java.io.IOException: An established connection was aborted by the software in your host machine

Introduction

“An established connection was aborted by the software in your host machine” is a very uncommon error when dealing with a network aspect of a project. It is an incredibly vague error which has a multitude of meanings and reasons and can range from a harmless warning to a layer-breaking error. The task of this paper is to document learned research about this error, what issues it can cause to the integrity of the persistence layer of a project, and show what proof is possible to provide to satisfy the learning outcome of a second persistence layer.

What does this error mean?

To assess the situation this error points to, we have to recognise it’s multiple meanings.

This IOException in my instance is caused when calling a function in the API. In my specific case, I possess 2 data sources or persistence layers in my project, as it is a necessity for satisfying my semester - one data source for content of the web application as well as one for managing users and user data. Note that during the development process I have switched back and forth with the secondary data source between a local database, like the primary one, and an online database hosted by my university, to test more efficiently.

While the primary (content) database works without issue and API calls are easily completed, any API call to the secondary database results in this error alongside a 404 Not Found error. What does this mean?

This means that a piece of software is blocking my access to the second persistence layer, either a result of an error in configuration, or an issue within the internal software of my personal device.

What is the severity of this error?

As mentioned in the introduction, the severity of the error varies depending on the specific construction of the software and the use case.

In the best case scenario, it is as harmful as a warning could be. It displays that there is an issue with the network but the called function still works and can be ignored. In the worst case scenario, which is this one, it completely blocks whatever network function should happen without any

What have I done to solve it?

Like every error a developer faces in the project, you start solving it with research. My findings include, but are not limited to, multiple articles and stackoverflow posts, which is normal, but along with these there was an abnormal amount of results leading to Minecraft-related forum posts. Something interesting to note is Minecraft is a video game reliant on the system running it to have Java installed, which is what my backend is built on.

A common note in both of these sources of information point to not an issue with Java, but something to do with the network functions, as both my instance of the error and the commonly met instances in both a code environment and the game environment is a process connected to a network task and API calls. In Minecraft forums, the main complaints concerning this error happen when one attempts to connect to a multiplayer server. Meanwhile in code forums, the general situation is that this error occurs during a server call or an API call.

What were the general solutions? Most, if not all solutions were external work that attempted to either clean up anything that can tamper with the networking.

- If the server is hosted from the same device, the service can be reset.
- Flush the device's DNS (Domain Name System) cache. This is generally done in case a user meets a Not Found error when they should not.
- Disable firewall/antivirus in case it is what is the source of the blocking

These are the most commonly met solutions and the only ones with replies from users that have proved successful. Unfortunately, this could not be achieved in this instance.

How can I prove that there actually is a connection?

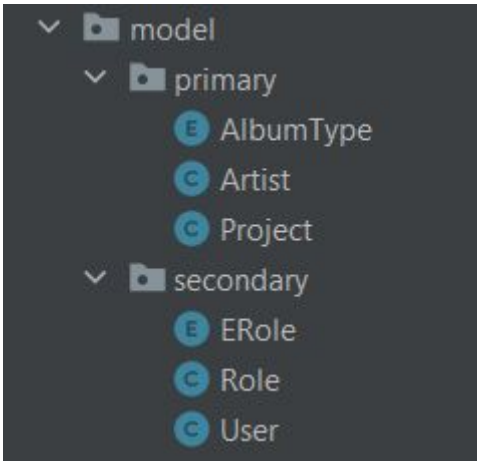
While researching this error I've also learned much about managing 2 database systems. While the delivered product will have a basic double-datasource configuration, some time during the development process it was much more complex, involving verification of the database tables before making the connection to the JPA Repositories. Unfortunately, due to errors with the database configuration, this was not possible to implement completely in the allotted time.

The way the verification worked is the affected databases would be scanned with every table, then dropped until the data sources are empty, then filled again with the scanned data. But due to an inconsistency with the repositories and the database, the configuration would break in the middle and the databases will stay empty.

For now, this is the closest proof I can provide that the connection is properly set, because the affected datasource is scanned as well.

What have I learned from studying how to use 2 persistence layers?

I have learned much on how to organise multiple data sources while attempting to fix this issue. Initially, I have only fed two data sources in the application.resources folder. I have later discovered that that is actually not enough. To properly set up two data sources, their respective resources require to be separated. So for instance: models that belong to a datasource need to be secluded in one package, while the models of the second need to be in another.



Along with that, there needs to be at least one configuration file to create the beans to both databases, so that the connection is made and the repositories and thus services and controllers can make use of the data. Since it is a Configuration file, it is run every time the application is built, so there is no need to apply it anywhere else in the project.

```
@Configuration
@EnableTransactionManagement
@EnableJpaRepositories(
        basePackages = {"com.audioappraiser.audioapp.model.primary"},
        entityManagerFactoryRef = "entityManagerFactory",
        transactionManagerRef = "transactionManager")
public class PrimaryDataSourceConfiguration {
    @Primary
    @Bean(name = "primaryDataSource")
    @ConfigurationProperties("datasource-primary")
    public DataSource primaryDataSource() { return DataSourceBuilder.create().build(); }

    @Primary
    @Bean
    NamedParameterJdbcOperations jdbcOperations(
            @Qualifier("primaryDataSource") DataSource sqlServerDs) {
        return new NamedParameterJdbcTemplate(sqlServerDs);
    }
}
```

Conclusion

This JavaIOException is a vague and confusing error to receive and leads to many, even more vague and inconsistent solutions. I hope this document has helped compact more of the useful information and possible fixes and satisfies the learning outcome of the second persistence layer despite this issue.