Low Level Design

Team Gen ChimpanZ's

November 9, 2022

Team Leader

Mark Fastner

Team Members

Liam Joseph Abalos

Anh Huynh

Justin Le

Aster Lee

Brendan Paing

Github: https://github.com/markfastner/Nuclei

Table of Contents

Registration **2**
- Success Case 2
- Failure Case 1 4
- Failure Case 2 5
- Failure Case 3 6
- Failure Case 4 8
- Failure Case 5 8
- Failure Case 6 9

Logging **10**
- Success Case 1 10
- Success Case 2 11
- Success Case 3 12
- Success Case 4 13
- Failure Case 1 14
- Failure Case 2 15
- Failure Case 3 16
- Failure Case 4 17
- Failure Case 5 18

Registration Code **21**

Logging Code **30**

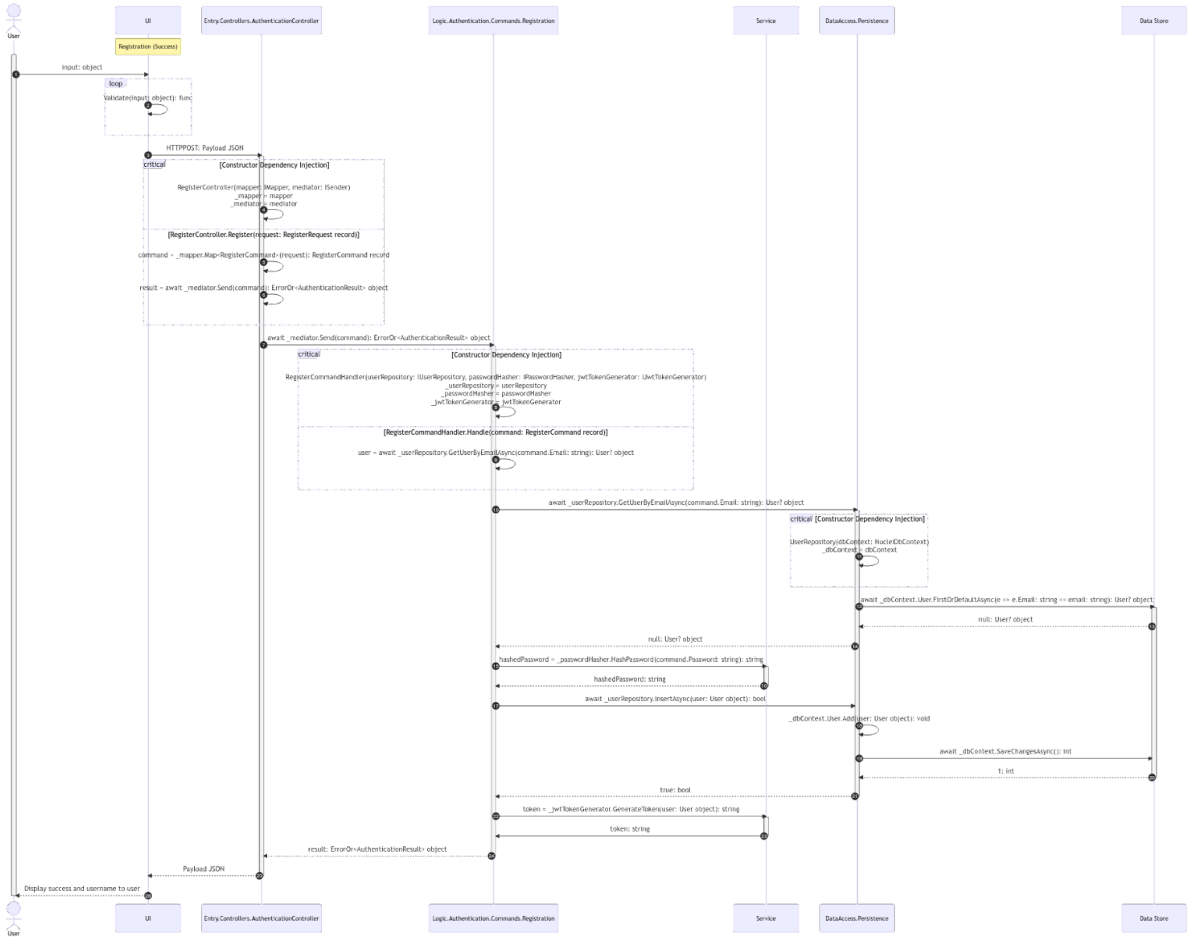
References **41**

Version Changelog **41**

Registration

Success Case

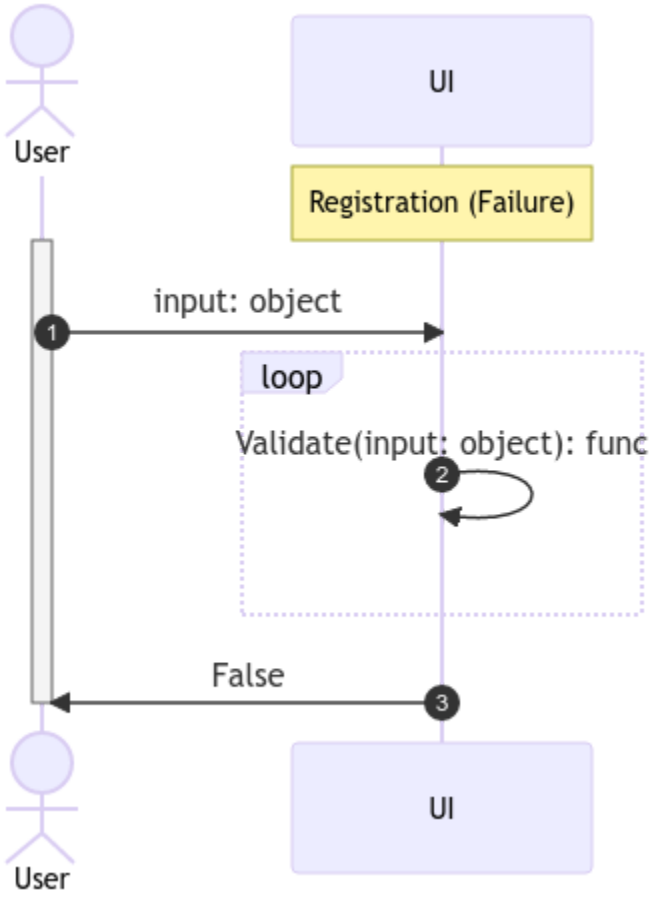
User registers with a valid email address and valid password. The system, without timing out, assigns the email as the user's unique username, displays a success message to the interface, and associates the unique username/email to the user.



Failure Case 1

User registers with an invalid email address

The user attempts to create an account but enters an incorrect email. Inside the UI layer we check to make sure that the information given by the user is of correct form. If the user inputs an incorrect email we will notify the user that the email is incorrect and they will need to enter a new one. We loop until the email that the user enters is a correct email(infinitely).
Fig. 2



Technical Description:
After the user chooses to register an account they are prompted to enter their information(first name, last name,email,password). The email they have entered is of incorrect form. Inside the front end component logic(.js) we use the yup library to define and check our input standards. If the email is not a real email the validatInfo(User) returns false and the user is then prompted

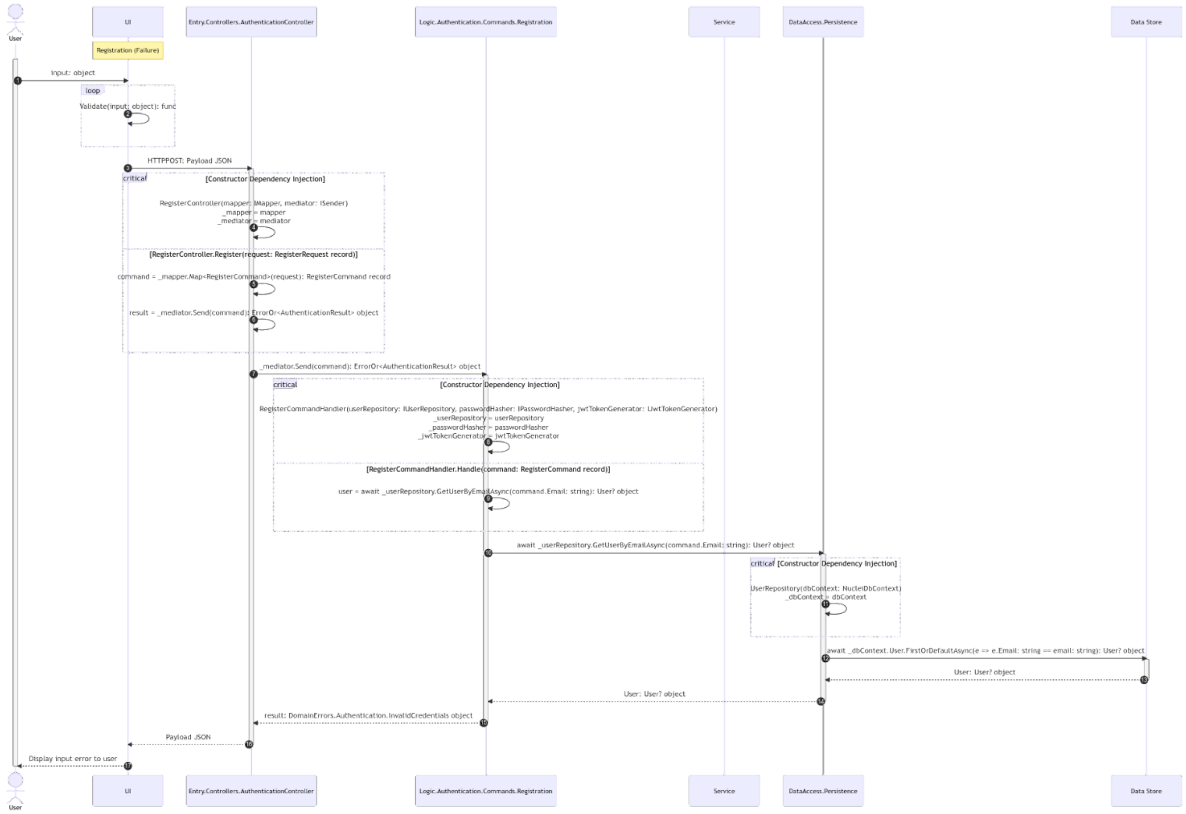
informing them of the mistake and they are able to enter new information. The user is able to enter their information however many times they want.

Failure Case 2

User registers with a valid email address, but it is not unique within the system. This causes our system to not assign a username.

After acceptable information is entered the UI layer sends an HTTP request to the entry layer. Upon receiving the HTTP request the Entry layer sends the request to the validator inside the business logic layer. In the business layer a check is made to see if the account already exists. To do this the business layer gives the information of the account to the data access layer which checks the data store if the account already exists inside the system. The email address already exists inside the data store so the account is not created and an “email already exists” failure response is sent back to the business logic layer. The business logic layer sends an invalid email failure response to the entry layer which sends the failure response to the ui notifying the user, “Invalid email provided. Retry again or contact system administrator”.

