**Technical Assignment for a Software Developer at**

**NEXT BASKET**

**Part 1: Introduction and Our Philosophy**

**1.1. Welcome to NEXT BASKET: Beyond the Code**

Welcome to the NEXT BASKET selection process. We are a company driven by an engineering culture that values innovation, creativity, and solving real, large-scale problems. We believe the best products are created by teams where each member is not just an executor but a partner in our shared mission.

We are looking for individuals with a proactive and conscious approach to software development—engineers who strive to understand the business context behind every task and contribute with ideas and expertise to build the future of our products. This technical assignment is the first step through which we invite you to demonstrate not only what you can do but also how you think.

**1.2. Why is this process different? Our commitment to transparency and modern engineering**

We understand that the requirements in this assignment are different from the standard. Our goal is not to make it difficult for you but to create a maximally transparent and objective process that allows us to assess the skills that are truly important to us. This approach is designed to help us understand your way of working in depth.

**Regarding the video recording:** The purpose of the video recording is not for you to create a perfect presentation or to speak continuously. On the contrary, we want to see your authentic workflow. This allows us:

* **To distinguish candidates who build the solution themselves** from those who rely primarily on ready-made code found online.
* **To get a complete picture of your technical thinking and working style**—how you structure your project, how you approach debugging, and what decisions you make on the fly.
* **To ensure transparency and commitment to the task** by observing the real effort invested in solving it.

**Regarding AI assistants:** The use of AI tools like GitHub Copilot, ChatGPT, or Anthropic is now an integral part of a modern software engineer's workday. We not only permit it but require it because we want **to assess your ability to use AI effectively**. It is valuable for us to see how an engineer uses these tools to increase productivity—how they ask the right questions, how they validate and adapt the generated code, and how they use AI as a powerful assistant, not as a substitute for critical thinking.

The combination of a video recording and the mandatory use of AI is a powerful filter that evaluates not only technical skills but also mastery of modern workflows, transparency in problem-solving, and adaptability. Candidates who accept this challenge demonstrate a high level of confidence and cultural alignment with our principles of innovative and transparent engineering.

**Part 2: General Requirements and Rules of the Assignment**

**2.1. Your Mission: Choose One Task**

Your mission is to choose and solve **only one** of the four technical tasks described in Part 3. We recommend that you carefully review all tasks and choose the one that best matches your experience and interests. Select the task that will give you the best opportunity to demonstrate your strengths as an engineer.

**2.2. Video Recording Requirements: Show Us How You Think**

**Goal:** To observe your authentic development process—from the initial project structure to the final details.

**Content of the recording:**

* It is mandatory to show your entire screen or at least the key windows you use: your IDE/text editor, the terminal/console, the browser, and most importantly—**the interface of the AI assistants you use** (e.g., the web interface of ChatGPT, the GitHub Copilot extension in the editor, etc.).

**Duration and format:**

* The total duration of the recording(s) should be for the whole problem solving **за цялото решаване на задачата**
* The recording can be made in one or several sessions. There is no requirement for it to be continuous. You can record only the key moments of development where you are solving the main problems.

**Tools:**

* You can use any screen recording tool you are comfortable with. We recommend free and easy-to-use tools like **OBS Studio, Loom**, or **Screenity** (a Chrome extension).

**Audio commentary:**

* Verbal commentary ("thinking out loud") during the recording is **entirely optional**. It is encouraged as it provides additional context to your decisions, but its absence will not negatively affect your evaluation.

**2.3. Mandatory Use of an AI Assistant: Your Modern Toolkit**

Demonstrating work with AI assistants such as **GitHub Copilot, ChatGPT, Anthropic, Tabnine**, or others is a **key and mandatory part** of the evaluation. We expect to see how you integrate these tools into your workflow for tasks such as:

* Generating boilerplate code (template code for servers, configurations, etc.).
* Writing automated tests.
* Refactoring existing code for better readability or performance.
* Debugging errors and searching for solutions.
* Researching architectural approaches or explaining complex concepts.

