SPACE SURFER

LOW LEVEL DESIGN DOCUMENT

TEAM NAME: PIXEL PALS

TEAM MEMBERS: BRANDON GALICH
CARINE GORDILLO
SARAH SANTOS
SARAH PHAN

TEAM LEADER: KAY KAYALE

DATE: DECEMBER 18, 2023

GITHUB: GITHUB.COM/CARINEGORDILLO/CECS491

Version History

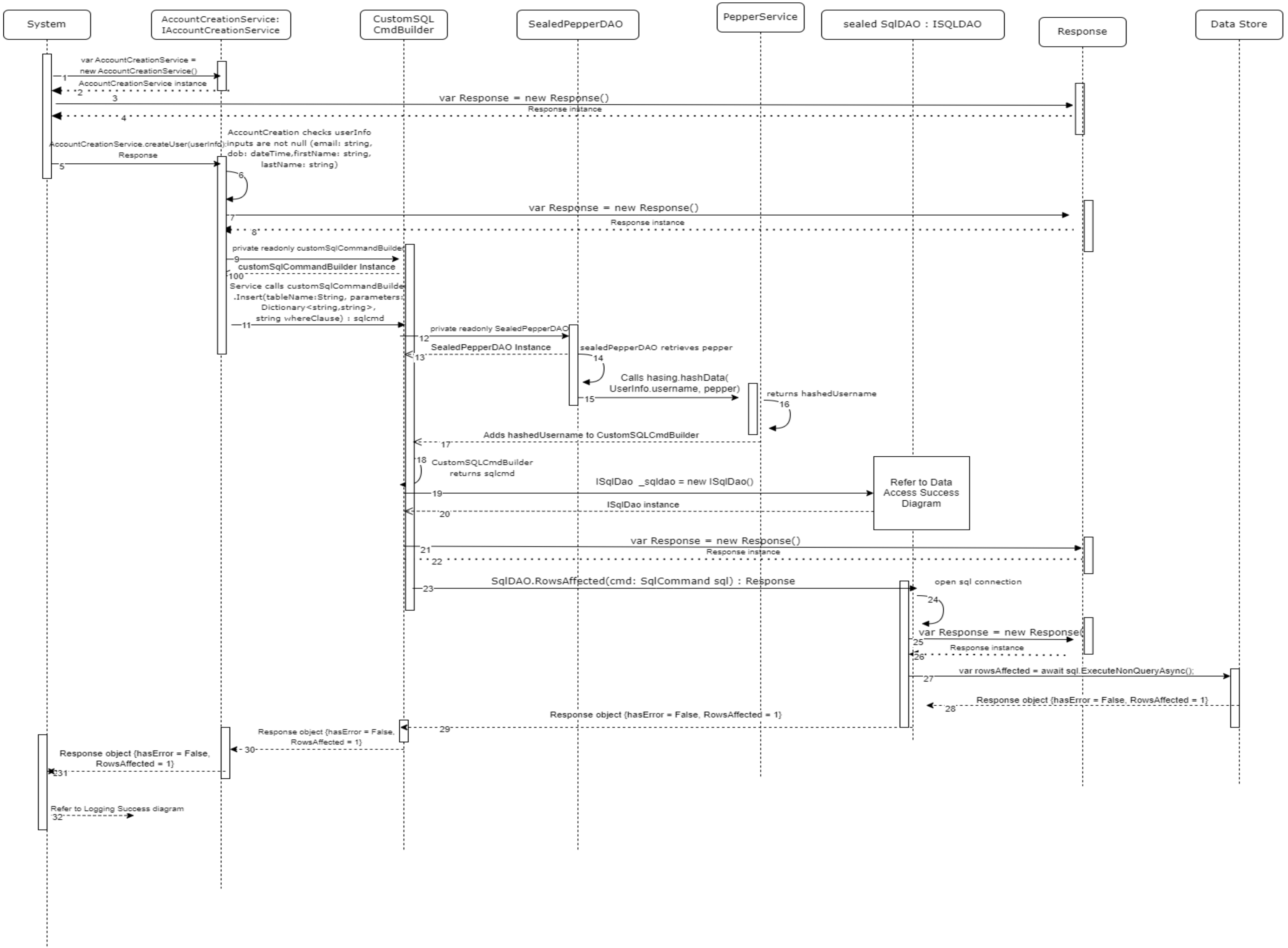
Version	Date	Changes
1	December 18, 2023	Initial submission

Table of Contents:

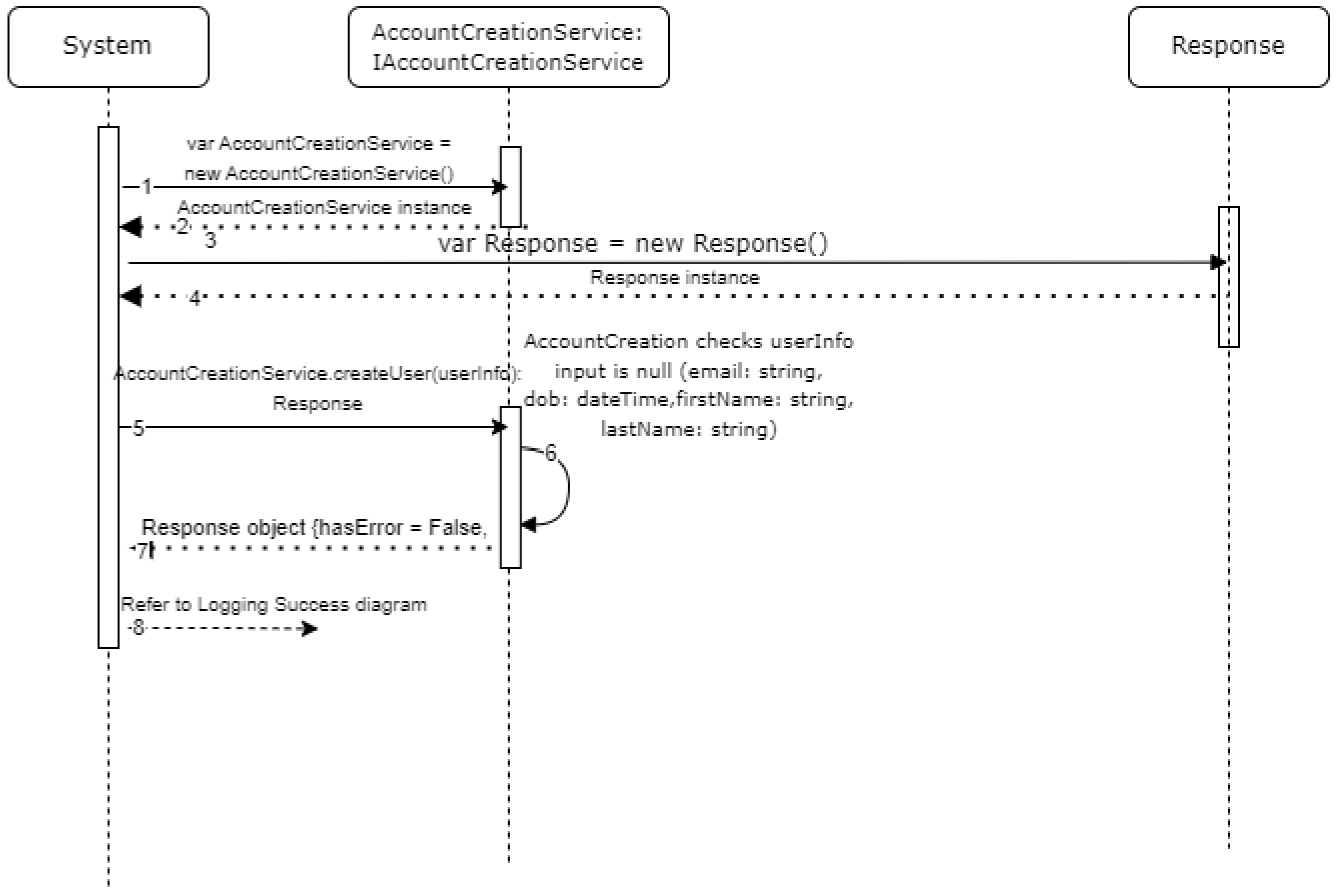
Introduction:

The purpose of this document is to provide a detailed overview of how each feature in our web application functions. Each feature will have its own diagram showing the flow of data through each layer that feature will go through. Our diagrams will address all success and failure outcomes so that all scenarios regarding that feature will be accounted for. With the help of these diagrams, developers and clients will gain a better understanding of how each feature will work and interact with the different layers that allow our application to function.

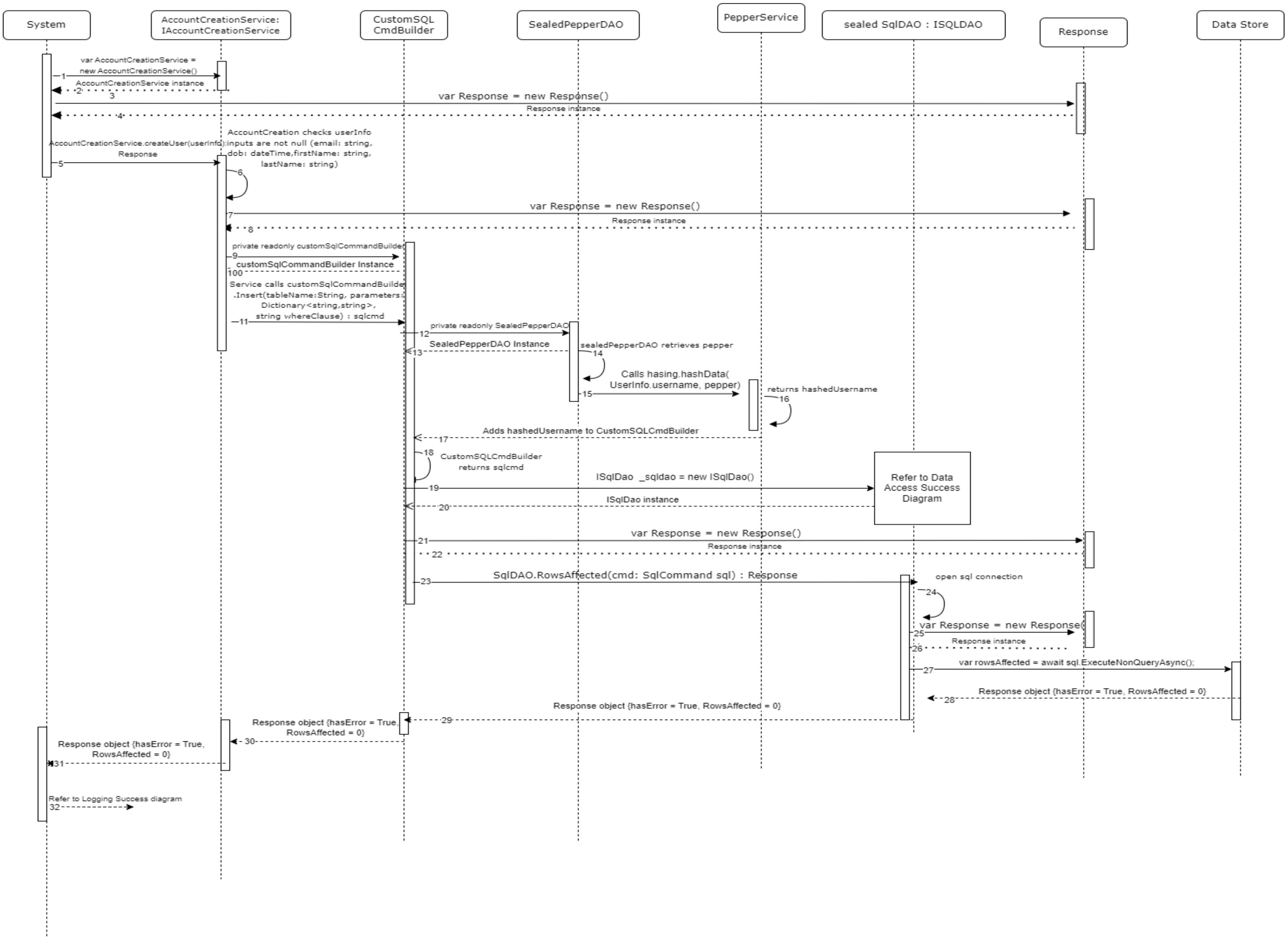
AccountCreation.Success


System
AccountCreationService: IAccountCreationService
CustomSQL CmdBuilder
SealedPepperDAO
PepperService
sealed SqlDAO : ISQLDAO
Response
Data Store
var AccountCreationService = new AccountCreationService()
1
AccountCreationService instance
2
3
var Response = new Response()
Response instance
4
AccountCreationService.createUser(userInfo): Response
5
AccountCreation checks userInfo inputs are not null (email: string, dob: dateTime,firstName: string, lastName: string)
6
7
var Response = new Response()
Response instance
8
private readonly customSqlCommandBuilder
9
customSqlCommandBuilder Instance
100
Service calls customSqlCommandBuilder .Insert(tableName:String, parameters: Dictionary<string,string>, string whereClause) : sqlcmd
11
private readonly SealedPepperDAO
12
SealedPepperDAO Instance
13
sealedPepperDAO retrieves pepper
14
Calls hasing.hashData(UserInfo.username, pepper)
15
returns hashedUsername
16
Adds hashedUsername to CustomSQLCmdBuilder
17
18
CustomSQLCmdBuilder returns sqlcmd
ISqlDao _sqldao = new ISqlDao()
19
Refer to Data Access Success Diagram
ISqlDao instance
20
var Response = new Response()
21
Response instance
22
SqlDAO.RowsAffected(cmd: SqlCommand sql) : Response
23
open sql connection
24
var Response = new Response(
25
Response instance
26
var rowsAffected = await sql.ExecuteNonQueryAsync();
27
Response object {hasError = False, RowsAffected = 1}
28
Response object {hasError = False, RowsAffected = 1}
29
Response object {hasError = False, RowsAffected = 1}
30
Response object {hasError = False, RowsAffected = 1}
231
Refer to Logging Success diagram
32