Fig. 3

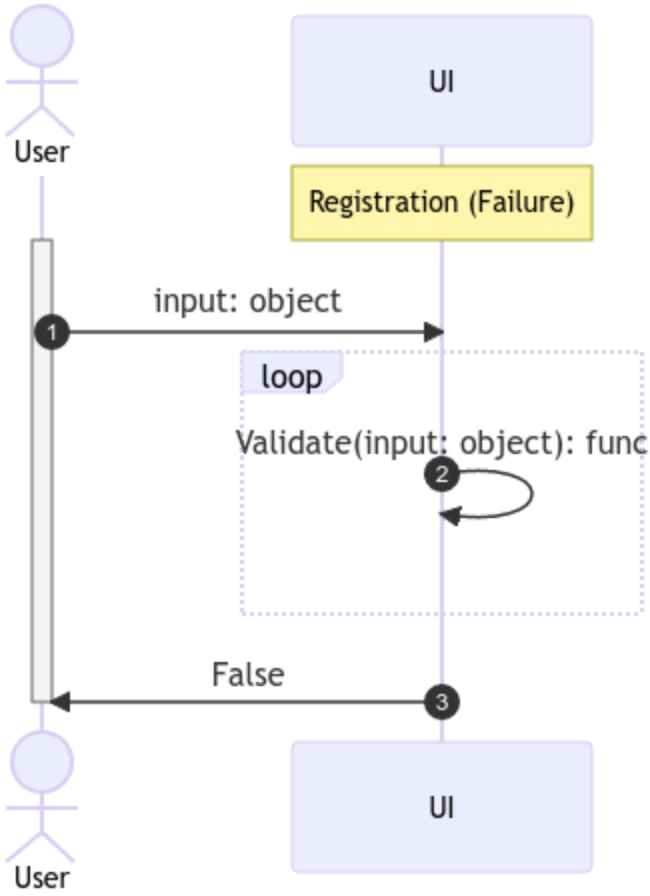


After acceptable information is entered by the user and an http request is made the mediation will find a handler for the respective request. The first thing the handler does is validate that the email exists inside our database. This is done through calling the GetUserByEmail function inside our business logic layer which has an email parameter. This function is a part of the data access layer which takes in the email and returns a nullable user object. Inside the getUserByEmail() function we use dbContext to query the database for the firstorDefault instance of the email. In this case the email already exists inside the database so the getUserByEmail() returns a user to the business logic layer. Inside the business logic layer we will have a condition that if the getUserByEmail function returns a user that an error of the type ErrorOr<AuthenticationResult> is sent back to the entry layer. This error is then sent to the ui layer which will display to the user the error message.

Failure Case 3

User registers with an invalid password

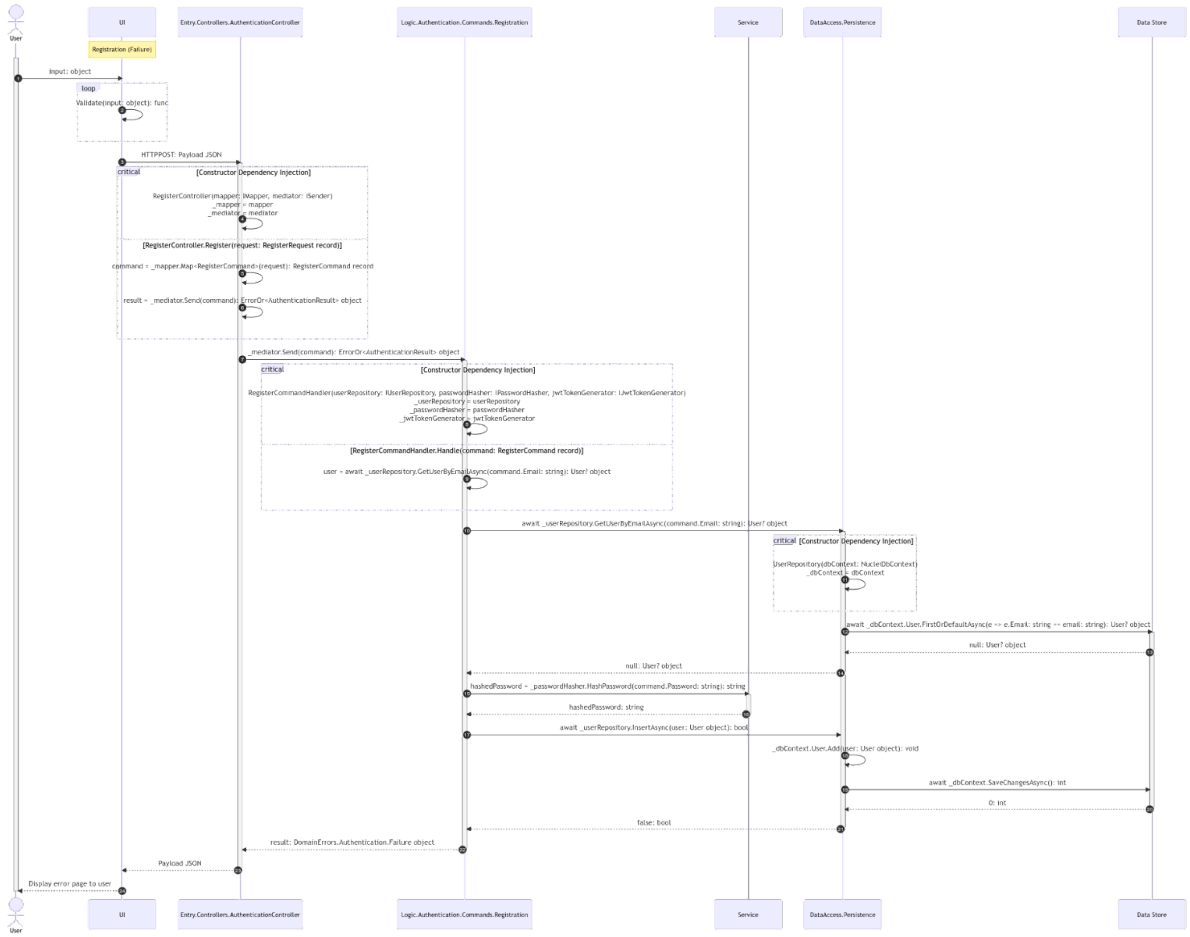
The user attempts to create an account but enters an incorrect password. Inside the UI layer we check to make sure that the information given by the user is of correct form. If the user inputs an incorrect password we will notify the user that the password is incorrect and they will need to enter a new one.



Technical Description:
After the user chooses to register an account they are prompted to enter their information(first name, last name,email,password). The password they have entered is of incorrect form. Inside the front end component logic(.js) we use the yup library to define and check our input standards. The password does not meet the following standards: Passwords must be at least 8 characters, must contain at least 1 capital and 1 lowercase character, at least 1 number, and at least 1 special character. The validatInfo(User) returns false and the user is then prompted informing them of the mistake and they are able to enter new information. The user is able to enter their information however many times they want.

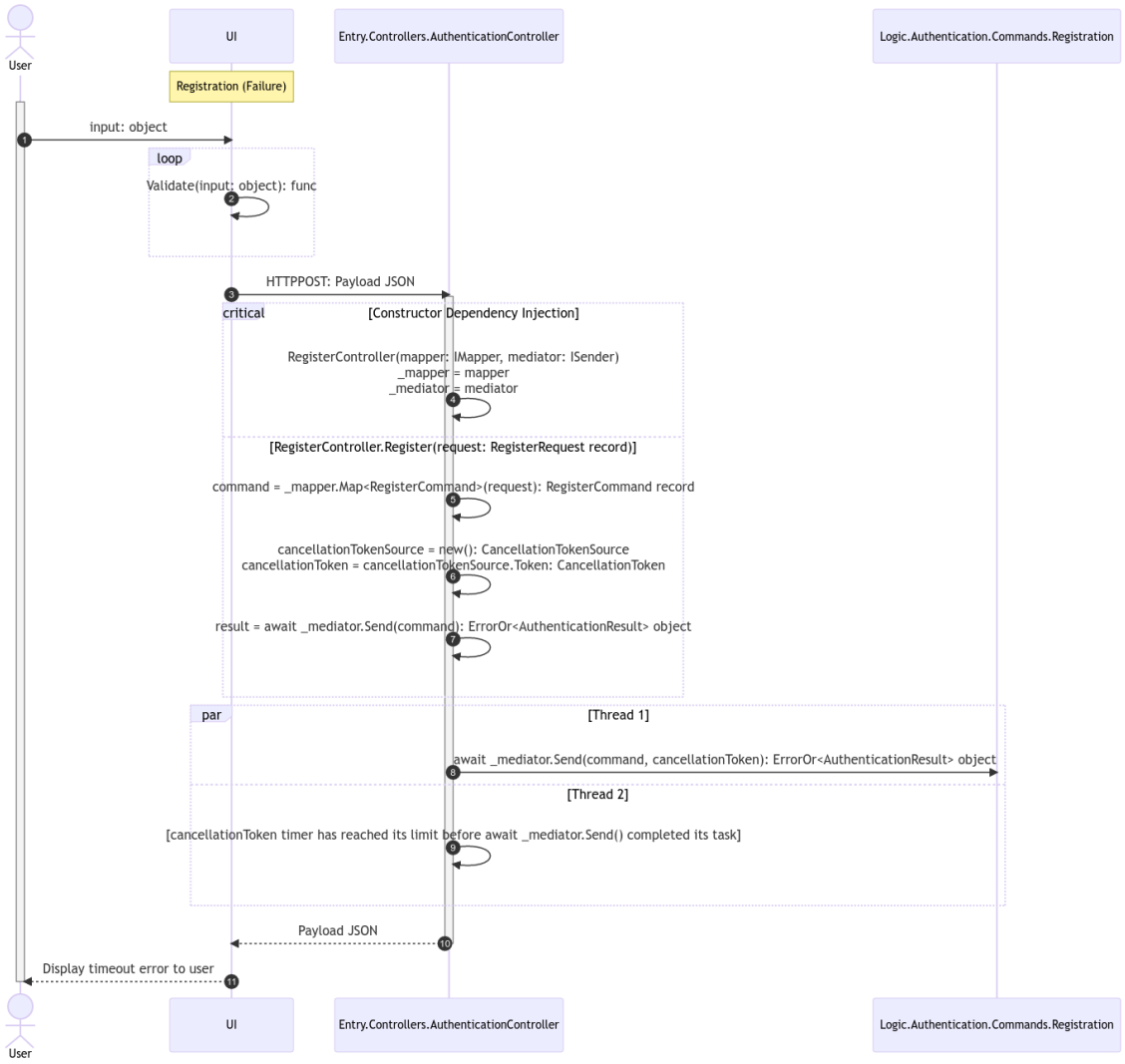
Failure Case 4

User enters correct information but the user is not able to be created inside the database.(database full)



