CTIS411 SENIOR PROJECT-I SOFTWARE DESIGN DESCRIPTION (SDD)

BilPay

Okyay Say

Team 4

Canberk Demirci

Altuğ Ankaralı

Fatih Yavuz

Yiğit Usta

Table of Contents

1 Design Strategy & Principles

Object Oriented Decomposition

When talking about decomposition in software context, two things come two minds. Functional decomposition and object oriented decomposition. In functional decomposition, steps of processes are divided into modules and functions, on the other hand, object oriented decomposition sees software as objects that model entities in domain.
The whole idea is to solve complex problems by breaking into smaller pieces.

For example, Bilpay Mobile App has a qr code scanner that parses a qr code and returns the object representation of qr which is a transaction. So, qrCodeScanner, qRCode and transaction are different objects.

Separation of Concerns

Before talking about separation of concerns, we must first answer the question, “what is a concern?”. A concern is a particular interest in other words focus in a software. The synonyms of concern are, feature, behaviour. Concern applies to both high and low levels. The official definition of separation of concerns is the process of breaking computer program into distinct features that overlap in functionality as little as possible.

In our case, we are using Swift in mobile app, PHP in the interface of blockchain and JavaScript for the Web SDK. All these support object creation and we separate the concerns into objects.
We are using MVC in mobile app to separate the content from presentation and data storage and processing from content. We are also using Service Oriented Architecture to separate concerns into services.

2 Processing & Control

Sending money will be handled in a distributed manner since the transactions are based on a blockchain.

RESTful API will work in a sequential manner, it will simply take the requests, communicate with the blockchain network if necessary, connect to database and data or make changes if necessary and respond back the client. All these processes will be done in a sequential manner. For example: Connection to the database and communication with the blockchain network will not happen concurrently, one will have to be completed before the other one can be started.

iOS app, Android App, and the web GUI for back-office will work asynchronously due to the nature of the programming languages they are written in. One task does not have to necessarily block another task from happening. They can occur at the same time.

3 Commercial Of-The-Shelf (COTS) Software

The BilPay Payment System comprises a mobile app to send money, view transaction history and make payments. It will also consist of a RESTful API server, blockchain nodes, SDK for integration with an e-commerce website and a web GUI for the back-office to handle management of the users.

The blockchain transactions will be handled by the **Stellar,** an open-source decentralized protocol for digital currency to flat currency transfering allowing transactions without geological limits between any currencies.

API server will run on an server using **Nginx** software.

The programming language for the RESTful API is **PHP**, **Laravel Framework** will be used for **CRUD** purposes. It will communicate with the **MySQL database**.

PHPUnit will be used to test modules of the RESTful API.

MySQL database will be used to store api token, email, hash of the password, name, and the public key of user's wallet. It will also hold information related to transactions, such as merchant's email, transaction creation date, transaction status (pending or received), transaction amount and the hash of the transaction in **Stellar** protocol.

Let's Encrypt certificate authority will be used for generate SSL certificates without any cost.

BitBucket will be used to store the git repositories, **git hooks** will be used for continuous deployment.

BilPay iOS app will be developed in **Swift, BilPay Android app** using **React Native.** Stellar protocol's related SDKs will be used in both.

BilPay SDK’s front-end implementation will be available as a **ReactJS** component which can be installed using **npm**, and it will be available as vanilla JavaScript through a CDN. **BilPay SDK’s back-end** implementation will be optional and it will be available in **npm** for development in **NodeJS,** and it will be available in **composer package manager** for development in **PHP**. BilPay SDK will consume endpoints of the **RESTful API**. Implementations in more languages and package managers might be rolled out later. **BilPay SDK** will be open-source and anyone can view it’s source code in GitHub. **BilPay SDK’s front-end** implementation will be tested using **Puppeteer** and **Jest**. **BilPay SDK’s back-end** implementation will be tested using **Mocha for NodeJS** and **PHPSpec for PHP**.

Back Office’s GUI will be written as a web application. It will use **ReactJS** and **Redux** to synchronise the state and the DOM components and, **Next.js** for server side rendering. Back Office’s GUI will also consume endpoints of the RESTful API.

4 Non-Functional & Quality Requirements

- The BilPay shall ensure that data of all users will be stored securely.

The passwords will not be stored in plain format, they will be hashed first and the hashes will be stored in the database. Each time a user tries to log in, the hash will be taken again and it will be compared against the database records.

- Passwords shall never be viewable at the point of entry or at any other time.

The passwords will appear as asterisks.

- The system shall be accessible from mobile and web user interfaces.

iOS and Android apps will allow users to access the system from mobile. Web GUI for the back-office will be responsive therefore it will also allow access from mobile and web user interfaces. BilPay SDK will also be available to mobile and web users because it will be written using web technologies.