AccountCreation.InvalidUserInfo

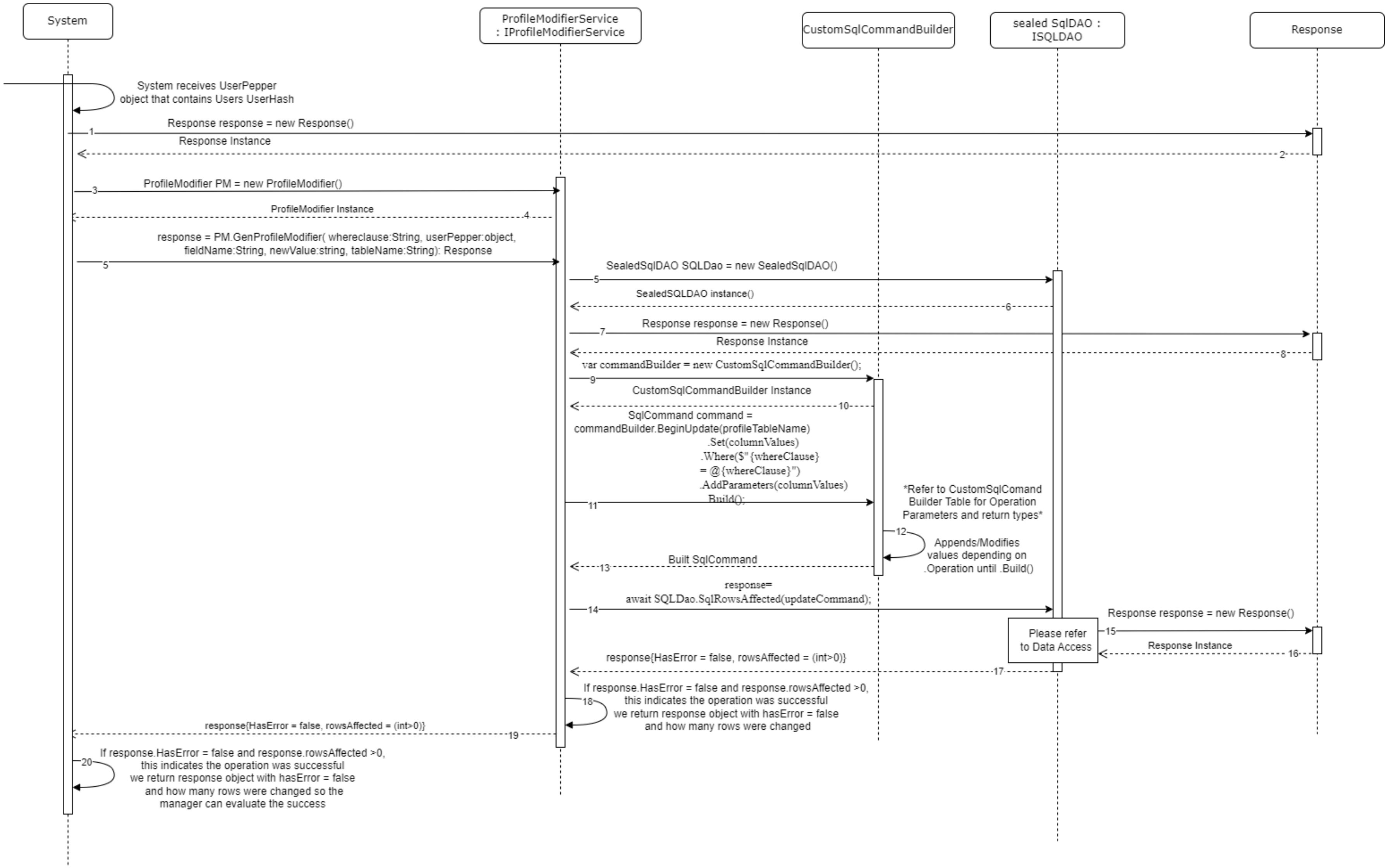

System
AccountCreationService:
IAccountCreationService
Response
var AccountCreationService =
new AccountCreationService()
1
AccountCreationService instance
2
3
var Response = new Response()
Response instance
4
AccountCreation checks userInfo
input is null (email: string,
dob: dateTime,firstName: string,
lastName: string)
AccountCreationService.createUser(userInfo):
Response
5
6
Response object {hasError = False,
7
Refer to Logging Success diagram
8

AccountCreation.InvalidTableInsert


System
AccountCreationService: IAccountCreationService
CustomSQL CmdBuilder
SealedPepperDAO
PepperService
sealed SqlDAO : ISQLDAO
Response
Data Store
1 var AccountCreationService = new AccountCreationService()
2 AccountCreationService instance
3 var Response = new Response()
4 Response instance
5 AccountCreationService.createUser(userInfo): Response
6 AccountCreation checks userInfo inputs are not null (email: string, dob: dateTime,firstName: string, lastName: string)
7 var Response = new Response()
8 Response instance
9 private readonly customSqlCommandBuilder
100 customSqlCommandBuilder Instance
11 Service calls customSqlCommandBuilder .Insert(tableName:String, parameters: Dictionary<string,string>, string whereClause) : sqlcmd
12 private readonly SealedPepperDAO
13 SealedPepperDAO Instance
14 sealedPepperDAO retrieves pepper
15 Calls hasing.hashData(UserInfo.username, pepper)
16 returns hashedUsername
17 Adds hashedUsername to CustomSQLCmdBuilder
18 CustomSQLCmdBuilder returns sqlcmd
19 ISqlDao _sqldao = new ISqlDao()
Refer to Data Access Success Diagram
20 ISqlDao instance
21 var Response = new Response()
22 Response instance
23 SqlDAO.RowsAffected(cmd: SqlCommand sql) : Response
24 open sql connection
25 var Response = new Response(
26 Response instance
27 var rowsAffected = await sql.ExecuteNonQueryAsync();
28 Response object {hasError = True, RowsAffected = 0}
29 Response object {hasError = True, RowsAffected = 0}
30 Response object {hasError = True, RowsAffected = 0}
31 Response object {hasError = True, RowsAffected = 0}
32 Refer to Logging Success diagram

UpdateProfile.Success

Profile Modification Success



System receives UserPepper object that contains Users UserHash

1 Response response = new Response()

2 Response Instance

3 ProfileModifier PM = new ProfileModifier()

4 ProfileModifier Instance

5 response = PM.GenProfileModifier(whereclause:String, userPepper:object, fieldName:String, newValue:string, tableName:String): Response

5 SealedSqlDAO SQLDao = new SealedSqlDAO()

6 SealedSQLDAO instance()

7 Response response = new Response()

8 Response Instance

9 var commandBuilder = new CustomSqlCommandBuilder();

10 CustomSqlCommandBuilder Instance

11 SqlCommand command =
commandBuilder.BeginUpdate(profileTableName)
.Set(columnValues)
.Where($"{whereClause}
= @{whereClause}")
.AddParameters(columnValues)
.Build();

Refer to CustomSqlComand Builder Table for Operation Parameters and return types

12 Appends/Modifies values depending on .Operation until .Build()

13 Built SqlCommand

14 response=
await SQLDao.SqlRowsAffected(updateCommand);

Please refer to Data Access

15 Response response = new Response()

16 Response Instance

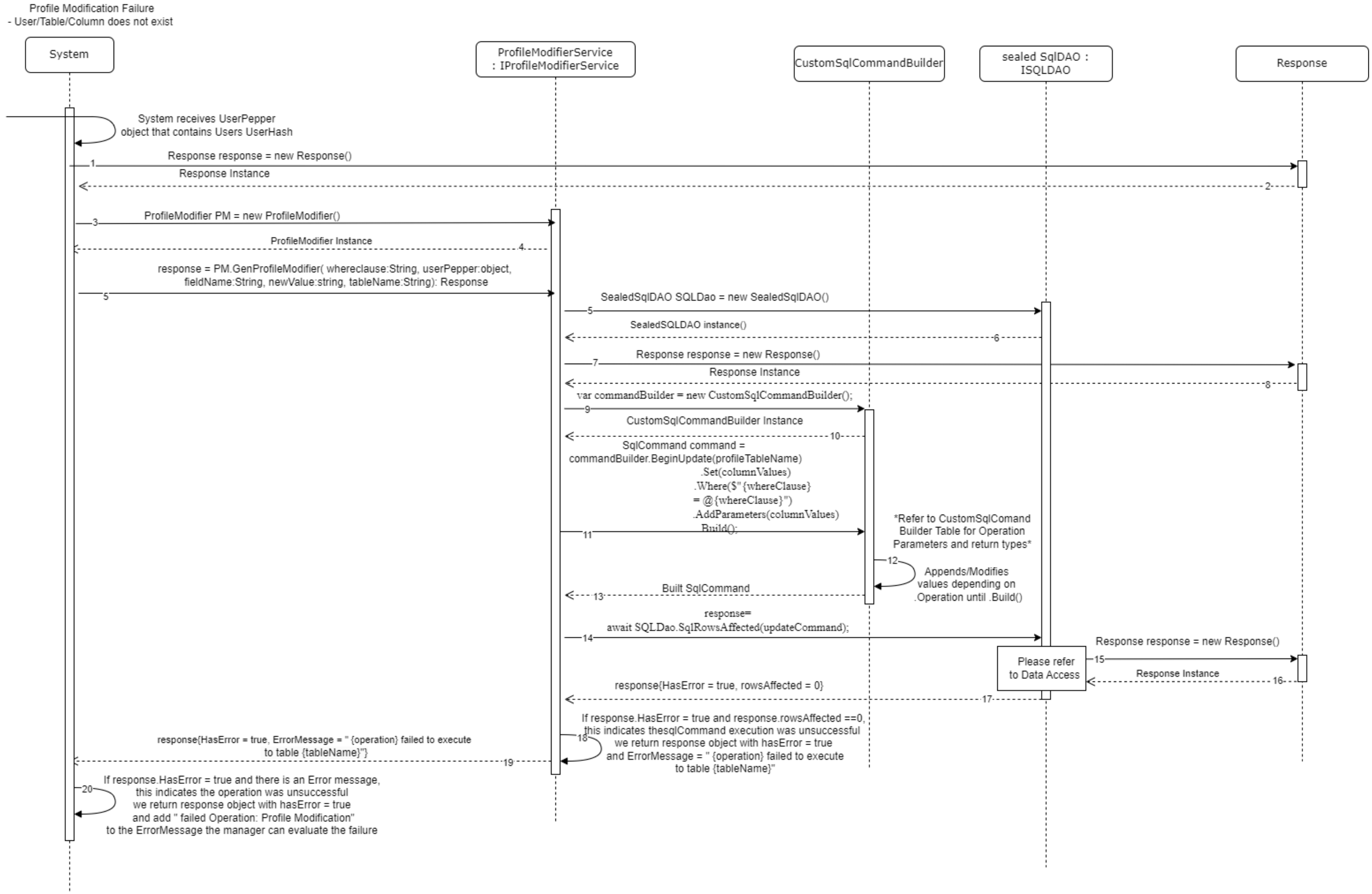
17 response{HasError = false, rowsAffected = (int>0)}

18 If response.HasError = false and response.rowsAffected >0, this indicates the operation was successful we return response object with hasError = false and how many rows were changed

19 response{HasError = false, rowsAffected = (int>0)}

20 If response.HasError = false and response.rowsAffected >0, this indicates the operation was successful we return response object with hasError = false and how many rows were changed so the manager can evaluate the success

UpdateProfile.Failure.Data/Table/ColumnDoesNotExist



Profile Modification Failure
- User/Table/Column does not exist

System

ProfileModifierService : IProfileModifierService

CustomSqlCommandBuilder

sealed SqlDAO : ISQLDAO

Response

System receives UserPepper object that contains Users UserHash

1 Response response = new Response()

2 Response Instance

3 ProfileModifier PM = new ProfileModifier()

4 ProfileModifier Instance

5 response = PM.GenProfileModifier(whereclause:String, userPepper:object, fieldName:String, newValue:string, tableName:String): Response

5 SealedSqlDAO SQLDao = new SealedSqlDAO()

6 SealedSQLDAO instance()

7 Response response = new Response()

8 Response Instance

9 var commandBuilder = new CustomSqlCommandBuilder();

10 CustomSqlCommandBuilder Instance

11 SqlCommand command =
commandBuilder.BeginUpdate(profileTableName)
.Set(columnValues)
.Where($"{whereClause}
= @{whereClause}")
.AddParameters(columnValues)
.Build();

Refer to CustomSqlComand Builder Table for Operation Parameters and return types

12 Appends/Modifies values depending on .Operation until .Build()

13 Built SqlCommand

14 response=
await SQLDao.SqlRowsAffected(updateCommand);