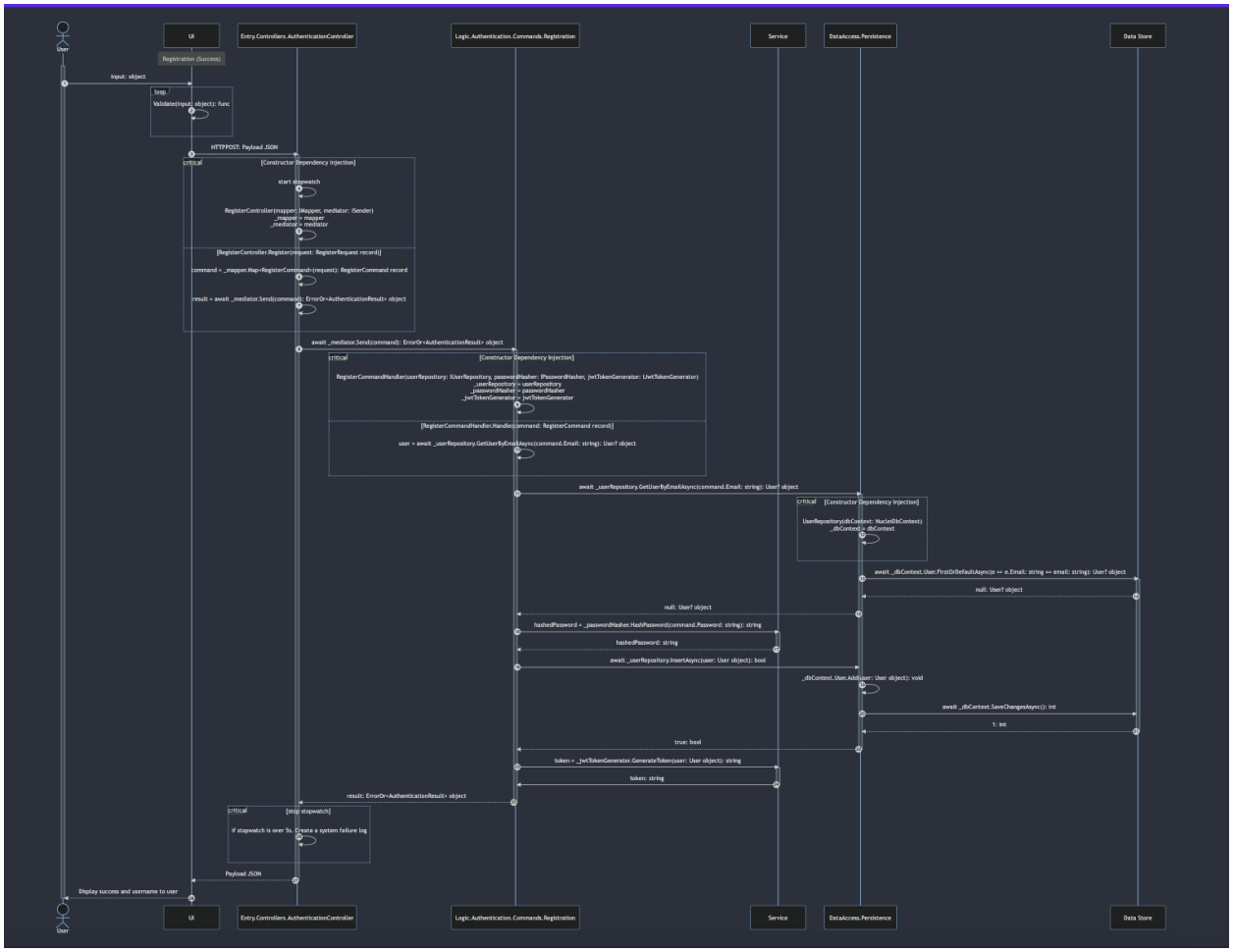
Failure Case 5

User registers, but the request timed out and data never reaches the Entry layer.



Failure Case 6

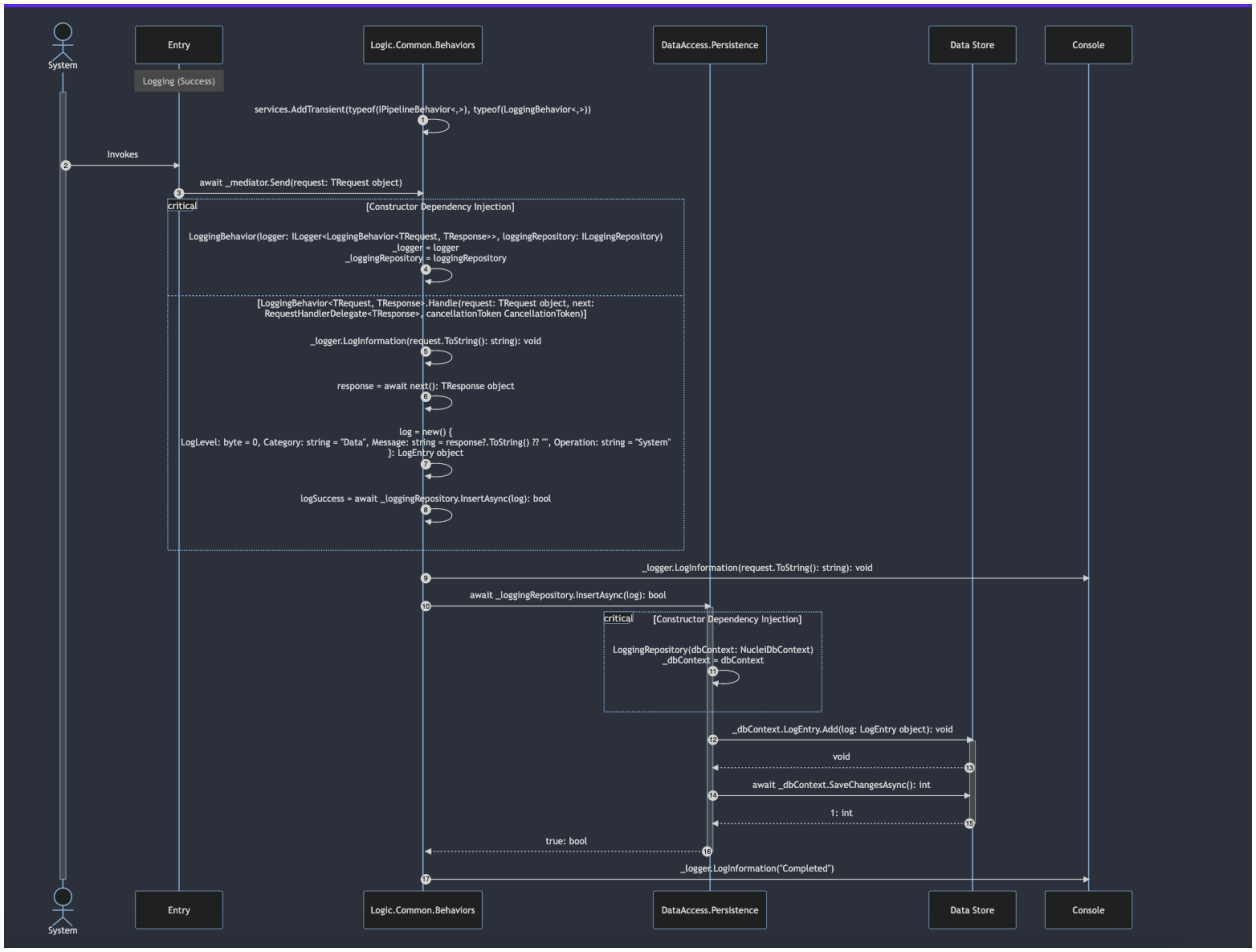
User registers, an account is created and the username is returned back to the user. Process takes longer than 5 seconds



