**WellnessPay VP of Tech Candidate – 1-Hour Technical Challenge**

***Objective***

Evaluate architecture skills, coding ability, and product thinking in under an hour — focusing on **backend logic**, **frontend architecture**, and **scalability planning**.

**Part 1: Mini System Design (20 mins)**

**Prompt:**

Imagine you’re building a feature for WellnessPay that allows users to:

* Connect a bank account via Plaid
* View transactions
* Auto-identify HSA/FSA-eligible items
* Flag items that need a Letter of Medical Necessity (LMN)

**Tasks:**

* Briefly outline the architecture (you can sketch or write it out).
* What tools/frameworks/languages would you use and why?
* How would you ensure HIPAA compliance?

**What You’re Looking For:**

* Use of modern stack (e.g., Node, React, Supabase, Stripe, Plaid)
* Thoughtful use of APIs & AI
* Data privacy thinking

**Response:**

Potential high level architecture for the wellness pay system:

**Front End**: Next.js / React with super cute shadcn UI components & Redux for state management

**Middle Tier**: Next.js server side components, to integrate numerous back end systems, but primarily with a Supabase backend database / security infrastructure of the wellness pay application.

**Back End**: Supabase (primary Wellness pay data store, security paradigm, etc..), Plaid for KYB, bank account details integration, etc.., Python Django backend for proprietary ML/AI model inference, NLP processing, vector search / similarity search capabilities, and other capabilities that python shines over Javascript on, and for payment rails infrastructure / merchant management, Stripe platform. Next.js server side components would be the hub for integrating these various Restful backend services platforms with the front end. Note: I worry a little bit about the potential for no code implementations that might be created in supabase that are not tracked in version control systems if various people are working on the platform and data model design and incremental modifications would have to be done with CLI or sql script migrations or some other way instead of using the user interface capabilities to be able to consistently migrate data model changes between environments.

**Security:** The wellness pay system would leverage off Supabase’s HIPPA compliance and security infrastructure, using roles-based security, data encryption at rest and data in transit encryption features, and enforcement of client data privacy requirements of the system. Next.js would integrate directly with the Supabase security system. From a system perspective this would help with HIPPA compliance, but company / organizational program with HIPPA compliant cyber security standards and policies would also be required beyond simply technical security controls of the platform. With potential payment processing requirements and PCI DSS security standards potentially being a requirement, Stripe infrastructure would be used to satisfy those security requirements, however quarterly scans and organizational cyber security policies and procedures may also be required over time as the platform code base evolves.

**CI/CD**: Github actions, simple implementation that works well

**Cloud Platform:** AWS, seems to be the best fit from a Supabase backend, so Next.js front end app and (optional) Python proprietary services would have to be hosted in the same region with latency between front and back-end systems optimized as much as possible.

**ML/AI:** If LLM application tier requirements are required in the platform or a trained classifier proprietary model(s) are required to classify transactions, the AWS eco system supports Claude as a good LLM choice, with Sagemaker utilized for more proprietary classification models deployment and training if required, such as training random forrest classifiers. If necessary, an Azure subscription could be spun up if OpenAI models were required, in which case the ML/AI eco system of choice would be Azure ML studio / foundry instead of Sagemaker since Microsoft is slightly leading in this area.

**Architecture Rationale:**

Considering the potential architecture from the front end through to the back end, React is a more popular framework for front end systems these days than say Angular. Additionally, although not exactly analogous, React development knowledge is somewhat transferable to React Native system development, which is a wonderful mobile application framework, if business requirements change over time requiring mobile solutions. Next.js is the most popular full stack React framework in 2025 at this point, so Its api routes capability would act as the intermediary in integrating with the Supabase main backend as well as optional python restful services if Python’s ML / NPL processing capability is required. Additionally, api routes would integrate other critical backend services such as integrating with Plaid for bank account integration and Stripe for merchant management, under writing, payment processing, and all systems integrated together in the Next.js api routes / server-side components.

