# **DWS Dev Challenge Solution Design Approach**

# Summary of Existing Functionality

Provided a simple REST service with some very basic functionality to add and read an account.

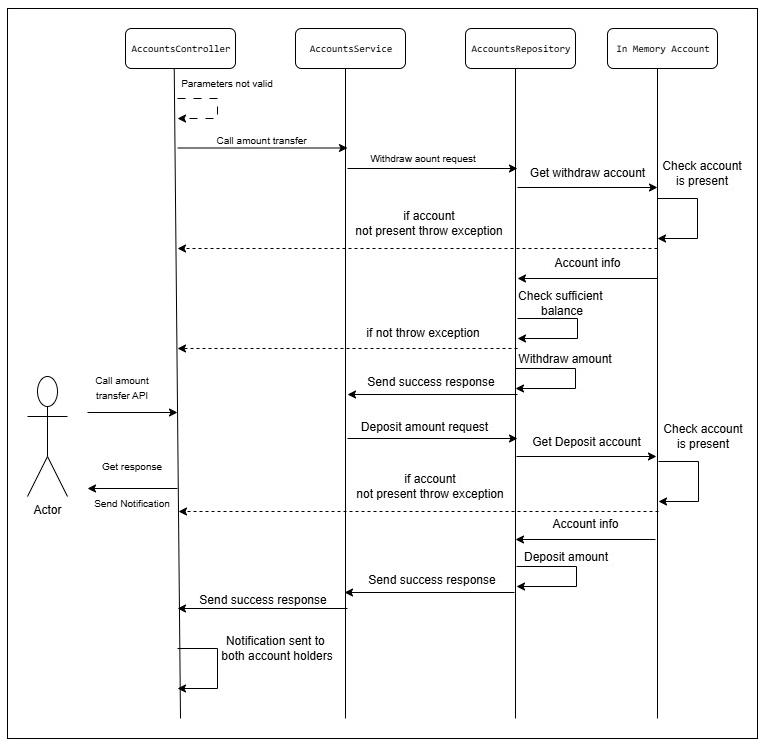
It is a standard gradle project, running on Spring Boot.

# Requirement Details

* Add functionality for a transfer of money between accounts. Transfers should be specified by providing:
  + - * accountFrom id
      * accountTo id
      * amount to transfer between accounts
* The amount to transfer should always be a positive number.
* It should not be possible for an account to end up with negative balance.
* Whenever a transfer is made, a notification should be sent to both account holders, with a message containing id of the other account and amount transferred. For this purpose please use the NotificationService interface
* This feature should be implemented in a thread-safe manner. Solution should never deadlock, should never result in corrupted account state, and should work efficiently for multiple transfers happening at the same time.

# [Solution Design](https://softwaredominos.com/home/software-design-development-articles/high-level-solution-design-documents-what-is-it-and-when-do-you-need-one/)

* Created new REST API for a transfer of money between accounts.
* This API has following inputs
  + - * accountFrom id
      * accountTo id
      * amount to transfer between accounts
* Validate account from id and account to id should be in memory data.
* Validate amount to transfer should always be a positive number.
* Check from account balance should be more or equals to amount. If not then throe the exception
* Withdraw amount from account and deposit in to account.
* If any error occurred while depositing amount then revert the withdraw transection from account.
* Used lock per account to avoid deadlock. Used acquire locks in the same order always.
* After successful transfer is made, a notification sent to both account holders, with a message containing id of the other account and amount transferred.



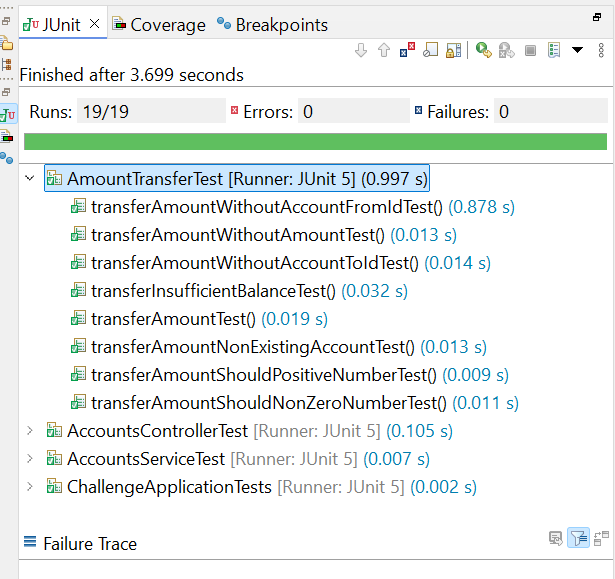
# **Improvements/add, given more time**

* Implement user authentication and authorization
* Create bank account with more information (e.g., name address, phone number).
* Add rate limiter for request in specific time
* Database can be used for account and balance management
* Use transaction management
* Use JPA for database operations
* Can be use Kafka messaging broker for amount transfer and more fault tolerance

