PASTA worksheet

Stages	Sneaker company
I. Define business and security objectives	Make **2-3 notes** of specific business requirements that will be analyzed. ● *Will the app process transactions?* ● *Does it do a lot of back-end processing?* ● *Are there industry regulations that need to be considered?*
II. Define the technical scope	List of technologies used by the application: ● *API* ● *PKI* ● *AES* ● *SHA-256* ● *SQL* Write **2-3 sentences** (40-60 words) that describe why you choose to prioritize that technology over the others.
III. Decompose application	Sample data flow diagram
IV. Threat analysis	List **2 types of threats** in the PASTA worksheet that are risks to the information being handled by the application. ● *What are the internal threats?* ● *What are the external threats?*
V. Vulnerability analysis	List **2 vulnerabilities** in the PASTA worksheet that could be exploited. ● *Could there be things wrong with the codebase?* ● *Could there be weaknesses in the database?* ● *Could there be flaws in the network?*
VI. Attack modeling	Sample attack tree diagram
VII. Risk analysis and impact	List **4 security controls** that you've learned about that can reduce risk.

Stage 1

Yes, the app will process transactions. Sneaker company wants to connect sellers and shoppers.
Enough backend processing so that the users have both a smooth and secure checkout process.
Yes, we are handling customer's SPII so all parts of the app that handle that must be in compliance with PCI DSS. SPII must also be in compliance with PIPEDA.

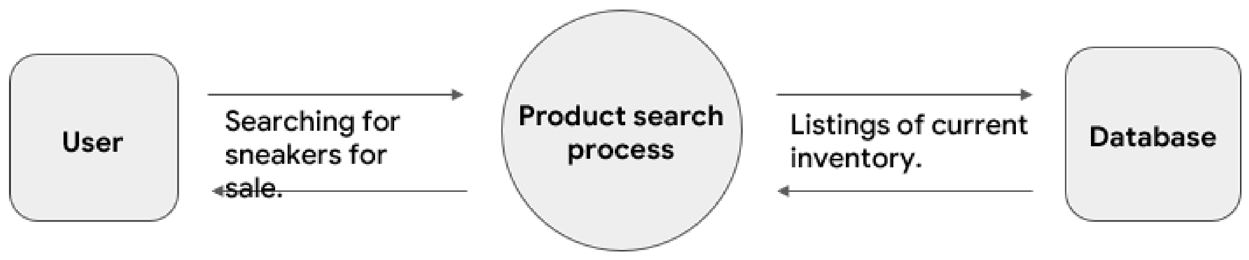
Stage 2

I would analyze the API first as it is the first point of contact with the user and potential threat actor. The biggest attack surface is here as there might be improper access controls, data leaks etc 2nd would be the SQL, and see if the app is vulnerable via the input to SQL injection, privilege separation and that the DB isn't publicly accessible. The remainder SHA-256, PKI, AES should be done in that order. If you want additional security, you could use SHA-4096 as the hash with salting and add in bcrypt, scrypt or Argon2 (preferred). SHA-256 could be used though SHA-4096 is better for Digital Signatures with ECDSA. For API key integration, it should be implemented as HMAC-SHA-4096 for authentication tokens and key signing. We can also use SHA-4096 for file integrity and file downloads. Use Argon2 for password hashing (SHA-family not enough due to modern GPUs and server farms). Check the Public-Key Infrastructure next for self-signed keys or expired certs or poor key storage policies. AES is generally fine unless put in ECB mode.

Stage 3

Data flow diagram

Note: This data flow diagram represents a single process. Data flow diagrams for an application like this are normally much more complex.


User
Searching for sneakers for sale.
Product search process
Listings of current inventory.
Database

Stage 4

Internal threats come from employees. If the employees mishandle user data by accessing or leaking it, or issuing unauthorized refunds or even just spying on a specific user's purchasing habits. Data exfiltration is very possible (screenshots, making copies of any database via SQL, abusing APIs especially unapproved APIs). Code injection, adding in a Backdoor for Developers. Credential sharing, poor offboarding (former employees still having access), employees performing internal fraud by applying discounts to friends or manipulating orders or spending company money in an unauthorized way. Abusing logging and monitoring
External threats can come from poor system design: SQL injection, Credential Stuffing, Broken authentication, XSS possibilities, MitM attacks, Credit Card skimming, Broken Access Controls, DDoS attempts (especially if successful), API abuse if API is poorly designed, Phishing attacks, Malware Injections

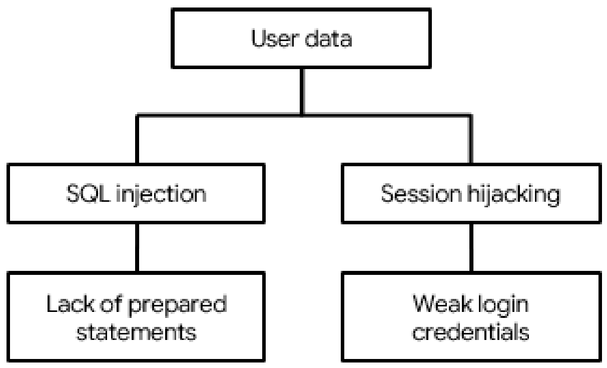
Stage 5

Listed above what vulnerabilities

Stage 6

Sample attack tree

Note: Applications like this normally have large, complex attack trees with many branches.



Stage 7

On the authentical and access control side, Strong Password Policy with MFA, lockouts, RBAC and principle of least privilege. For the input, we need to validate and sanitize it by using parametrized queries like using Python with psycopg2 and adding in the inputs as data rather than have it be interpreted as code:

```
cursor.execute(“SELECT * FROM users WHERE username = %s”,
(user_input,))
```

Using %s with parameters and the execute function, this allows proper escaping of “, ‘ and ; characters. The above code sends the SQL query and parameters separately to the PostgreSQL engine.
Secure Communication needs to be had. Use TLS everywhere to prevent MitM. Enforce HTTPS, and use Secure Cookies
Secure Storage: use Strong Hashing i.e. Argon2 for passwords, tokenize payment data, AES-256 is good for encrypting data at rest.
Monitoring and Logging: good for audits, detecting anomalies and has alerting+SIEM
Network Hardening: Web App Firewall implemented, Network Segmentation, Rate Limiting on APIs and verify all internal access with strict id checks
File Backups (encrypted), have incident response playbooks and security drills
Manage risk from any third party vendor (SaaS or PaaS) and evaluate them for compliance.