- BilPay shall achieve 99.9% up time.

Blockchain will only be down if all the nodes in it are shutdown, since there are many nodes in the network, this won't be likely to happen. RESTful API server will provide 99.9% availability by leveraging the power of load balancers and back-up servers.

- If the system is non-operational, the system shall present a user with notification informing them that the system is unavailable.

The mobile applications and the web GUI will notify the users if they are unable to receive responses or receiving responses with 5xx status code from the RESTful API server or Blockchain.

- The BilPay shall be fully scalable for any technical and non-technical operations.

Scalability will simply be provided by application of load balancers and alternative servers.

- The BilPay shall ensure that users' national security numbers whether correct or not.

National security numbers will be verified by an algorithm to be used by the RESTful API Server.

- The initial system shall be able to handle the entry of payments by customers at a minimum rate of 15 second.
  Stellar protocol allows 1,000 transactions per second.

- Complete report summaries of the current business day’s payments shall be available one minute after transaction completion.
  Transaction data will be available in the database immediately. Therefore current business day’s transactions will be created inside the mobile app quickly.

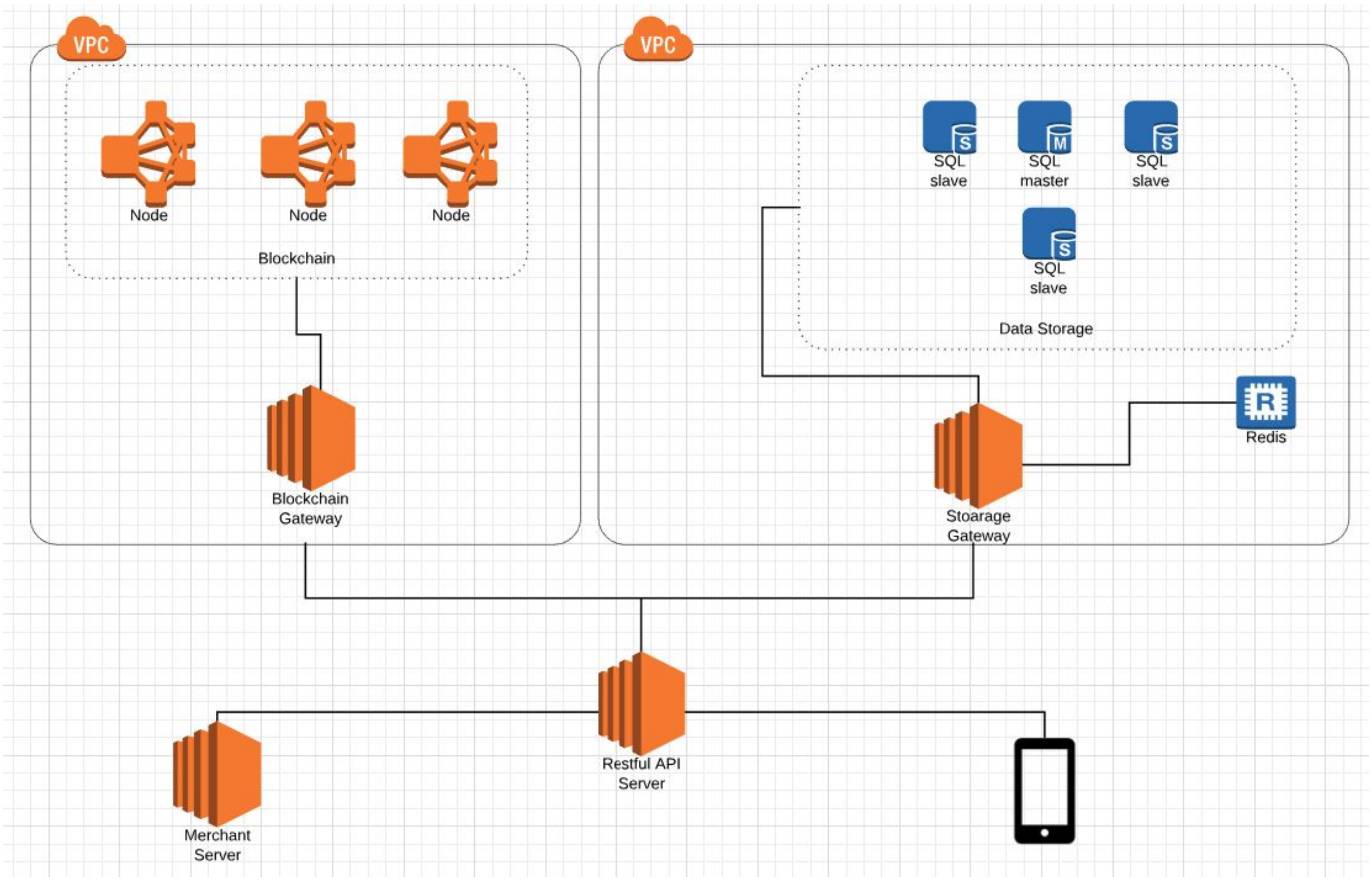
- Routine maintenance that is executed while users are active shall not cause a perceptible increase in response time for any function of more than 5% over the response time when no maintenance process is executing.
  Load balancers and back-up servers will allow us to do this.