Please refer to Data Access

15 Response response = new Response()

16 Response Instance

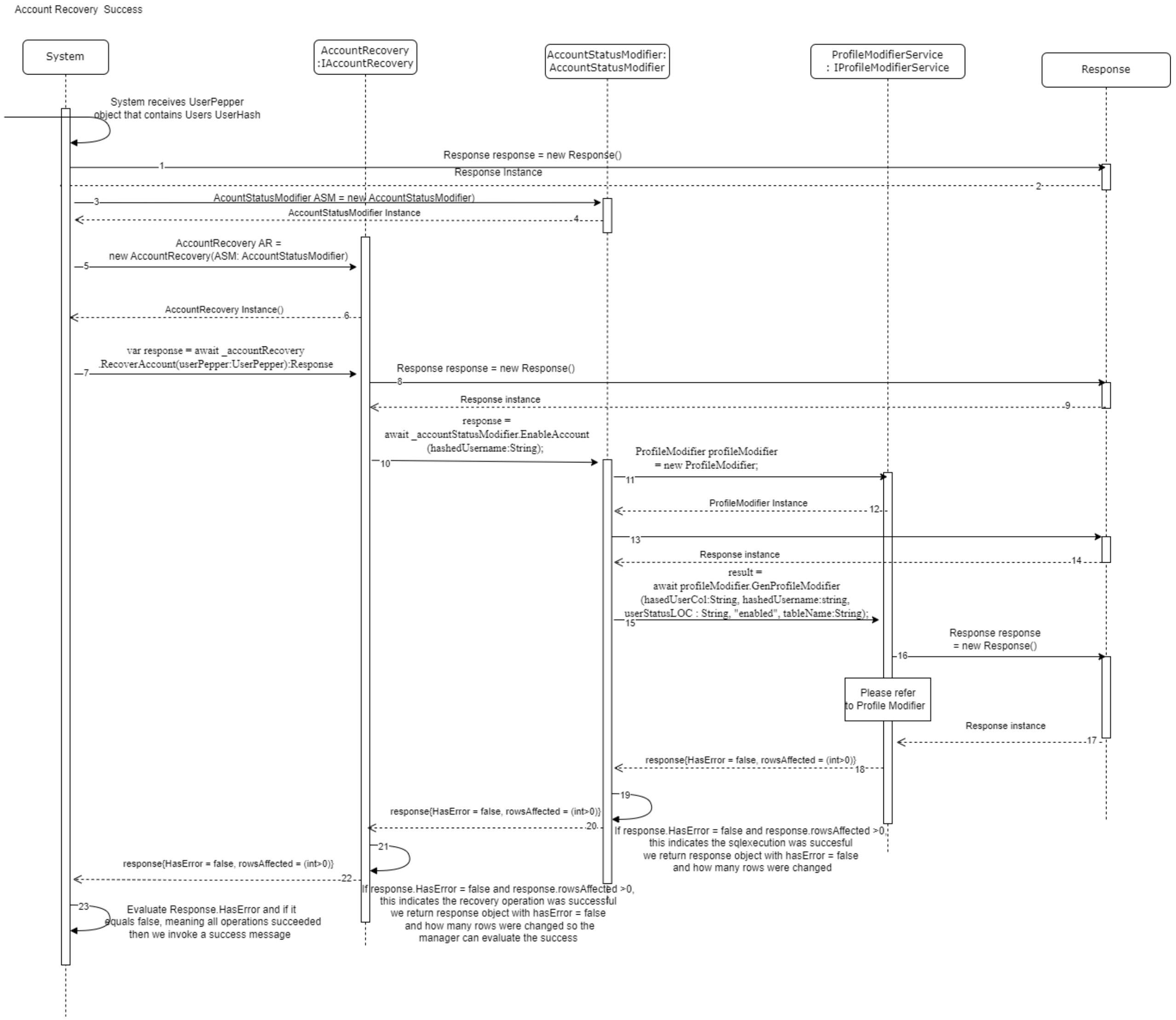
17 response{HasError = true, rowsAffected = 0}

18 If response.HasError = true and response.rowsAffected ==0, this indicates thesqlCommand execution was unsuccessful we return response object with hasError = true and ErrorMessage = " {operation} failed to execute to table {tableName}"

19 response{HasError = true, ErrorMessage = " {operation} failed to execute to table {tableName}"}

20 If response.HasError = true and there is an Error message, this indicates the operation was unsuccessful we return response object with hasError = true and add " failed Operation: Profile Modification" to the ErrorMessage the manager can evaluate the failure

AccountRecovery.Success



Account Recovery Success

System

AccountRecovery
:IAccountRecovery

AccountStatusModifier:
AccountStatusModifier

ProfileModifierService
: IProfileModifierService

Response

System receives UserPepper
object that contains Users UserHash

1 Response response = new Response()

2 Response Instance

3 AcountStatusModifier ASM = new AccountStatusModifier)

4 AccountStatusModifier Instance

5 AccountRecovery AR =
new AccountRecovery(ASM: AccountStatusModifier)

6 AccountRecovery Instance()

7 var response = await _accountRecovery
.RecoverAccount(userPepper:UserPepper):Response

8 Response response = new Response()

9 Response instance

10 response =
await _accountStatusModifier.EnableAccount
(hashedUsername:String);

11 ProfileModifier profileModifier
= new ProfileModifier;

12 ProfileModifier Instance

13

14 Response instance

15 result =
await profileModifier.GenProfileModifier
(hasedUserCol:String, hashedUsername:string,
userStatusLOC : String, "enabled", tableName:String);

16 Response response
= new Response()

Please refer
to Profile Modifier

17 Response instance

18 response{HasError = false, rowsAffected = (int>0)}

19

If response.HasError = false and response.rowsAffected >0,
this indicates the sqlexecution was succesful
we return response object with hasError = false
and how many rows were changed

20 response{HasError = false, rowsAffected = (int>0)}

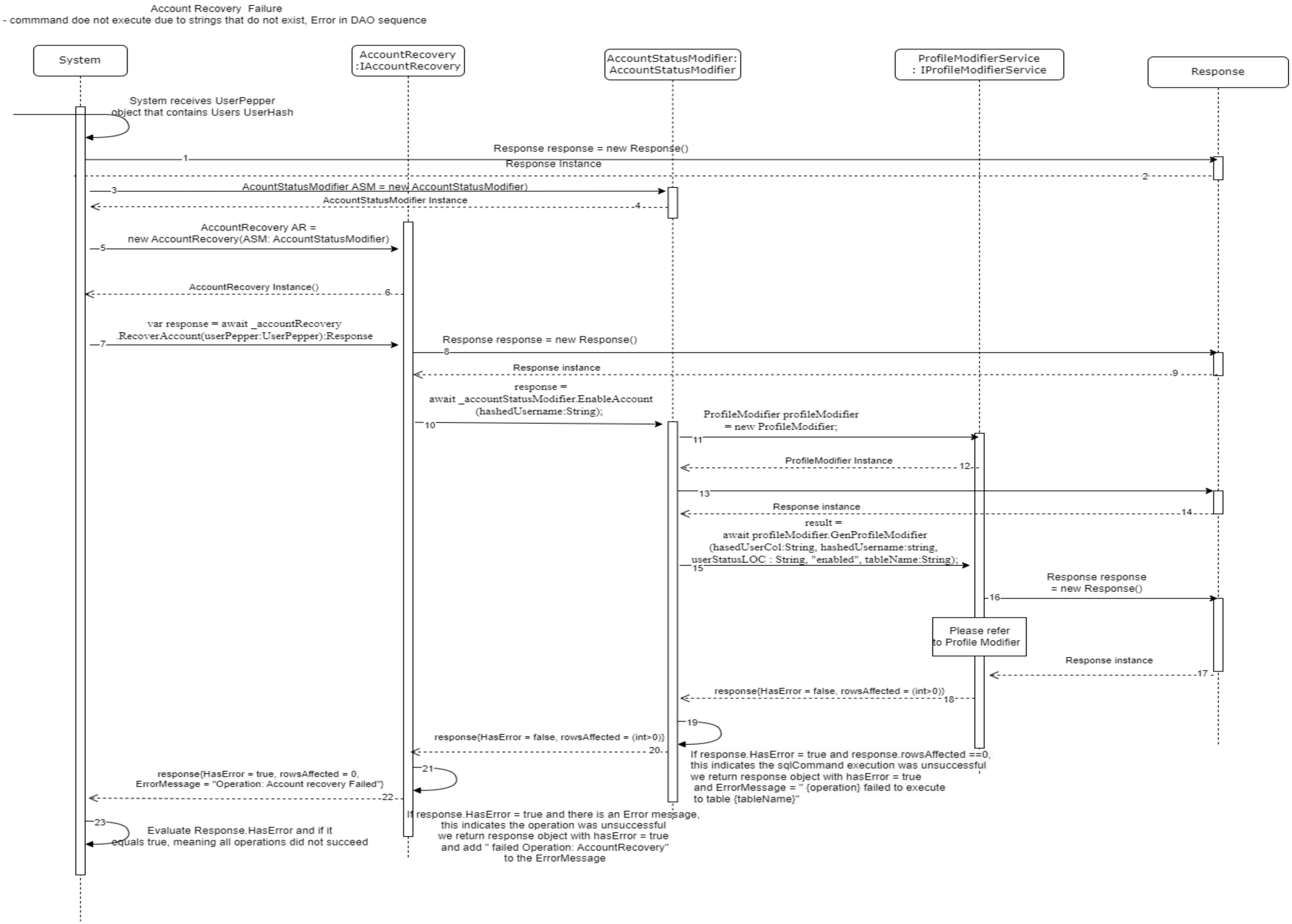
21

If response.HasError = false and response.rowsAffected >0,
this indicates the recovery operation was successful
we return response object with hasError = false
and how many rows were changed so the
manager can evaluate the success

22 response{HasError = false, rowsAffected = (int>0)}

23 Evaluate Response.HasError and if it
equals false, meaning all operations succeeded
then we invoke a success message

AccountRecovery.Failure



Account Recovery Failure
- commmand doe not execute due to strings that do not exist, Error in DAO sequence

System

AccountRecovery
:IAccountRecovery

AccountStatusModifier:
AccountStatusModifier

ProfileModifierService
: IProfileModifierService

Response

System receives UserPepper
object that contains Users UserHash

1 Response response = new Response()

2 Response Instance

3 AcountStatusModifier ASM = new AccountStatusModifier)

4 AccountStatusModifier Instance

5 AccountRecovery AR =
new AccountRecovery(ASM: AccountStatusModifier)

6 AccountRecovery Instance()

7 var response = await _accountRecovery
.RecoverAccount(userPepper:UserPepper):Response

8 Response response = new Response()

9 Response instance

10 response =
await _accountStatusModifier.EnableAccount
(hashedUsername:String);

11 ProfileModifier profileModifier
= new ProfileModifier;

12 ProfileModifier Instance

13

14 Response instance

15 result =
await profileModifier.GenProfileModifier
(hasedUserCol:String, hashedUsername:string,
userStatusLOC : String, "enabled", tableName:String);

16 Response response
= new Response()

Please refer
to Profile Modifier

17 Response instance

18 response{HasError = false, rowsAffected = (int>0)}

19

If response.HasError = true and response.rowsAffected ==0,
this indicates the sqlCommand execution was unsuccessful
we return response object with hasError = true
and ErrorMessage = " {operation} failed to execute
to table {tableName}"

