Thrivory: Head of Product Engineering Take-Home Practical

Background

Thrivory is a healthcare fintech platform focused on providing alternative financing solutions for healthcare practices. In practice, Thrivory accelerates insurance reimbursements, giving medical practices faster access to the funds they've earned (through a non-recourse claim-to-payment platform).

As the Head of Product Engineering, you will be responsible for building and leading teams that develop robust, scalable solutions in this domain. A key aspect of this role is evaluating and hiring strong engineers who can drive Thrivory's technical vision forward. This take-home assignment puts you in the hiring manager's shoes. You will design a coding challenge for a hypothetical mid-to-senior backend/fullstack engineer candidate who would report to you. Through this exercise, we aim to assess your ability to create a meaningful technical evaluation that reflects Thrivory's product and engineering needs.

Objective

This practical assignment assesses your skills in:

- **Architectural & System Design Thinking:** Crafting challenges that require candidates to design systems or components with sound architecture.
- **Practical Coding & Problem Solving:** Ensuring the challenge involves real coding tasks that solve non-trivial problems relevant to our domain.
- **Real-World Trade-Off Awareness:** Evaluating how well you incorporate considerations of trade-offs (e.g. simplicity vs. complexity, performance vs. maintainability) in the challenge.
- **API Design & Integration:** Reflecting Thrivory's platform needs by involving API design or integration with external services/data.
- **Security, Scalability & Maintainability:** Emphasizing the importance of secure coding, scalable solutions, and clean, maintainable code in the challenge and its evaluation criteria.

Your Role and Task

Imagine you are hiring a mid-to-senior Backend or Fullstack Engineer for Thrivory. You need to develop a take-home coding challenge to assess that candidate's abilities. The challenge should be grounded in a scenario relevant to Thrivory's domain (healthcare, fintech, or product platform). For example, dealing with financial transactions, healthcare claims data, or an internal product API. The preferred implementation languages are Python or PHP (Laravel) (our primary stacks), but the candidate may use any language they are comfortable with.

Deliverables

You are required to submit three components for this practical:

1. **The Coding Challenge Description:** Write a clear, structured prompt for the take-home assignment you would give to the engineer candidate. This should read as if it's the instructions to that candidate. Be sure to include:
    - Scenario/Context: A brief description of the problem domain (e.g. a simplified feature or service related to Thrivory's healthcare fintech platform). For instance, you might base the challenge on building a mini API for processing insurance claims or a small module for a fintech ledger, anything that echoes real tasks an engineer at Thrivory might tackle.
    - Task Requirements: What exactly the candidate should build or implement. Define the scope clearly, for example, "Design and implement a RESTful API endpoint to do X," or "Build a small service that ingests Y and outputs Z." Make sure the requirements touch on both functionality and quality (e.g. error handling, edge cases).
    - Constraints & Data: Any input data or data models the candidate should consider (you can provide sample data or schemas if needed). If external integration is expected (e.g. calling a third-party API or reading a file), clarify how to approach it (possibly by stubbing or using a fake service).
    - Preferred Tech: Note that while any language is acceptable, Python or PHP (Laravel) are preferred. If the challenge would particularly benefit from one of these languages' ecosystems (for example, using a specific library or framework), you can mention it, but it's not required that the candidate use it.
    - Deliverable Expectations: Explain what the candidate should deliver back. Typically, this would be the source code (in a repository or zip), plus any documentation. Encourage them to include a README with setup instructions, a brief design explanation, and even unit tests if appropriate.

For instance, you might say *“Include instructions on how to run your solution, and any tests or assumptions you considered.”.*
- Time Guidance: (Optional but recommended) Give an expected time range or effort level. For example, “This challenge is designed to be doable in about 4-6 hours of focused work,” so candidates can gauge scope and depth. Emphasize that completeness and clarity are more important than a huge scope, it’s fine if they don’t implement every edge case as long as they can explain their decisions.

2. **Justification & Evaluation Criteria:** After writing the challenge prompt, provide a brief explanation (as your justification) for choosing this exercise. This section is for the hiring committee (us at Thrivory) to understand your thought process, and *it would not be shown to the engineer candidate*. In this justification, address the following:
    - Relevance of the Challenge: Why is this problem a good representation of the work at Thrivory? Explain how it ties into our product or technical goals. For example, if your challenge involves processing healthcare claims or designing a payment API, note that this mirrors real tasks on our platform (handling claims data, ensuring financial calculations are correct, etc.).
    - Skills and Behaviors Assessed: Break down which skills the challenge will reveal. For instance, does it test database design, API design, concurrency, error handling, etc.? If the challenge has an architectural component, mention that you expect the candidate to demonstrate design thinking (e.g. how they structure the project or any discussion of system design). If it involves integration, mention expecting them to handle external API calls or data parsing. Tie these to the bullet points in the Objective (architecture, coding, trade-offs, API, security, scalability, maintainability).
    - Real-World Trade-Offs: Describe any trade-offs the candidate might face in the challenge and how a strong candidate would navigate them. For example, maybe the candidate has to decide between a quick hard-coded solution vs. a more extensible design, or how much error checking to implement given time constraints. You can note that you intentionally designed the task to see how they prioritize correctness vs. completeness, or simplicity vs. performance, etc.
    - Evaluation Plan: How would you, as the hiring manager, evaluate the submitted work? Outline the criteria you would use to grade or assess the solution. This can be a list of aspects you will look at: code correctness (does it meet the requirements?), code quality (readability, organization, following best practices), testing (did they include unit tests or

demonstrate how they'd test their code?), documentation and clarity (is their approach clearly explained?), and alignment with the focus areas (did they consider security, scalability, etc. appropriately for the task?). Also mention how you'd weigh these aspects. For example, you might prioritize correctness and clarity over minor inefficiencies. The goal is to show us that you know what a good vs. mediocre solution looks like.

