.Id)
                        .Single()
            });

        public static void InsertIntoBranchTable(ReVersion_DatabaseContext db, string repoName, string
    branchName) =>
            db.Branches.Add(
                (from repository in db.Repositories
                 where repository.Name == repoName
                 select new Branch
                 {
                     RepositoryId = repository.Id,
                     Name = branchName
                 })
                .Single()
                );

        public static void InsertIntoRepoPermissionsTable(ReVersion_DatabaseContext db, string repoName,
    string username) =>
            db.RepositoryPermissions.Add(
                (from repository in db.Repositories
                 where repository.Name == repoName
                 from user in db.Users
                 where user.UserName == username
                 select new RepositoryPermission
                 {
                     PermittedUser = user.Id,
                     RepositoryId = repository.Id
                 })
                .Single()
                );

        public static void InsertIntoEventLog(ReVersion_DatabaseContext db, string repoName, string
    branchName, string username, string message, string type)
        {
            db.EventLogs.Add(
                (from user in db.Users
                 from branch in db.Branches
                 join repository in db.Repositories on branch.RepositoryId equals repository.Id
                 where user.UserName == username
                 && branch.Name == branchName
                 && repository.Name == repoName
                 select new EventLog
                 {
                     Type = type,
                     BranchId = branch.Id,
                     UserId = user.Id,
                     Message = message
                 })
                .Single()
               );
        }
```

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

```
                        .Single());

    public static void UpdateHierarchyDatumInBranchesTable(ReVersion_DatabaseContext db, string repoName, string branchName, string hierarchyDatum)
    {
        Branch branchEntity =
            (from repository in db.Repositories
             where repository.Name == repoName
             join branch in db.Branches on repository.Id equals branch.RepositoryId
             where branch.Name == branchName
             select branch)
            .Single();

        branchEntity.VersionNumber += 1;
        branchEntity.LatestFileHierarchy = hierarchyDatum;
    }

    public static LockState ReleaseLock(ReVersion_DatabaseContext db, string repoName, string branchName)
    {
        Branch branchEntity =
            (from repository in db.Repositories
             where repository.Name == repoName
             join branch in db.Branches on repository.Id equals branch.RepositoryId
             where branch.Name == branchName
             select branch)
            .Single();

        if (!branchEntity.Locked)
        {
            return LockState.AlreadyUnlockedConflict;
        }

        branchEntity.Locked = false;
        return LockState.SuccessfulLockOperation;
    }

    public static LockState SafeReleaseLock(ReVersion_DatabaseContext db, string repoName, string branchName, string username)
    {
        Branch branchEntity =
            (from repository in db.Repositories
             join branch in db.Branches on repository.Id equals branch.RepositoryId
             where repository.Name == repoName
             && branch.Name == branchName
             select branch)
            .Single();

        if (!branchEntity.Locked)
            return LockState.AlreadyUnlockedConflict;

        string usernameQuery =
            (from log in db.EventLogs
             where branchEntity.Id == log.BranchId
             && log.Type == "place_lock"
             join user in db.Users on log.UserId equals user.Id
             orderby log.LoggedAt descending, log.Id descending
             select user.UserName)
            .First();

        if (username != usernameQuery)
            return LockState.LockedByDifferentUser;

        branchEntity.Locked = false;
        return LockState.SuccessfulLockOperation;
    }

    public static void InsertIntoPermissionRequestTable(ReVersion_DatabaseContext db, string repoName, int eventId) =>
        db.PermissionRequests.Add(new PermissionRequest
        {
            EventId = eventId,
            RepositoryId = (from repository in db.Repositories
                            where repository.Name == repoName
                            select repository.Id).Single()
        });
```

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

```
<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>ReVersion VCS</title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/normalize/8.0.1/normalize.css" />
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.7.4/css/bulma.css" />
    <link rel="stylesheet" href="index.css" />

    <script src="https://cdn.jsdelivr.net/npm/vue"></script>
</head>
<body>
    <section class="section">
        <div class="container has-background-primary column is-8 is-offset-2"
             style="text-align:center">
            <h1 class="title">
                ReVersion VCS
            </h1>
            <p class="subtitle">
                Version Control <i>as a</i><strong>service</strong>!
            </p>
        </div>
    </section>

    <section id="userData" class="section columns" v-if:"show">
        <div class="field column">
            <div class="column is-three-fifths is-offset-2">
                <div class="control">
                    <input class="input is-primary" type="text" placeholder="username" />
                </div>
                <div class="control">
                    <input class="input is-info" type="password" placeholder="password" />
                </div>
            </div>
            <div class="column">
                <div class="columns">
                    <div class="column is-3 is-offset-3">
                        <input class="button is-primary" type="submit" value="Login" />
                    </div>
                    <div class="column is-3">
                        <a class="button is-info" href="#">Sign Up</a>
                    </div>
                </div>
            </div>
        </div>
    </section>