20 response{HasError = false, rowsAffected = (int>0)}

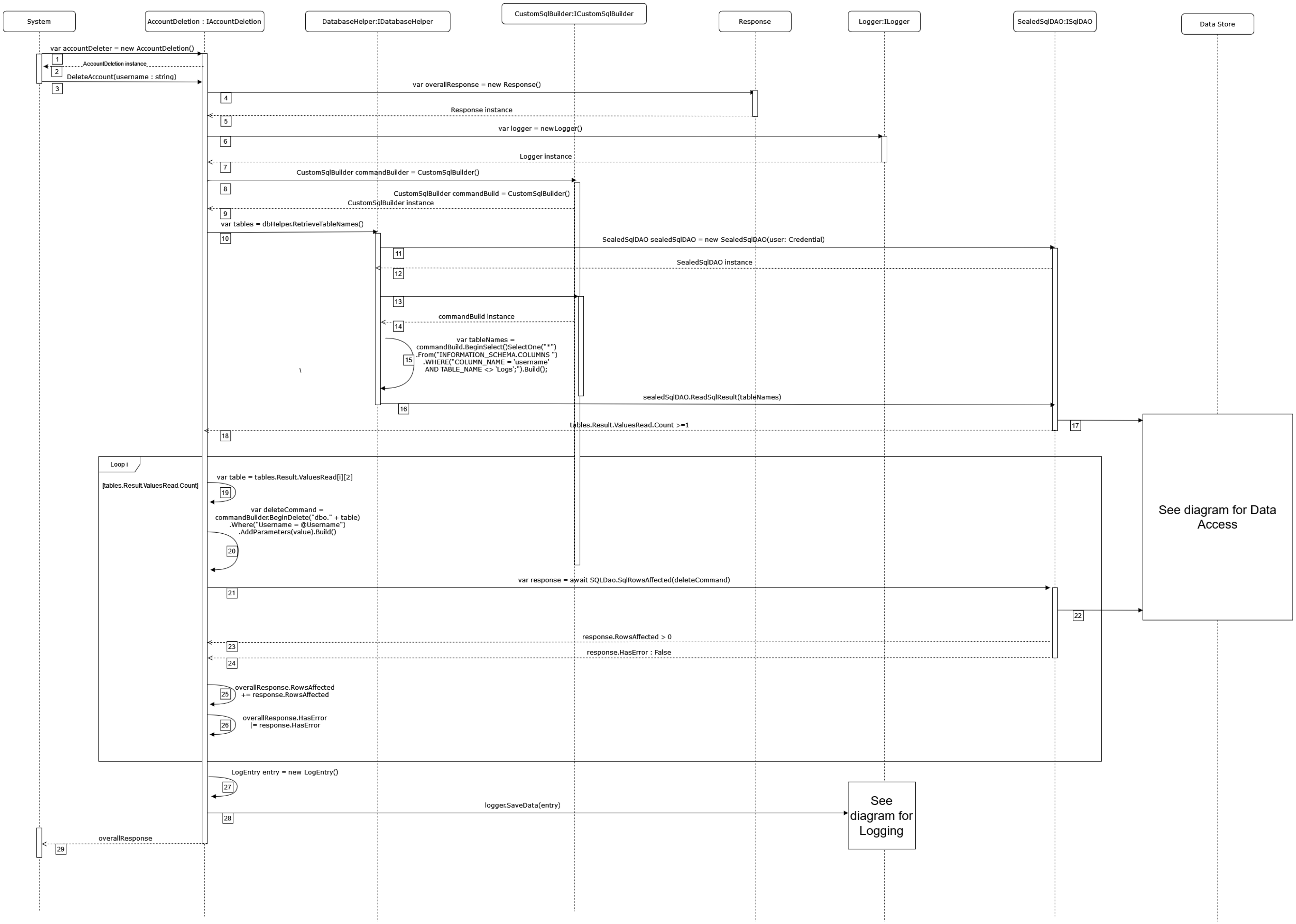
21

If response.HasError = true and there is an Error message,
this indicates the operation was unsuccessful
we return response object with hasError = true
and add " failed Operation: AccountRecovery"
to the ErrorMessage

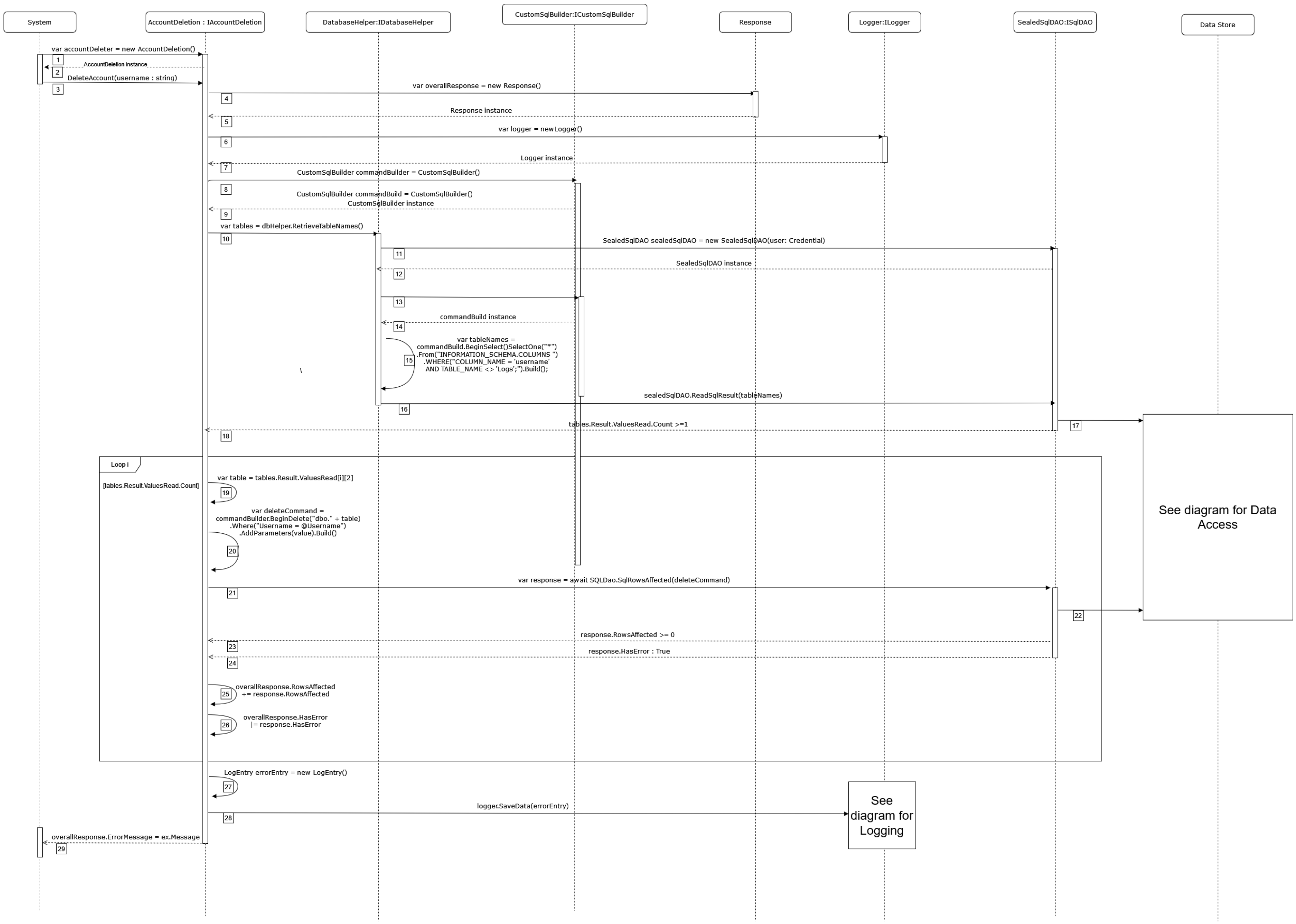
22 response{HasError = true, rowsAffected = 0,
ErrorMessage = "Operation: Account recovery Failed"}

23 Evaluate Response.HasError and if it
equals true, meaning all operations did not succeed

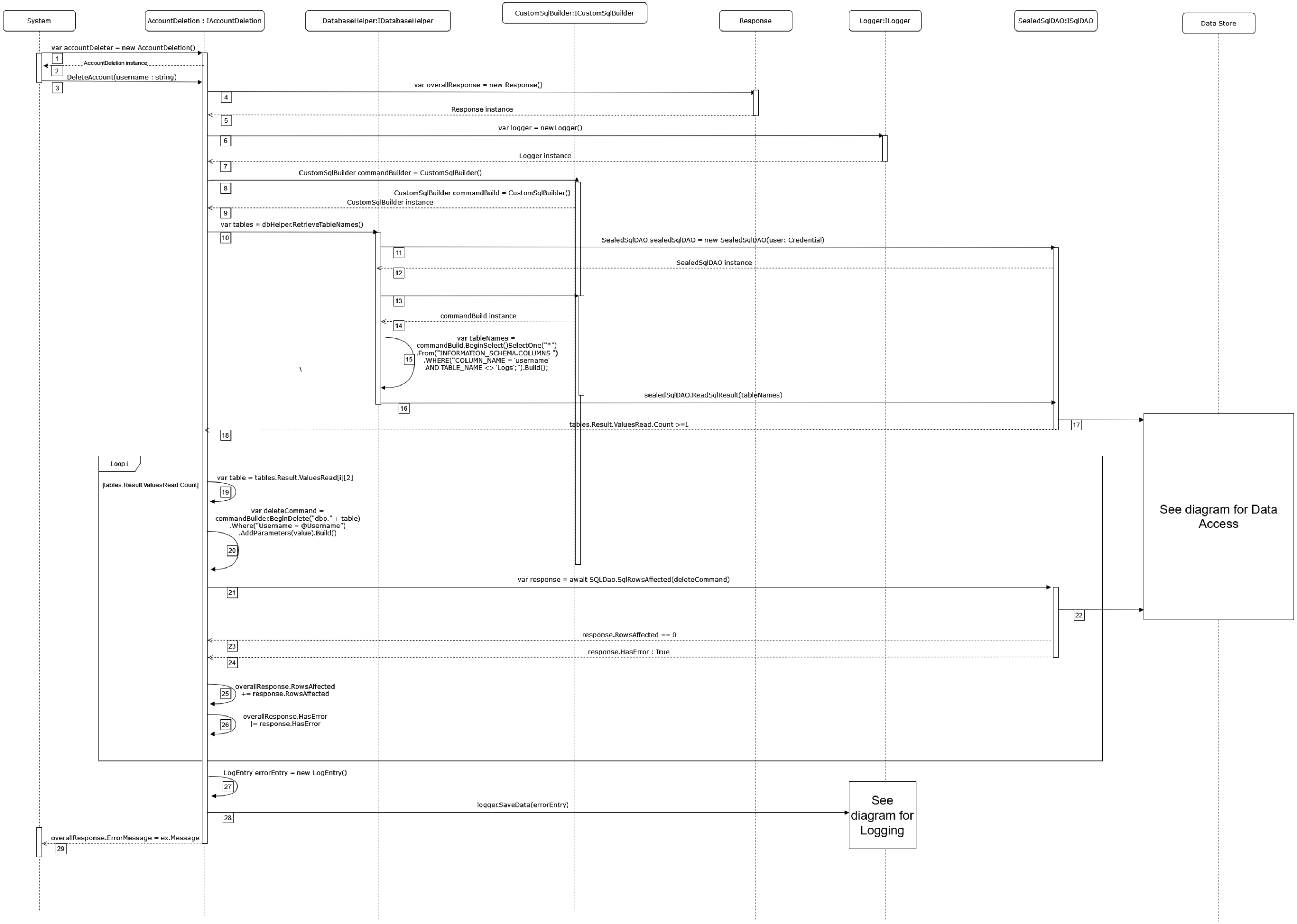
AccountDeletion.Success


System
AccountDeletion : IAccountDeletion
DatabaseHelper:IDatabaseHelper
CustomSqlBuilder:ICustomSqlBuilder
Response
Logger:ILogger
SealedSqlDAO:ISqlDAO
Data Store
var accountDeleter = new AccountDeletion()
1
AccountDeletion instance
2
DeleteAccount(username : string)
3
var overallResponse = new Response()
4
Response instance
5
var logger = newLogger()
6
Logger instance
7
CustomSqlBuilder commandBuilder = CustomSqlBuilder()
8
CustomSqlBuilder commandBuild = CustomSqlBuilder()
CustomSqlBuilder instance
9
var tables = dbHelper.RetrieveTableNames()
10
SealedSqlDAO sealedSqlDAO = new SealedSqlDAO(user: Credential)
11
SealedSqlDAO instance
12
13
commandBuild instance
14
var tableNames =
commandBuild.BeginSelect()SelectOne("*")
.From("INFORMATION_SCHEMA.COLUMNS ")
.WHERE("COLUMN_NAME = 'username'
AND TABLE_NAME <> 'Logs';").Build();
15
sealedSqlDAO.ReadSqlResult(tableNames)
16
17
tables.Result.ValuesRead.Count >=1
18
Loop i
[tables.Result.ValuesRead.Count]
var table = tables.Result.ValuesRead[i][2]
19
var deleteCommand =
commandBuilder.BeginDelete("dbo." + table)
.Where("Username = @Username")
.AddParameters(value).Build()
20
var response = await SQLDao.SqlRowsAffected(deleteCommand)
21
22
response.RowsAffected > 0
23
response.HasError : False
24
overallResponse.RowsAffected
+= response.RowsAffected
25
overallResponse.HasError
|= response.HasError
26
See diagram for Data Access
LogEntry entry = new LogEntry()
27
logger.SaveData(entry)
28
See diagram for Logging
overallResponse
29