# Testing

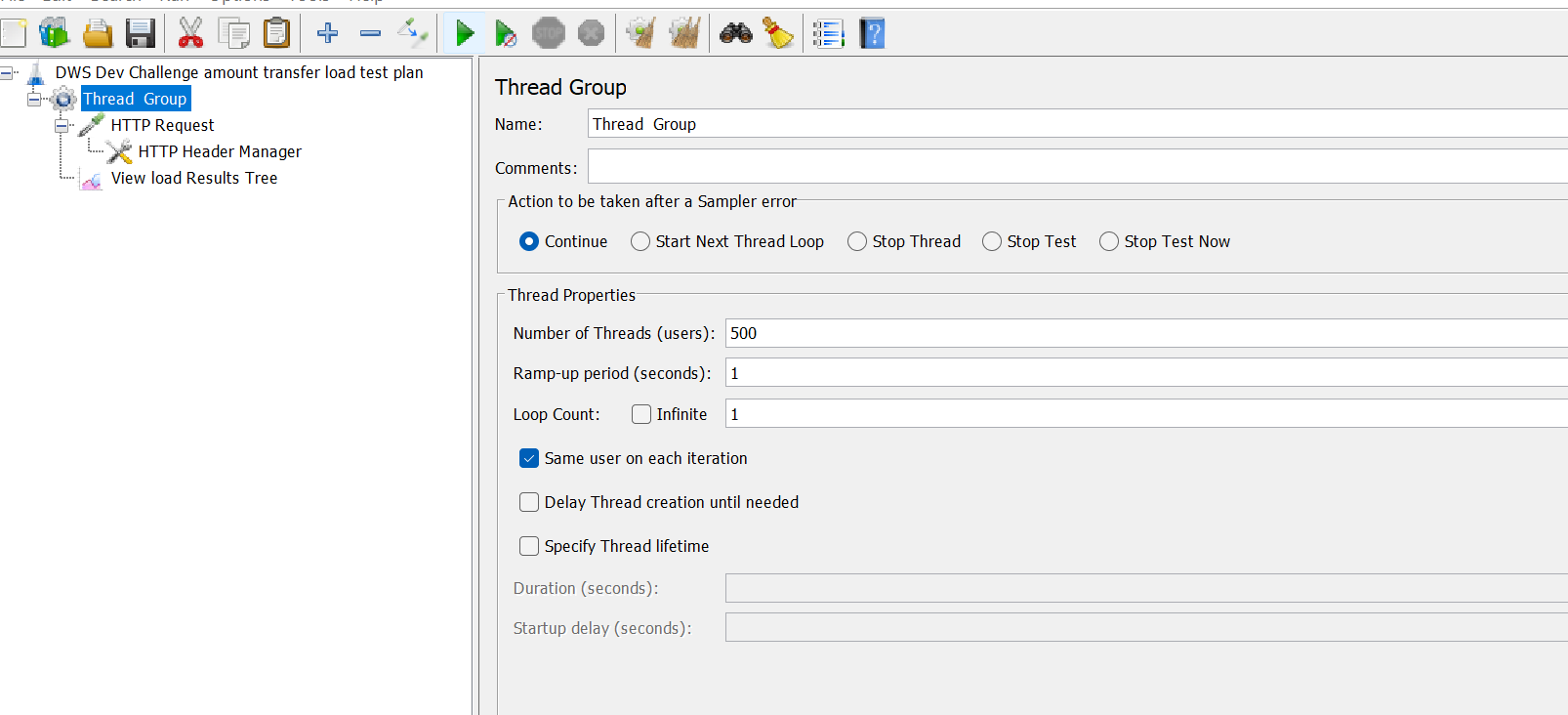
Adde following JUnit test cases for amount transfer

