RASD.docx

Main document changes and comments

| Page 1: Style Definition | Marco Petri | 22/12/2020 21:35:00 |
|---|---|---|

TOC 4

| Page 1: Deleted | Marco Petri | 22/12/2020 22:16:00 |
|---|---|---|

0.1

| Page 1: Inserted | Marco Petri | 22/12/2020 22:16:00 |
|---|---|---|

1.0

| Page 1: Deleted | Marco Petri | 22/12/2020 22:16:00 |
|---|---|---|

20

| Page 1: Inserted | Marco Petri | 22/12/2020 22:16:00 |
|---|---|---|

22

| Page i: Inserted | Marco Petri | 22/12/2020 22:16:00 |
|---|---|---|

**Page ii: Deleted** **Marco Petri** **22/12/2020 22:16:00**

**Page ii: Inserted** **Marco Petri** **22/12/2020 22:16:00**

**Error! Hyperlink reference not valid.**

**Page ii: Inserted** **Marco Petri** **22/12/2020 22:17:00**

**Error! Bookmark not defined.**

**Page ii: Inserted** **Marco Petri** **22/12/2020 22:16:00**

**Error! Hyperlink reference not valid.**

**Page ii: Inserted** **Marco Petri** **22/12/2020 22:17:00**

**Error! Bookmark not defined.**

**Page ii: Inserted** **Marco Petri** **22/12/2020 22:16:00**

**Error! Hyperlink reference not valid.**

**Page ii: Inserted** **Marco Petri** **22/12/2020 22:17:00**

**Error! Bookmark not defined.**

**Page ii: Inserted** **Marco Petri** **22/12/2020 22:16:00**

**Error! Hyperlink reference not valid.**

**Page ii: Inserted** **Marco Petri** **22/12/2020 22:17:00**

**Error! Bookmark not defined.**

**Page ii: Inserted** **Marco Petri** **22/12/2020 22:16:00**

**Error! Hyperlink reference not valid.**

**Page ii: Inserted** **Marco Petri** **22/12/2020 22:17:00**

**Error! Bookmark not defined.**

**Page ii: Inserted** **Marco Petri** **22/12/2020 22:16:00**

**Error! Hyperlink reference not valid.**

**Page ii: Inserted** **Marco Petri** **22/12/2020 22:17:00**

**Error! Bookmark not defined.**

**Page 1: Inserted** **asus** **18/12/2020 10:59:00**

and employees

**Page 2: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 2: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 2: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 2: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 2: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 2: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 2: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 2: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 2: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 2: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 2: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 2: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 2: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 3: Formatted Table** **Marco Petri** **18/12/2020 10:54:00**

Formatted Table

**Page 3: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 3: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 3: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 3: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 3: Formatted Table** **Marco Petri** **18/12/2020 10:54:00**

Formatted Table

**Page 3: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 3: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 3: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 3: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 3: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 3: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 3: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 3: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 3: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 4: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 4: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 4: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 4: Deleted** **Marco Petri** **18/12/2020 11:01:00**

## Machine phenomena

| Phenomena | Controller | Description |
|---|---|---|
| 1 | M | System computes the number of a client |
| 2 | M | System calls a number |
| 3 | M | System verifies if a number is the number of the customer which should enter |
| 4 | M | System computes the estimated waiting time (for each customer) |
| 5 | M | System computes mean duration of a visit for a long-term customer |
| 6 | M | System evaluates the distance between the customer and the store |
| 7 | M | System finds alternative time slots |
| 8 | M | System finds other near stores |

**Page 4: Commented [MP1]** **Marco Petri** **01/12/2020 15:22:00**

ROSSI DOESN'T LIKE MACHINE PHENOMENA IN RASD

**Page 5: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 5: Formatted Table** **Marco Petri** **18/12/2020 10:54:00**

Formatted Table

**Page 5: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 5: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 5: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 5: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 5: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 5: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 5: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 5: Formatted** **Marco Petri** **22/12/2020 22:03:00**

English (United Kingdom)

**Page 5: Inserted** **Marco Petri** **22/12/2020 17:56:00**

+ **Booking**: a booking refers to a reservation asked by the client to visit a store, sometimes it is called visit;

**Page 5: Formatted** **Marco Petri** **22/12/2020 17:56:00**

Font: Not Bold

**Page 5: Inserted** **Marco Petri** **22/12/2020 17:56:00**

;

+

**Page 5: Deleted** **Marco Petri** **22/12/2020 17:56:00**

.