Logging

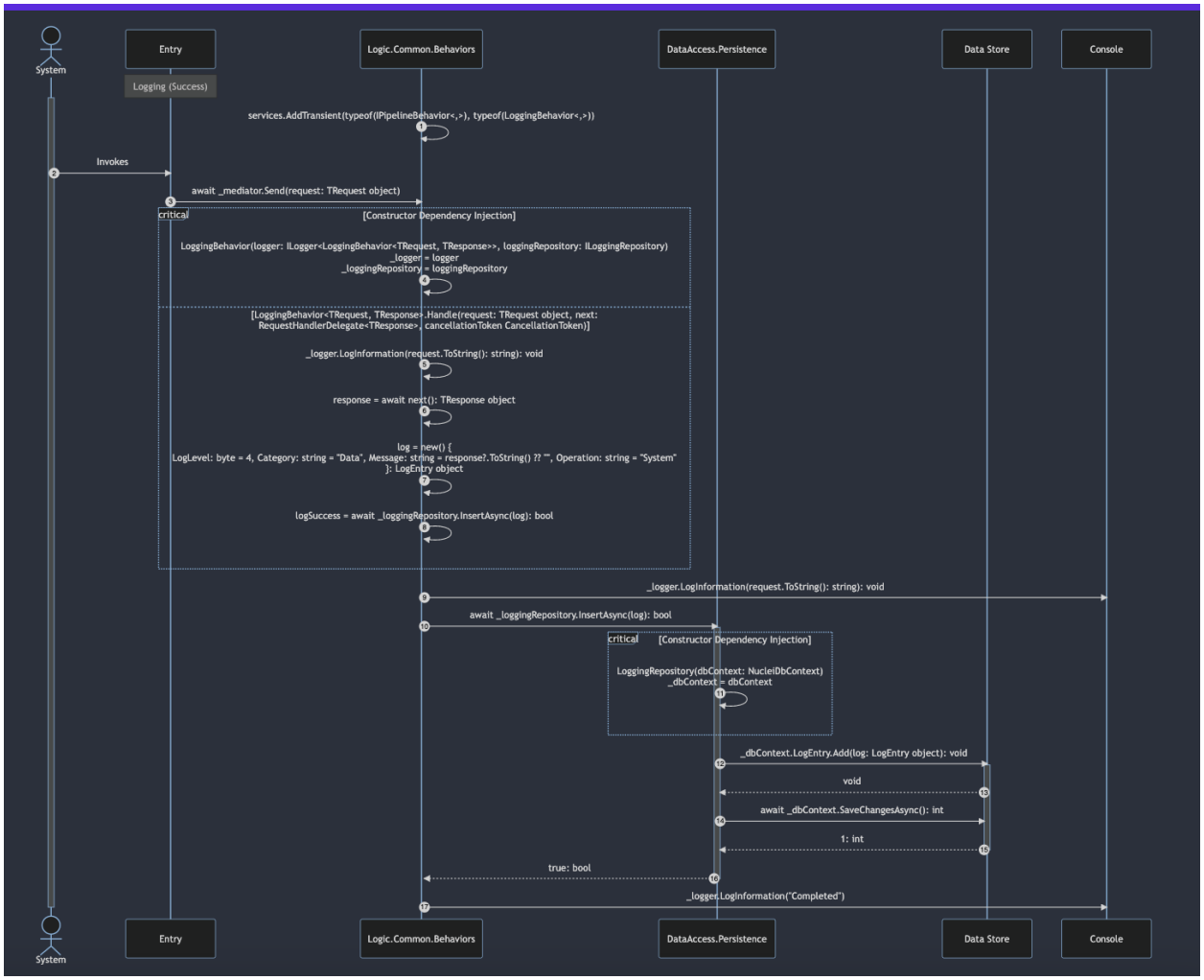
Success Case 1

The system logs system success events



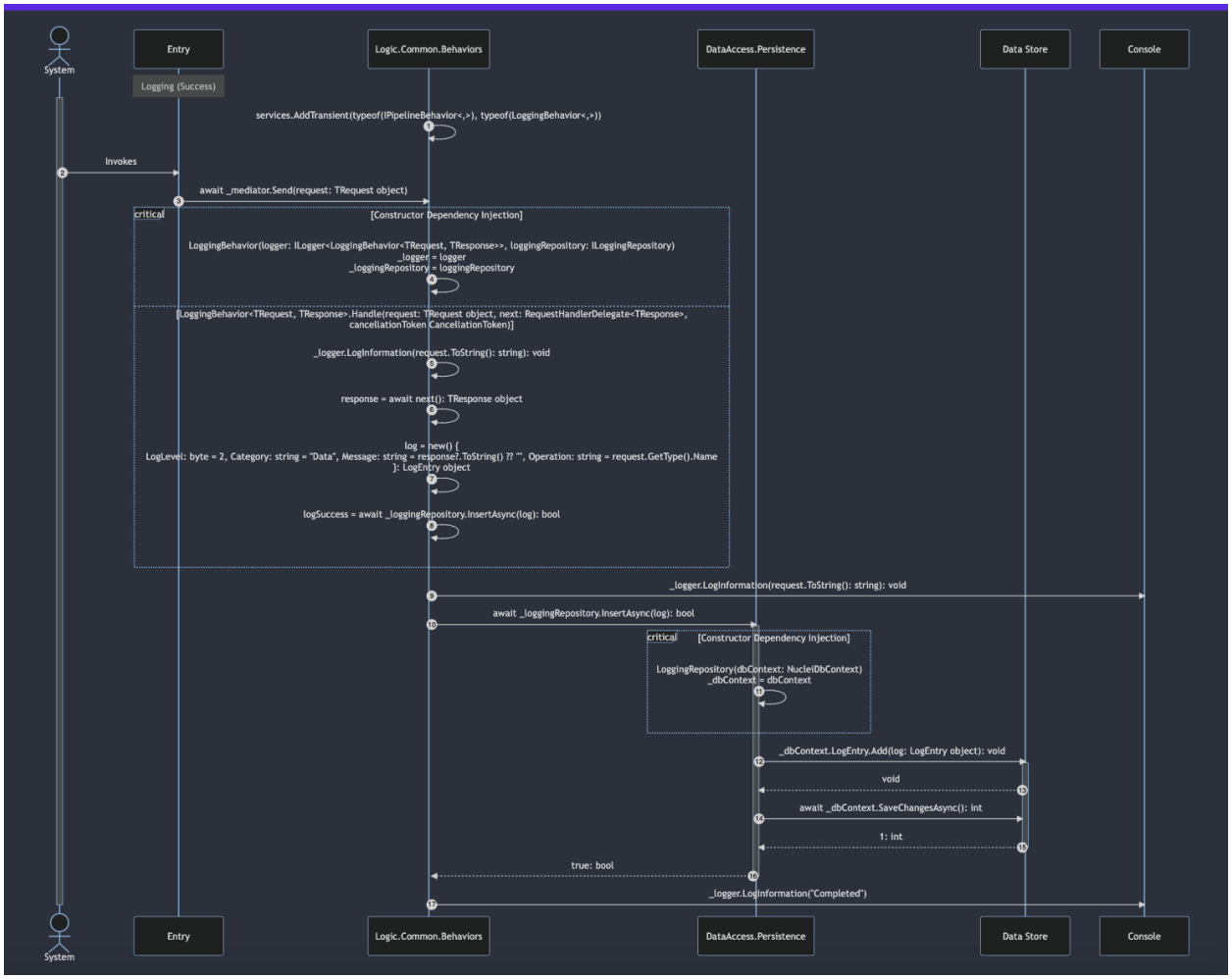
Success Case 2

The system logs system failure events



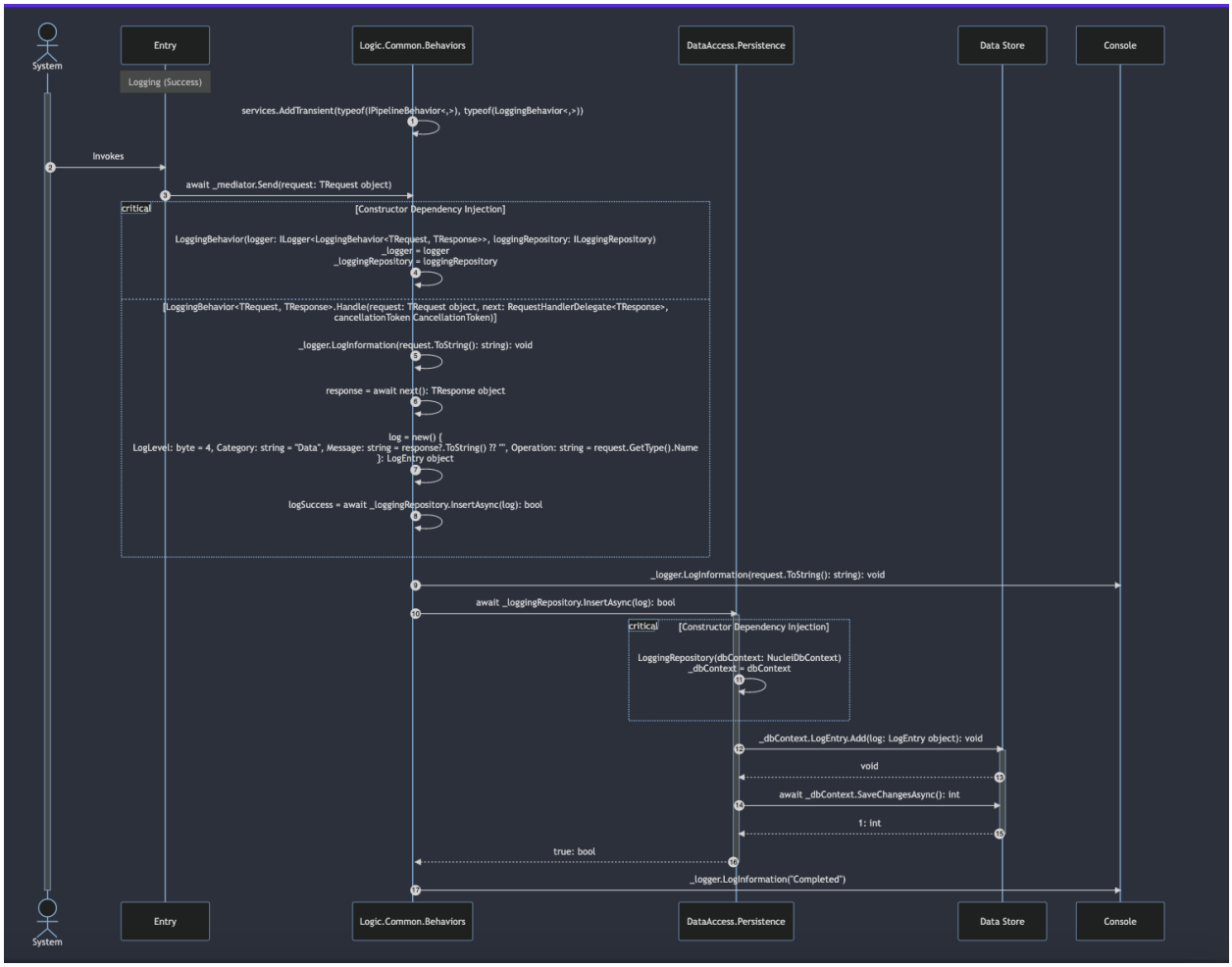
Success Case 3

The system logs user success events



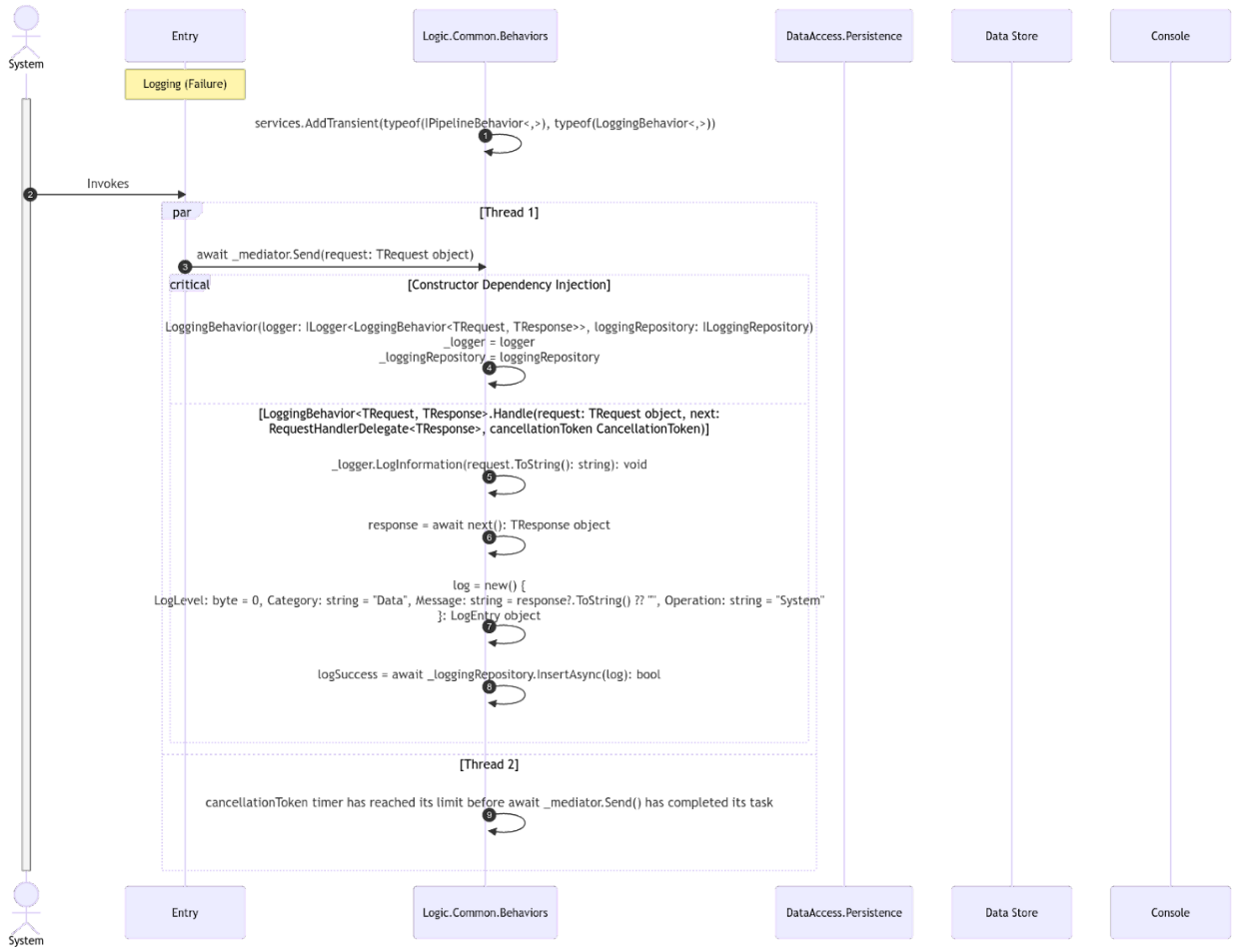
Success Case 4

The system logs user failure events



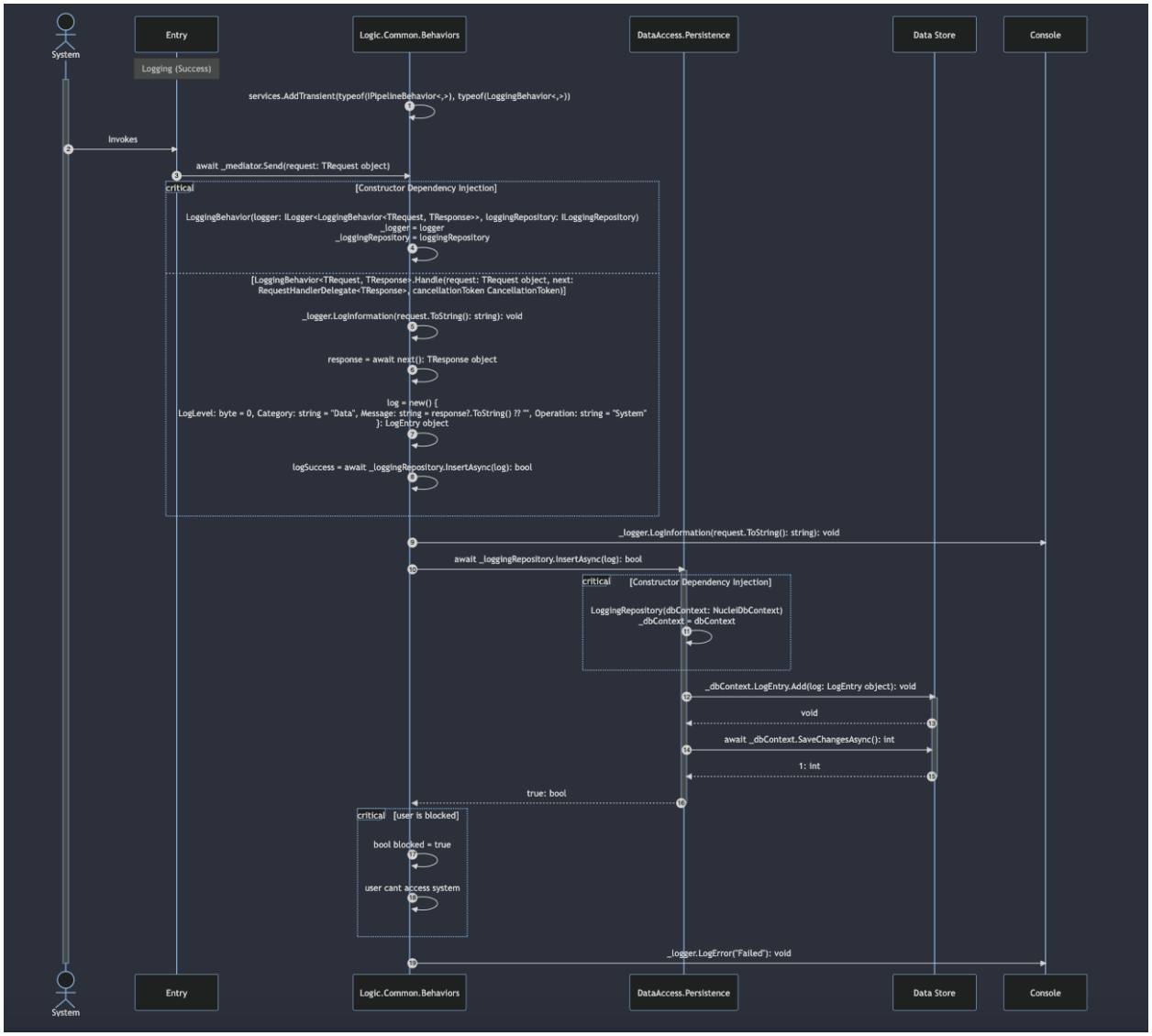
Failure Case 1

The logging pipeline timed out.



