Ticket Salad Requirements and design document

Contents

1 Software architecture overview

Figure 1 shows a high-level overview of the software architecture. In particular it shows the decomposition of the system into layers with abstract responsibilities, the core architectural components of the system and the concrete frameworks to be used when realizing these architectural components.

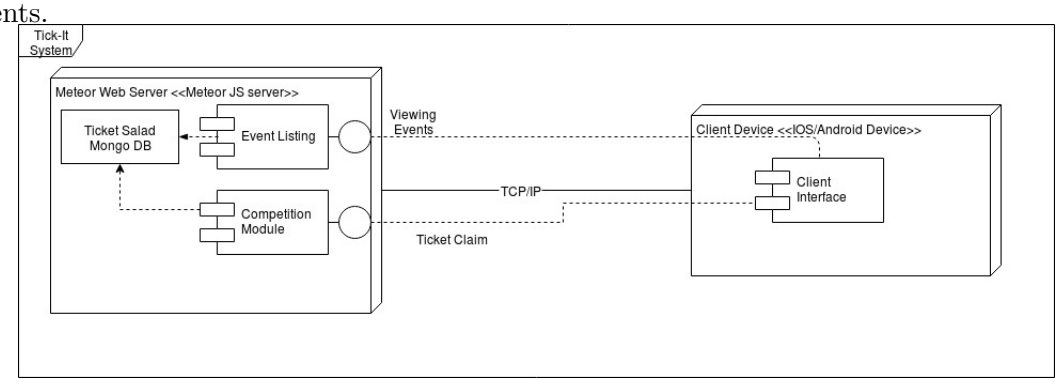


Figure 1: A high-level overview of the software architecture of the Ticket-Salad System.

2 Overall software architecture

This section specifies the software architecture requirements and the software architecture design for the first level of granularity the system as a whole. The output will be the high level software architecture components, the infrastructure between them and the tactics used at the first level of granularity to realize the quality requirements for the system. Subsequent sections will focus on the software architecture requirements and design for these high-level architectural components. Note that many of the architectural requirements (particularly the quality requirements) will be propagated down to lower level architectural components.

2.1 Architecture requirements

2.1.1 Access and integration requirements

This section discusses the software architecture requirements that is the requirements around the software infrastructure within which the application functionality is to be developed. The purpose of this infrastructure is to address the non-functional requirements. In particular, the architecture

requirements specify

- The architectural responsibilities which need to be addressed.
- The access and integration requirements for the system.
- The quality requirements
- The architecture constraints specified by the client.

2.1.2 Quality requirements

The quality requirements are the requirements around the quality attributes of the systems and the services it provides. This includes requirements like maintainability, flexibility, extensibility, performance, scalability, security, auditabilty, usability, and testability requirements.

2.1.2.1 Flexibility

It is important that the system architecture is such that one can easily add different access channels to the system. This is particularly important in the context of the proliferation of connected devices in a world transiting to fully embrace the Internet of Things (IoT).

Furthermore, persistence architectures and reporting infrastructures are rapidly evolving as can be seen from the rapid growth of NoSQL databases, semantic knowledge repositories and big data stores. In this context it is important that the application functionality is not locked into any specific persistence technology and that one is able to easily modify the persistence provider and reporting framework.

2.1.2.2 Maintainability

Amongst the most important quality requirements for the system is *maintainability*. It should be easy to maintain the system in the future. To this end

- Future developers should be able to easily understand the system.
- The technologies chosen for the system should be available for a reasonably long time.
- Developers should be easily and quickly be able to change aspects of the functionality the system provides.

- Developers should be easily and quickly be able to add new functionality to the system.

2.1.2.3 Scalability

- Initially the system must be able to support a relatively large group of users, i.e, supporting a maximum of 1000 users.
- The architecture of the system should allow for increasing the amount of users to includes several thousands of users.

2.1.2.4 Performance

- All operations that do not require database communication should respond in less than 0.5 seconds.
- All operations that communicate and modify the database should be processed in less than 7 seconds

The above figures do not include effect of internet speed and device speed have on the system.

2.1.2.5 Reliability

The system should provide by default a reasonable level of availability and reliability and should be deployable within configurations which provide a high level availability, supporting

- Fail-over safety of all components
- A deployment without single points failure Rapid deployment of new/changing functionality is required for this system.

2.1.2.6 Security

The system needs to support

- Login authentication against a NoSQL user repository.
- Credit card verification against mastercard and visa repositories.
- Payment verification using mastercard and visa services.

