Implementation of Large-Scale Cloud-Based Systems

Author: Kamil Rafikov
Email: mailbox@kamil-rafik.com
WhatsApp: +996 755 439 777
Telegram: kamilrafikov

Updated at January 3rd, 2026.
Edition: 10 (final fact-checking not yet performed).

Table of Contents

Introduction

The only purpose of this document is to demonstrate for potential employers my knowledge of modern software development and skills in designing and managing development of large scale software systems.

All architectural designs presented further are just samples not related to projects implemented at my previous jobs. That's why they match relatively standard requirements only. No collection of business analytics has been performed for writing the designs. You should not and cannot use this document as for development of production stage software as for development of MVPs.

Design and development process

Approaches and practices

Planning

Depending on particular project the following documents may be need to be written on the planning stage with different level of details: a) general functional and technical specifications, b) low-level specifications of internal and external APIs, c) input/output formats of data for the largest functional blocks, d) list of possible bottlenecks and technical/organizational issues, e) logical and physical data storage schemas, f) mockups of UI and diagrams of UI workflow, g) general interaction/inheritance

diagram of modules and components with only main relationships rendered, h) numeric limits for different metrics of the system, i) development standards.

Also, the following may need to be considered: a) in some projects significant part of functionality may be encapsulated in standartized micromodules that may be developed independently by multiple low-skilled developers simultaneously (or even generated by AI); b) in addition to business-oriented part of functionality the large project may need significant amount of internal custom tools to perform utilitarian tasks (however, AI may simplify this part significantly too); c) active usage of design patterns terminology may cause development of unnecessary functionality, and may be a sign hiding lack of qualification in some developers; d) any change in UI/UX design in a completed part of the project may require up to 10x changes in code (just an oversimplification but it is good to know); that's why it is desired to stabilize UI/UX design as early as possible, or if such stabilization is impossible, it is desired to stabilize at least the ways how the changes will be introduced in the future to allow developers to automate these procedures.

Scrumban

Considering that requirements of managerial/financial team and possibilities of development/design team may mismatch often, I would propose the following *Scrumban* approach for development of complex projects having a target to develop new product for the market but not just support some business processes.

a) On initial project development stage which may take up to 1 calendar year where complexity of the system and connectivity between all its parts is not yet high for an experienced professional (certainly if development is made from the scratch), all teams in the business may use Scrum.

b) But then paths diverge. Development/design team performs all main work further with use of Kanban. Managerial/financial team chooses completed tasks which are well-tested on staging servers already, and then with use of Scrum performs minor fixes and subsequent release (minor fixes, additional testing, and release may be even performed by separate development/design team).

If the goal is just to support existing business processes then Kanban may be completely enough for all teams. If the whole structure of the project suggests low connectivity between micromodules and/or microservices then Scrum may be continued to be used always instead of Scrumban.

Such approach may not meet modern enterprise project management standards, but for relatively small organization with less than 50 workers where everyone knows everyone and projects are rather complex, it may be good enough.

TDD

Test-driven development is acceptable approach if at least one of the following conditions are met: a) there is rather high budget from the very beginning, b) managerial team has skills to control large team of mid-level developers (10-30 persons or more), instead of hiring fewer number of highly qualified

ones, c) mission-critical system is developed, d) project is in the mature development stage and all new development supports well-established business processes.

Open source community style

Many years ago I have read that some gaming companies (e.g. Valve) apply open source community development style to development of commercial products. All could change since those times, but this approach may be reviewed as acceptable too in some situations.

Different practices

Some of the following may need to be made in large complex projects in different amount depending on particular circumstances and stage of the project development.

- Defining new project-specific and product-specific metalanguage that consists of business domain terms and terminology taken from used technologies.
- Expecting political and business games that sometimes have rather destructive flare.
- Blurring managerial and other roles between several persons in the team instead of strict assignments.
- Balancing between detailed marketing researches and simple breakdowns of prioritized tasks/features.
- Documenting as system level design, as code, as hardware environment and configurations.
- Code reviews, sometimes multi-step reviews.
- Fixing quality standards as on paper as with use of automation tools.
- Tracking interchangeability of developers by assigning them appropriate tasks from different parts of the system.
- Paying much attention to design of the office if in-office development is performed.