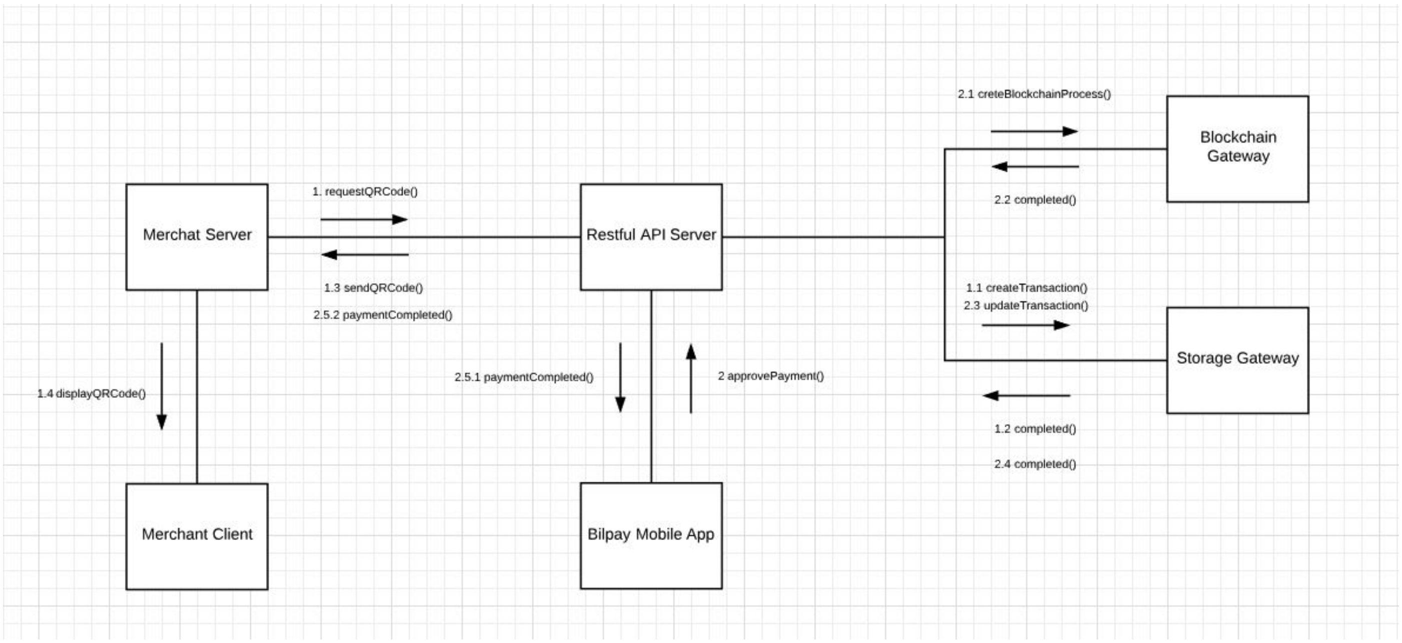
- The system shall prevent access to failed functions while providing access to all currently operational functions.
  The RESTful API server will only respond to requests with a 5xx for the failed functions.

- All payment transactions shall pass BilPay’s internal security validation before committing transaction update.
  Transactions will be validated using the receiver’s and sender’s public keys by checking them against the database.

