

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Goal of the project . . . . .	2
1.2	How we worked . . . . .	2
1.2.1	Trello . . . . .	2
1.2.2	Git . . . . .	2
1.2.3	Discord . . . . .	2
1.3	Submission . . . . .	3
1.4	Assumptions . . . . .	3
1.5	Chapters . . . . .	3
1.5.1	Structured Analysis . . . . .	3
1.5.2	UML . . . . .	3
1.5.3	Structured Design . . . . .	4
<b>2</b>	<b>Structured Analysis</b>	<b>4</b>
<b>3</b>	<b>UML</b>	<b>4</b>
3.1	Use Cases . . . . .	4
3.2	Classes . . . . .	5
3.3	Command Lifecycle . . . . .	5
<b>4</b>	<b>Structured Design</b>	<b>6</b>
<b>5</b>	<b>Epilogue</b>	<b>6</b>

# 1 Introduction

This section aims to clarify the goal of the project, as well as describing the philosophy and the tools used by this team to efficiently collaborate and achieve better results. In addition, any assumptions made about the project are reported and a short description of the all content is given.

## 1.1 Goal of the project

Given the description of a software application, an e-broker<sup>1</sup> system, use a variety of diagrams(UML, dataflow etc.) in order to better analyze the requirements, functional or not, of the application.

## 1.2 How we worked

### 1.2.1 Trello

To better collaborate and organize our work, we used *Trello*, a flexible project management tool, by assigning weekly tasks to each member of the team.

### 1.2.2 Git

In order to exploit work parallelism, while keeping track of all progress the rest of the team made in the meantime, we used the famous VSC *git*<sup>2</sup> using *this* branching model. For those unfamiliar with git, it is a distributed version-control system, tracking changes in source code during software development to better coordinate work among programmers. Whenever a task was completed, the corresponding member requested a review. Then, the rest of the team was able to review the set of changes, discuss potential modifications, or even enhance the commit. Eventually, in most cases, the request was approved and merged into the main branch.

### 1.2.3 Discord

As in every team project, in order to optimize performance, meetings really were a necessity. We sure had some face-to-face meetings, but that was not always possible. So, another tool we used is *Discord*, a platform designed for video gaming communities, that specializes in text, image, video and audio communication between users in a chat channel. Using Discord's video-chatting and screen-sharing features helped us co-ordinate and co-operate easy and fast. As a team, we agreed on having short but frequent meetings(2 or 3, 10 minute meetings per day) in order to take full advantage of time, optimizing performance in general.

---

<sup>1</sup>An e-broker is a brokerage house that allows you to buy and sell stocks and obtain investment information from its Web site

<sup>2</sup>One can take a look at the source code of this project or even contribute, [\*here\*](#)

### 1.3 Submission

Work load was pretty well distributed amongst all members of this team throughout the whole process of deployment. The one member responsible of reviewing the final version of this pdf and submitting it to e-class is *Konstantinos Koyias*.

### 1.4 Assumptions

Despite the detailed and informative description of the project, there were a few points we had to intervene, make our own assumptions and decide on what is best to follow. The most important of those were:

- 
- 

### 1.5 Chapters

#### 1.5.1 Structured Analysis

This chapter contains:

- The dataflow diagramm of the e-Broker system.
- Dataflow diagramms for levels 1 and 2 of procedure decomposition.
- The process decomposition tree.
- Specifications for:
  1. The process of transmitting order in Structured English.
  2. The process of supply estimation using a Decision Table.
  3. The process of a client's log in the e-Broker system using a Tree.
  4. Removing a command from the Data Dictionary.

#### 1.5.2 UML

In this chapter, we used the general-purpose, developmental, modeling language UML in order to visualize the design of the e-Broker system. Specifically, the following were included:

- ▷ Use Case diagramm of the system.
- ▷ Class diagramm of the system.
- ▷ State-machine diagramm of the entity "Command".

To better model the stock selling/buying operations we included:

- ▷ A main success case Scenario and some alternative ones.

- ▷ An Activity diagramm.
- ▷ A detailed Class diagramm.
- ▷ A Sequence Diagram.
- ▷ A Communication Diagram.

### 1.5.3 Structured Design

This is the chapter where a systematic methodology using

- a Programm Structure diagramm and
- a piece of pseudo-code that formulates the control unit of
  - the main transformation
  - the M3 transformation
  - the M4 transformation

and the presentation unit of

- the M2 transformation

for the corresponding Programm Structure diagramm.

determines the design specifications of the application.