Critical documentation

Only critical sections of documentation that are necessary in every project are highlighted here: a) local installation instruction, b) release instruction, c) vaulted list of credentials to used external system and internal administrator-level credentials in the system, d) main system requirements and metrics, e) instruction for basic manual testing of primary system functionality, f) history of the most critical emergencies and methods of their resolution, g) contacts of all key team members, h) standard instruction for restoring the system after failure, i) locations of all system logs.

Using AI for development

By the autumn of 2025th year I have not used yet AI actively for development. However, at October of 2025th I performed an experiment with ChatGPT 5 Plus with asking it for assistance while developing

an application with several tens classes on a technological stack that I'm not completely proficient in. Here are the approximate results from the point of view of experienced backend developer:

a) it gives very good responses on questions concerning usage and correct implementation of software design patterns; however, my opinion may be biased by the fact that in the most part of my previous projects the attention to correctness and active usage of design patterns was limited; so, upon 3-6 months of active development with extensive usage of all kinds of software design patterns, no necessity in AI for this purpose may be present;

b) it generates good draft versions of frontend components; obviously, they cannot be used as is, but they minimize work with documentation for a case if frontend stack is not well known (as it is in my case); I may use these draft versions as a general guidance and write final versions of components more quickly;

c) it provides good review of large modules consisting of several tens components (I uploaded one module in zipped format into the system); 10%-30% of review recommendations were reasonable and needed to be really applied;

d) for a case when I needed to configure from the scratch some complex "out of the box" functionality requiring configuration in about 10 places in .env file and code, it gives about 50% of correct answers; so, it is rather helpful too.

One co-worker told me that using AI in agent mode would give much more impressive results. However, I work primarily with old proprietary projects with not so good codebase and architecture, where some solutions may be non-trivial and require much attention. That's why I would not want the AI to interrupt my work and flow of thoughts so deeply. It may bring more harm than benefit in such situation (maybe not immediately but in long-term perspective).

Summary:

- I will definitely use AI as "smart search engine" to resolve some non-trivial architectural and technical issues or to recover in memory formal theoretical definitions for naming parts of code;
- I will definitely use AI as "draft code generator" for a case of working with weakly known or completely new stacks of technogies;
- and I will definitely use AI as "reviewer" before passing results of my work to human review.

All this should improve my performance and quality of work for 10%-25%, as I suppose.

However, I do not expect yet to use AI as "equal" developer-assistant during the next 1-3 years at least.

Hiring

Classifications of software developers

Classifications provided further are my own inventions, are based on my practical experience, and do not correspond to market buzzwords that may be met online. It may be impossible to classify hired person clearly if this person has experience of working in the business for more than 10 years. So, the provided classifications should be used as a general guidance, no more.

Classification by style of thinking:

a) “analysts” − usually having corporate or research/academic background (obviously, if they are older than 30), usually having more “employee psychology” than “entrepreneur psychology”, more reliable and useful on long-term large projects;

b) “creators” − having very “fashionable” skillset sometimes, preferring to operate as subcontractors but not as employees, having good portfolio often, may be much better in writing code than other types of developers;

c) “engineers” − having background in DevOps, system administration, and industrial engineering, having experience in low-level software development sometimes, having experience in working in complex mission-critical projects sometimes, having “exotic” or somewhat outdated skillset sometimes.

Classification by project size/length that can be handled:

a) “sprinters” − these people can work on single project for no more than several months (obviously, they concentrate on simple projects primarily and may be moved out of the job market by AI relatively soon);

b) “stayers” − these people can work on single project up to several years (either full-time or part-time);

c) “permanent workers” − these people can work on single project for many years full-time.

It is easy to guess that “permanent workers” prefer to work in well-established businesses. So, startup or studio business should practically rely on the first two types of developers, and expect full rotation of the team even once per year sometimes (which is definitely not pleasant and requires special attention to the project knowledge base).