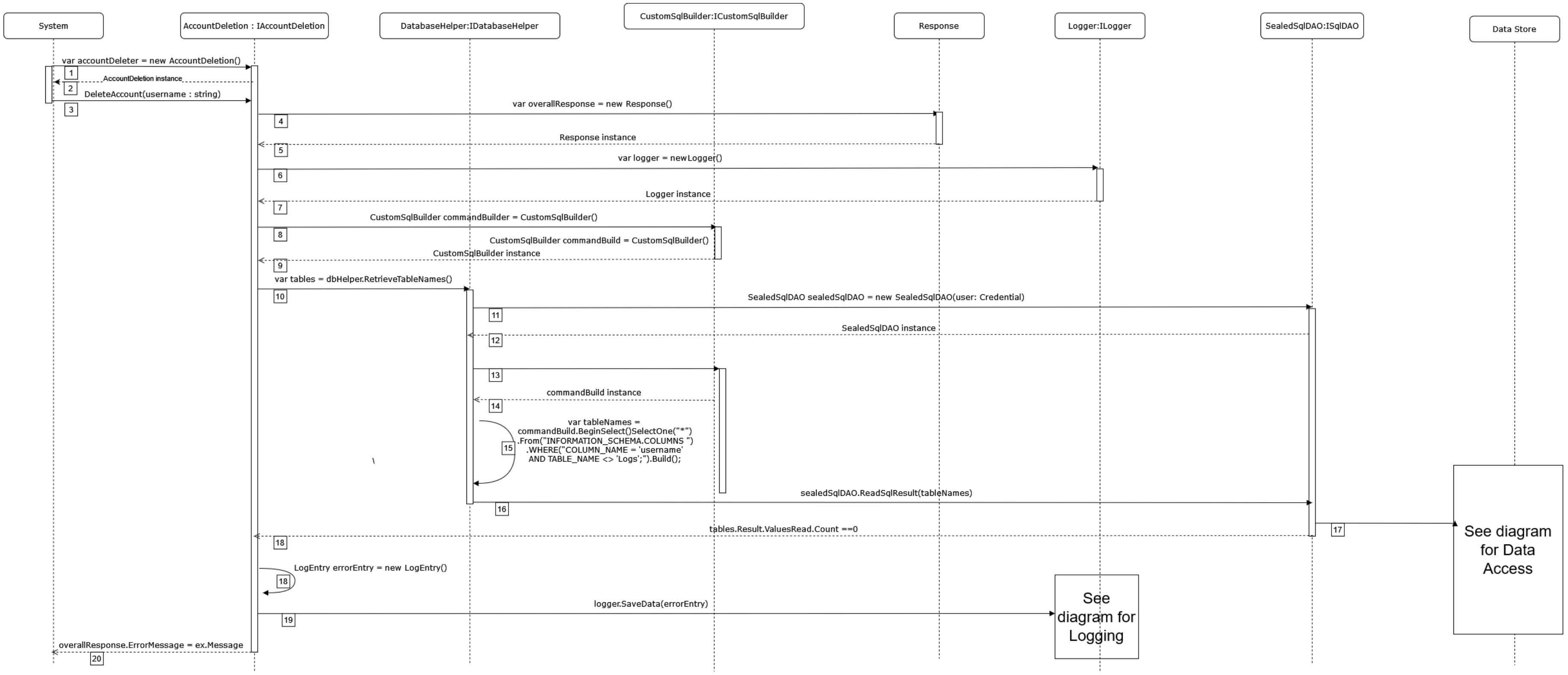
AccountDeletion.Failure


System
AccountDeletion : IAccountDeletion
DatabaseHelper:IDatabaseHelper
CustomSqlBuilder:ICustomSqlBuilder
Response
Logger:ILogger
SealedSqlDAO:ISqlDAO
Data Store
var accountDeleter = new AccountDeletion()
1
AccountDeletion instance
2
DeleteAccount(username : string)
3
var overallResponse = new Response()
4
Response instance
5
var logger = newLogger()
6
Logger instance
7
CustomSqlBuilder commandBuilder = CustomSqlBuilder()
8
CustomSqlBuilder commandBuild = CustomSqlBuilder()
CustomSqlBuilder instance
9
var tables = dbHelper.RetrieveTableNames()
10
SealedSqlDAO sealedSqlDAO = new SealedSqlDAO(user: Credential)
11
SealedSqlDAO instance
12
13
commandBuild instance
14
var tableNames =
commandBuild.BeginSelect()SelectOne("*")
.From("INFORMATION_SCHEMA.COLUMNS ")
.WHERE("COLUMN_NAME = 'username'
AND TABLE_NAME <> 'Logs';").Build();
15
\
sealedSqlDAO.ReadSqlResult(tableNames)
16
17
tables.Result.ValuesRead.Count >=1
18
See diagram for Data Access
Loop i
[tables.Result.ValuesRead.Count]
var table = tables.Result.ValuesRead[i][2]
19
var deleteCommand =
commandBuilder.BeginDelete("dbo." + table)
.Where("Username = @Username")
.AddParameters(value).Build()
20
var response = await SQLDao.SqlRowsAffected(deleteCommand)
21
22
response.RowsAffected >= 0
23
response.HasError : True
24
overallResponse.RowsAffected
+= response.RowsAffected
25
overallResponse.HasError
|= response.HasError
26
LogEntry errorEntry = new LogEntry()
27
logger.SaveData(errorEntry)
28
See diagram for Logging
overallResponse.ErrorMessage = ex.Message
29

AccountDeletion.Failure.Empty


System
AccountDeletion : IAccountDeletion
DatabaseHelper:IDatabaseHelper
CustomSqlBuilder:ICustomSqlBuilder
Response
Logger:ILogger
SealedSqlDAO:ISqlDAO
Data Store
var accountDeleter = new AccountDeletion()
1
AccountDeletion instance
2
DeleteAccount(username : string)
3
var overallResponse = new Response()
4
Response instance
5
var logger = newLogger()
6
Logger instance
7
CustomSqlBuilder commandBuilder = CustomSqlBuilder()
8
CustomSqlBuilder commandBuild = CustomSqlBuilder()
CustomSqlBuilder instance
9
var tables = dbHelper.RetrieveTableNames()
10
SealedSqlDAO sealedSqlDAO = new SealedSqlDAO(user: Credential)
11
SealedSqlDAO instance
12
13
commandBuild instance
14
var tableNames = commandBuild.BeginSelect()SelectOne("*") .From("INFORMATION_SCHEMA.COLUMNS ") .WHERE("COLUMN_NAME = 'username' AND TABLE_NAME <> 'Logs';").Build();
15
sealedSqlDAO.ReadSqlResult(tableNames)
16
17
tables.Result.ValuesRead.Count >=1
18
See diagram for Data Access
Loop i
[tables.Result.ValuesRead.Count]
var table = tables.Result.ValuesRead[i][2]
19
var deleteCommand = commandBuilder.BeginDelete("dbo." + table) .Where("Username = @Username") .AddParameters(value).Build()
20
var response = await SQLDao.SqlRowsAffected(deleteCommand)
21
22
response.RowsAffected == 0
23
response.HasError : True
24
overallResponse.RowsAffected += response.RowsAffected
25
overallResponse.HasError |= response.HasError
26
LogEntry errorEntry = new LogEntry()
27
logger.SaveData(errorEntry)
28
See diagram for Logging
overallResponse.ErrorMessage = ex.Message
29

AccountDeletion.DatabaseHelper.Failure



System

AccountDeletion : IAccountDeletion

DatabaseHelper:IDatabaseHelper

CustomSqlBuilder:ICustomSqlBuilder

Response

Logger:ILogger

SealedSqlDAO:ISqlDAO

Data Store

1 var accountDeleter = new AccountDeletion()

2 AccountDeletion instance

3 DeleteAccount(username : string)

4 var overallResponse = new Response()

5 Response instance

6 var logger = newLogger()

7 Logger instance

8 CustomSqlBuilder commandBuilder = CustomSqlBuilder()

CustomSqlBuilder commandBuild = CustomSqlBuilder()

9 CustomSqlBuilder instance

10 var tables = dbHelper.RetrieveTableNames()

11 SealedSqlDAO sealedSqlDAO = new SealedSqlDAO(user: Credential)

12 SealedSqlDAO instance

13

14 commandBuild instance

15 var tableNames =
commandBuild.BeginSelect()SelectOne("*")
.From("INFORMATION_SCHEMA.COLUMNS ")
.WHERE("COLUMN_NAME = 'username'
AND TABLE_NAME <> 'Logs';").Build();

\

16 sealedSqlDAO.ReadSqlResult(tableNames)

17

See diagram for Data Access

18 tables.Result.ValuesRead.Count ==0

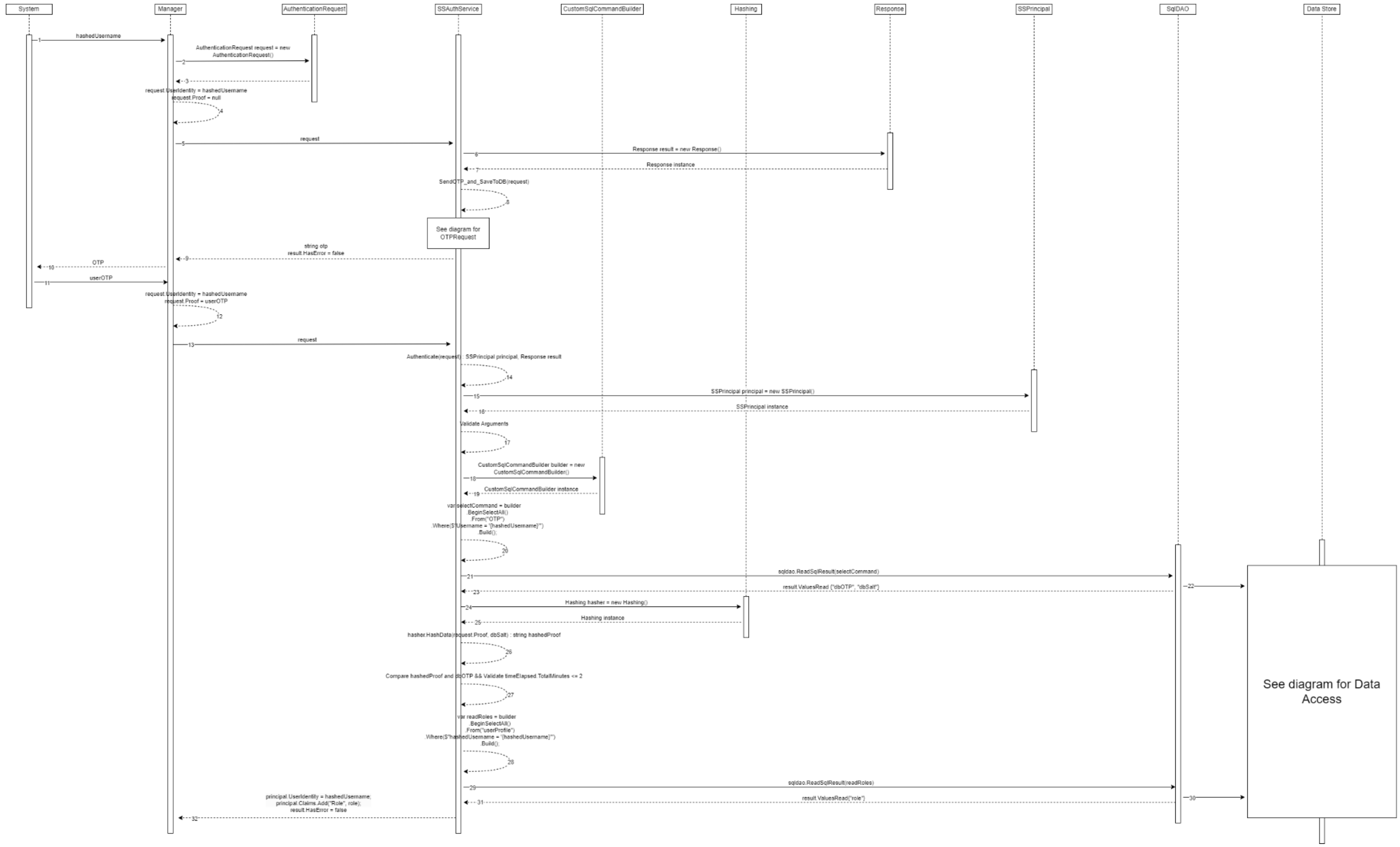
18 LogEntry errorEntry = new LogEntry()

19 logger.SaveData(errorEntry)

