Please review this document in its entirety prior to starting the initial stage of the interview/assessment process.

Web API Resources

1. Candidates need to wrap the assessment in Spring Boot if they are Web API resources

Top reasons applicants fail the initial stage of interview process/assessment.

1. **Unit Tests**-these are required and should be clear and easy to decipher. Regression, UAT
2. Must have a **Readme** files with instructions as to how you would run the app. **Get/Rewards, Get/Rewards (customer ID,) GET/transactions.**
3. Variable Naming Convention. If variable A holds the transaction data, then label it transaction data. Must be built so that another developer can easily find the file/information in the future.
4. Keep the implementation as simple as possible so it's easy for someone new to read, understand, and easily navigate the packages.
5. Must have **Gitignore** file and. **mvn/wrapper**
6. Do not upload external files as a folder.
7. Follow component-based software engineering principles when completing the assessment. Ensure the code is clean.
8. Copy and Pasting Code. Candidates who do this are highly likely to fail the assessment unless the code is modified and is good code. Candidates who just copy the code without ensuring the quality are highly likely to fail. The client has reviewed enough assessment over the years to quickly identify this.
9. When feasible, utilize the newest technology and features you are familiar with. Example: Javascript features like ES6 built in methods. Java version 8 or higher vs version 6.
10. Avoid using unnecessary 3rd party libraries.

Feedback on prior resources who did not pass the test

1*. There are a couple of issues. The controller only has one method, which exposes rewards for ALL customers instead of letting you retrieve one by id. (This is fine with our requirements; it's just my pet peeve.) There's a "Uil" class, which I assume is supposed to be "Util". I wish there were more tests. The "three months" is enforced only by the data returned by the DAO; in a real-world scenario, this solution would require old data to be purged from the database or reliance on a remote service that only returns three months' worth of data to start with.*

2. *I feel like he didn't think through the API design first.*
It seems like he used a TreeMap to calculate the Monthly Rewards and then just kind of lazily leaked this out to the consumer without thinking about how they might use it.
It'd be more convenient, at least for rendering it in a UI, to have an array of objects with month and amount properties.
There's other things I could nit-pick like some poor names such as <u>updateBeginDate</u> and CommonUtil, fields are package-private instead of private, and a lack of <u>unit tests</u> (which he noted in the README though) (the 1 unit test doesn't assert on the points).

***3**. Feedback: There’s no gitignore, which is leading to extra things getting included (.iml file, build package, etc.) No tests. The only endpoint is POST /rest/points which takes a request body full of transactions (month + amount) and formats them into a map of total points by month. The $100 = 0 points bug is there, too.*

***4** There are a couple of really strange things the code is doing (like their bizarre way of determining whether a transaction is part of the current month) and a bug for $100. The tests don’t add clarity, either.*

WebAPI Developer

A retailer offers a rewards program to its customers, awarding points based on each recorded purchase.

A customer receives 2 points for every dollar spent over $100 in each transaction, plus 1 point for every dollar spent between $50 and $100 in each transaction.

(e.g., a $120 purchase = 2x$20 + 1x$50 = 90 points).

Given a record of every transaction during a three-month period, calculate the reward points earned for each customer per month and total.

- Solve using Spring Boot
- Create a RESTful endpoint
- Make up a data set to best demonstrate your solution
- Check solution into GitHub

Developer Hiring Process:

1. Candidate Submission: Vendor submits candidate profile including link to homework. Homework is reviewed by lead dev.
2. Technical Screen: 30-minute WebEx *(camera on)* between lead developer and candidate to review homework in detail.
3. Interview: 60-minute webex *(camera on)* with technical leaders and candidate to meet candidate, assess technical abilities, review experience and culture fit.