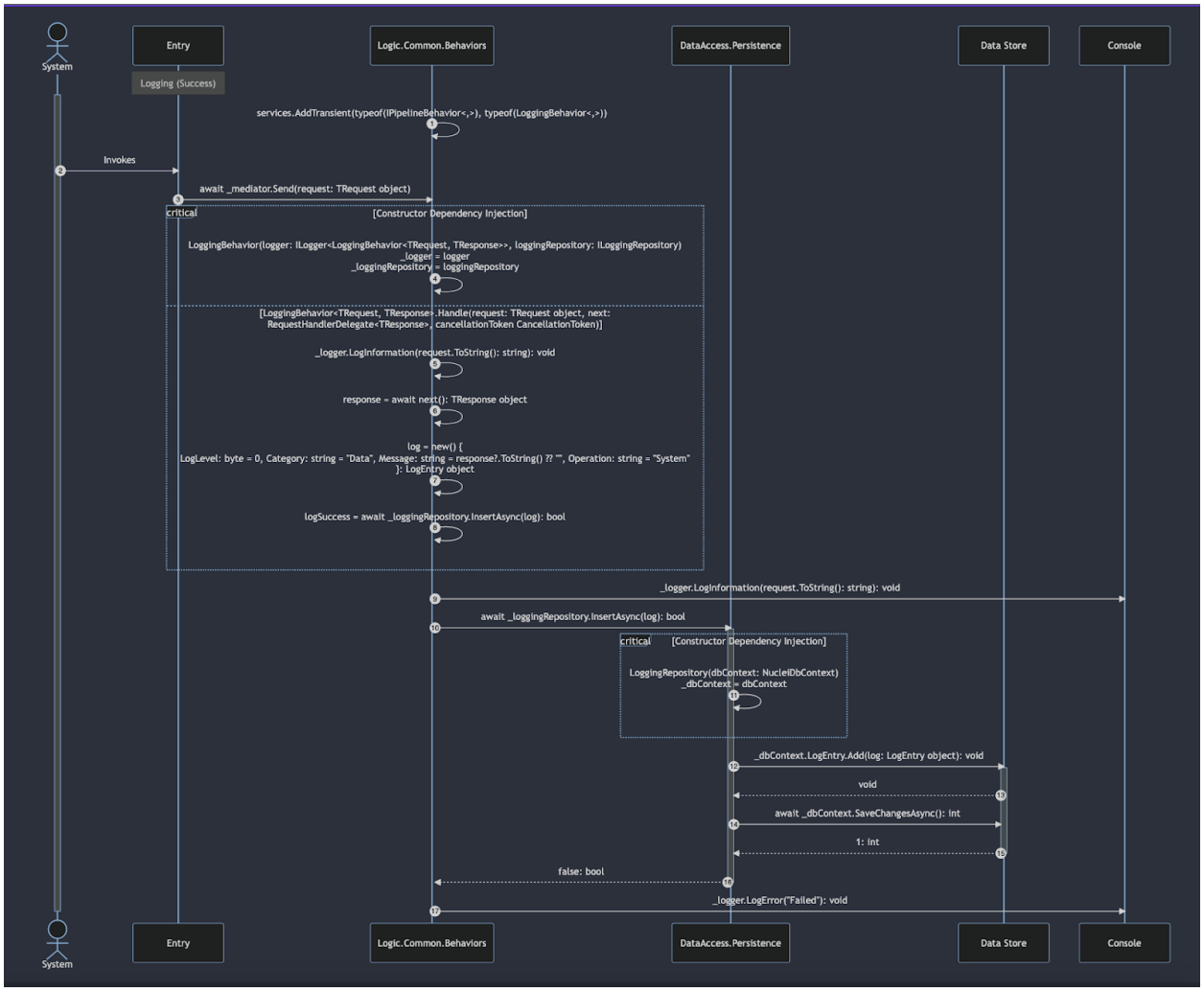
Failure Case 2

The logging process blocks a user from interacting with the system?



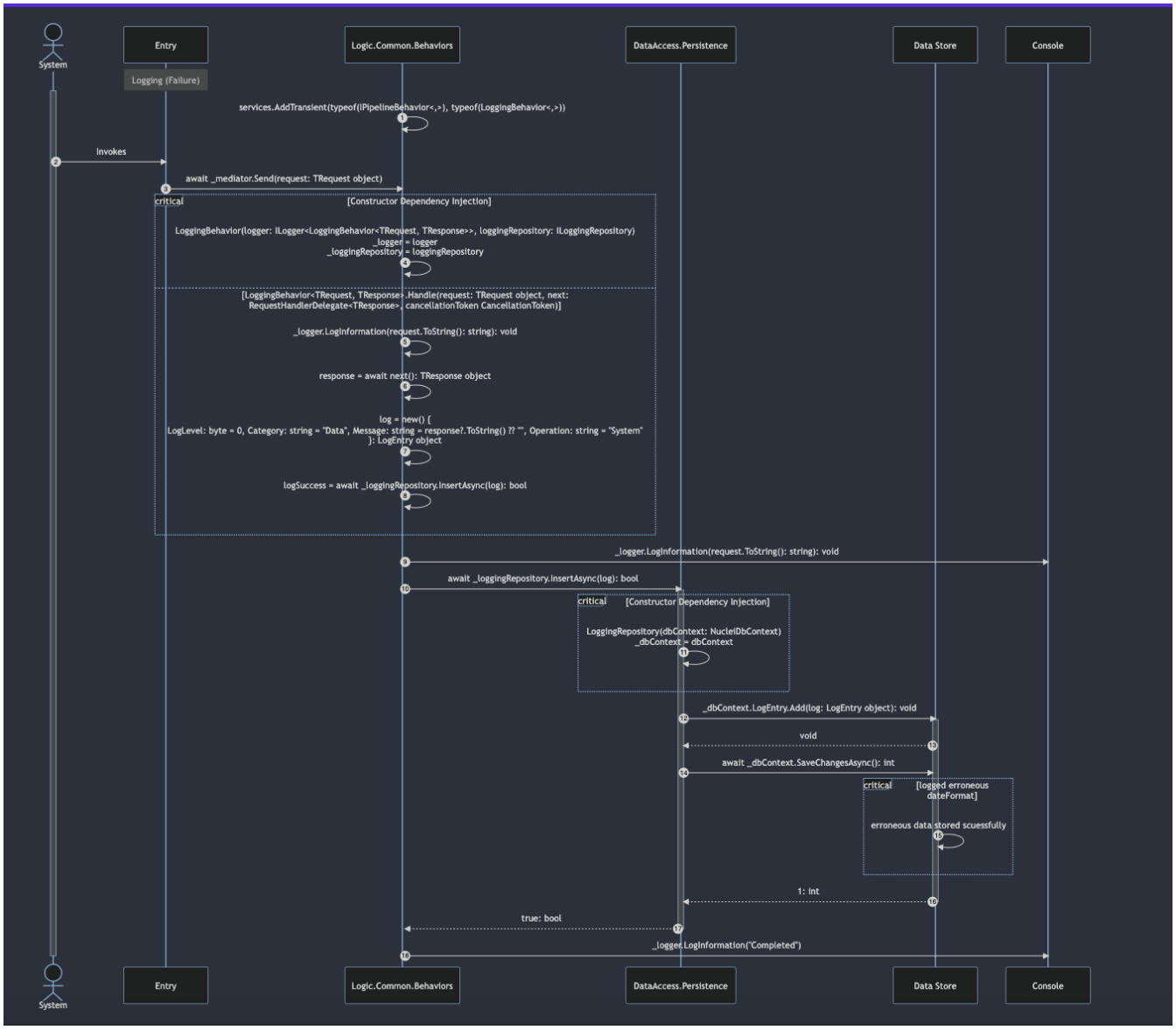
Failure Case 3

The logging pipeline did not time out, but did not save to persistent data storage.