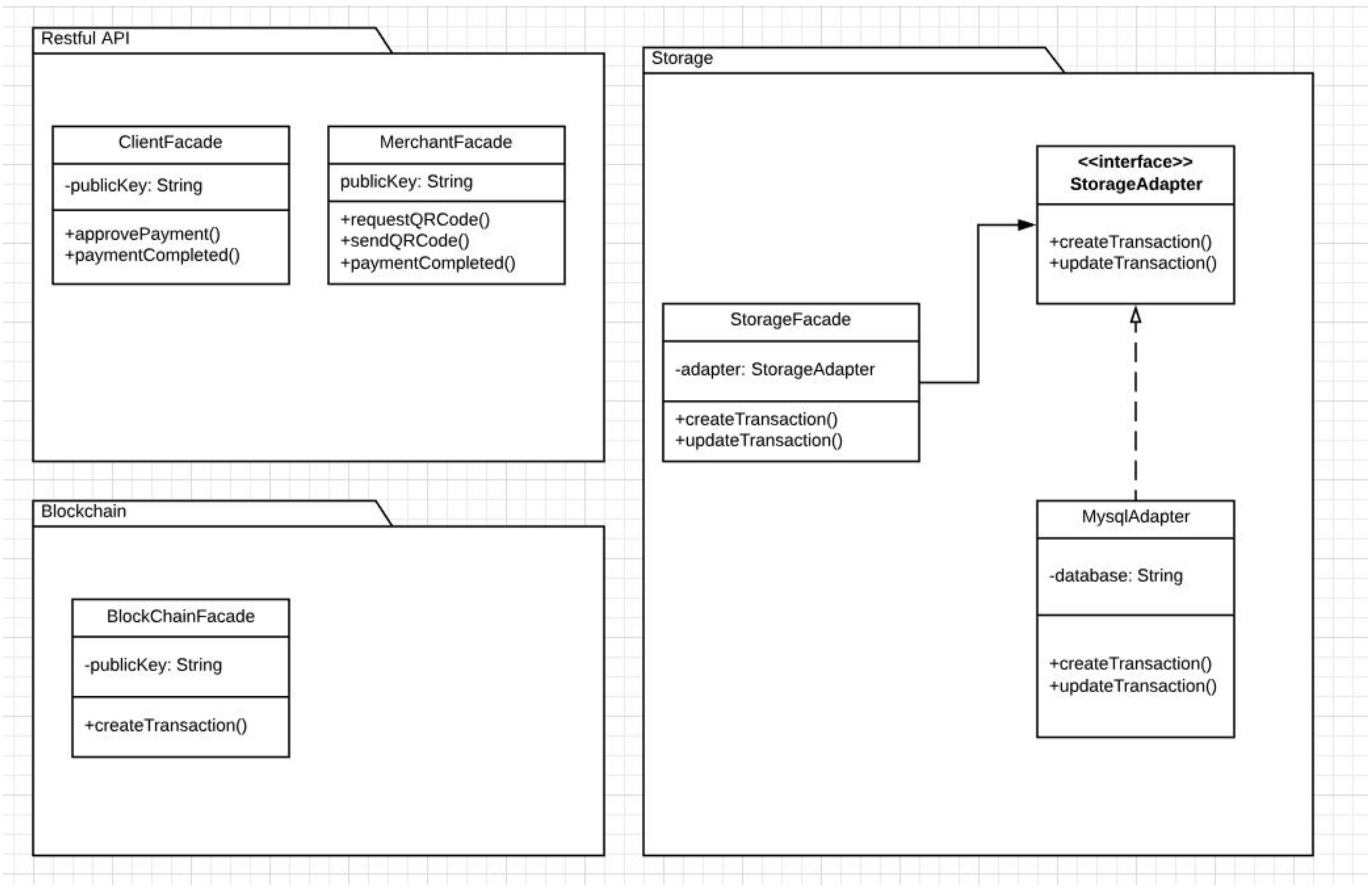
- QR Code payment transactions shall be created and validated by BilPay.
  BilPay SDK will request RESTful API Server to generate a transaction by using merchant’s e-mail, amount to be paid and merchant’s shop name. RESTful API server will respond with transaction details which contains the transaction id, then the BilPay SDK will start polling the RESTful API Server using the transaction id to obtain the transaction status. When the mobile app reads the QR Code, it will obtain the transaction id and request the RESTful API server to mark the transaction as completed, if the transaction is not found or completed already the validation will fail. If the transaction is found and the status is pending, it will be marked as received and a notification will be sent to merchant and also the payment will be shown as completed in the e-commerce website.