Filtering of candidates

The provided approach may be considered as the most reasonable one basing on my practical experience.

a) Check quality of CV writing (just as a basic test of ability to express thoughts clearly).

b) Check that CV contains something from the following list: educational awards, degree in Computer Science (degreees in Physics, Mathematics, Life Sciences, Engineering are acceptable too), knowledge of several foreign languages, participation in open source projects (projects must be reviewed if present), online commercial portfolio (must be reviewed too if present).

c) Check that CV contains all necessary technologies used in the projects or their analogs.

d) Talk with candidate and ask about numeric metrics/limits of the developed projects.

e) Technical interview may be not necessary, but if it is preferred to organize a lot of meetings to discuss ongoing tasks then such interview may be desired to understand communicative skills.

f) Ask the candidate to perform code analysis test task with sending to the candidate a file with several hundreds lines of “dirty” code.

If developers of type “engineer” described above are hired then formal educational degree, good references, background check, and technical interview are obligatory.

Implementation of general functionality

Workflow

The following specifications are written with assumption that development will be performed in multiple stages by small distributed team working for strictly limited budget.

Limits

It is supposed that projects written with use of the provided specifications will be scaled to maximum possible audience globally, and must support extremely large data flows from millions of users with minimal latency.

Technologies

Choice of technologies is dictated by widespread usage and broad range of fields they may be applied to.

- Either PHP or Python (or both) may be used as the primary tools to build backend part of every project. Laravel is preferred as PHP framework, and Django is preferred as Python framework. (Node.js and Go may be used too for some performance sensitive tasks.)
- General purpose data management may be handled by MySQL, PostgreSQL, Redis, ClickHouse, and object storage for files.
- Specialized data management may be handled by TimescaleDB (for timeseries data), PostGIS (for spatial data), Pgvector (for vector data), AGE (for graph data), Elasticsearch etc. Choice is

based on relation of the most part of the mentioned tools to PostgreSQL to avoid zoo of technologies. Obviously, more specialized tools may be used too.

- Events management may be handled by Kafka or RabbitMQ.
- Opentelemetry is recommended to be used for collecting technical data on system behavior, and in some cases as lower level of data collection for business analytics.
- React, React Native, and TypeScript are the preferred choices for building browser-running and mobile-running frontends. (I'm not specialized in frontend development, so I could miss some modern trends in this field, and consultation with frontend developer is necessary.)
- Docker may be used for containerization for local development. However, its well-known security issues may require to search another containerization solution for production-stage deployment (certainly, if you have not DevOps specialized on resolving such issues).
- Kubernetes may be used for handling deployment.
- In addition to standard standalone web-servers (e.g. Nginx), some web-server technologies deeply integrated into the backend language infrastructure may be used for maximum performance (e.g. Swoole). Also, in a cloud environment it is possible to migrate some functionality to a serverless architecture.
- Choice of CI/CD tools depends on used repository service (e.g GitHub Actions, GitLab CI).
- CDN may be used to improve usability if the geographically distributed auduence is targeted.

Architecture

Development of backend software may be performed in several steps (if your budget is very limited):

a) monolithic application with low connectivity between modules; each module operates with subset of data models in storages and does not access data models of other modules directly; each module provides service class with API for being accessed by other modules; common model classes will not be implemented; at this step all software may be hosted on single server without much necessity of cloud infrastructure;

b) at the next step modules are extracted from monolithic application and encapsulated into separate microservices; at this step complete migration to cloud is necessary;

   - in the beginning, for simplicity, the "knowledge" of API of every service may be encapsulated into common connections library which is hosted in separate repository and included as dependency into every microservice; interaction between services will be performed synchrounously as with REST APIs as in procedural style with use of appropriate libraries;
   - optionally, API gateway service may be implemented to handle all interaction between microservices and provide for developers and DevOps central monitoring point of all interactions in the system;

- optionally, in case if interaction between modules/microservices is not unidirectional and supposes high latencies, a common event bus may be implemented where every module/microservice will put and collect events with data for processing; in such case separate monitoring module may need to be developed for administrator area of the system to track state of the bus and results of operations; results of operations also may be implemented in the format of events, and modules/microservices should handle them independently and asynchronously;
- optionally, in case of large number of microservices and different versions of microservices, some standard service discovery tool (e.g. Consul) may be used instead of storing settings in configuration files;

c) as on step A as on step B particular pieces of code which represent minor overcomplex parts of functionality and/or updated much more often than main code of microservices may be extracted into FaaS infrastructure (e.g. AWS Lambda).