    <section id="requestTable" class="section" v-if="show">
        <div class="container">
            <h1 class="title" style="text-align:center">Repository Permission Requests</h1>
            <table class="table is-hoverable is-fullwidth">
                <thead>
                    <tr>
                        <th scope="col">Repository Name</th>
                        <th scope="col">Requesting User</th>
                        <th scope="col">Timestamp</th>
                    </tr>
                </thead>
                <tbody>
                    <tr v-for="perReq in permissionRequests">
                        <td v-for="value in perReq">
                            {{value}}
                        </td>
                    </tr>
                </tbody>
            </table>
        </div>
    </section>


    <section id="hierarchy" class="section">
        <div class="container">
            <div class="menu" id="menu">
                <ul>
```

```
                    <input class="menuInput" type="radio" name="menu" id="archive" checked>
                    <li class="menuListItem">
                        <label for="archive" class="menuTitle"><i class="fa fa-folder"></i>Archive</label>
                        <a class="menuItem" href="#">New File</a>
                        <a class="menuItem" href="#">Open File</a>
                        <a class="menuItem" href="#">Save As...</a>
                        <a class="menuItem" href="#">Exit</a>
                    </li>
                    <input class="menuInput" type="radio" name="menu" id="edit">
                    <li class="menuListItem">
                        <label for="edit" class="menuTitle"><i class="fa fa-edit"></i>Edit</label>
                        <a class="menuItem" href="#">Copy</a>
                        <a class="menuItem" href="#">Cut</a>
                        <a class="menuItem" href="#">Paste</a>
                        <a class="menuItem" href="#">Undo</a>
                    </li>
                    <input class="menuInput" type="radio" name="menu" id="tools">
                    <li class="menuListItem">
                        <label for="tools" class="menuTitle"><i class="fa fa-gavel"></i>Tools</label>
                        <a class="menuItem" href="#">Build</a>
                        <a class="menuItem" href="#">Macros</a>
                        <a class="menuItem" href="#">Command</a>
                        <a class="menuItem" href="#">Snippets</a>
                    </li>
                    <input class="menuInput" type="radio" name="menu" id="preferences">
                    <li class="menuListItem">
                        <label for="preferences" class="menuTitle"><i class="fa fa-gears"></i>Preferences<
    /label>
                        <a class="menuItem" href="#">Browser</a>
                        <a class="menuItem" href="#">Settings</a>
                        <a class="menuItem" href="#">Packages</a>
                        <a class="menuItem" href="#">Theme</a>
                    </li>
                </ul>
            </div>
        </div>
</section>

    <script src="index.js"></script>
</body>
</html>
```

```
@import url(https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css);
@import url(https://fonts.googleapis.com/css?family=Montserrat:400,700|Open+Sans:400,300);


body {
    font-family: 'Montserrat', sans-serif;
}

#menu {
    position: fixed;
    display: flex;
    align-items: flex-start;
    width: 80%;
    height: 100%;
    background-color: #111;
}


li.menuListItem, ul label.menuTitle, ul, a {
    width: 100%;
    color: #FFF;
    font-family: 'Montserrat', 'Open Sans', sans-serif;
    display: block;
    font-weight: bold;
}

    ul label {
        height: 35px;
    }

    ul li.menuListItem {
        height: 35px;
        overflow: hidden;
        transition: all .3s;
    }

li.menuListItem {
    display: block;
    background-color: #363636;
}

label.menuTitle {
    font-size: 14px;
    background: linear-gradient(#111, #2f2f2f);
    padding: 10px 15px;
    cursor: pointer;
    transition: all .25s;
}

a.menuItem {
    font-size: 12px;
    text-decoration: none;
    color: #FFF;
    display: block;
    padding: 10px 25px;
    transition: all .25s;
}

    a.menuItem:hover {
        background-color: #444;
        box-shadow: inset 5px 0px 0px 0px #fff;
    }

label.menuTitle:hover {
    text-shadow: 0px 0px 10px #fff;
}

input.menuInput[type="radio"] {
    display: none;
}

#edit:checked + li.menuListItem, #archive:checked + li.menuListItem, #tools:checked + li.menuListItem, #
    preferences:checked + li.menuListItem {
    height: 179px;
}

```

```
i {
    margin-right: 12px;
}

@media screen and (max-width: 600px) {
    #menu {
        width: 100%;
        position: relative;
    }

    main {
        width: 100%;
        position: relative;
    }
}
```

```
var requests = new Vue({
    el: '#requestTable',
    data: {
        permissionRequests: [
            { name: 'repo1', user: 'user1', timestamp: 'right now!' },
            { name: 'repo2', user: 'user2', timestamp: 'idk awhile ago' },
            { name: 'repo3', user: 'user3', timestamp: 'like forever ago!' },
            { name: 'repo4', user: 'user4', timestamp: 'holy crap idk!' },
            { name: 'repo5', user: 'user5', timestamp: 'ugh...yesterday?' }
        ],
        show: true
    }
})

var login = new Vue({
    el: '#userData',
    data: {
        show: true
    }
})
```