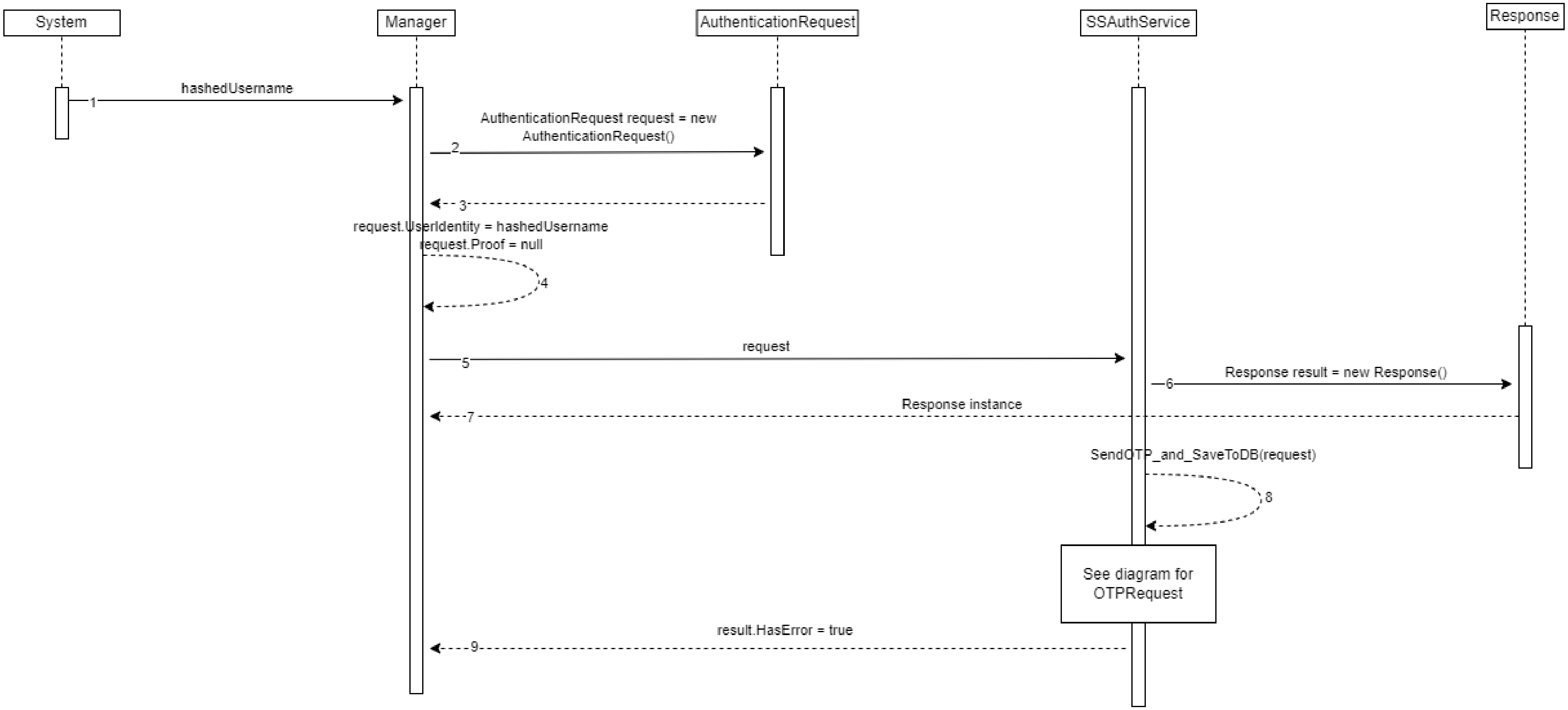
See diagram for Logging

20 overallResponse.ErrorMessage = ex.Message

Authentication.Success

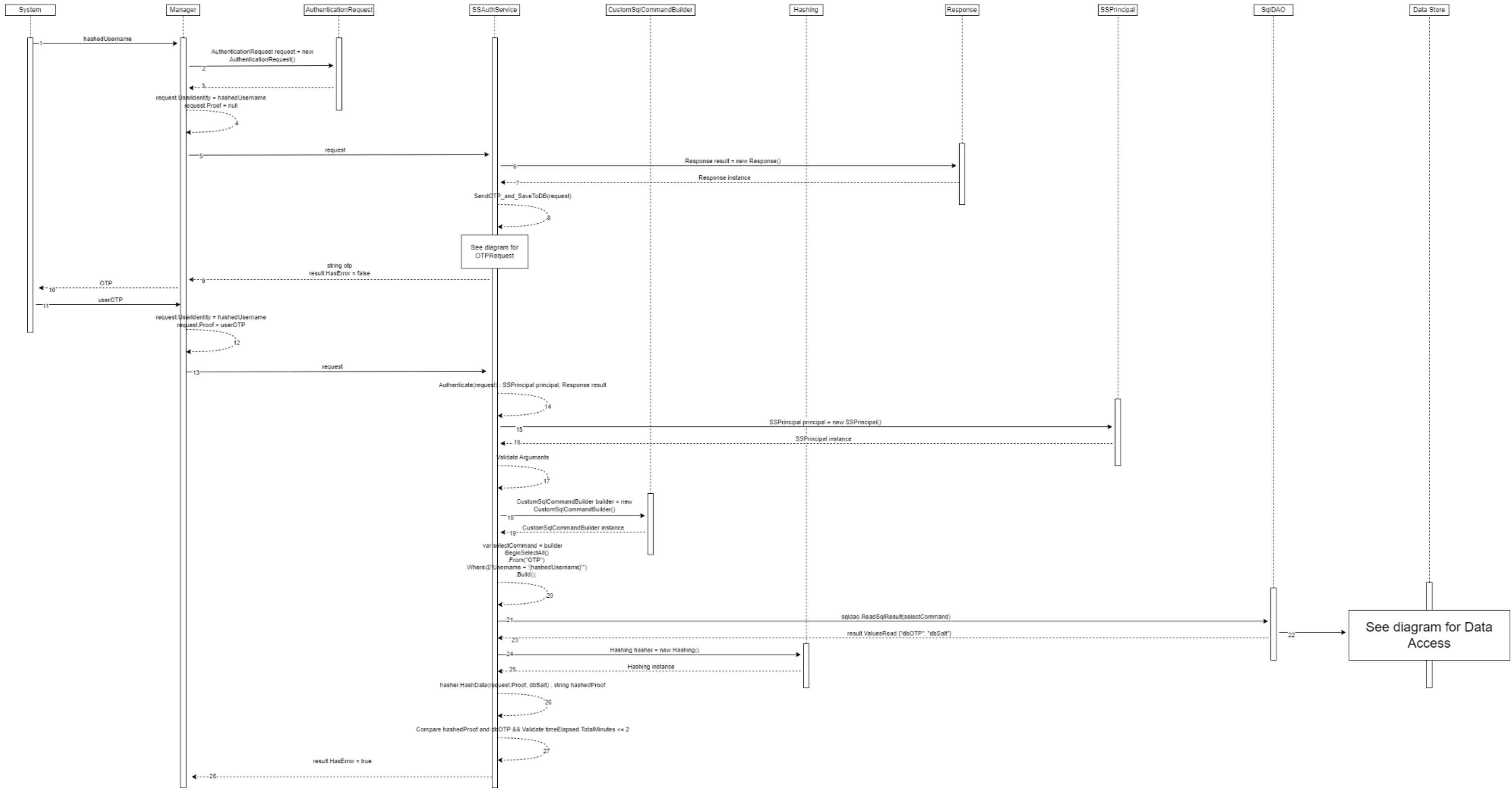

System
Manager
AuthenticationRequest
SSAuthService
CustomSqlCommandBuilder
Hashing
Response
SSPrincipal
SqlDAO
Data Store
1
hashedUsername
2
AuthenticationRequest request = new AuthenticationRequest()
3
request.UserIdentity = hashedUsername
request.Proof = null
4
5
request
6
Response result = new Response()
7
Response instance
SendOTP_and_SaveToDB(request)
8
See diagram for OTPRequest
9
string otp
result.HasError = false
10
OTP
11
userOTP
request.UserIdentity = hashedUsername
request.Proof = userOTP
12
13
request
Authenticate(request) : SSPrincipal principal, Response result
14
15
SSPrincipal principal = new SSPrincipal()
16
SSPrincipal instance
Validate Arguments
17
18
CustomSqlCommandBuilder builder = new CustomSqlCommandBuilder()
19
CustomSqlCommandBuilder instance
var selectCommand = builder
.BeginSelectAll()
.From("OTP")
.Where($"Username = '{hashedUsername}'")
.Build();
20
21
sqldao.ReadSqlResult(selectCommand)
22
23
result.ValuesRead {"dbOTP", "dbSalt"}
24
Hashing hasher = new Hashing()
25
Hashing instance
hasher.HashData(request.Proof, dbSalt) : string hashedProof
26
Compare hashedProof and dbOTP && Validate timeElapsed.TotalMinutes <= 2
27
var readRoles = builder
.BeginSelectAll()
.From("userProfile")
.Where($"hashedUsername = '{hashedUsername}'")
.Build();
28
29
sqldao.ReadSqlResult(readRoles)
30
31
result.ValuesRead("role")
principal.UserIdentity = hashedUsername;
principal.Claims.Add("Role", role);
result.HasError = false
32
See diagram for Data Access

AuthenticationFailure.UnregisteredUser

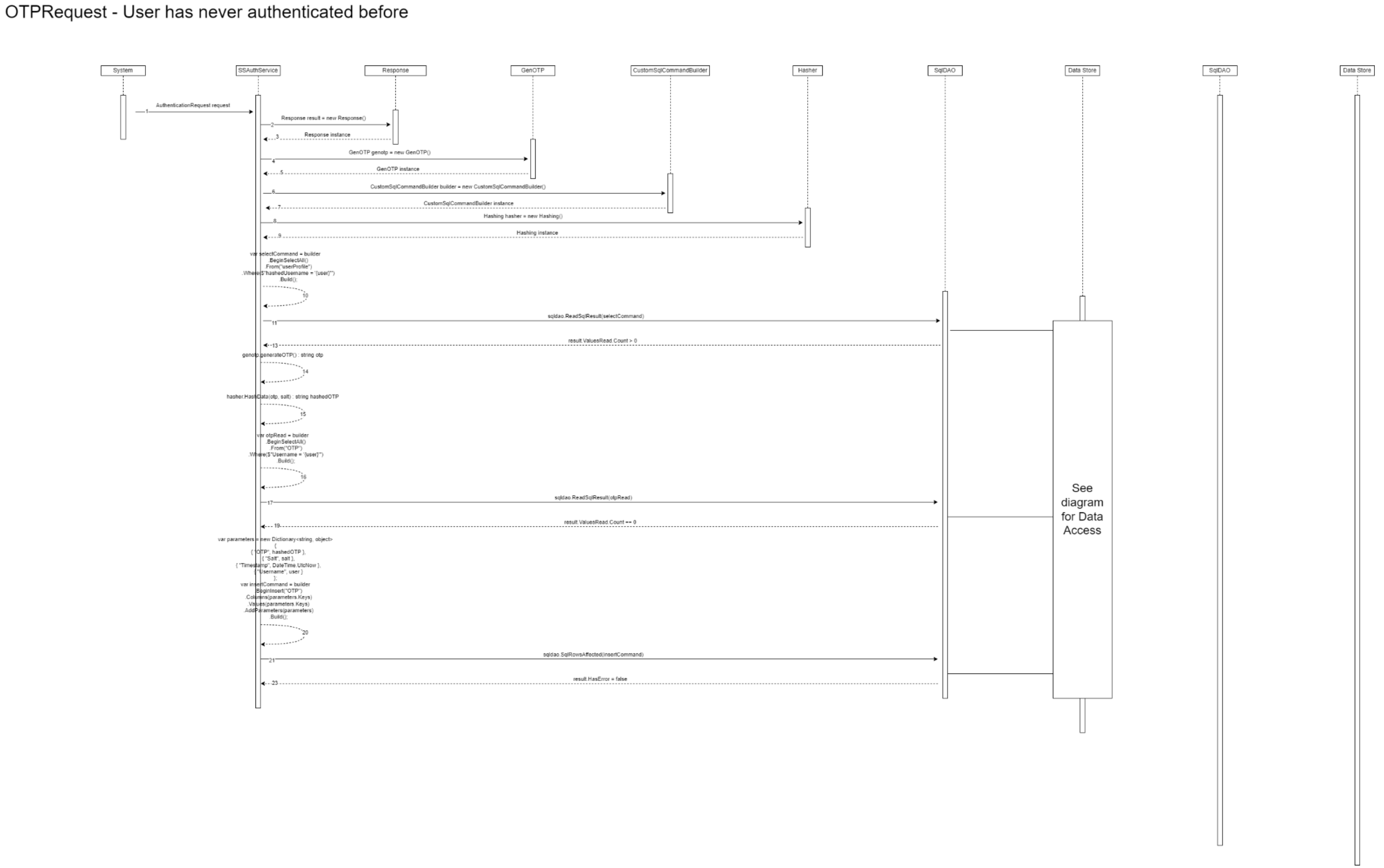

System
Manager
AuthenticationRequest
SSAuthService
Response
1
hashedUsername
AuthenticationRequest request = new AuthenticationRequest()
2
3
request.UserIdentity = hashedUsername
request.Proof = null
4
request
5
6
Response result = new Response()
Response instance
7
SendOTP_and_SaveToDB(request)
8
See diagram for OTPRequest
result.HasError = true
9

AuthenticationFailure.ExpiredOTP

Authentication Failure - Expired OTP


System
Manager
AuthenticationRequest
SSAuthService
CustomSqlCommandBuilder
Hashing
Response
SSPrincipal
SqlDAO
Data Store
1 hashedUsername
2 AuthenticationRequest request = new AuthenticationRequest()
3
request.UserIdentity = hashedUsername
request.Proof = null
4
5 request
6 Response result = new Response()
7 Response instance
SendOTP_and_SaveToDB(request)
8
See diagram for OTPRequest
9 string otp
result.HasError = false
10 OTP
11 userOTP
request.UserIdentity = hashedUsername
request.Proof = userOTP
12
13 request
Authenticate(request) : SSPrincipal principal, Response result
14
15 SSPrincipal principal = new SSPrincipal()
16 SSPrincipal instance
Validate Arguments
17
18 CustomSqlCommandBuilder builder = new CustomSqlCommandBuilder()
19 CustomSqlCommandBuilder instance
var selectCommand = builder
.BeginSelectAll()
.From("OTP")
.Where($"Username = '{hashedUsername}'")
.Build();
20
21 sqldao.ReadSqlResult(selectCommand)
22
See diagram for Data Access
23 result.ValuesRead {"dbOTP", "dbSalt"}
24 Hashing hasher = new Hashing()
25 Hashing instance
hasher.HashData(request.Proof, dbSalt) : string hashedProof
26
Compare hashedProof and dbOTP && Validate timeElapsed.TotalMinutes <= 2
27
result.HasError = true
28

OTPRequest - User has never authenticated before


OTPRequest - User has never authenticated before
System
SSAuthService
Response
GenOTP
CustomSqlCommandBuilder
Hasher
SqlDAO
Data Store
SqlDAO
Data Store
1 AuthenticationRequest request
2 Response result = new Response()
3 Response instance
4 GenOTP genotp = new GenOTP()
5 GenOTP instance
6 CustomSqlCommandBuilder builder = new CustomSqlCommandBuilder()
7 CustomSqlCommandBuilder instance
8 Hashing hasher = new Hashing()
9 Hashing instance
10
11 sqldao.ReadSqlResult(selectCommand)
13 result.ValuesRead.Count > 0
genotp.generateOTP() : string otp
14
hasher.HashData(otp, salt) : string hashedOTP
15
16
17 sqldao.ReadSqlResult(otpRead)
19 result.ValuesRead.Count == 0
20
21 sqldao.SqlRowsAffected(insertCommand)
23 result.HasError = false
See diagram for Data Access

```
var selectCommand = builder
    .BeginSelectAll()
    .From("userProfile")
    .Where($"hashedUsername = '{user}'")
    .Build();
```