Requests to external services and providing resources to external services should be implemented in asynchrounous style to parallelize execution.

Only the main approaches to interaction between microservices are listed in the step B, all others look like too exotic and unnecessary for the most part of projects.

Obviously, depending on business tasks you may go in opposite direction: to start from development of separate microservices, and unite them into the single system at later stages of the project.

Storage

From the very beginning of the project each domain area of the system may be given separate database or set of databases. Standard domain areas requiring separate databases may be specified as the following: analytics, logging, descriptions of objects in object storage, access control, finances, general business logic, customer-oriented business logic, provider-oriented business logic.

If project is targeted at the audience with uneven geographic distribution then partitioning/sharding of data by geographic criteria may be necessary. If project provides data that are accessed unevenly depending on datetime markers then datetime-based partitioning may be applied.

Data of every logical entity belonging to different types (number, text etc) and having different read/write scenarios may be distributed across different physical tables for maximum optimization.

Extremely frequently accessed data may be placed in key-value in-memory storages.

Replication should be used to keep availability of databases for emergency cases.

All stored customer-related data (including data in logs) should be easily identified via unique GDPR marker (which may be just internal customer ID) related to customer account to have ability to find and erase them from the system completely, including all caches.

All externally available data identifiers should be non-sequential for security reasons.

Analytics

Collection of analytics in all projects may need to be performed for several purposes: a) technical monitoring data that are necessary to debug software; b) business intelligence data that are necessary to analyze usage of the system and make business decisions; c) reporting for end-users who use the system for providing their services. Analytics logic for every purpose may be encapsulated into separate module, and finally, keep data of every module in separate set of databases. However, all modules may use common set of libraries and common approach to break development by stages.

a) Initially all analytics may be implemented through usage of external software for text logs analysis and periodical generation of summary report tables in scheduled jobs.

b) At the next stage generation of events may be implemented on all levels of the system, combination of offline and online data analysis may be used.

c) Complexity of data structure of collected events may be raised gradually, and include the following in the final version: type, timestamp, location in code, type and ID of performed operation, operation data, all actors involved into operation, result of operation.

Security and GDPR must be considered at all steps of analytics development.

Logging

Logging may be implemented in different way on each stage of project development:

a) log all to files that are automatically rotated and zipped either by framework tools or by system level tools;

b) output logged data into stream that is handled by centralized cloud infrastructure, so that all logged data may be queried later through built-in cloud tools; optionally, for cost optimization, output logged data into ClickHouse databases; also, for cost optimization, different storage/rotation rules may be applied to different levels/types of log messages (obviously, amount of informational messages will be overwhelming, but for financial modules even they must be stored for lengthy standard-specified periods of time);

c) implement structured business oriented logging where logged messages are represented by complex data structures and may be interconnected within single business transaction/operation (so, for example, multiple complex messages logged from multiple classes/methods may be analyzed as single data unit); partially, it intersects with tasks of analytics modules, implementation depends on how development/business processes are built and distributed across organization.

Security and GDPR must be considered at all steps of logging development.

Scheduled jobs

Each module will have its own set of scheduled jobs. On initial stage scheduled jobs will be implemented with use of frameworks' built-in functionality. Later, migration to Kubernetes CronJobs may be performed upon necessity. Critical jobs must be marked in code appropriately with providing results of execution to monitoring module in administrative area of the system.

