Conception Phase - Task 1

The primary goal of this project is to develop a high-performance data system that provides endpoints for ingesting new data into the storage system and publicly available endpoints that will be used by front-ends to access the stored data. The system-design aims to meet the following quality expectations: Efficiency, flexibility, scalability, reliability, maintainability and robustness.
To meet the quality expectations, the system needs to scale horizontally. This will ensure increasing data loads and user requests will not deteriorate request latency and system performance. For this purpose, the chosen framework and programming language is .NET in combination with C#, which has not only high performance but additionally is optimized for publicly facing endpoints and containerization. To increase maintainability, it enforces a clean-code structure as well as coding-schemes, which will ensure coding efficiency and code maintainability by its simplistic coding style. For transmission efficiency, incoming data will be forwarded to gRPC endpoints. RPC creates a reliable and robust data stream which is already structured and serializable. The database of choice is PostgreSQL. It offers an official Docker-image, node sharding and extensive support for NoSQL-like behavior through a special row type JSONB (binary JSON). Since PostgreSQL is an open source, ACID-compatible, on-premise database, it is widely used and almost all cloud providers offer PostgreSQL-nodes. This creates a lot of flexibility and no upfront costs or dependence on cloud providers for test-integrations or prototyping. A relational database with support for semi-structured data, in line with the chosen dataset, offers greater value than document-based-database systems (e.g. MongoDB, ArangoDB) or key-value-stores (e.g. Redis).

The programming paradigm pans out to Extreme Programming (XP), since it greatly scales down to small teams or single seats. At first, tickets with all the supposed features will be created. In the development phase, they will be sorted by priority and then worked on until the feature reaches the predefined acceptance criteria. After reaching acceptance, the feature should be secured by adding unit-tests and increase test coverage until the outer acceptance will be reached. Documentation will be auto-generated from protobuf files and code comments after reaching feature completeness.