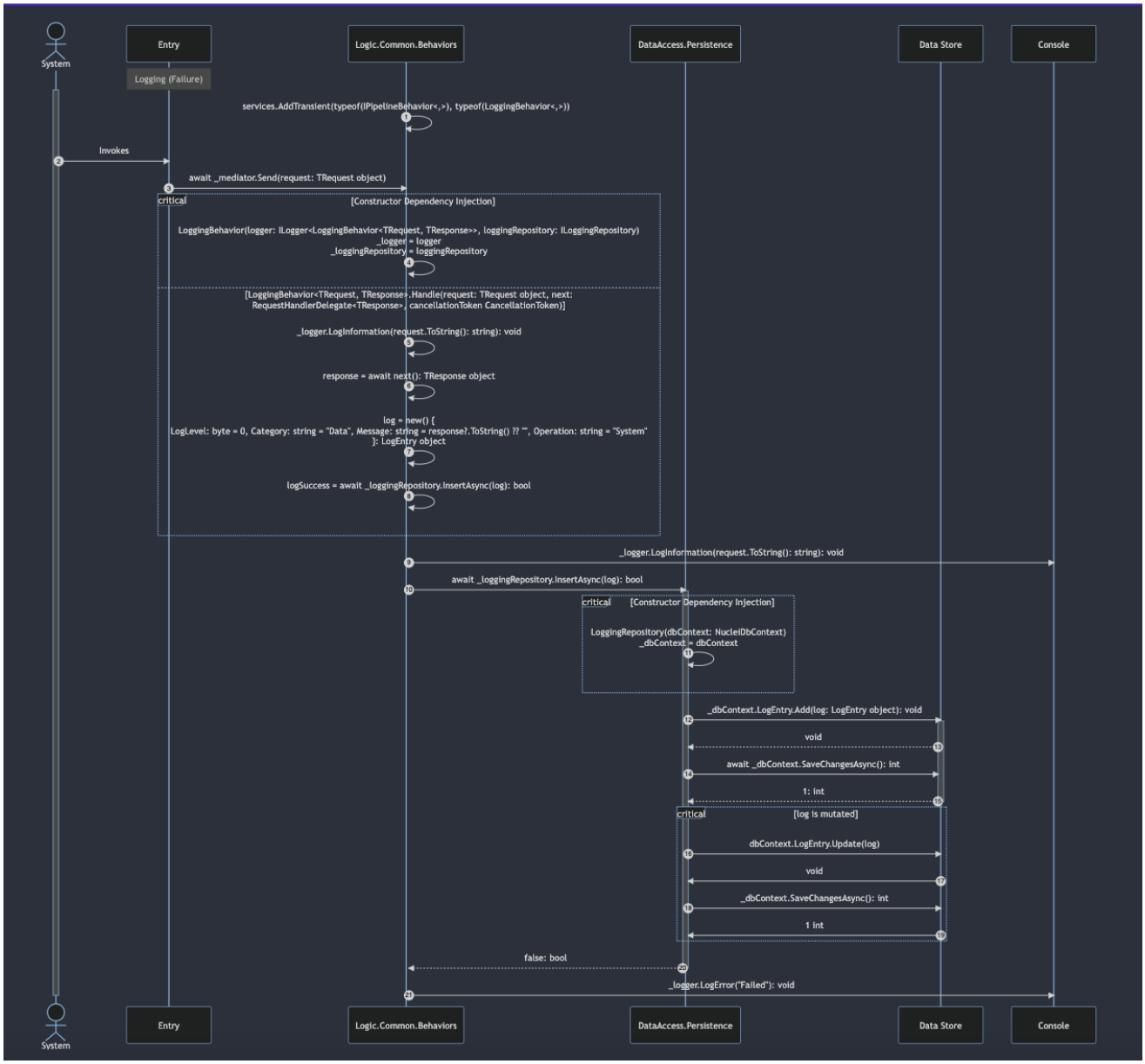
Failure Case 4

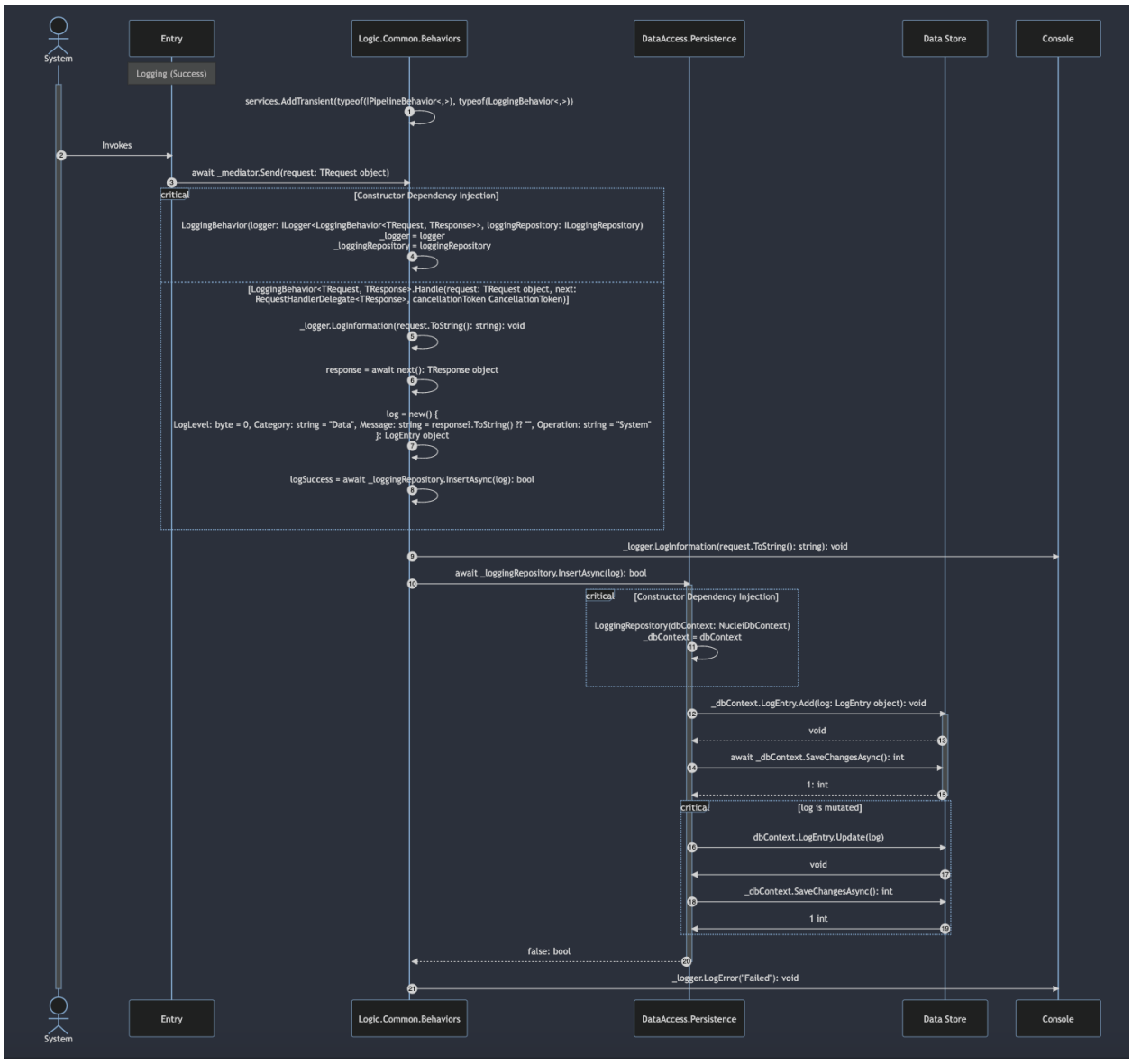
The logging pipeline did not time out, but there are erroneous discrepancies in the data which was stored.



Failure Case 5

Saved log entries in the data store are mutated in some way.


System
Entry
Logic.Common.Behaviors
DataAccess.Persistence
Data Store
Console
Logging (Failure)
services.AddTransient(typeof(IPipelineBehavior<,>), typeof(LoggingBehavior<,>))
Invokes
await _mediator.Send(request: TRequest object)
critical
[Constructor Dependency Injection]
LoggingBehavior(logger: ILogger<LoggingBehavior<TRequest, TResponse>>, loggingRepository: ILoggingRepository)
_logger = logger
_loggingRepository = loggingRepository
[LoggingBehavior<TRequest, TResponse>.Handle(request: TRequest object, next: RequestHandlerDelegate<TResponse>, cancellationToken CancellationToken)]
_logger.LogInformation(request.ToString(): string): void
response = await next(): TResponse object
log = new() {
LogLevel: byte = 0, Category: string = "Data", Message: string = response?.ToString() ?? "", Operation: string = "System"
}: LogEntry object
logSuccess = await _loggingRepository.InsertAsync(log): bool
_logger.LogInformation(request.ToString(): string): void
await _loggingRepository.InsertAsync(log): bool
critical
[Constructor Dependency Injection]
LoggingRepository(dbContext: NucleiDbContext)
_dbContext = dbContext
_dbContext.LogEntry.Add(log: LogEntry object): void
void
await _dbContext.SaveChangesAsync(): int
1: int
critical
[log is mutated]
dbContext.LogEntry.Update(log)
void
_dbContext.SaveChangesAsync(): int
1 int
false: bool
_logger.LogError("Failed"): void
System
Entry
Logic.Common.Behaviors
DataAccess.Persistence
Data Store
Console


System
Entry
Logic.Common.Behaviors
DataAccess.Persistence
Data Store
Console
Logging (Success)
services.AddTransient(typeof(IPipelineBehavior<,>), typeof(LoggingBehavior<,>))
Invokes
await _mediator.Send(request: TRequest object)
critical
[Constructor Dependency Injection]
LoggingBehavior(logger: ILogger<LoggingBehavior<TRequest, TResponse>>, loggingRepository: ILoggingRepository)
_logger = logger
_loggingRepository = loggingRepository
[LoggingBehavior<TRequest, TResponse>.Handle(request: TRequest object, next:
RequestHandlerDelegate<TResponse>, cancellationToken CancellationToken)]
_logger.LogInformation(request.ToString(): string): void
response = await next(): TResponse object
log = new() {
LogLevel: byte = 0, Category: string = "Data", Message: string = response?.ToString() ?? "", Operation: string = "System"
}: LogEntry object
logSuccess = await _loggingRepository.InsertAsync(log): bool
_logger.LogInformation(request.ToString(): string): void
await _loggingRepository.InsertAsync(log): bool
critical
[Constructor Dependency Injection]
LoggingRepository(dbContext: NucleiDbContext)
_dbContext = dbContext
_dbContext.LogEntry.Add(log: LogEntry object): void
void
await _dbContext.SaveChangesAsync(): int
1: int
critical
[log is mutated]
dbContext.LogEntry.Update(log)
void
_dbContext.SaveChangesAsync(): int
1 int
false: bool
_logger.LogError("Failed"): void

Registration Code

Success Case

```
sequenceDiagram
    actor User
    participant ui as UI
    participant e as Entry.Controllers.AuthenticationController
    participant bl as Logic.Authentication.Commands.Registration
    participant s as Service
    participant da as DataAccess.Persistence
    participant ds as Data Store
    autonumber
    note over ui: Registration (Success)
    activate User
    User->>ui: input: object
    loop
        ui->>ui: Validate(input: object): func
    end
    ui->>+e: HTTPPOST: Payload JSON
    critical Constructor Dependency Injection
        e->>e: RegisterController(mapper: IMapper, mediator:
ISender)<br/>_mapper = mapper<br/>_mediator = mediator
    option RegisterController.Register(request: RegisterRequest record)
        e->>e: command = _mapper.Map<RegisterCommand>(request):
RegisterCommand record
        e->>e: result = await _mediator.Send(command):
ErrorOr<AuthenticationResult> object
    end
    e->>+bl: await _mediator.Send(command): ErrorOr<AuthenticationResult>
object
    critical Constructor Dependency Injection
        bl->>bl: RegisterCommandHandler(userRepository: IUserRepository,
passwordHasher: IPasswordHasher, jwtTokenGenerator:
IJwtTokenGenerator)<br/>_userRepository =
userRepository<br/>_passwordHasher = passwordHasher<br/>_jwtTokenGenerator
= jwtTokenGenerator
    option RegisterCommandHandler.Handle(command: RegisterCommand record)
        bl->>bl: user = await
_userRepository.GetUserByEmailAsync(command.Email: string): User? object
    end
```

```
    bl->>+da: await _userRepository.GetUserByEmailAsync(command.Email:
string): User? object
    critical Constructor Dependency Injection
        da->>da: UserRepository(dbContext: NucleiDbContext)<br/>_dbContext
= dbContext
    end
    da->>+ds: await _dbContext.User.FirstOrDefaultAsync(e => e.Email:
string == email: string): User? object
    ds-->>da: null: User? object
    da-->>bl: null: User? object
    bl->>+s: hashedPassword =
_passwordHasher.HashPassword(command.Password: string): string
    s-->>-bl: hashedPassword: string
    bl->>da: await _userRepository.InsertAsync(user: User object): bool
    da->>da: _dbContext.User.Add(user: User object): void
    da->>ds: await _dbContext.SaveChangesAsync(): int
    ds-->>-da: 1: int
    da-->>-bl: true: bool
    bl->>+s: token = _jwtTokenGenerator.GenerateToken(user: User object):
string
    s->>-bl: token: string
    bl-->>-e: result: ErrorOr<AuthenticationResult> object
    e-->>-ui: Payload JSON
    ui-->>User: Display success and username to user
deactivate User
```

Failure Case 1

```
sequenceDiagram
    actor User
    participant ui as UI
    autonumber
    note over ui: Registration (Failure)
    activate User
    User->>ui: input: object
    loop
        ui->>ui: Validate(input: object): func
    end
    ui->>User: False
deactivate User
```

Failure Case 2

```
sequenceDiagram
    actor User
    participant ui as UI
    participant e as Entry.Controllers.AuthenticationController
    participant bl as Logic.Authentication.Commands.Registration
    participant s as Service
    participant da as DataAccess.Persistence
    participant ds as Data Store
    autonumber
    note over ui: Registration (Failure)
    activate User
    User->>ui: input: object
    loop
        ui->>ui: Validate(input: object): func
    end
    ui->>+e: HTTPPOST: Payload JSON
    critical Constructor Dependency Injection
        e->>e: RegisterController(mapper: IMapper, mediator:
ISender)<br/>_mapper = mapper<br/>_mediator = mediator
    option RegisterController.Register(request: RegisterRequest record)
        e->>e: command = _mapper.Map<RegisterCommand>(request):
RegisterCommand record
        e->>e: result = _mediator.Send(command):
ErrorOr<AuthenticationResult> object
    end
    e->>+bl: _mediator.Send(command): ErrorOr<AuthenticationResult> object
    critical Constructor Dependency Injection
        bl->>bl: RegisterCommandHandler(userRepository: IUserRepository,
passwordHasher: IPasswordHasher, jwtTokenGenerator:
IJwtTokenGenerator)<br/>_userRepository =
userRepository<br/>_passwordHasher = passwordHasher<br/>_jwtTokenGenerator
= jwtTokenGenerator
    option RegisterCommandHandler.Handle(command: RegisterCommand record)
        bl->>bl: user = await
_userRepository.GetUserByEmailAsync(command.Email: string): User? object
    end
```

```
    bl->>+da: await _userRepository.GetUserByEmailAsync(command.Email:
string): User? object
    critical Constructor Dependency Injection
        da->>da: UserRepository(dbContext: NucleiDbContext)<br/>_dbContext
= dbContext
    end
    da->>+ds: await _dbContext.User.FirstOrDefaultAsync(e => e.Email:
string == email: string): User? object
    ds-->>-da: User: User? object
    da-->>-bl: User: User? object
    bl-->>-e: result: DomainErrors.Authentication.InvalidCredentials
object
    e-->>-ui: Payload JSON
    ui-->>User: Display input error to user
deactivate User
```