**2.4. Submitting Your Solution**

After you have finished, please send us the following three components:

1. **A link to a public Git repository** (e.g., on GitHub, GitLab) containing the entire source code of your solution.
2. **A README.md file** in the root of the repository that contains clear and concise instructions for installing dependencies, running, and testing the project.
3. **A public link to the video file(s)** (e.g., via Google Drive, Dropbox, an Unlisted video on YouTube, Loom). Please ensure the link is accessible for viewing without requiring special permissions or registration.

**Part 3: Technical Tasks (Choose One)**

**3.0. Task Selection Guide**

The four tasks are designed to test different aspects of software engineering and to attract candidates with different profiles and experience. This table will help you make an informed choice based on your strengths.

| Task | Key Technologies and Concepts | Expected Complexity | Recommended Profile |
| --- | --- | --- | --- |
| **1: Event Feed** | WebSocket, Real-Time Communication, Multi-Tenancy, Backend (Node/Python), Frontend (Vanilla JS) | Medium | Backend or Full-Stack Developer with an interest in real-time systems. |
| **2: Document API** | REST API, Security (RBAC), Multi-Tenancy, File Storage, Authentication | Medium | Backend Developer with a focus on security, API design, and data management. |
| **3: Fraud Detection** | REST API, Rules Engine, Machine Learning Integration (scikit-learn), Data Processing | Medium to High | Backend Developer with an interest in Data Science or an engineer who wants to show the ability to integrate ML models. |
| **4: AI Checkout** | E-commerce Logic, LLM Integration, Docker, Cloud Deployment, CI/CD, Testing, System Design | High | Senior Full-Stack or Backend Developer with experience in DevOps, cloud technologies, and building end-to-end systems. |

**3.1. Task 1: Real-Time Multi-Tenant Event Feed**

**Problem**

Build a real-time event broadcasting system where multiple tenants can send and receive their own events in real-time, with strict tenant isolation.

**What to Build**

* **Backend Service (Node.js/Express or Python/FastAPI):**
  + WebSocket server that handles connections with tenant authentication.
  + Simple in-memory event storage per tenant.
  + REST endpoint to post events: POST /events (with tenant header).
  + WebSocket broadcasts events only to same-tenant connections.
* **Frontend (Simple HTML/JS):**
  + Basic page with tenant login (dropdown: "Tenant A" or "Tenant B").
  + Real-time event list that updates via WebSocket.
  + Form to send new events.
  + Clear visual indication when events arrive.

**Data Model**

JSON

{

"id": "uuid",

"tenant\_id": "tenant\_a",

"message": "User logged in",

"timestamp": "2025-01-20T10:30:00Z"

}

**Requirements**

* **Tenant Isolation:** Tenant A never sees Tenant B's events.
* **Real-time:** Events appear in UI within 1 second.
* **Simple Auth:** Use tenant ID in a header or query parameter (no complex JWT required).
* **In-Memory Only:** No database required.
* **Automated Testing:** The solution must include an appropriate level of automated tests.

**Success Criteria**

* Two browser windows (representing different tenants) show different, isolated event streams.
* Posting an event from one tenant appears only in that tenant's window(s).
* The code is clean, well-documented, and includes clear setup instructions in the README.md.

**3.2. Task 2: Multi-Tenant Document API**

**Problem**

Create a secure document storage API where multiple tenants can upload, list, and download their files with role-based access control (RBAC).

**What to Build**

* **Core API (Node.js/Express or Python/Flask):**
  + POST /documents - Upload a file (using form-data).
  + GET /documents - List a user's accessible documents.
  + GET /documents/:id - Download a specific document.
  + DELETE /documents/:id - Delete a document (admin only).

**Authentication**

* Simple token-based authentication with predefined users.
* Users belong to tenants and have roles (admin or user).

**Data Model**

JSON

{

"id": "doc123",

"tenant\_id": "company\_a",

"filename": "contract.pdf",

"uploaded\_by": "user1",

"upload\_date": "2025-01-20",

"access\_level": "tenant"

}

*Note: access\_level can be "tenant" (visible to all in the tenant) or "private" (visible only to the uploader and admins).*

**Access Rules**

* Users can only see documents belonging to their own tenant.
* Regular users see all "tenant" level documents plus their own "private" documents.
* Admins see all documents within their tenant (both "tenant" and "private").