Access control

Access control to different system machines and services should be based on private/public pair of key unique for every developer. Revocation of the access should be made by DevOps with a single click that deletes this key from access control service. Moreover, all such access should be made possible within corporate VPN only. It is complete responsibility of DevOps to configure this part of the system and to document it appropriately.

Access control for administrative area and service providers area of the systems may be implemented as a combination of two-factor authentication with use of authenticator app on smartphone, one-time passwords for initial login, and automated password recovery workflow for critical curcumstances. External or custom service for risk-based authentication may be used.

Access control for customers may be implemented as a separate module that allows login with multiple standard external identification providers (like mass email services, popular messengers, and phones). Implementing custom password management functionality is not recommended, one-time login codes sent to external identification providers are enough.

Security keys stored in databases should be rotated automatically every N days depending on account type. Failed logins should be audited.

On initial stage RBAC may be completely enough to satisfy business requiremets. Later, migration to ABAC may need to be performed.

Financial functionality

Financial part of system functionality should meet the following requirements: a) separate logging storage, b) structured logging that keeps data for every step in multi-step operation, c) ability to rollback multistep operation and mass operations at once (either via custom command-line tool or in UI), d) notifications for administrators on non-standard situations.

All this is necessary to avoid negative end-user experience and conflicts with managerial/financial teams due to inevitable technical issues.

Settings

Settings management can be developed with several iterations: a) initially all is configured in .env file and Docker configuration files, b) later, settings may be moved into cloud infrastructure and/or

Kubernetes, c) upon raising complexity of the system, implementation of feature flags service may be necessary.

Console commands

Every console commands modifying data should use a) standard option to be executed in dry mode, b) standard location for output log that is written on every execution (this log may be kept for N days), c) transactional approach with automated rollback on failure. It is desired to develop template of such command from the very beginning of the project.

QA

In case if highly qualified development team is hired and the project is not mission-critical then only smoke tests and unit tests for the most important parts of functionality (20% − 50% of the system) may be necessary. In microservice environment, contract testing (including versioning) may be also necessary. Full scale testing is needed upon reaching some maturity by the projects only.

Containerization

If Docker is used for containerization then every separate application/service in the system should be configured with several Docker Compose configuration overrides for development, staging, and production environments. Configurations should override each other with maximum reusability. Separate shell scripts should be written to build, start, and stop containers with simple commands. (Such approach may be outdated for enterprise environment, and all this should be managed completely by Kubernetes tools. However, on initial stages of project development, before integration with Kubernetes, it may be completely enough.)

To minimize amount of debugging on production servers all Docker configuration for multi-service system should allow all services to be started in development environment and be connected to IDE debugger. Such configuration may require common IDE project configuration that may be downloaded from separate repository and imported locally by every developer in the team.

You should note that Docker has multiple security issues and for mission-critical projects you should consider other alternatives (that may be compatible with Docker and Docker Compose on configuration files formats level).

Deployment

Even after completion of automation of deployment with Kubernetes the main developers of the project should be able to login directly into the databases and run console commands in production environment manually. (Yes, it creates high security risks, but may allow to resolve critical issues quickly.) Logging functionality and local debugging configuration should be fully completed before

migration to Kubernetes to minimize such intrusion into production environment for debugging and hotfixing.

Scaling and load balancing

On initial stage of project development you may use Docker Compose for scaling the system with running any number of microservice instances. Load balancing between instances may be implemented either with use of Nginx as reverse proxy or with use of API gateway tools.

Later, migration to native cloud tools may be performed.

Backup

Backup may be made in according to the following principles:

a) all databases may need daily backup for the latest N (7-30) days; these backups may be rotated with deletion of those that are older than N days; backup should be made into storage that allows immediate restore;

b) every N days long-term (up to M years) backup of all databases may be made into low-cost storage that does not allow immediate restore;

c) object storages may have less strict backup requirements, but it is project-specific;

d) financial logs and logs from manually executed commands may be backed up immediately to low-cost storage that does not allow immediate restore for number of years specified in standards.

All backups must be obligatory verified for integrity, sensitive backups must be encrypted.