3. **Baseline Solution / Rubric:** Finally, provide a sample solution outline or a grading rubric for the challenge you wrote. This serves as a benchmark to demonstrate what you consider a successful implementation. You can approach this in one of two ways (or a combination of both):
    - Sample Solution: Describe (at a high level) an exemplary solution to the challenge. This could include the core approach, key components or algorithms, and perhaps pseudo-code or simplified code snippets illustrating how the candidate might solve specific parts. You don't need to write the full code for every part, but you should highlight the important pieces. For example, if the challenge was to build a REST API for claim processing, your sample solution might outline the data model (e.g. classes or database schema for claims), the API endpoint structure, and the logic for determining funding eligibility, including error handling. Mention any edge cases your solution covers (and that you'd expect strong candidates to handle or at least discuss).
    - Rubric: In addition or alternatively, present a rubric with criteria and what constitutes "good," "average," or "poor" performance for each. For instance:
        - *Correctness:* Does the solution fulfill all requirements (e.g. all API endpoints work as specified, calculations are accurate)? A good solution meets all requirements and handles edge cases; a poor one has bugs or missing functionality.
        - *Code Quality:* Is the code well-structured, readable, and maintainable? A top solution might use clear modular structure or adhere to clean code principles (meaningful naming, DRY code, etc.), whereas a weaker one might be tangled or lack organization.
        - *Architecture/Design:* Did the candidate choose an appropriate design for the problem? For example, using a sensible data model, layering (if applicable), and considering future growth. A great submission might even include a short explanation of how they'd extend or scale the solution.
        - *Trade-offs & Decisions:* Did the candidate discuss or implicitly make wise decisions about trade-offs? Note if your sample solution makes certain choices (e.g. using an in-memory approach for

    simplicity, with a comment that in a real system you'd use a database, a strong candidate might do the same).
    - *Testing & Documentation:* Did they include tests or at least test their code manually? Did they document assumptions or provide a README? A model solution would include a brief README explaining how to run it and any key assumptions/choices.
    - *Security & Robustness:* Even if the challenge is small, did the candidate consider security (for example, input validation to prevent bad data, awareness of sensitive data if any)? Did they handle errors gracefully (avoiding crashes or leaking information)? For scalability, if applicable, did they write the code in a way that wouldn't obviously fail with larger inputs or multiple users?
  - You may combine the above by writing a narrative of an ideal solution *and* bulleting out the criteria. The goal is to convince us that you know what excellence looks like in the context of this challenge. Your baseline should set a standard that you would expect a hireable mid-to-senior engineer to meet or exceed.

Guidelines and Expectations

- **Relevance:** The challenge should be realistic for Thrivory's line of business. We want to see that you understand the kind of engineering problems we solve (e.g. handling healthcare data, financial calculations, integrating with third-party services like EHR systems or payment gateways, etc.). You don't need deep domain knowledge in the prompt, but it should feel relevant to either healthcare, fintech, or digital product platforms.
- **Clarity & Scope:** Make sure your challenge description is clear and unambiguous. A candidate reading it should understand what is expected. The scope should be reasonable. Complex enough to be interesting, but scoped to be completed in a day or two. (As a rule of thumb, avoid an overly broad "build a whole application" scope; instead focus on a critical component or a few interconnected tasks.)
- **Focus on Key Themes:** In your justification and rubric, explicitly tie back to the key themes (architecture, coding, trade-offs, API, security, scalability, maintainability). We will look for those connections. For instance, if your challenge doesn't naturally involve scalability concerns, you could still mention how a candidate might discuss scaling their solution in their write-up.
- **Evaluation Mindset:** Approach this as if you are actually going to use this challenge to hire someone. What would you want to see in their submission? The best responses will demonstrate an understanding of what differentiates a great

engineer from an average one in practical terms. Be specific in your criteria (e.g., rather than just saying “code quality,” mention things like readability, use of idiomatic constructs, etc.).

- **Format:** Your submission (this practical) can be organized in a clear way (you can use the above Deliverables 1, 2, 3 as section headings in your document). Use bullet points or tables in the rubric if it improves clarity. Remember, part of this exercise is also to assess how you communicate complex ideas in written form, so clean structure and writing will count.

Conclusion

This take-home practical is an opportunity to showcase your technical leadership and evaluative thinking. By creating a well-considered coding challenge (and accompanying justification and solution guidelines), you demonstrate your ability to align hiring practices with Thrivory’s technical goals and standards. We are excited to see how you would ensure only the best engineers join our team through a thoughtful evaluation process. Good luck, and if you have any questions about the assignment, feel free to reach out for clarification. We look forward to reviewing your submission and gaining insight into your approach to technical hiring and problem definition.