**Functional Requirements:**

* Login / user creation / credit card application
* Handle Credit Card Payment Processing
* Post Transactions / View Card Balance / View Transaction History
* View Daily, Weekly, Monthly Stats

**Nonfunctional Requirements**

* Availability
* Security
* Compliance with regulations
* Reliability
* Scalability
* Eventual Consistency
* Performance (low latency)
* Idempotent operations (even if an operation is sent more than once, it is only applied once)

**APIs:**

* User Creation API (username, password, email, phone, authentication data (ssn, ccn, dob, ln)
  + Process data, check data storage for authentication data to match with user
* Credit Card Application (Full Legal Name, SSN, Address, Employment Status, income, dob)
  + Process data, send to external company for credit score, approve or deny, set credit amount, email verification, send card
* Login (username, password, security question, 2 factor authentication)
  + Check credentials in storage, if valid, create session ID.
* Handle Credit Card Processing – Receive payment info from processor, decode message, check user/cc in database, return valid or not, store information
* Post Transactions, view card balance, view transaction history – Data storage to cache, pulled to user interface
* View Daily, Weekly, Monthly Stats – Data to Queue to ETL to get enriched data set, pushed to data warehouse, pulled to cache.

**Database:**

* For account data use relational database because of ACID properties, easier to store, update and retrieve relational data, does not scale as well to lots of requests but for user data, the number of requests will be more contained (will not be writing to this database super frequently), may read more frequently but this can be handled by having more read machines and fewer write machines.
* For transaction data, we also use a relational database for ACID properties and easy update, and data retrieval. Will have more write servers and fewer read servers because the reads will be less frequent. Can set time to live on data and delete after 3 months. Meanwhile, we pass data through ETL and into data lake offline. Can also have a cache to get data to user faster.
* Can use data warehouse like snowflake because it guarantees ACID, it works with SQL, it scales easily when required, and it’s designed for easy data analysis and machine learning, etc.
* ACID – Atomicity (transaction w/ multiple operations is all or nothing), Consistency (transactions bring database from one valid state to another), Isolation (concurrent operations results are the same as if they were done in correct order), Durability (Once a transaction is committed, it remans committed no matter what)

**Features:**

* Load balancers, web servers, application servers, relational databases, data lakes, Sequencers, Distributed Queue, Cache, ETL-AWS Glue, encoding/decoding, TCP/TLS, Pub-Sub System

**Data Schema:**

* user\_id, username, password, first name, last name, email, phone, SSN, Security Questions, credit card number, DoB, Employment Status, Income
* UserId, transactionId, timestamp, amount, merchant

**Estimations:**

* 100 Mil cards in circulation
* 20 Mil transactions per day
* 20 Mil transactions / 24/60/60 approx. 250 qps
* 10 Mil account logins + new account + credit cards. 125 qps
* Our system should be able to handle about 400 qps
* AWS RDS allows 500 concurrent requests per region (8 regions)

A diagram of a software company

Description automatically generated