Failure Case 3

```
sequenceDiagram
    actor User
    participant ui as UI
    autonumber
    note over ui: Registration (Failure)
    activate User
    User->>ui: input: object
    loop
        ui->>ui: Validate(input: object): func
    end
    ui->>User: False
deactivate User
```

Failure Case 4

```
sequenceDiagram
    actor User
    participant ui as UI
    participant e as Entry.Controllers.AuthenticationController
    participant bl as Logic.Authentication.Commands.Registration
```

```
    participant s as Service
    participant da as DataAccess.Persistence
    participant ds as Data Store
    autonumber
    note over ui: Registration (Failure)
    activate User
    User->>ui: input: object
    loop
        ui->>ui: Validate(input: object): func
    end
    ui->>+e: HTTPPOST: Payload JSON
    critical Constructor Dependency Injection
        e->>e: RegisterController(mapper: IMapper, mediator:
ISender)<br/>_mapper = mapper<br/>_mediator = mediator
    option RegisterController.Register(request: RegisterRequest record)
        e->>e: command = _mapper.Map<RegisterCommand>(request):
RegisterCommand record
        e->>e: result = _mediator.Send(command):
ErrorOr<AuthenticationResult> object
    end
    e->>+bl: _mediator.Send(command): ErrorOr<AuthenticationResult> object
    critical Constructor Dependency Injection
        bl->>bl: RegisterCommandHandler(userRepository: IUserRepository,
passwordHasher: IPasswordHasher, jwtTokenGenerator:
IJwtTokenGenerator)<br/>_userRepository =
userRepository<br/>_passwordHasher = passwordHasher<br/>_jwtTokenGenerator
= jwtTokenGenerator
    option RegisterCommandHandler.Handle(command: RegisterCommand record)
        bl->>bl: user = await
_userRepository.GetUserByEmailAsync(command.Email: string): User? object
    end
    bl->>+da: await _userRepository.GetUserByEmailAsync(command.Email:
string): User? object
    critical Constructor Dependency Injection
        da->>da: UserRepository(dbContext: NucleiDbContext)<br/>_dbContext
= dbContext
    end
    da->>+ds: await _dbContext.User.FirstOrDefaultAsync(e => e.Email:
string == email: string): User? object
    ds-->>da: null: User? object
```

```
    da-->>bl: null: User? object
    bl->>+s: hashedPassword =
_passwordHasher.HashPassword(command.Password: string): string
    s-->>-bl: hashedPassword: string
    bl->>da: await _userRepository.InsertAsync(user: User object): bool
    da->>da: _dbContext.User.Add(user: User object): void
    da->>ds: await _dbContext.SaveChangesAsync(): int
    ds-->>-da: 0: int
    da-->>-bl: false: bool
    bl-->>-e: result: DomainErrors.Authentication.Failure object
    e-->>-ui: Payload JSON
    ui-->>User: Display error page to user
deactivate User
```

Failure Case 5

```
sequenceDiagram
    actor User
    participant ui as UI
    participant e as Entry.Controllers.AuthenticationController
    participant bl as Logic.Authentication.Commands.Registration
    autonumber
    note over ui: Registration (Failure)
    activate User
    User->>ui: input: object
    loop
        ui->>ui: Validate(input: object): func
    end
    ui->>+e: HTTPPOST: Payload JSON
    critical Constructor Dependency Injection
        e->>e: RegisterController(mapper: IMapper, mediator:
ISender)<br/>_mapper = mapper<br/>_mediator = mediator
    option RegisterController.Register(request: RegisterRequest record)
        e->>e: command = _mapper.Map<RegisterCommand>(request):
RegisterCommand record
        e->>e: cancellationTokenSource = new():
CancellationTokenSource<br/>cancellationToken =
cancellationTokenSource.Token: CancellationToken
        e->>e: result = await _mediator.Send(command):
ErrorOr<AuthenticationResult> object
    end
```

```
    par Thread 1
        e->>+bl: await _mediator.Send(command, cancellationToken):
ErrorOr<AuthenticationResult> object
    and Thread 2
        e->>e: [cancellationToken timer has reached its limit before await
_mediator.Send() completed its task]
    end
    e-->>-ui: Payload JSON
    ui-->>User: Display timeout error to user
deactivate User
```

Failure Case 6

```
sequenceDiagram
   actor User
   participant ui as UI
   participant e as Entry.Controllers.AuthenticationController
   participant bl as Logic.Authentication.Commands.Registration
   participant s as Service
   participant da as DataAccess.Persistence
   participant ds as Data Store
   autonumber
   note over ui: Registration (Success)
   activate User
   User->>ui: input: object
   loop
       ui->>ui: Validate(input: object): func
   end
   ui->>+e: HTTPPOST: Payload JSON
   critical Constructor Dependency Injection
       e->>e: start stopwatch
       e->>e: RegisterController(mapper: IMapper, mediator:
ISender)<br/>_mapper = mapper<br/>_mediator = mediator
   option RegisterController.Register(request: RegisterRequest record)
       e->>e: command = _mapper.Map<RegisterCommand>(request):
RegisterCommand record
```

```
        e->>e: result = await _mediator.Send(command): 
ErrorOr<AuthenticationResult> object
    end
    e->>+bl: await _mediator.Send(command): ErrorOr<AuthenticationResult> 
object
    critical Constructor Dependency Injection
        bl->>bl: RegisterCommandHandler(userRepository: IUserRepository, 
passwordHasher: IPasswordHasher, jwtTokenGenerator: 
IJwtTokenGenerator)<br/>_userRepository = 
userRepository<br/>_passwordHasher = passwordHasher<br/>_jwtTokenGenerator 
= jwtTokenGenerator
    option RegisterCommandHandler.Handle(command: RegisterCommand record)
        bl->>bl: user = await 
_userRepository.GetUserByEmailAsync(command.Email: string): User? object
    end
    bl->>+da: await _userRepository.GetUserByEmailAsync(command.Email: 
string): User? object
    critical Constructor Dependency Injection
        da->>da: UserRepository(dbContext: NucleiDbContext)<br/>_dbContext 
= dbContext
    end
    da->>+ds: await _dbContext.User.FirstOrDefaultAsync(e => e.Email: 
string == email: string): User? object
    ds-->>da: null: User? object
    da-->>bl: null: User? object
    bl->>+s: hashedPassword = 
_passwordHasher.HashPassword(command.Password: string): string
    s-->>-bl: hashedPassword: string
    bl->>da: await _userRepository.InsertAsync(user: User object): bool
    da->>da: _dbContext.User.Add(user: User object): void
    da->>ds: await _dbContext.SaveChangesAsync(): int
    ds-->>-da: 1: int
    da-->>-bl: true: bool
    bl->>+s: token = _jwtTokenGenerator.GenerateToken(user: User object): 
string
    s->>-bl: token: string
```

```
    bl-->>-e: result: ErrorOr<AuthenticationResult> object
    critical stop stopwatch
        e->>e: if stopwatch is over 5s. Create a system failure log
    end
    e-->>-ui: Payload JSON
    ui-->>User: Display success and username to user
deactivate User
```

Logging Code

Success Case 1

```
sequenceDiagram
    actor System
    participant e as Entry
    participant bl as Logic.Common.Behaviors
    participant da as DataAccess.Persistence
    participant ds as Data Store
    participant c as Console
    autonumber
    note over e: Logging (Success)
    activate System
    bl->>bl: services.AddTransient(typeof(IPipelineBehavior<,>),
typeof(LoggingBehavior<,>))
    System->>e: Invokes
    e->>+bl: await _mediator.Send(request: TRequest object)
    critical Constructor Dependency Injection
        bl->>bl: LoggingBehavior(logger: ILogger<LoggingBehavior<TRequest,
TResponse>>, loggingRepository: ILoggingRepository)<br/>_logger =
logger<br/>_loggingRepository = loggingRepository
    option LoggingBehavior<TRequest, TResponse>.Handle(request: TRequest
object, next: RequestHandlerDelegate<TResponse>, cancellationToken
CancellationToken)
        bl->>bl: _logger.LogInformation(request.ToString(): string): void
        bl->>bl: response = await next(): TResponse object
        bl->>bl: log = new() {<br/>LogLevel: byte = 0, Category: string =
"Data", Message: string = response?.ToString() ?? "", Operation: string =
"System"<br/>}: LogEntry object
        bl->>bl: logSuccess = await _loggingRepository.InsertAsync(log):
bool
    end
    bl->>c: _logger.LogInformation(request.ToString(): string): void
    bl->>+da: await _loggingRepository.InsertAsync(log): bool
    critical Constructor Dependency Injection
        da->>da: LoggingRepository(dbContext:
NucleiDbContext)<br/>_dbContext = dbContext
```

```
    end
    da->>+ds: _dbContext.LogEntry.Add(log: LogEntry object): void
    ds-->>da: void
    da->>ds: await _dbContext.SaveChangesAsync(): int
    ds-->>-da: 1: int
    da-->>-bl: true: bool
    bl->>c: _logger.LogInformation("Completed");
deactivate System
```

Success Case 2

```
sequenceDiagram
    actor System
    participant e as Entry
    participant bl as Logic.Common.Behaviors
    participant da as DataAccess.Persistence
    participant ds as Data Store
    participant c as Console
    autonumber
    note over e: Logging (Success)
    activate System
    bl->>bl: services.AddTransient(typeof(IPipelineBehavior<,>),
typeof(LoggingBehavior<,>))
    System->>e: Invokes
    e->>+bl: await _mediator.Send(request: TRequest object)
    critical Constructor Dependency Injection
        bl->>bl: LoggingBehavior(logger: ILogger<LoggingBehavior<TRequest,
TResponse>>, loggingRepository: ILoggingRepository)<br/>_logger =
logger<br/>_loggingRepository = loggingRepository
    option LoggingBehavior<TRequest, TResponse>.Handle(request: TRequest
object, next: RequestHandlerDelegate<TResponse>, cancellationToken
CancellationToken)
        bl->>bl: _logger.LogInformation(request.ToString(): string): void
        bl->>bl: response = await next(): TResponse object
        bl->>bl: log = new() {<br/>LogLevel: byte = 4, Category: string =
"Data", Message: string = response?.ToString() ?? "", Operation: string =
"System"<br/>}: LogEntry object
        bl->>bl: logSuccess = await _loggingRepository.InsertAsync(log):
bool
```

```
    end
    bl->>c: _logger.LogInformation(request.ToString(): string): void
    bl->>+da: await _loggingRepository.InsertAsync(log): bool
    critical Constructor Dependency Injection
        da->>da: LoggingRepository(dbContext:
NucleiDbContext)<br/>_dbContext = dbContext
    end
    da->>+ds: _dbContext.LogEntry.Add(log: LogEntry object): void
    ds-->>da: void
    da->>ds: await _dbContext.SaveChangesAsync(): int
    ds-->>-da: 1: int
    da-->>-bl: true: bool
    bl->>c: _logger.LogInformation("Completed");
deactivate System
```