**Requirements**

* **Tenant Isolation:** Company A cannot access Company B's files under any circumstances.
* **Role-Based Access:** Permissions for admins vs. regular users must be strictly enforced.
* **File Storage:** Use the local filesystem, storing files in a structure like ./uploads/<tenant\_id>/<filename>.
* **Security:** Implement input validation and protect against path traversal vulnerabilities.
* **Automated Testing:** The solution must include an appropriate level of automated tests.

**Pre-configured Test Data**

JavaScript

const USERS = {

"admin\_a": { tenant: "company\_a", role: "admin", token: "token\_admin\_a" },

"user\_a": { tenant: "company\_a", role: "user", token: "token\_user\_a" },

"admin\_b": { tenant: "company\_b", role: "admin", token: "token\_admin\_b" }

};

**Success Criteria**

* A file uploaded by user\_a can be accessed by admin\_a but not by admin\_b.
* An admin can delete any document within their tenant, but a regular user cannot.
* The API returns clear and appropriate error messages for unauthorized access attempts.

**3.3. Task 3: Simple Fraud Detection API**

**Problem**

Build a transaction analysis API that flags potentially fraudulent transactions using a combination of a rules engine and a simple machine learning model.

**What to Build**

* **Transaction Processing API:**
  + POST /transactions - Submit a transaction for analysis.
  + GET /transactions/flagged - List recently flagged transactions.
  + The system must provide real-time processing with an immediate fraud score.

**Detection Logic**

1. **Rules Engine:** Implement simple if/then rules with configurable thresholds.
2. **ML Component:** Use scikit-learn's IsolationForest for anomaly detection.
3. **Scoring:** Combine the flags from the rules engine with the ML anomaly score to produce a final risk assessment.

**Transaction Model**

JSON

{

"id": "tx123",

"user\_id": "user456",

"amount": 1500.00,

"location": "US",

"timestamp": "2025-01-20T10:30:00Z",

"merchant\_category": "electronics"

}

**Detection Rules**

* Amount > $5000 = **High risk**.
* Same user making transactions in different countries within 1 hour = **High risk**.
* Amount > 10x the user's average transaction amount = **Medium risk**.

**Requirements**

* **ML Integration:** The IsolationForest model should be trained on startup using synthetically generated data.
* **Real-time:** Transaction processing should complete in under 200ms.
* **Configurable Rules:** The thresholds for the rules should be easily adjustable (e.g., via environment variables or a config file).
* **Clear Scoring:** The API response for a transaction should include a risk level (low/medium/high) and the reasons for the assessment.
* **Automated Testing:** The solution must include an appropriate level of automated tests.

**Simplified ML Approach**

Python

# On application startup

# Pre-generate synthetic training data

training\_data = generate\_synthetic\_transactions(1000)

# Initialize and train the model

model = IsolationForest(contamination=0.1)

model.fit(feature\_matrix)

# For each incoming transaction

ml\_score = model.decision\_function([transaction\_features])

rule\_flags = check\_rules(transaction)

final\_risk = combine\_scores(ml\_score, rule\_flags)

**Success Criteria**

* A transaction with a high amount (e.g., $6000) is correctly flagged as "high risk".
* The ML model successfully detects outliers in transaction patterns that are not caught by the rules.
* The API returns a risk assessment in under 500ms.
* The submission includes 5-10 test transactions with their expected outcomes documented in the README.md.

**3.4. Task 4: AI-Powered Checkout & Upsell**

**Problem**

Build a tenant-aware e-commerce mini-workflow that:

1. Offers catalogue, cart, and checkout APIs.
2. Calls a Large Language Model (LLM) at checkout to suggest up-sell items.
3. Persists completed orders.
4. Runs both locally via Docker Compose and is deployable to a free-tier cloud provider.