* Amount should be positive number
* Amount should be non-zero number
* Test without account from id
* Test without account to id
* Test without transfer amount
* After exception amount should be reverted back

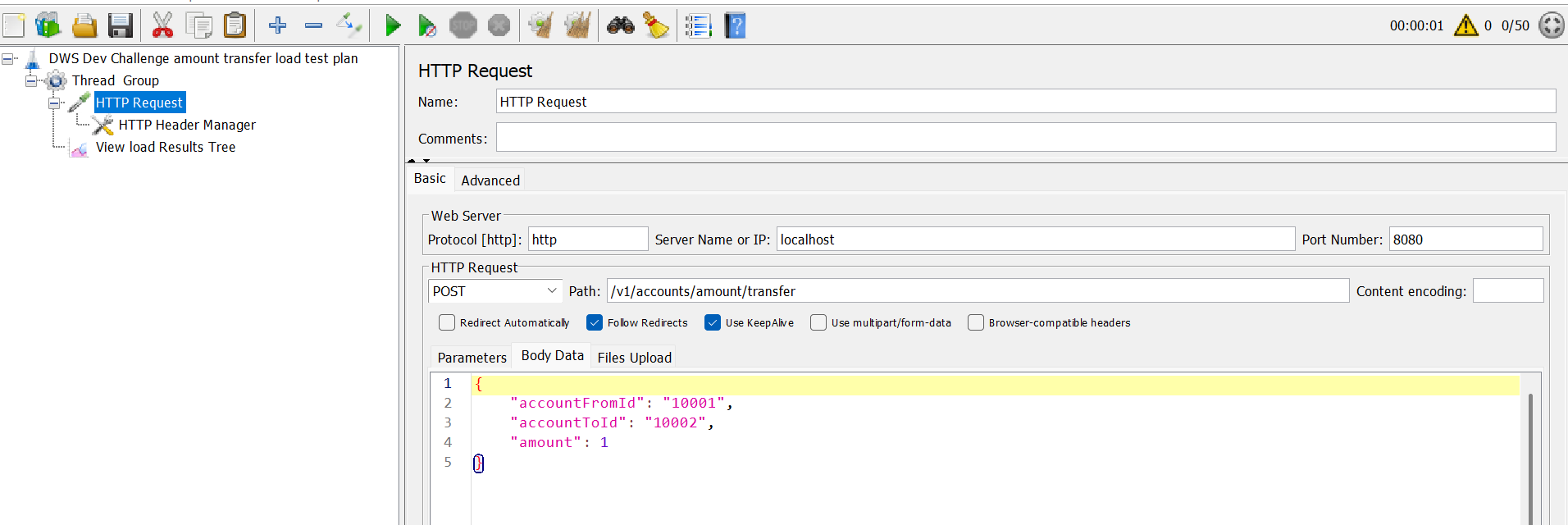


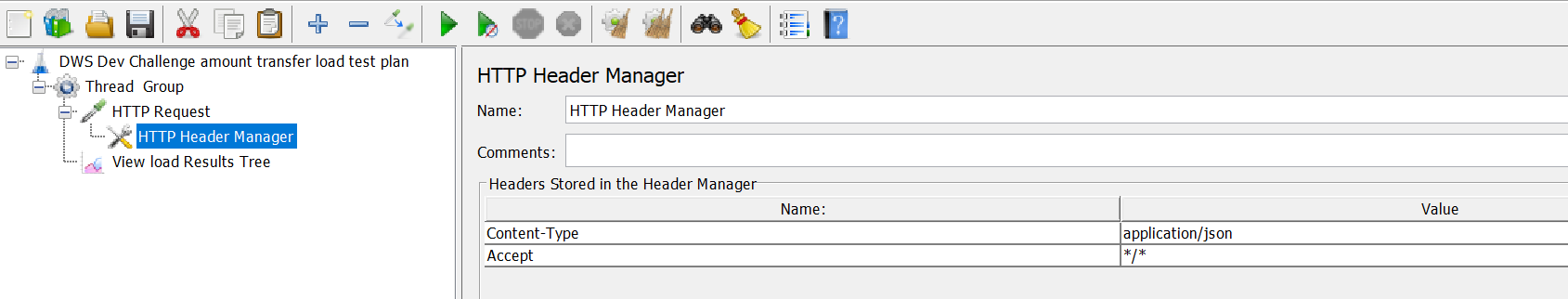
# Tress Testing for deadlock

* Tress testing did using JMeter
* Created Test plan with Thread Group of 500 threads can execute in 1 minute