Handling large-scale mature projects

In this section I would like to highlight some aspects that may be important if you are performing enhancements of someone's large-scale mature project, or if your own project has grown up already. By "large-scale" and "mature" I mean the project with the large technical debt, built on the zoo of technologies, and consisting of several hundreds microservices, frontends, mobile apps, data storages etc.

a) Migrate as much as possible to asynchronous communication between system components. Minimize synchronous operations to critical tasks only and limit them with one level call.

b) Minimize hard writes to data storages as much as possible, keep it for critical operations only.

c) Separate reading and writing to allow independent optimization and scaling.

d) Use materialized views for reading complex sets of data.

e) Apply real-time OLAP systems to your data infrastructure.

f) If your data sources/storages are very diverse applying data lake level above all of them may be beneficial.

g) Apply service mesh layer to all your microservices.

h) Improve communication between microservices with such approaches as avoiding repeated calls to failed microservices, implementing specialized algorithms for repeated calls, isolating failed parts of the system from others to avoid total collapse.

i) Implement zero-trust networking policy.

j) Migrate secure operations into trusted execution environments.

k) Apply automated terraforming approach to all your cloud environments.

l) Consider using automated compliance check if such compliance is necessary in the system.

m) Standartize tracing of all operations in the system.

n) In addition to standard test coverage, you may apply predictive QA trained on logs data and artificial simulation of failures.

o) Migrate to trunk-based development approach for building features that should be delivered immediately.

p) Implement progressive delivery with feature flags to minimize influence of new releases on working parts of the system.

q) Implement not only single local development environment in Docker but multiple ones for different versions of the system.

Examples of system design

Basic examples of system design are provided just to highlight deeper my approach to implementation of modern applications. Breakdown into modules and entities is not performed because it is usually extremely simple, only general analysis of each case is given.

Tours booking system

Requirements

- Providers of tours enter into the system information about tours, including locations, events, variable options, costs, and amount of available tickets.
- Customers browse tours, compile desired set of locations, events, options, tickets and perform purchase.
- The system charges commission from all sales.

Particularities

Number of providers is relatively small and can be handled easily. Information about provided tours is relatively static. Only variable options, dates, and amount of available tickets are changed regularly.

For low-cost tours the system reserves all charged costs in payment gateway until getting confirmation of performed tour services from provider. Final charges and payouts are performed only after that.

Idempotency keys may be necessary for multiple operations.

Integrations with payment gateway service, geolocation service, maps service are necessary. External or custom fraud detection system may be necessary.

Real-time notification exchange module between providers, customers, and administrators is necessary.

Financial compliance standards must be met.

Possible issues

During the process of compiling a tour the system should continuously provide feedback to customer about availability of tickets in chosen tour component to avoid spending customer’s time for selecting components with all sold tickets already. Real-time notifications with WebSockets may be used for this purpose.

At the moment of putting compiled tour into the cart the limited number of tickets in every tour component must be recalculated and locked, so there will be races between customers. Key-value in-memory storages (e.g. Redis) may be used for this purpose.

However, as in all similar systems minor overbooking may be allowed which will be later resolved with technical support and providers.

Seasonal peaks may increase system load by multiple times in comparison with standard load. Preliminary load testing is obligatory for such situation. Automated scaling and load balancing with standard cloud tools is necessary.

Field worker AI assistant

Requirements

- Field workers service complex technological objects with using attached head cameras and microphones.
- Videostreams from cameras (including audio track) are sent to the system for AI analysis.
- Upon completion of real time analysis recommendations from AI in audio format are sent to field workers and played while they perform their service work.

Particularities

Significant amount of model training may be necessary before system will be used in production mode. Up to several tens thousand cases.

Final output from the system should never come to field workers directly to avoid disturbing them and desinforming them. Separate person qualified for interpreting AI recommendations should observe situation from outside, receive recommendations, and retell them to field workers, if necessary.

Government security compliance standards must be met.

Possible issues

The most resource consuming part of the system is real-time analysis of videostreams to parse them into text, generate recommendation, and convert recommendations into audio format. However, if it will be done in external AI system then no performance issues should arise. OpenAI and Llama provide such functionality already.