**What to Build**

| Component | Core Responsibilities | Minimum Scope |
| --- | --- | --- |
| **Catalog API** | List products, prices, and stock per tenant. | Read from an in-memory list or a JSON file; provide endpoints to list all products and fetch one by ID. |
| **Cart & Checkout API** | Create/update a cart; on checkout, reserve stock and create an order. | Implement REST routes for cart creation/update and a checkout route that validates stock. |
| **AI Upsell Service** | When UPSELL\_ENABLED is true, request ≤3 complementary products for the current cart and add their explanations. | Decouple into its own module; may call OpenAI or a local model; must log the prompt and the response. |
| **Order Store** | Persist final orders. | Use an in-memory map or an SQLite file. |
| **Public Storefront (SPA)** | A one-page UI: product list, "Add to cart" button, Checkout button, upsell display, and order confirmation. | Can be implemented with plain React, HTMX, or vanilla JS. |

**Extra credit:** Implement a feature flag UPSELL\_ENABLED and a promo code SUMMER10 for a 10% discount.

**Requirements**

| Area | Concrete Expectations |
| --- | --- |
| **E-commerce logic** | Stock must be checked atomically on checkout. The final price calculation should be: subtotal, then apply the 10% promo discount (if present), then add 20% VAT. |
| **Tenant isolation** | All resources (products, carts, orders) must be scoped by tenant\_id. Any cross-tenant access attempt must fail with an error. |
| **AI integration** | The upsell logic must be isolated in its own file/module. The checkout process must succeed even if the LLM call fails or is disabled via the feature flag. |
| **Quality gates** | The project must be configured with a linter and a code formatter. It must include at least 5 unit tests and 1 integration test. A GitHub Actions workflow running the tests on push is required. |
| **Docs** | Provide a short OpenAPI (Swagger) or Postman specification for the APIs. Include 2-4 diagrams (e.g., context, sequence) explaining the architecture. The README.md must contain clear local and cloud run instructions. |
| **Deployability** | The command docker-compose up must work locally. Provide a live URL of the application deployed on a free-tier cloud service (e.g., Heroku, Vercel, Render). |

**Success Criteria**

* The checkout process correctly reserves stock, applies the discount and VAT, and stores the final order.
* When enabled, the upsell service returns relevant product suggestions with explanations.
* Toggling the UPSELL\_ENABLED feature flag works without requiring code changes or a restart.
* All CI tests pass; the code and documentation are clean and idiomatic; both local (Docker) and cloud deployments are functional and accessible.

**Part 4: Evaluation Criteria and Conclusion**

**4.1. What Will We Evaluate? A Look Beyond Working Code**

Your solution will be evaluated based on the following criteria, which give us a comprehensive view of your engineering skills. We are looking not just for working code, but for an elegant, well-thought-out, and sustainable solution.

1. **Problem-Solving Approach:** (Evaluated primarily from the video)
   * How do you analyze the task requirements?
   * How do you plan and structure your solution before you start writing the main code?
   * What technical trade-offs do you make during development, and what is the logic behind them?
2. **Architecture and Code Quality:**
   * Modularity, cleanliness, and clear structure of the code.
   * Adherence to best practices and conventions for your chosen language and framework.
   * Is your solution easy for another developer to understand and build upon?
3. **Efficiency and Resourcefulness in Using AI Assistants:**
   * **High score:** You demonstrate skillful use of AI to accelerate routine tasks (generating boilerplate, tests, documentation), to explore alternative solutions, for refactoring, or for debugging. You show critical thinking towards AI suggestions by verifying, adapting, and improving them.
   * **Low score:** You simply copy large blocks of code without a clear understanding, rely on AI to do all the work for you, and struggle to proceed when the AI assistant provides an incorrect or incomplete answer.
4. **Adherence to Requirements and Success Criteria:**
   * Does the solution meet all functional and non-functional requirements described in the task (e.g., tenant isolation, real-time updates, security rules, RBAC)?
5. **Testing and Quality:**
   * Presence, adequacy, and coverage of automated tests (unit, integration), where applicable and required by the task (especially in Task 4).
6. **Completeness of the Submitted Material:**
   * Quality and clarity of the README.md file.
   * Ease of running and testing the project.
   * Accessibility and correctness of the provided links to the repository and video recording.

**4.2. Next Steps and Thank You**

  After you submit your solution, we will review it within a few business days and get back to you with feedback. Successful candidates will be invited to the next step of the process—a technical interview, during which we will discuss your solution in detail, as well as other technical topics.

We want to sincerely thank you for the time, effort, and commitment you will invest in this assignment. We highly value your interest in NEXT BASKET and look forward to seeing your solution