2.1.2.7 Auditability

The system will have a log for all login, logout, bidding and financial transactions, the following will be stored in the log:

- User id
- Date/time stamp of the interaction
- Response from the interaction
- Before and after states of the users profile

2.1.2.8 Testability

All services offered by the system must be testable through

1. Automated unit tests testing components in isolation using mock objects.
2. Automated integration tests where components are integrated within the actual environment which test,
    - The services are provided if all pre-conditions are met
    - That all post conditions hold true after the service is provided

2.1.2.9 Usability

3 Functional Requirements

R.1 Ticket Salad shall allow the user to sign up.
R.1.1Ticket Salad shall allow a user to enter their required information.
R.1.2 Ticket Salad shall create a user account based on the information.
R.1.3 Ticket Salad shall reject account creation if the information is not valid.
R.1.4 Ticket Salad shall secure the account information in secure database.

R.2 Ticket shall allow the user to Log In
R.2.1 Ticket Salad shall allow a user to log in to their account.
R.2.2 Ticket Salad shall allow a user to log out of their account.
R.2.3Ticket Salad shall reject a login based off invalid details.
R.2.4 Ticket Salad shall hide all in-app views if a user is logged out.

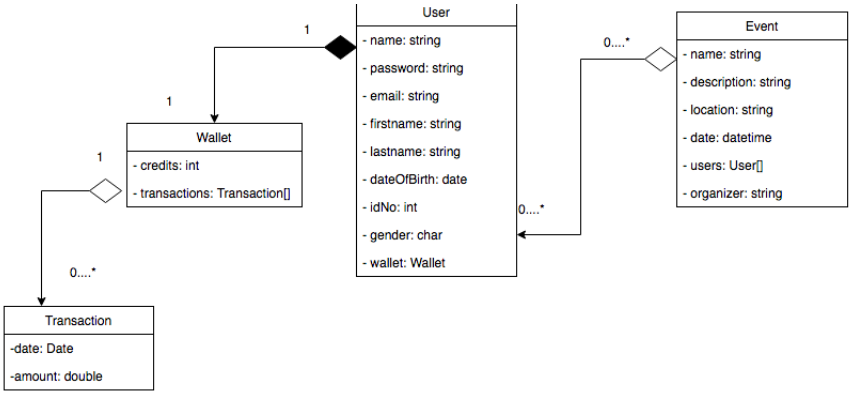
R.3 Ticket Salad shall allow the user to view events
R3.1 Ticket Salad shall allow a user to view all current ticket events.
R.3.2 Ticket Salad shall allow a user to search through the displayed events.
R.3.3 Ticket Salad shall display more information in the expanded event.
R.3.4 Ticket Salad shall allow a user to expand a certain event.
R.3.5 Ticket Salad shall allow a user to bid on an event from the expanded view.
R.3.6 Ticket Salad shall allow a user to bid on an event directly from the events view.
R.3.7 Ticket Salad shall notify users when an event closes
R.3.8 Ticket Salad shall notify a user if they win an event

R.4 Ticket Salad shall allow the user to gain access to their profile
R.4.1 Ticket Salad shall allow a user to view their profile.
R.4.2 Ticket Salad shall allow a user to edit information regarding their profile.
R.4.3 Ticket Salad shall allow a user to delete their account at their request.
R.4.4 Ticket Salad shall reject changes to the users profile if the information is invalid.

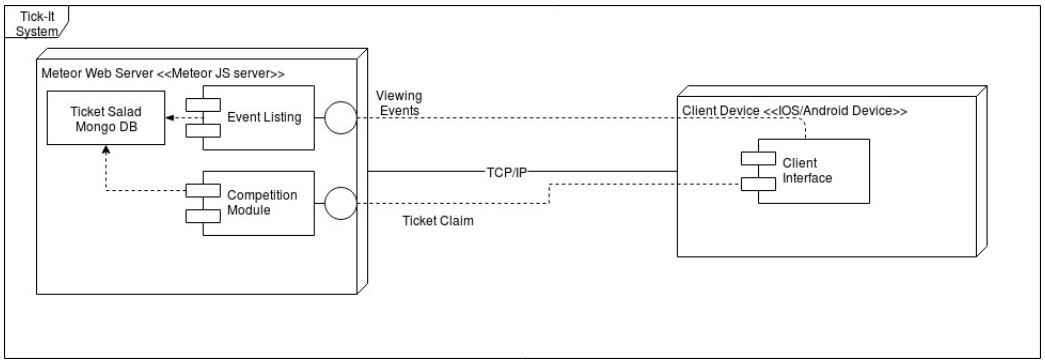
R.5 Ticket Salad shall allow the user to access their claim credits
R.5.1 Ticket Salad shall allow a user to view their claim credit balance.
R.5.2 Ticket Salad shall allow a user to purchase more claim credits.
R.5.3 Ticket Salad shall allow a user to claim bids if they possess enough credits.