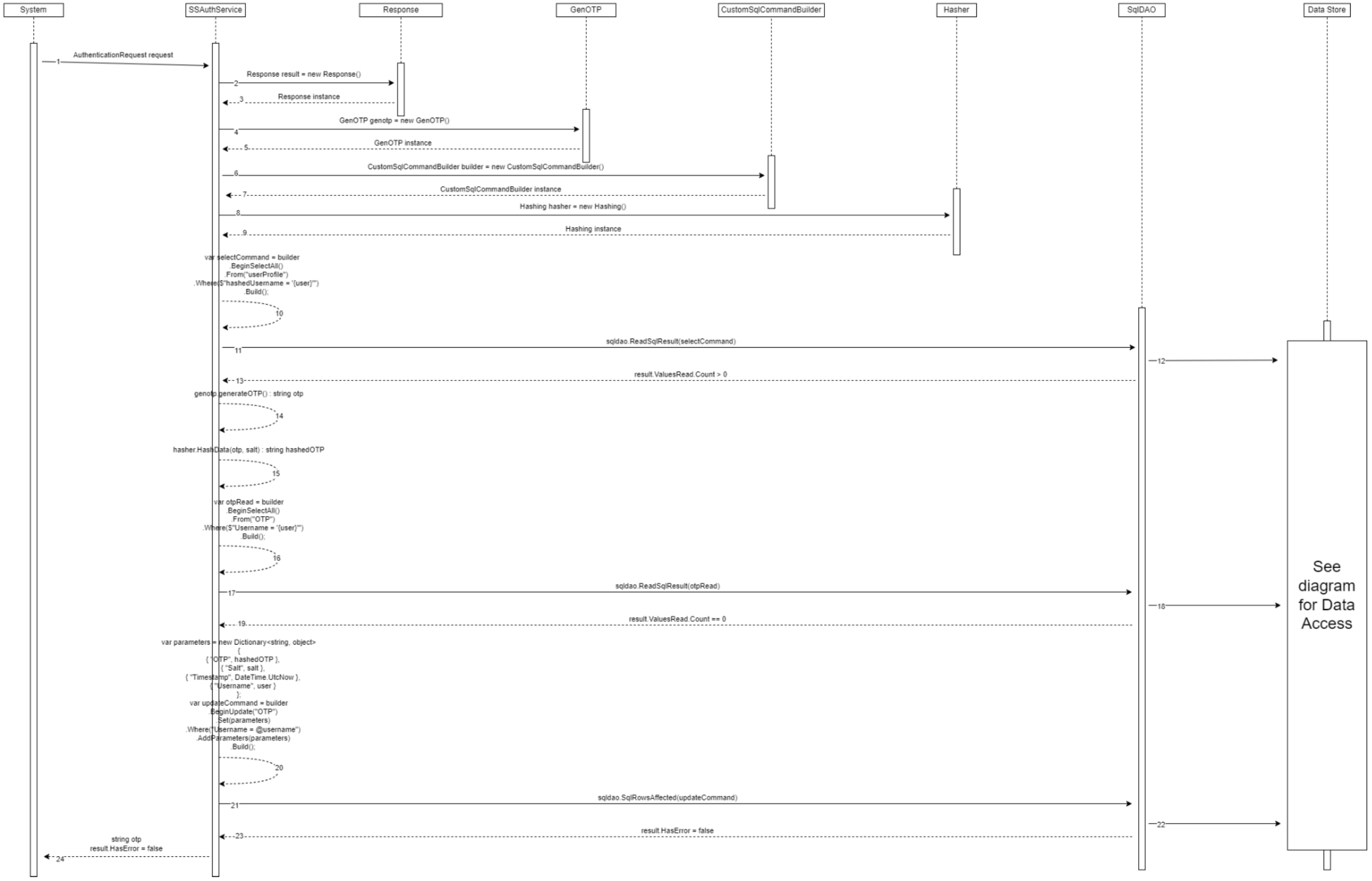
```
var otpRead = builder
    .BeginSelectAll()
    .From("OTP")
    .Where($"Username = '{user}'")
    .Build();
```

```
var parameters = new Dictionary<string, object>
{
    { "OTP", hashedOTP },
    { "Salt", salt },
    { "Timestamp", DateTime.UtcNow },
    { "Username", user }
};
var insertCommand = builder
    .BeginInsert("OTP")
    .Columns(parameters.Keys)
    .Values(parameters.Keys)
    .AddParameters(parameters)
    .Build();
```

OTPRequest - User has authenticated before

OTPRequest - User has authenticated before

Key difference on line 20 & 21


System
SSAuthService
Response
GenOTP
CustomSqlCommandBuilder
Hasher
SqlDAO
Data Store
1 AuthenticationRequest request
2 Response result = new Response()
3 Response instance
4 GenOTP genotp = new GenOTP()
5 GenOTP instance
6 CustomSqlCommandBuilder builder = new CustomSqlCommandBuilder()
7 CustomSqlCommandBuilder instance
8 Hashing hasher = new Hashing()
9 Hashing instance
10
11 sqldao.ReadSqlResult(selectCommand)
12
13 result.ValuesRead.Count > 0
14
15
16
17 sqldao.ReadSqlResult(otpRead)
18
19 result.ValuesRead.Count == 0
20
21 sqldao.SqlRowsAffected(updateCommand)
22
23 result.HasError = false
24
See diagram for Data Access

```
var selectCommand = builder
    .BeginSelectAll()
    .From("userProfile")
    .Where($"hashedUsername = '{user}'")
    .Build();
```

```
genotp.generateOTP() : string otp
```

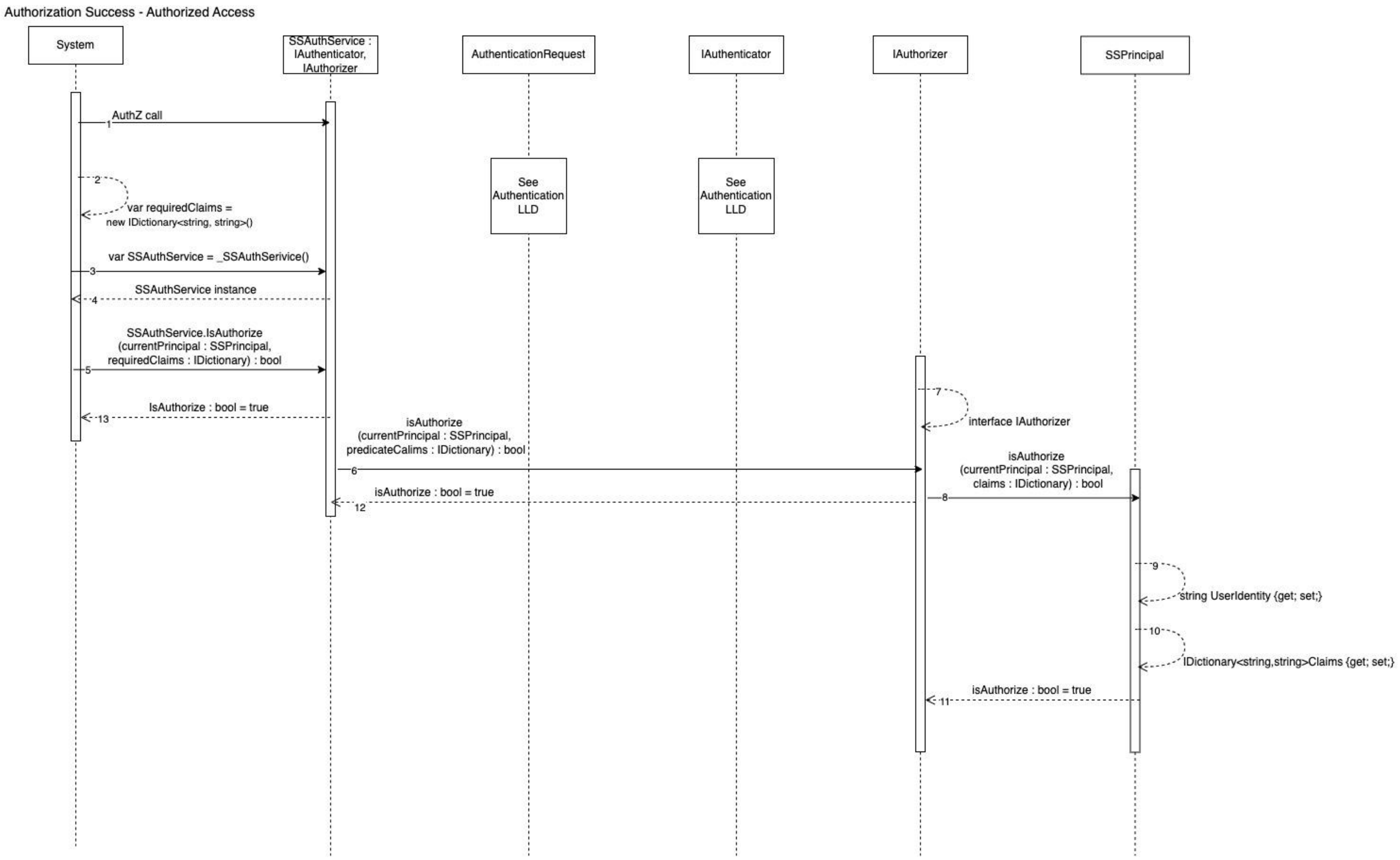
```
hasher.HashData(otp, salt) : string hashedOTP
```

```
var otpRead = builder
    .BeginSelectAll()
    .From("OTP")
    .Where($"Username = '{user}'")
    .Build();
```

```
var parameters = new Dictionary<string, object>
{
    { "OTP", hashedOTP },
    { "Salt", salt },
    { "Timestamp", DateTime.UtcNow },
    { "Username", user }
};
var updateCommand = builder
    .BeginUpdate("OTP")
    .Set(parameters)
    .Where("Username = @username")
    .AddParameters(parameters)
    .Build();
```

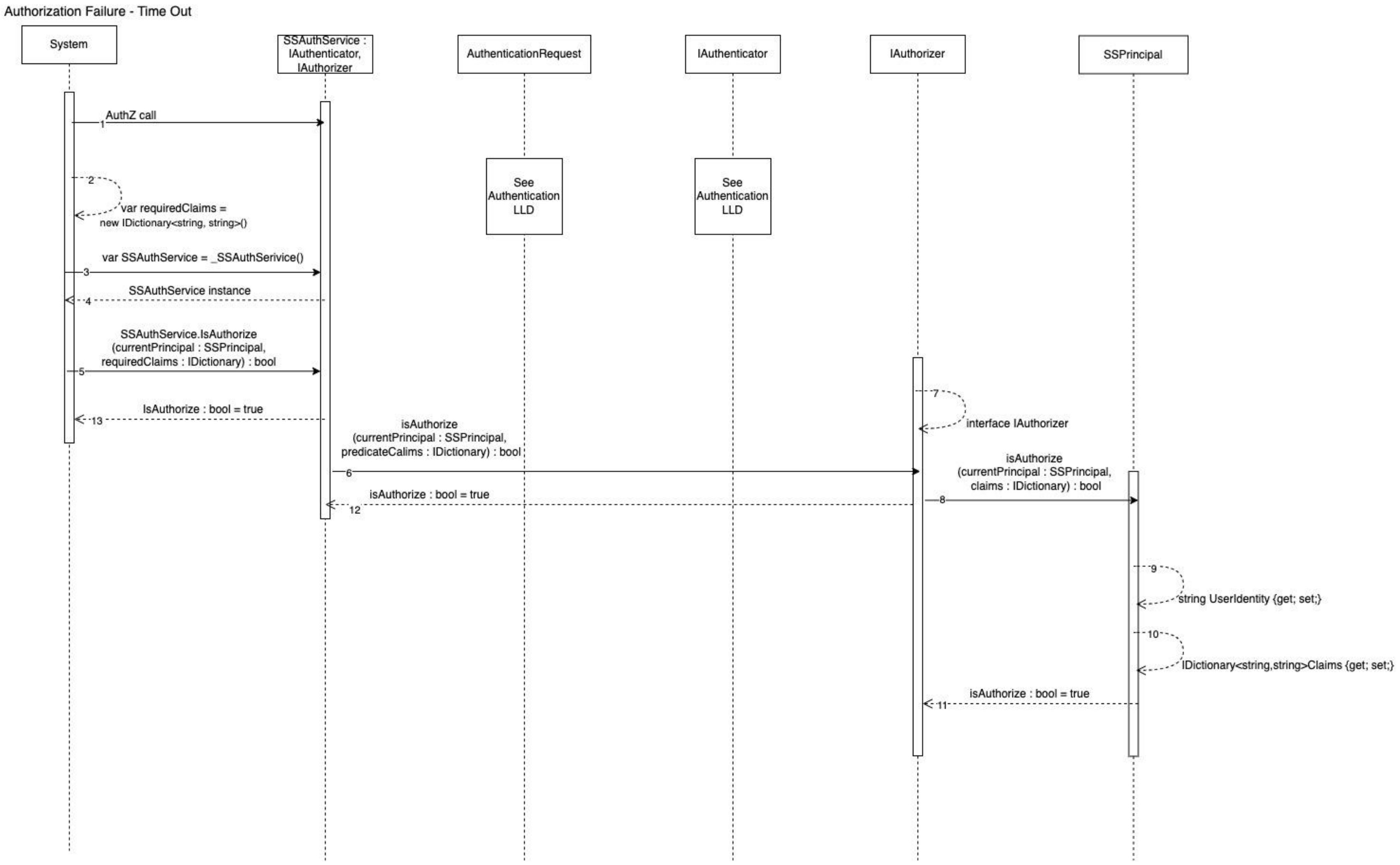
```
string otp
result.HasError = false
```