Success Case 3

```
sequenceDiagram
    actor System
    participant e as Entry
    participant bl as Logic.Common.Behaviors
    participant da as DataAccess.Persistence
    participant ds as Data Store
    participant c as Console
    autonumber
    note over e: Logging (Success)
    activate System
    bl->>bl: services.AddTransient(typeof(IPipelineBehavior<,>),
typeof(LoggingBehavior<,>))
    System->>e: Invokes
    e->>+bl: await _mediator.Send(request: TRequest object)
    critical Constructor Dependency Injection
        bl->>bl: LoggingBehavior(logger: ILogger<LoggingBehavior<TRequest,
TResponse>>, loggingRepository: ILoggingRepository)<br/>_logger =
logger<br/>_loggingRepository = loggingRepository
    option LoggingBehavior<TRequest, TResponse>.Handle(request: TRequest
object, next: RequestHandlerDelegate<TResponse>, cancellationToken
CancellationToken)
        bl->>bl: _logger.LogInformation(request.ToString(): string): void
```

```
        bl->>bl: response = await next(): TResponse object
        bl->>bl: log = new() {<br/>LogLevel: byte = 2, Category: string =
"Data", Message: string = response?.ToString() ?? "", Operation: string =
request.GetType().Name<br/>}: LogEntry object
        bl->>bl: logSuccess = await _loggingRepository.InsertAsync(log):
bool
    end
    bl->>c: _logger.LogInformation(request.ToString(): string): void
    bl->>+da: await _loggingRepository.InsertAsync(log): bool
    critical Constructor Dependency Injection
        da->>da: LoggingRepository(dbContext:
NucleiDbContext)<br/>_dbContext = dbContext
    end
    da->>+ds: _dbContext.LogEntry.Add(log: LogEntry object): void
    ds-->>da: void
    da->>ds: await _dbContext.SaveChangesAsync(): int
    ds-->>-da: 1: int
    da-->>-bl: true: bool
    bl->>c: _logger.LogInformation("Completed");
deactivate System
```

Success Case 4

```
sequenceDiagram
    actor System
    participant e as Entry
    participant bl as Logic.Common.Behaviors
    participant da as DataAccess.Persistence
    participant ds as Data Store
    participant c as Console
    autonumber
    note over e: Logging (Success)
    activate System
    bl->>bl: services.AddTransient(typeof(IPipelineBehavior<,>),
typeof(LoggingBehavior<,>))
    System->>e: Invokes
    e->>+bl: await _mediator.Send(request: TRequest object)
    critical Constructor Dependency Injection
```

```
        bl->>bl: LoggingBehavior(logger: ILogger<LoggingBehavior<TRequest,
TResponse>>, loggingRepository: ILoggingRepository)<br/>_logger =
logger<br/>_loggingRepository = loggingRepository
    option LoggingBehavior<TRequest, TResponse>.Handle(request: TRequest
object, next: RequestHandlerDelegate<TResponse>, cancellationToken
CancellationToken)
        bl->>bl: _logger.LogInformation(request.ToString(): string): void
        bl->>bl: response = await next(): TResponse object
        bl->>bl: log = new() {<br/>LogLevel: byte = 4, Category: string =
"Data", Message: string = response?.ToString() ?? "", Operation: string =
request.GetType().Name<br/>}: LogEntry object
        bl->>bl: logSuccess = await _loggingRepository.InsertAsync(log):
bool
    end
    bl->>c: _logger.LogInformation(request.ToString(): string): void
    bl->>+da: await _loggingRepository.InsertAsync(log): bool
    critical Constructor Dependency Injection
        da->>da: LoggingRepository(dbContext:
NucleiDbContext)<br/>_dbContext = dbContext
    end
    da->>+ds: _dbContext.LogEntry.Add(log: LogEntry object): void
    ds-->>da: void
    da->>ds: await _dbContext.SaveChangesAsync(): int
    ds-->>-da: 1: int
    da-->>-bl: true: bool
    bl->>c: _logger.LogInformation("Completed");
deactivate System
```

Failure Case 1

```
sequenceDiagram
    actor System
    participant e as Entry
    participant bl as Logic.Common.Behaviors
    participant da as DataAccess.Persistence
    participant ds as Data Store
    participant c as Console
    autonumber
    note over e: Logging (Failure)
    activate System
```

```
    bl->>bl: services.AddTransient(typeof(IPipelineBehavior<,>),
typeof(LoggingBehavior<,>))
    System->>e: Invokes
    par Thread 1
        e->>+bl: await _mediator.Send(request: TRequest object)
        critical Constructor Dependency Injection
            bl->>bl: LoggingBehavior(logger:
ILogger<LoggingBehavior<TRequest, TResponse>>, loggingRepository:
ILoggingRepository)<br/>_logger = logger<br/>_loggingRepository =
loggingRepository
        option LoggingBehavior<TRequest, TResponse>.Handle(request:
TRequest object, next: RequestHandlerDelegate<TResponse>,
cancellationToken CancellationToken)
            bl->>bl: _logger.LogInformation(request.ToString(): string):
void
            bl->>bl: response = await next(): TResponse object
            bl->>bl: log = new() {<br/>LogLevel: byte = 0, Category:
string = "Data", Message: string = response?.ToString() ?? "", Operation:
string = "System"<br/>}: LogEntry object
            bl->>bl: logSuccess = await
_loggingRepository.InsertAsync(log): bool
        end
    and Thread 2
        bl->>bl: cancellationToken timer has reached its limit before
await _mediator.Send() has completed its task
    end
deactivate System
```

Failure Case 2

Failure Case 3

```
sequenceDiagram
   actor System
   participant e as Entry
   participant bl as Logic.Common.Behaviors
   participant da as DataAccess.Persistence
```

```
   participant ds as Data Store
   participant c as Console
   autonumber
   note over e: Logging (Success)
   activate System
   bl->>bl: services.AddTransient(typeof(IPipelineBehavior<,>),
typeof(LoggingBehavior<,>))
   System->>e: Invokes
   e->>+bl: await _mediator.Send(request: TRequest object)
   critical Constructor Dependency Injection
       bl->>bl: LoggingBehavior(logger: ILogger<LoggingBehavior<TRequest,
TResponse>>, loggingRepository: ILoggingRepository)<br/>_logger =
logger<br/>_loggingRepository = loggingRepository
   option LoggingBehavior<TRequest, TResponse>.Handle(request: TRequest
object, next: RequestHandlerDelegate<TResponse>, cancellationToken
CancellationToken)
       bl->>bl: _logger.LogInformation(request.ToString(): string): void
       bl->>bl: response = await next(): TResponse object
       bl->>bl: log = new() {<br/>LogLevel: byte = 0, Category: string =
"Data", Message: string = response?.ToString() ?? "", Operation: string =
"System"<br/>}: LogEntry object
       bl->>bl: logSuccess = await _loggingRepository.InsertAsync(log):
bool
   end
   bl->>c: _logger.LogInformation(request.ToString(): string): void
   bl->>+da: await _loggingRepository.InsertAsync(log): bool
   critical Constructor Dependency Injection
       da->>da: LoggingRepository(dbContext:
NucleiDbContext)<br/>_dbContext = dbContext
   end
```

```
   da->>+ds: _dbContext.LogEntry.Add(log: LogEntry object): void
   ds-->>da: void
   da->>ds: await _dbContext.SaveChangesAsync(): int
   ds-->>-da: 1: int
   da-->>-bl: false: bool
   bl->>c: _logger.LogError("Failed"): void
deactivate System
```

Failure Case 4

```
sequenceDiagram
   actor System
   participant e as Entry
   participant bl as Logic.Common.Behaviors
   participant da as DataAccess.Persistence
   participant ds as Data Store
   participant c as Console
   autonumber
   note over e: Logging (Success)
   activate System
   bl->>bl: services.AddTransient(typeof(IPipelineBehavior<,>),
typeof(LoggingBehavior<,>))
   System->>e: Invokes
   e->>+bl: await _mediator.Send(request: TRequest object)
   critical Constructor Dependency Injection
       bl->>bl: LoggingBehavior(logger: ILogger<LoggingBehavior<TRequest,
TResponse>>, loggingRepository: ILoggingRepository)<br/>_logger =
logger<br/>_loggingRepository = loggingRepository
   option LoggingBehavior<TRequest, TResponse>.Handle(request: TRequest
object, next: RequestHandlerDelegate<TResponse>, cancellationToken
CancellationToken)
       bl->>bl: _logger.LogInformation(request.ToString(): string): void
       bl->>bl: response = await next(): TResponse object
```

```
        bl->>bl: log = new() {<br/>LogLevel: byte = 0, Category: string =
"Data", Message: string = response?.ToString() ?? "", Operation: string =
"System"<br/>}: LogEntry object
        bl->>bl: logSuccess = await _loggingRepository.InsertAsync(log):
bool
    end
    bl->>c: _logger.LogInformation(request.ToString(): string): void
    bl->>+da: await _loggingRepository.InsertAsync(log): bool
    critical Constructor Dependency Injection
        da->>da: LoggingRepository(dbContext:
NucleiDbContext)<br/>_dbContext = dbContext
    end
    da->>+ds: _dbContext.LogEntry.Add(log: LogEntry object): void
    ds-->>da: void
    da->>ds: await _dbContext.SaveChangesAsync(): int
    critical logged erroneous dateFormat
        ds->>ds: erroneous data stored scuessfully
    end
    ds-->>-da: 1: int
    da-->>-bl: true: bool
    bl->>c: _logger.LogInformation("Completed");

deactivate System
```

Failure Case 5

```
sequenceDiagram
    actor System
    participant e as Entry
    participant bl as Logic.Common.Behaviors
    participant da as DataAccess.Persistence
    participant ds as Data Store
    participant c as Console
    autonumber
    note over e: Logging (Success)
```

```
    activate System
    bl->>bl: services.AddTransient(typeof(IPipelineBehavior<,>),
typeof(LoggingBehavior<,>))
    System->>e: Invokes
    e->>+bl: await _mediator.Send(request: TRequest object)
    critical Constructor Dependency Injection
        bl->>bl: LoggingBehavior(logger: ILogger<LoggingBehavior<TRequest,
TResponse>>, loggingRepository: ILoggingRepository)<br/>_logger =
logger<br/>_loggingRepository = loggingRepository
    option LoggingBehavior<TRequest, TResponse>.Handle(request: TRequest
object, next: RequestHandlerDelegate<TResponse>, cancellationToken
CancellationToken)
        bl->>bl: _logger.LogInformation(request.ToString(): string): void
        bl->>bl: response = await next(): TResponse object
        bl->>bl: log = new() {<br/>LogLevel: byte = 0, Category: string =
"Data", Message: string = response?.ToString() ?? "", Operation: string =
"System"<br/>}: LogEntry object
        bl->>bl: logSuccess = await _loggingRepository.InsertAsync(log):
bool
    end
    bl->>c: _logger.LogInformation(request.ToString(): string): void
    bl->>+da: await _loggingRepository.InsertAsync(log): bool
    critical Constructor Dependency Injection
        da->>da: LoggingRepository(dbContext:
NucleiDbContext)<br/>_dbContext = dbContext
    end
    da->>+ds: _dbContext.LogEntry.Add(log: LogEntry object): void
    ds-->>da: void
    da->>ds: await _dbContext.SaveChangesAsync(): int
    ds-->>-da: 1: int
    critical log is mutated
        da->>ds:  dbContext.LogEntry.Update(log);
        ds->>da: void
        da->>ds: _dbContext.SaveChangesAsync(): int
        ds->>da: 1 int
```

```
    end
    da-->>-bl: false: bool
    bl->>c: _logger.LogError("Failed"): void
deactivate System
```

References

1. https://mermaid-js.github.io
2. https://github.com/amantinband
3. Email, “Core Components Requirements - Professor Vatanak Vong”, 10/17/2022

Version Changelog

Version	Submission Date	Changelog
1	10/28/22	Initial Draft Version. Added Success Cases of Registration and Logging.
2	11/02/22	Added Failure Cases of Registration. Changed Success Cases of Registration and Logging.
3	11/09/22	Added Failure Cases of Logging. Finalization.