The Supabase backend as a service would be the main backend of the system, leveraging Supabase’s HIPPA compliance and data privacy features. Given that classifying transactions will likely grow to be significantly complex, potentially requiring proprietary model(s) that are potentially designed in a sophisticated blended ensemble design of multiple (LLM or proprietary) models classifying independently and results ‘voted’ upon to provide final high quality and sophisticated classification across a variety of merchant data sets. The ‘classification engine’ so to speak, might eventually have to reside in a Python service infrastructure noted above as a core sophisticated ML infrastructure with a very strong automated testing regime wrapped around it from a process and maintenance standpoint.

**Part 2: Backend Code Challenge (25 mins)**

**Prompt:**

Write a simple Express.js (or preferred stack) API that:

* Accepts a POST with a mock transaction (e.g., merchant name, amount, MCC)
* Checks if the item is HSA/FSA eligible using a mocked list
* Returns a JSON response with:
  + eligible: true/false
  + needsLMN: true/false

**Bonus:** Add a confidence score or a mocked AI classification (hardcoded is fine).

**Deliverables:**

* Code file or GitHub repo
* Should work in Postman or simple curl

**What You’re Looking For:**

* Clean code
* Proper structure and logic
* Clear responses and data handling

**Response:**

I’ve largely being using Next.js and python recently, so I implemented the sample api in Next.js even though it’s a little overkill in comparison to Express.js.

>> git clone <https://github.com/ColinWrinch/wellnesspay.git> (with whatever tool you prefer)

>> cd wellnesspay

>> npm install

>> npm run build

>> npm run dev

Testing curl commands as follows (using MS Dos format):

echo "============================================================================================="

echo "Scenario 1: Video store rental transaction"

curl -X POST http://localhost:3000/api/txn-hsa-fsa-classification ^

-H "Content-Type: application/json" ^

-d "{\"merchantName\": \"Blockbuster Video\", \"txnAmount\": 49.99, \"txnDescription\": \"Titanic Rental\", \"mcc\": \"1000\", \"sku\": \"654321\", \"upc\": \"654321\"}"

echo "============================================================================================="

echo "Scenario 2: Pharmacy blood pressure monitor purchase"

curl -X POST http://localhost:3000/api/txn-hsa-fsa-classification ^

-H "Content-Type: application/json" ^

-d "{\"merchantName\": \"Colin Local Pharmacy\", \"txnAmount\": 99.99, \"txnDescription\": \"Blood Pressure Monitor\", \"mcc\": \"5912\", \"sku\": \"1001\", \"upc\": \"10011001\"}"

echo "============================================================================================="

echo "Scenario 3: Pharmacy chocolate bar purchase"

curl -X POST http://localhost:3000/api/txn-hsa-fsa-classification ^

-H "Content-Type: application/json" ^

-d "{\"merchantName\": \"Colin Local Pharmacy\", \"txnAmount\": 29.99, \"txnDescription\": \"Chocolate Bar\", \"mcc\": \"5912\", \"sku\": \"123456\", \"upc\": \"123456\"}"

echo "============================================================================================="

echo "Scenario 4: Pharmacy Insulin injection transaction, insulin not on item list"

curl -X POST http://localhost:3000/api/txn-hsa-fsa-classification ^

-H "Content-Type: application/json" ^

-d "{\"merchantName\": \"Colin Local Pharmacy\", \"txnAmount\": 29.99, \"txnDescription\": \"Insulin injection\", \"mcc\": \"5912\", \"sku\": \"123456\", \"upc\": \"123456\"}"

**Output Results of Curl Commands:**

A screenshot of a computer

AI-generated content may be incorrect.

**Part 3: Rapid Product Strategy Answer (10–15 mins)**

**Prompt:**

We’re preparing to onboard 100 merchants in Q4 via our HSA/FSA checkout plugin.

