## 3.

### a.risks

Power outage, network failure (or overload) or hardware failure in general could put the server out of reach for the clients.

### b.non-Risks

Updating or switching out parts should be fairly simple as they are well separated from the rest.

Client/server architecture helps keeping sensitive information away from the user, improving security.

### c.sensitivity points

Connection to the payment module is a must for the shop to sell things. Must be reliable and secure.

### d.tradeoff points

To improve security the system is running on a server and clients rather than having everything on the client. Which is a tradeoff in performance from having to communicate with the server.

## 

## 4.

## a. Information hiding

Unnecessary information is kept away from the different clients through the different interfaces they have to the server.

## b. Minimize coupling

Coupling is low-ish. Database looks disconnected from the rest of the system.

## c. Coherence

Components that work togeather are grouped away from the rest.

## d. Divide and conquer

Architecture is divided into manageable parts.

## e. Separation of concerns

The different parts don’t make changes to things they have nothing to do with.

## f. Keep it simple

It’s decently simple, but there seems to be a bit extra clutter in the component diagram.

## g. No circular dependencies

No

## h. Layering

System is layered although “application logic layer” which is nothing but interfaces. Database and external services are separated from the server and the client.

## i. Modularity

Switching out components/groups of components or reusing them should not be a problem.

## 5.

8/10, It is well divided, coherent and with low coupling. the use-case diagram had a… peculiar layout.