Also, large object storage must be used to keep all videodata obtained from field workers. Strict rules for backup should be implemented considering nature of collected videodata.

Technology is very fresh by 2025th year, so results in quality and latency are difficult to predict. The whole project may be classified as experimental one, may go out of the budget easily, and completion may be postponed up to several years. If custom model will be trained instead of using external services then project becomes even more experimental.

In practice, it is recommended to contact sales managers of companies providing video analysis models and obtain from them realistic estimates of quality of results. Also, considering nature of the project and amount of data for training it may be possible to obtain significant discounts from sales managers.

Self-learning AI assistant

Requirements

- Customer provides for the system planned reading lists and updates on progress.
- The system analyzes customer’s data, discusses with the customer the ideas taken from read material, interviews the customer to check whether materials have been absorbed fully, and provides for the customer recommendations on subsequent reading and purchasing goods/services.

Particularities

Significant amount of model training may be needed with several thousands categories of literature and similar amount of combination of purposes of self-learning. However, upon completion of training the model with test users, it will be possible to build several hundreds “portraits” of customers that further may be used to simplify/speed up communication and recommendations.

Catalog categories should support as tree-like structures as tagging.

Integration with reading list services and ebook provider platforms are desired.

Integration with external advertising management system is necessary to obtain contextual recommendations.

Possible issues

No bottlenecks or other issues are expected in the system.

Aggregator of small scale sellers

Requirements

- Small scale sellers register in the system and place there information on sold products.
- Customers perform search of sold products by multiple criteria and purchase products.
- System charges commission from sales.

Particularities

Catalog categories should support as tree-like structures as tagging.

System reserves charged amounts in payment gateway and performs final payout only upon getting information on completed delivery.

Search engine should be used to simplify search by multiple criteria (e.g. Elasticsearch).

Rating/review module for seller may be necessary.

Idempotency keys may be necessary for multiple operations.

Integrations with payment gateway service, geolocation service, maps service are necessary. External or custom fraud detection system may be necessary.

Real-time notification exchange module between sellers, customers, and administrators is necessary.

Financial compliance standards must be met.

Possible issues

Races between customers for remaining amounts of products may be possible. However, overselling is acceptable because delivery is not immediate and sellers may produce/obtain necessary products within reasonable amount of time. Also, order may be cancelled in communication with technical support and seller upon completion. So, no performance issues are present.

Augmented reality recommendations system

Requirements

- Providers place information on locations, services, and events into the system.
- The system builds customers' profiles upon registration and upon analysis of changes of customer's locations. Basing on these profiles the recommendations are generated.
- Customers moving through the area obtain aforementioned recommendations in real time. Information is displayed via augmented reality device. Recommendations may include paid advertisements.

Particularities

Number of providers is relatively small and information on locations, services, and events is relatively static. So, all these data may be handled very easily.

In opposite, number of customers may be very high and amount of geolocations data obtained from customers may be extremely large.

Preliminary breakdown of serviced territory into blocks of several tens meters width/length, matching all providers' data to these blocks, and predicting visited blocks by analyzing previous customers' behavior may be necessary for maximum optimization.

Locations categories should support as tree-like structures as tagging.

Integrations with geolocation service, maps service, external advertisement service are necessary.

Specialized low-latency spatial data management system may be necessary.

Possible issues

Real-time rebuilding of customer profiles and real-time update of recommendations may be the largest bottleneck. However, high load will be on initial stage of collecting data only which can be done with several thousands test users receiving fee for participation. Upon analysis of the collected test data it will be possible to build several hundreds typical "portraits" of customers, and later perform gradual matching of new customers to these "portraits". So, every customer will immediately receive large amount of relatively appropriate recommendations that will be later polished only. Obviously, amount of "portraits" may be extended later.

Summary

I hope this document demonstrates my capability to lead complex large-scale cloud-based projects during the whole lifecycle, I hope you enjoyed reading, and I expect long-term productive work with you. Thank you for spending your time!