## 2 Structured Analysis

f t

## 3 UML

In this chapter, all necessary UML diagrams for the e-Broker system will be created. In order to distinguish relationship types, we used this map:

{always: include, might: extend, generic: generalization}.

This gets a bit more clear in the following subsection.

### 3.1 Use Cases

After taking a good look at the description of the system, we came up with the following actors:

- Customer(Primary)
- Customer Support Department(Secondary)
- Athens Stock Exchange(Secondary)

and the corresponding use cases. Each customer can:

- **log in**

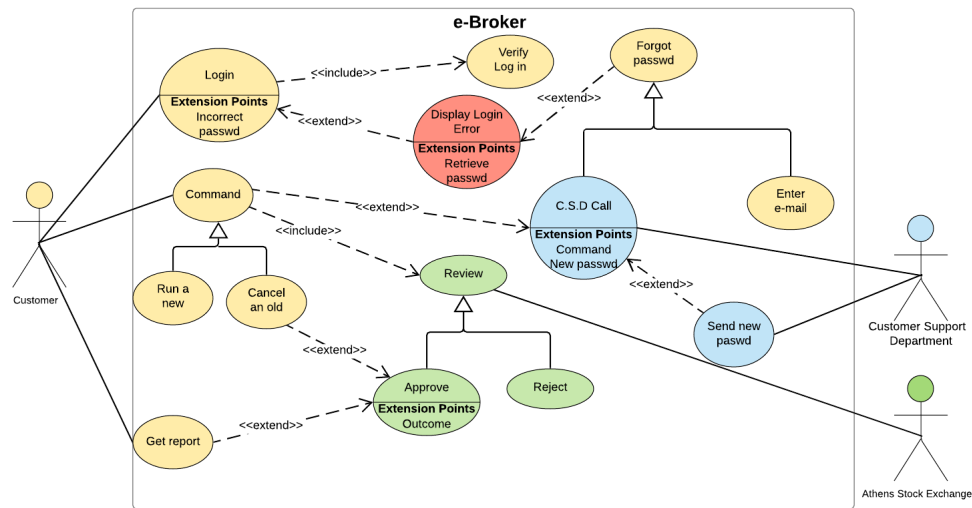
This action will **always** trigger a password check, and it **might** lead to an error message. In this case a **generic** “forgot password” request **might** happen, which can be either a password reminder via an e-mail, or a call to the *C.S.D* for a new password or/and a command to be executed.

- **command**

A **generic** command, can be either a creation or a cancellation, both of which will **always** have to be reviewed by the *A.S.E*. This **generic** review will either lead to an approval or a rejection. An approved command **might** at some point get cancelled, otherwise the *Customer* will

- **get a report**

so we came up with the following use case diagramm



## 3.2 Classes

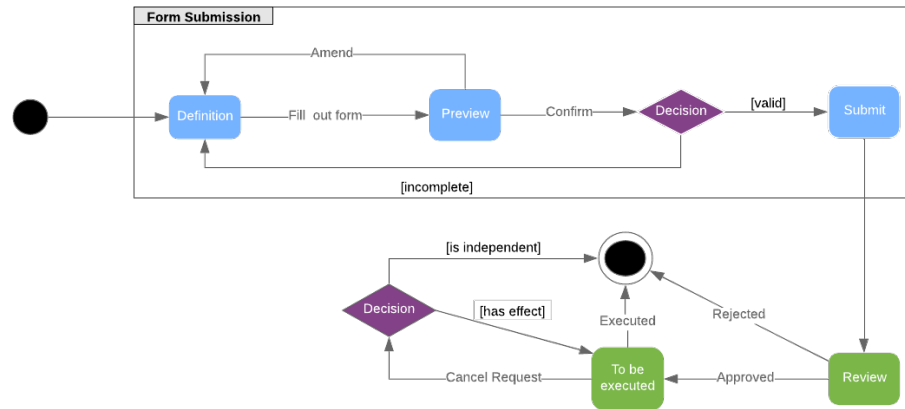
cls

## 3.3 Command Lifecycle

By reading the description of the system, one might notice that the lifetime of the *command* entity happens to be a really interesting one. Well, in that case, a common technique is using a state-machine diagramm to better visualize all possible outcomes of feature.

For a command to be executed, all properties it has, need to first be defined using a form.

After a successful submission, meaning all fields contain values of the right type, the command will be reviewed. In case of an approval, it will be executed as soon as possible. In the meantime, it can be cancelled. But, this is only possible in case of an independent command, meaning it has no recursive effect to any previously executed commands, because they can not be reverted now in any way. The diagram follows:



## 4 Structured Design

c a

## 5 Epilogue

k t