R.5.4 Ticket Salad shall deduct claim credits when a user bids on an event

4 Domain Model



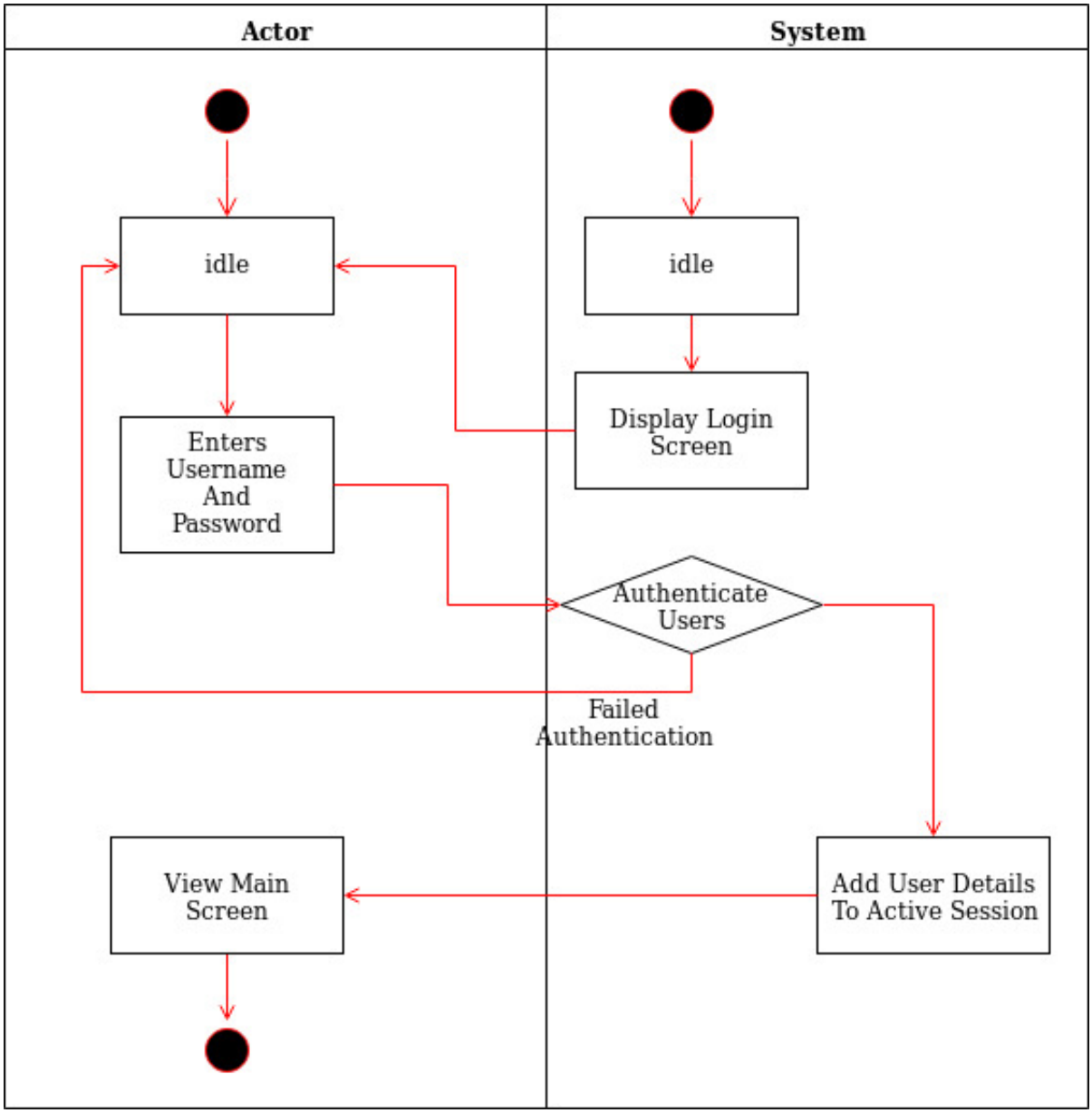
5 System Architecture



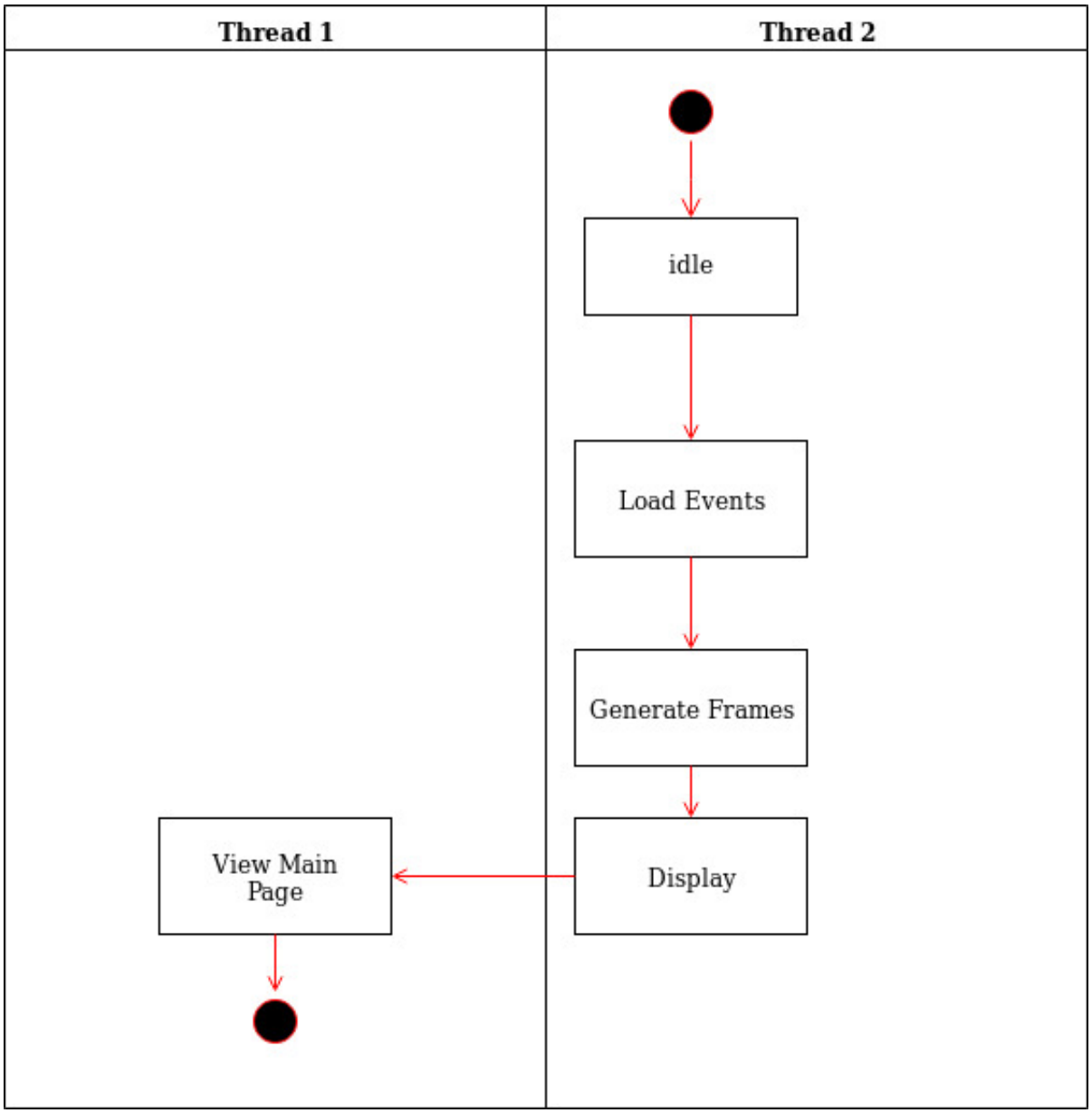
The architecture used in the Ticket Salad application is client server. The system runs it's main database functions on a central server along with certain server functions which can then be accessed by the client device in order to verify data used for the application which runs on the client device.

6 Use Cases

6.1 Logging in



6.2 Display Main screen



6.3 Event Bidding