5 Software Architecture Design (High-Level Design)



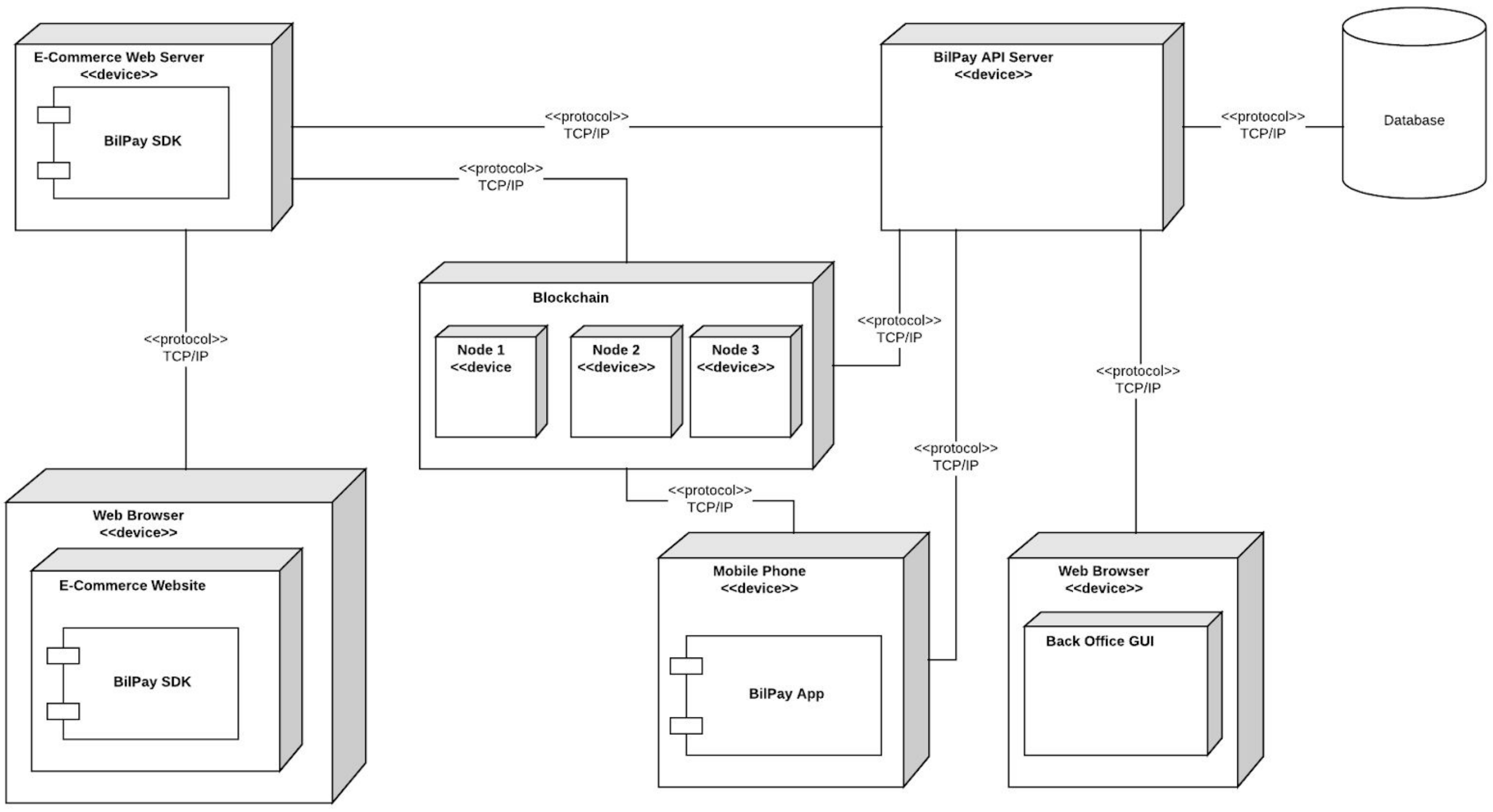


6 Subsystem Interface Design

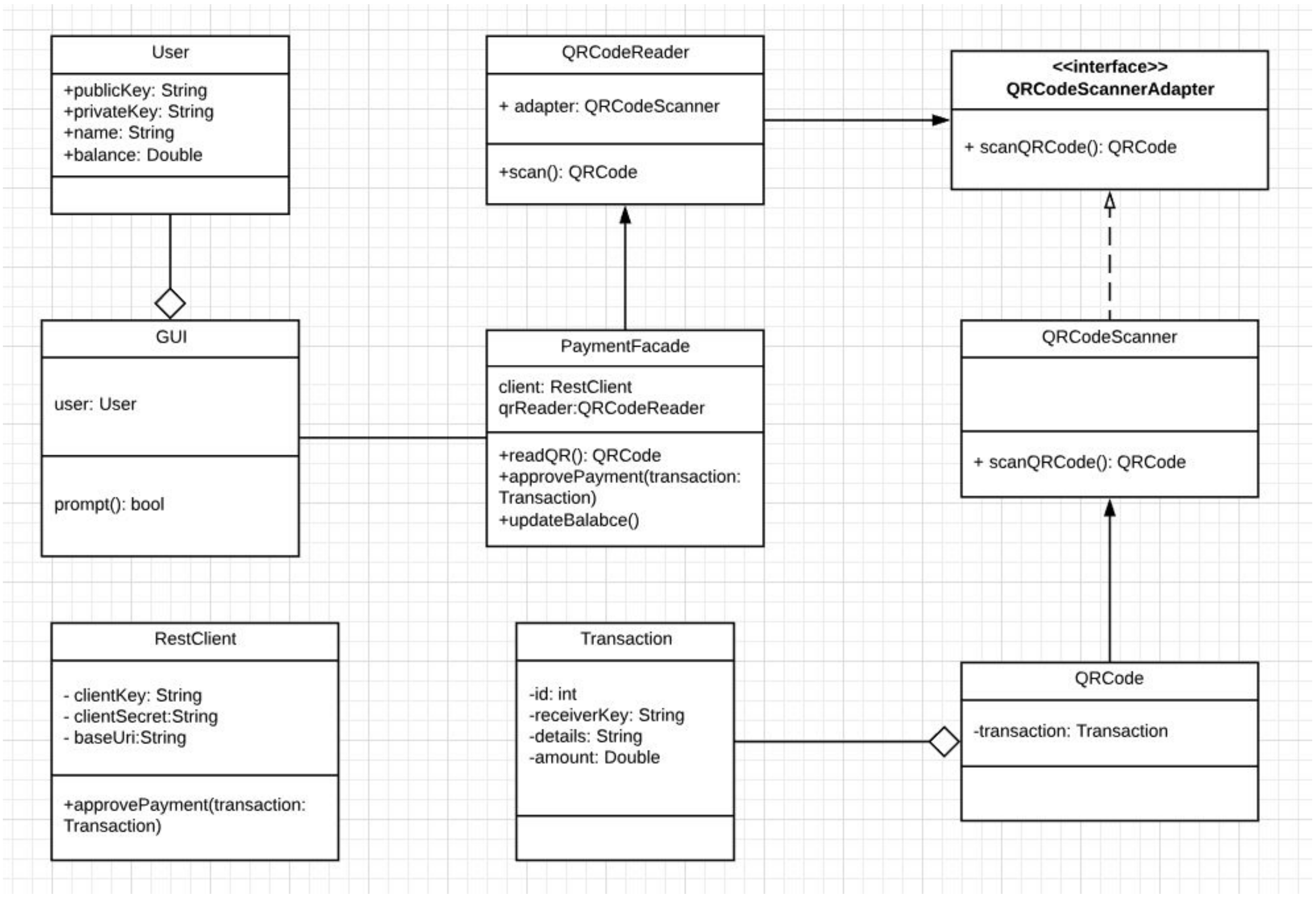


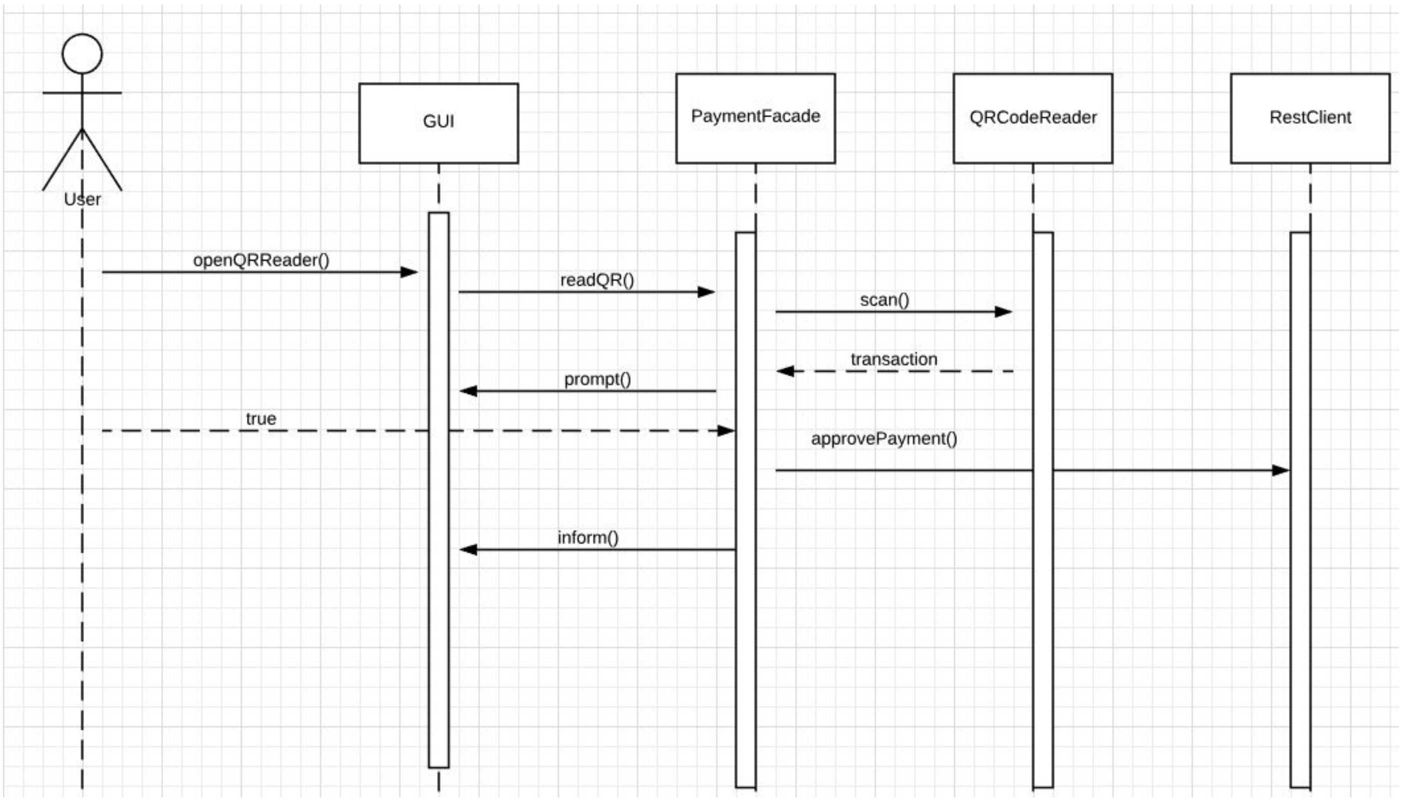
7 Physical Design

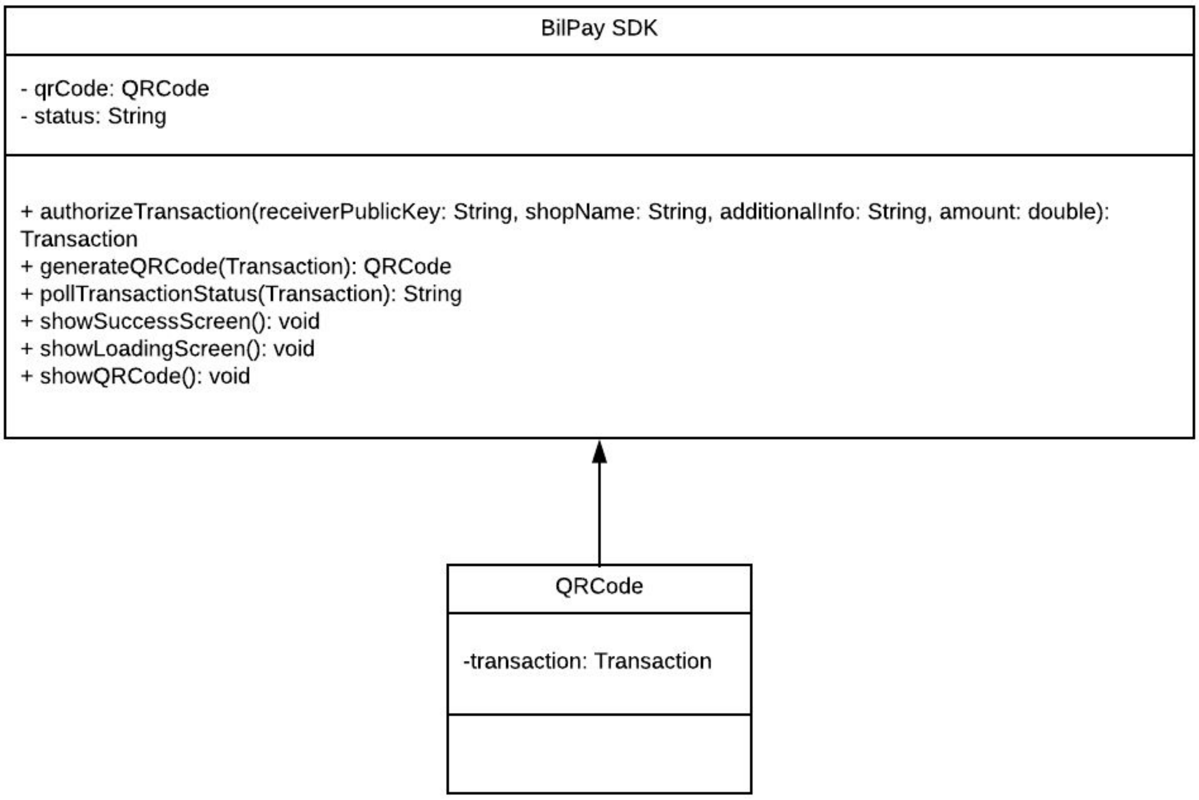
RESTful API will be deployed to a web server running Nginx.
Back-office's GUI will be deployed to a web server serving static files.
Database will be deployed to a Digital Ocean droplet.
BilPay SDK will be published in NPM and composer package managers and also will be available to download from GitHub. Merchant's e-commerce website will include **BilPay SDK** in it to communicate with BilPay API Server.
iOS app will be published in app store to download where as the **Android app** will be published in Google Play.
Blockchain nodes will be deployed using Kubernetes.