Authorization.Success


Authorization Success - Authorized Access
System
SSAuthService : IAuthenticator, IAuthorizer
AuthenticationRequest
IAuthenticator
IAuthorizer
SSPrincipal
See Authentication LLD
See Authentication LLD
1 AuthZ call
2 var requiredClaims = new IDictionary<string, string>()
3 var SSAuthService = _SSAuthSerivice()
4 SSAuthService instance
5 SSAuthService.IsAuthorize (currentPrincipal : SSPrincipal, requiredClaims : IDictionary) : bool
13 IsAuthorize : bool = true
6 isAuthorize (currentPrincipal : SSPrincipal, predicateCalims : IDictionary) : bool
12 isAuthorize : bool = true
7 interface IAuthorizer
8 isAuthorize (currentPrincipal : SSPrincipal, claims : IDictionary) : bool
9 string UserIdentity {get; set;}
10 IDictionary<string,string>Claims {get; set;}
11 isAuthorize : bool = true

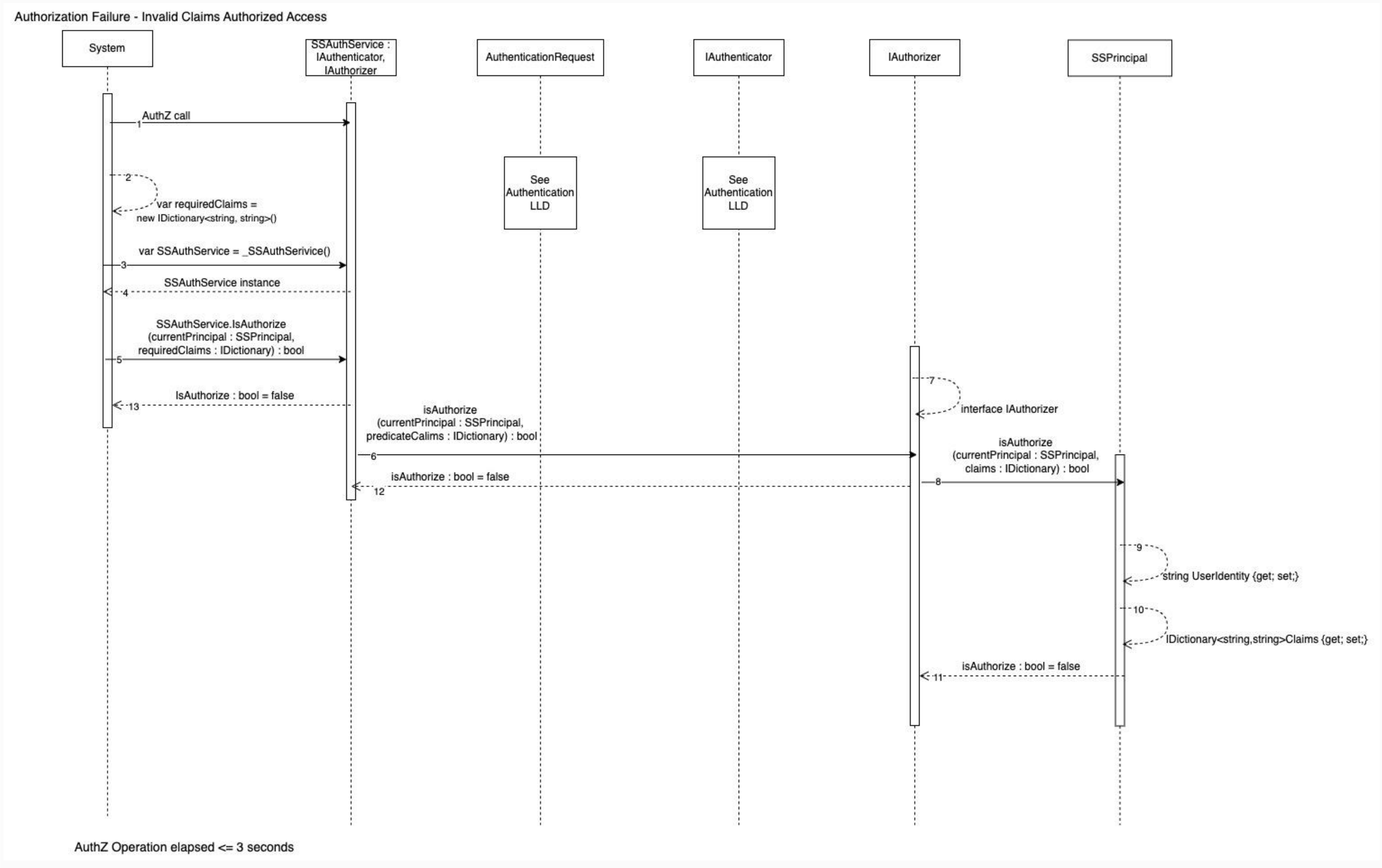
AuthZ Operation elapsed <= 3 seconds

Authorization.Failure.Timeout

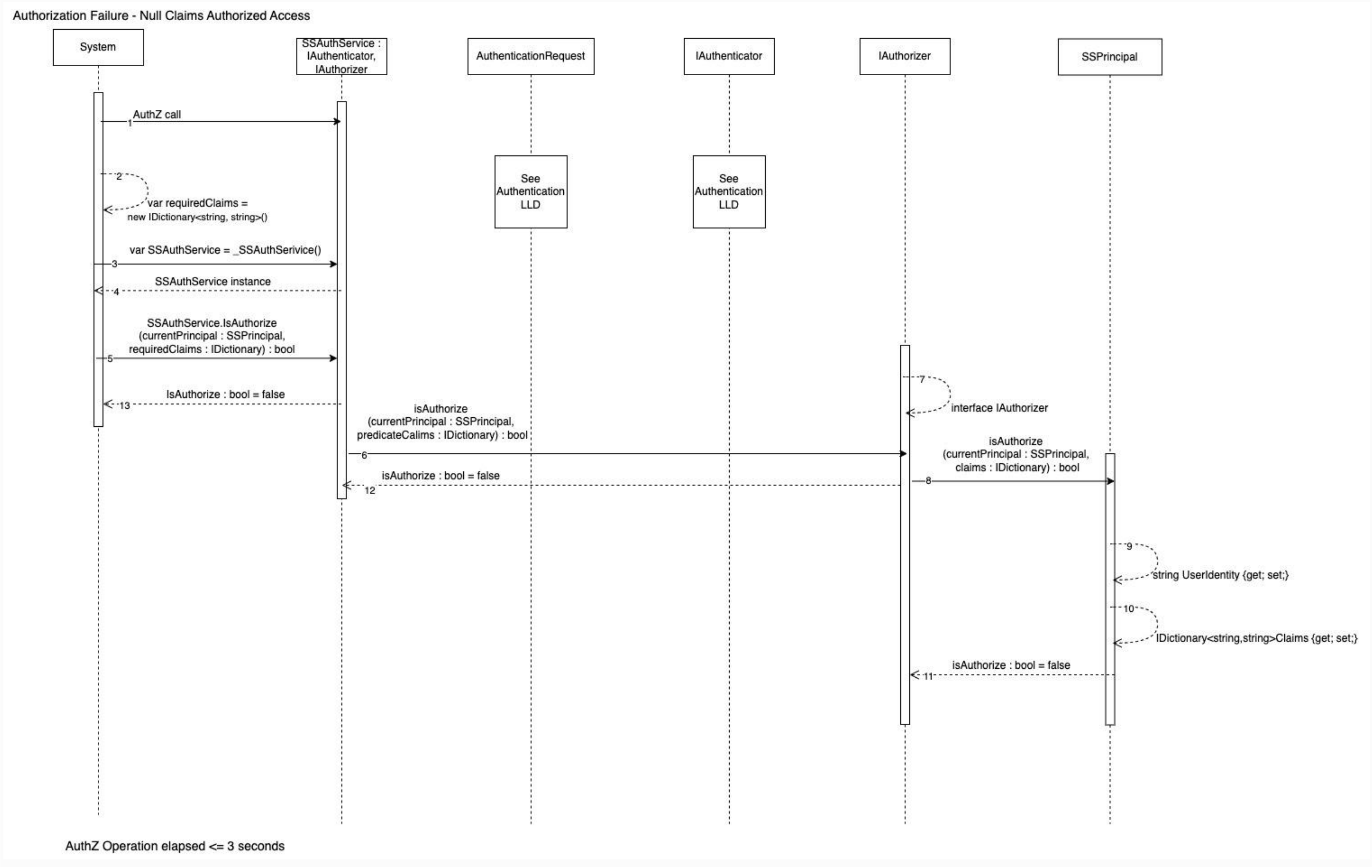

Authorization Failure - Time Out
System
SSAuthService : IAuthenticator, IAuthorizer
AuthenticationRequest
IAuthenticator
IAuthorizer
SSPrincipal
See Authentication LLD
See Authentication LLD
1 AuthZ call
2 var requiredClaims = new IDictionary<string, string>()
3 var SSAuthService = _SSAuthSerivice()
4 SSAuthService instance
5 SSAuthService.IsAuthorize (currentPrincipal : SSPrincipal, requiredClaims : IDictionary) : bool
6 isAuthorize (currentPrincipal : SSPrincipal, predicateCalims : IDictionary) : bool
7 interface IAuthorizer
8 isAuthorize (currentPrincipal : SSPrincipal, claims : IDictionary) : bool
9 string UserIdentity {get; set;}
10 IDictionary<string,string>Claims {get; set;}
11 isAuthorize : bool = true
12 isAuthorize : bool = true
13 IsAuthorize : bool = true

AuthZ Operation elapsed > 3 seconds

Authorization.Failure.InvalidClaims


Authorization Failure - Invalid Claims Authorized Access
System
SSAuthService : IAuthenticator, IAuthorizer
AuthenticationRequest
IAuthenticator
IAuthorizer
SSPrincipal
See Authentication LLD
See Authentication LLD
1 AuthZ call
2 var requiredClaims = new IDictionary<string, string>()
3 var SSAuthService = _SSAuthSerivice()
4 SSAuthService instance
5 SSAuthService.IsAuthorize (currentPrincipal : SSPrincipal, requiredClaims : IDictionary) : bool
6 isAuthorize (currentPrincipal : SSPrincipal, predicateCalims : IDictionary) : bool
7 interface IAuthorizer
8 isAuthorize (currentPrincipal : SSPrincipal, claims : IDictionary) : bool
9 string UserIdentity {get; set;}
10 IDictionary<string,string>Claims {get; set;}
11 isAuthorize : bool = false
12 isAuthorize : bool = false
13 IsAuthorize : bool = false
AuthZ Operation elapsed <= 3 seconds

Authorization.Failure.NullClaims


Authorization Failure - Null Claims Authorized Access
System
SSAuthService : IAuthenticator, IAuthorizer
AuthenticationRequest
IAuthenticator
IAuthorizer
SSPrincipal
See Authentication LLD
See Authentication LLD
1 AuthZ call
2 var requiredClaims = new IDictionary<string, string>()
3 var SSAuthService = _SSAuthSerivice()
4 SSAuthService instance
5 SSAuthService.IsAuthorize (currentPrincipal : SSPrincipal, requiredClaims : IDictionary) : bool
13 IsAuthorize : bool = false
6 isAuthorize (currentPrincipal : SSPrincipal, predicateCalims : IDictionary) : bool
12 isAuthorize : bool = false
7 interface IAuthorizer
8 isAuthorize (currentPrincipal : SSPrincipal, claims : IDictionary) : bool
9 string UserIdentity {get; set;}
10 IDictionary<string,string>Claims {get; set;}
11 isAuthorize : bool = false
AuthZ Operation elapsed <= 3 seconds

SQLCommandBuilder Methods: Parameters and Return Types

```
ICustomSqlCommandBuilder BeginInsert(string tableName);

ICustomSqlCommandBuilder Columns(IEnumerable<string> columns);

ICustomSqlCommandBuilder Values(IEnumerable<string> columns);

ICustomSqlCommandBuilder BeginUpdate(string tableName);

ICustomSqlCommandBuilder Set(Dictionary<string, object> columnValues);

ICustomSqlCommandBuilder Where(string whereClause);

ICustomSqlCommandBuilder BeginSelect();

ICustomSqlCommandBuilder BeginSelectAll();

ICustomSqlCommandBuilder SelectColumns(params string[] columns);

ICustomSqlCommandBuilder From(string tableName);

ICustomSqlCommandBuilder BeginDelete(string tableName);

ICustomSqlCommandBuilder Join(string joinTable, string fromColumn, string toColumn);

ICustomSqlCommandBuilder AddParameters(Dictionary<string, object> parameters);

SqlCommand Build();
```