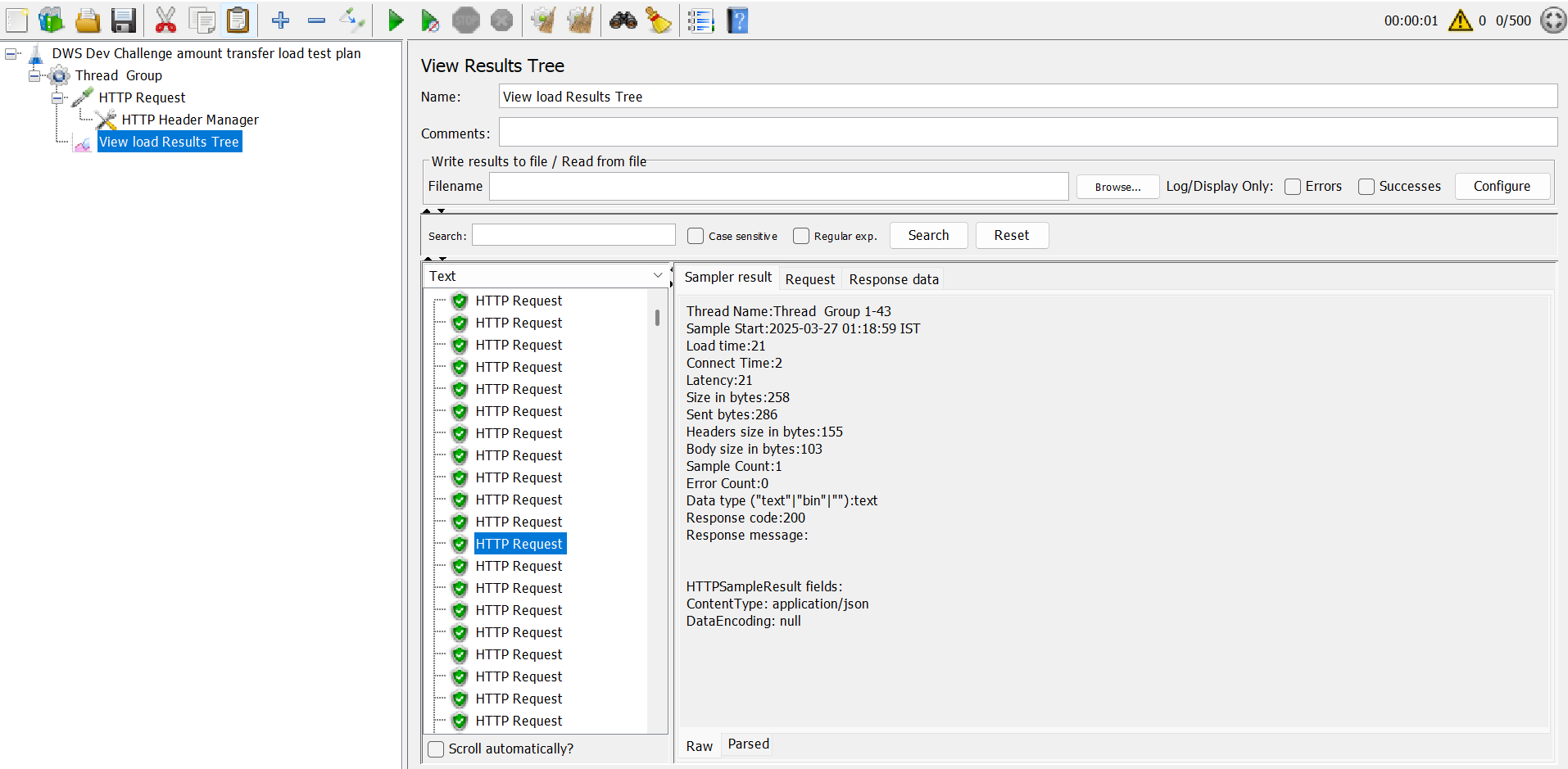


Created HTTP request





After run the stress testing showing following success result without deadlock



Console logs in application