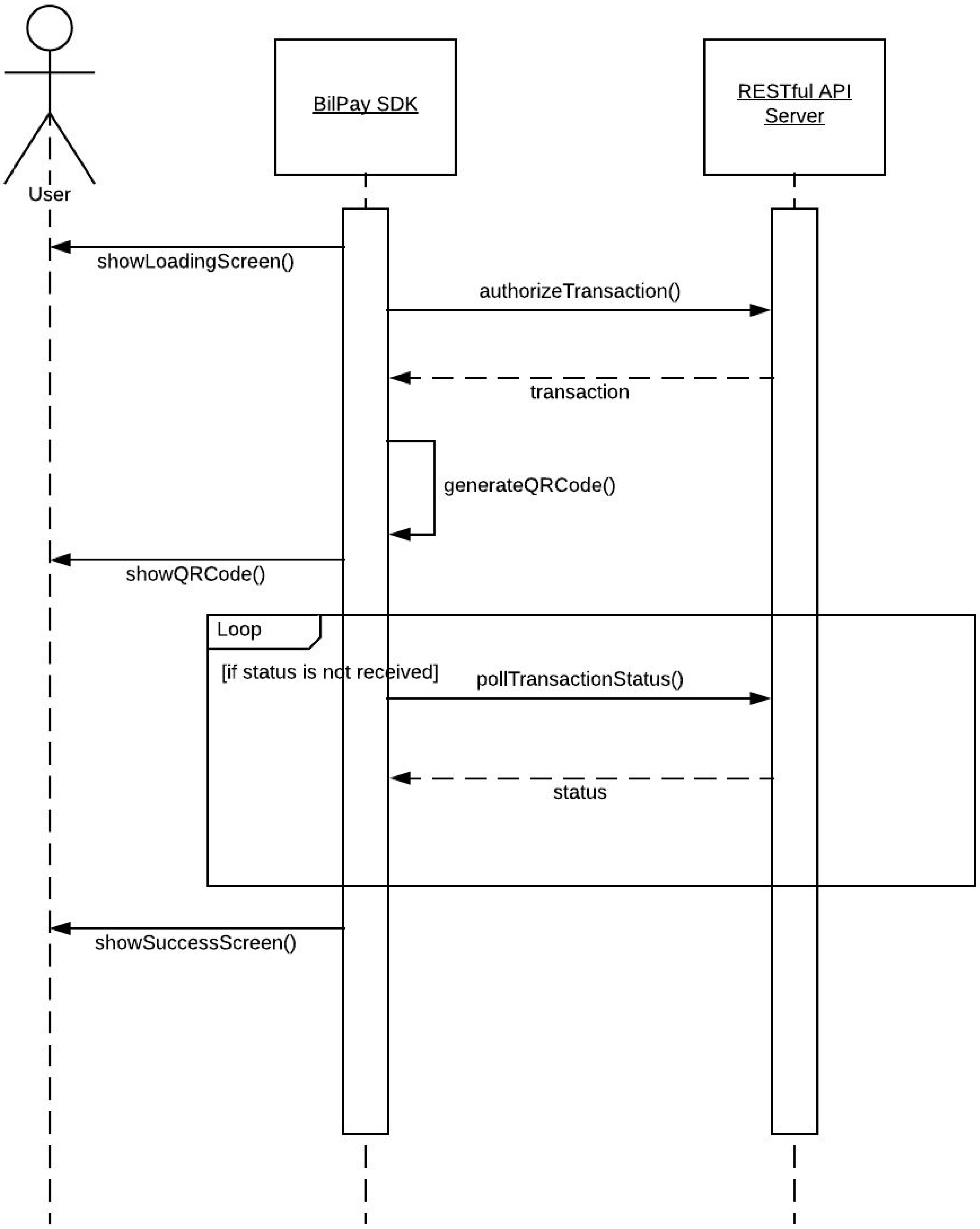


8 Subsystem Design (Low-Level Design, Object Design)




User
GUI
PaymentFacade
QRCodeReader
RestClient
openQRReader()
readQR()
scan()
transaction
prompt()
true
approvePayment()
inform()


BilPay SDK
- qrCode: QRCode
- status: String
+ authorizeTransaction(receiverPublicKey: String, shopName: String, additionalInfo: String, amount: double): Transaction
+ generateQRCode(Transaction): QRCode
+ pollTransactionStatus(Transaction): String
+ showSuccessScreen(): void
+ showLoadingScreen(): void
+ showQRCode(): void
QRCode
-transaction: Transaction


User
BilPay SDK
RESTful API Server
showLoadingScreen()
authorizeTransaction()
transaction
generateQRCode()
showQRCode()
Loop
[if status is not received]
pollTransactionStatus()
status
showSuccessScreen()