In 5 bullet points, explain how you’d:

1. Ensure clean onboarding (Stripe Connect, eligibility classification)
2. Maintain code quality across the plugin rollout
3. Track usage and improve performance
4. Keep merchant data secure
5. Manage a small dev team to build and scale fast

**Responses:**

1. Stripe connect hosted merchant onboarding would be used to leverage their merchant management / underwriting / identity verification / bank details (merchant KYC/KYB/onboarding) capability as merchant management, verification and onboarding is a significant process to build and maintain as a proprietary system. Eligibility classification lookup on processing merchant transactions would utilize an eligible product list, keying off SKU or UPC as applicable, potentially requiring some integration work to integrate the merchant’s product catalog may be required if there are novel considerations in merchant inventory on a merchant-by-merchant perspective. Eligible product classification would reside in the supabase backend and maintained throughout merchant onboarding.
2. Integration enhancements for merchants, changing and shifting business requirements, and increased technical sophistication and monitoring of the checkout plugin would force significant ongoing changes to the checkout plugin before it becomes increasingly more mature. In order to avoid breaking changes to previous onboarding implementations and to control code quality, strong team level version control (git workflow, code reviews, etc.) and unit testing paradigm would be required to manage quality of the checkout plugin as it evolves. Ideally external contractors would not be involved in this part of the system and instead full-time employees that have skin in the game would maintain the complexity of this module. Keeping the code base implementation to simple implementations and avoiding complex solutions would be critical in keeping the code base maintainable and avoiding having to create, maintain, and operate multiple different versions in production environments.
3. APM system integration and comprehensive logging frameworks would be required to manage and monitor and investigate usage of the platform at runtime (i.e. gather telemetry information on requests, view error responses, etc.) as well as centralized splunk or other logging aggregation infrastructure to perform detailed merchant / txn level investigations. Ongoing system performance and traffic ‘bursts’ would be to be monitored in order to capacity plan the platform as it scales with increasing numbers of merchants/customers. If the system is using an ORM framework for database access, almost assuredly raw sql optimizations will be required in core areas of the system to improve scalability and performance.
4. Merchant onboarding data (in particular their banking details) would largely be contained in Stripe’s onboarding infrastructure, however some merchant details (on a data need to know basis only) as required by the system would likely be integrated into the Wellness pay supabase backend infrastructure. Integrating merchant data with the front end would require, leveraging Stripe’s apis as well as supabase instrastructure would be aggregated on the server side in the Next.js architecture discussed earlier. On the back-end, Stripe is a sophisticated PCI compliant enterprise platform that securely maintains merchant information, and with Supabase’s capability on HIPPA security standards, back-end merchant data privacy and protection and standards should be sufficient for protecting both data at rest as well as data in transit in the wellness pay system.
5. Assuming that the platform is operational and we are looking to focus specifically on onboarding the 100 merchants or as many as possible that the business can hunt down in order to scale the business as fast as possible: the team would have to all be aligned on the task of maintaining the setup, user / merchant configuration, and changes required to the system to support the merchant onboardings. In preparations for the onboardings, the team would have to self-organize to isolate all the system requirements and job functions required to effectively onboard a merchant. For instance, one developer might be maintaining the stripe integrations with the merchant details, so they might review the data of the onboarded merchants, synch the merchant information into the Supabase database backend, etc. Another team member or developer might focus on the merchant’s inventory and verify product classification routines, with another team member potentially reviewing end to end processing and operation of the plugin and a final member enhancing unit testing scripts to prevent impacts to previously onboarded merchants.

With the team organized and prepped in terms of roles and responsibilities, merchants onboarding and system enhancements activity would be managed through centralized JIRA scrum or kanban boards, with daily standups of activities status’ discussed as a team and potentially breakout discussions around code management, change management, and testing onboarded merchants in dev/test/prod environments. Ideally the team would support a handful of merchants to start and with an increase in productivity in performing data integrations or merchant setups, the pace of merchant integrations would increase as the changes to the checkout plugin decrease over time. The processes of onboarding would get increasingly more automated and the checkout plugin increasingly more feature rich, fault tolerant, performant, and stable. Likely, in the middle of all this focus on onboarding merchants, business folks will begin thinking about the next round of features for the business, so some level of air cover by management would be required to keep the team focused on getting increasingly more proficient in merchant onboarding. When onboarding is more streamlined, some members of the team could probably start on the next versions of the wellness pay platform.

Generally speaking, the small team would be organized as an agile scrum team using JIRA for ticket tracking and estimation as the team level operation for software development activities in the platform. In order to support github action or other CI/CD processes, unit testing as well as feature flag infrastructure like Launchdarkly could be used to support regular high-quality releases to production environments.