**Payment Processor Communication Implementation**

Software Design & Programming

Distributed Computing

Joseph DiBiasi

University of Denver College of Professional Studies

10/4/2025

Faculty: Michael Schwartz

Director: Cathie Wilson

Dean: Michael J. McGuire

**Introduction**

Our payment processor is still in development, but the basic design has been fleshed out to give an idea of our workflow. The frontend user interface is provided using Angular 19. The design is basic and intended to be a simplified version of the checkout process. Once a user clicks checkout, a payment authorization request will be sent to the Java 21 backend. This request is sent with headers specifying the content type as JSON and containing the authorization token. This initial mockup of the payment processor then validates the payment authorization details. If validation is successful a 200 response is returned; if errors in validation occur they are stored and returned along with a 422 response. Connecting to the credit card network via a third-party payment gateway along with storing payment authorization information for nightly batch processing is currently only mocked up.

**Design and Implementation**

The initial design of this payment processor is focused on communication from the client side to the payment processor server and receiving a response. The current implementation demonstrates this through the use of a post call. All fields displayed on the user interface are able to be modified and, through the two way data binding provided by the ngModel, what the user sees on the interface is the same information that will be gathered up to be sent in the body of the post request when the user is satisfied with their checkout information and submits this information to the payment processor. Currently, there is no validation on the frontend. This was intentional to better demonstrate the communication process and return not only successful responses but also error responses if fields are left blank. Prior to release, the frontend will be converted to handle all shallow field validation, such as ensuring valid email or credit card number format. Additionally, The backend will only handle deep field validation; instances such as a payment authorization failing due to incorrect billing details.

Once gathered, this information is both sent and received in JSON format. The object is broken down into two main key-value pairs, billing details and shipping details. The amount of payment that is requested is included with billing details; product type and amount was considered superfluous for this demo product and not included. These two main objects of billing and shipping details are then broken down into smaller fields containing everything needed for payment validation and shipping the product once payment has been successfully authorized. Billing and shipping details share overlapping fields, this is intentional because while most shipping information is the same as the billing information that is not a guarantee. Customers are more than welcome to pay for an item then ship it to a different location. For the majority of cases where customer shipping and billing addresses are the same, the ‘Same as Shipping’ checkbox will quickly fill all overlapping billing details with the provided shipping information. Clicking the ‘Checkout’ button will send the post request.

The backend utilizes the Spring Boot framework to make this communication process easier. Spring Boot gives access to restful controller classes designed to receive requests and respond. Currently there is only one method controller route developed, the payment authorization method which utilizes post mapping to connect with the payment authorization request sent from the frontend. The actual processing of this payment authorization is kept separate from the controllers and instead is located in the payment processor service layer. This layer is currently still in development. Eventually this service layer will be responsible for the very important tasks of connecting to our third-party payment gateway and storing payment authorization details in our database for later batch processing. These two important tasks are only stubbed out at the moment but will be crucial to the success of our payment processing service. At the moment, our payment processor service layer is only able to validate whether or not the request contains data in all fields for billing and shipping details. If validation succeeds, a 200 okay response is returned; if validation fails a 422 unprocessable content error is returned, signifying that while the content type is correct the content itself contains errors. In the instance of an error being returned, additional information is also returned containing what fields contain errors. This will eventually expanded upon to allow customers the chance to re-attempt checkout after attempting to fix the erroneous fields.

**Demo Code**

This demonstration code is still far from deployment. Any attempt at demonstrating the current payment authorization workflow must be done using a local machine setup for development. Local machine setup includes, but is not limited to, installation of a Java 21 development kit, installation of the front end packages such as Angular 19, and compiling the code prior to running. Additional requirements may vary based on operating system. Once these needs have been met the backend payment processor service can be started by navigating to the file location and running the operating system command of mvn spring-boot:run; the payment processor UI can be started using the command npm start. Despite being an Angular project, a simple ng serve command will not suffice for demonstrating the UI locally as it needs to be started using a proxy config file to navigate the cross-origin resource (CORS) sharing issues. Browsers implement CORS protections to prevent webpages from making a request to a different origin, protecting against cross-site request forgery and other malicious attacks. Users on a windows machine will not have to worry about this as the provided start\_payment\_processor batch file can be run to start both payment processing service and UI.

**Dependencies and Assumptions**

This project is built on the assumption that more development is forthcoming. Crucial features of the payment processing workflow have not yet been implemented and only stubbed out as methods. Our payment processing service will need to fully implement these stubbed methods before it can be considered complete and one of these methods relies on a third-party connection that we have not yet made. This third-party could have a different structure to what we are anticipating, causing certain segments of the code to be rewritten to adapt.

Demonstrating this code on a machine locally is easy, demonstrating this machine on any machine locally is another task. There are many different operating systems and factors that can impact environment setup. Overall instructions to set up the local environment have been provided but the exact details will vary. We need to assume that this project can be shared and run successfully with the relevant parties while our next focus is on finding a stable deployed environment to show off progress.

**Summary**

Our payment processing service still has a lot of development needed before it can be considered the minimum viable product for release, but this small product is able to demonstrate a high level workflow of what it will achieve. This communication between client and server can be expanded to cover all the validations and security needs our merchant system will have. Any errors in this process can be handled and sent to the user interface where the system will decide what is to be done with it.

**References**

Kiernan, John. 2022. “How Credit Card Transaction Processing Works: Steps, Fees & Participants.” Edited by Alina Comoreanu. WalletHub. August 12, 2022. https://wallethub.com/edu/cc/credit-card-transaction/25511.

Steen, van Maarten and Andrew Tanenbaum. 2023. Distributed Systems. CreateSpace Independent Publishing Platform. <https://www.distributed-systems.net/index.php/books/ds4/>