+

**Page 5: Inserted** **Marco Petri** **22/12/2020 17:56:00**

+

**Visit**: a visit is the action of visiting a store entering it, we refer to it when we speak about bookings.

+

**Page 5: Formatted** **Marco Petri** **22/12/2020 17:56:00**

Font: Not Bold

**Page 7: Deleted** **Marco Petri** **22/12/2020 18:00:00**

## Reference documents

http://dati.istat.it/Index.aspx?DataSetCode=DCCV_ICT: is an ISTAT statistic about the usage internet diffusion in Italy;

Oracle Directory Server Enterprise Edition Deployment Planning Guide: is a document about the operations needed in the process of designing a system [Chapter 12]; https://docs.oracle.com/cd/E20295_01/html/821-1217/fjdch.html#scrolltoc;

29148-2018 - 29148-2018 - ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering - IEEE Standard: ISO/IEC/IEEE document about the building and designing of a system and RASD definition;

https://gdpr-info.eu/: contains the official PDF of the Regulation (EU) 2016/679 (General Data Protection Regulation).

https://www.iso.org/isoiec-27001-information-security.html: ISO/IEC 27001 international standard.

**Page 7: Moved to page 54 (Move #1) Marco Petri 22/12/2020 18:00:00**

http://dati.istat.it/Index.aspx?DataSetCode=DCCV_ICT: is an ISTAT statistic about the usage internet diffusion in Italy;

Oracle Directory Server Enterprise Edition Deployment Planning Guide: is a document about the operations needed in the process of designing a system [Chapter 12];

https://docs.oracle.com/cd/E20295_01/html/821-1217/fjdch.html#scrolltoc;

29148-2018 - 29148-2018 - ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering - IEEE Standard: ISO/IEC/IEEE document about the building and designing of a system and RASD definition;

https://gdpr-info.eu/: contains the official PDF of the Regulation (EU) 2016/679 (General Data Protection Regulation).

https://www.iso.org/isoiec-27001-information-security.html: ISO/IEC 27001 international standard.

**Page 7: Formatted asus 18/12/2020 09:39:00**

English (United Kingdom)

**Page 7: Formatted asus 18/12/2020 09:39:00**

English (United Kingdom)

**Page 7: Formatted asus 18/12/2020 09:39:00**

English (United Kingdom)

**Page 7: Formatted asus 18/12/2020 09:39:00**

English (United Kingdom)

**Page 7: Formatted asus 18/12/2020 09:39:00**

English (United Kingdom)

**Page 7: Formatted asus 18/12/2020 09:39:00**

English (United Kingdom)

**Page 7: Deleted Marco Petri 22/12/2020 18:00:00**

not cited in the first chapter

**Page 9: Deleted aa m 19/12/2020 12:11:00**

to his favourite store

**Page 9: Deleted aa m 19/12/2020 12:05:00**

**exit**

**Page 9: Inserted aa m 19/12/2020 12:05:00**

**change**

**Page 9: Deleted aa m 19/12/2020 12:04:00**

exits from

**Page 9: Inserted** aa m 19/12/2020 12:04:00

enters a new

**Page 9: Deleted** aa m 19/12/2020 12:04:00

his

**Page 9: Inserted** aa m 19/12/2020 12:04:00

. A pop-up message appears on the screen warning him that by doing this he would automatically exit the previous queue.

**Page 9: Deleted** aa m 19/12/2020 12:04:00

, shortening the queue for those behind him.

**Page 9: Inserted** aa m 19/12/2020 12:06:00

agrees and

**Page 9: Deleted** aa m 19/12/2020 12:06:00

then

**Page 9: Inserted** aa m 19/12/2020 12:03:00

## Queue exit

A worker realizes that he cannot attend his queue spot since he now has an important videocall to attend. He therefore logs into his account and exits from his queue, shortening the queue for those behind him.

**Page 13: Formatted** Marco Petri 22/12/2020 17:55:00

Left, Space After: 8 pt, Line spacing: Multiple 1.08 li

**Page 13: Deleted** Marco Petri 22/12/2020 17:55:00

| Page 13: Inserted | Marco Petri | 22/12/2020 17:55:00 |
|---|---|---|

**Page 13: Formatted** **Marco Petri** **22/12/2020 17:55:00**

Justified, Space After: 0 pt, Line spacing: Multiple 1.25 li

**Page 14: Deleted** **asus** **18/12/2020 09:44:00**

short-time

**Page 14: Inserted** **asus** **18/12/2020 09:44:00**

real-time

**Page 14: Inserted** **asus** **18/12/2020 09:54:00**

Visits can be for a single person or for multiple-people groups, with an upper limit to the size of a group.

**Page 14: Inserted** **asus** **18/12/2020 09:45:00**

he or she

**Page 14: Deleted** **asus** **18/12/2020 09:45:00**

if

**Page 15: Inserted** **asus** **18/12/2020 09:47:00**

(on entrance)

**Page 15: Inserted** **asus** **18/12/2020 09:47:00**

.

**Page 15: Deleted** **asus** **18/12/2020 09:47:00**

..

**Page 15: Inserted** **asus** **18/12/2020 09:47:00**

s

**Page 18: Deleted** **Marco Petri** **21/12/2020 19:32:00**

Introductory text to the chapter (how it is subdivided and what are we going to say in the chapter)

**Page 18: Inserted** **Marco Petri** **21/12/2020 19:32:00**

In chapter 3 we show the requirements for the S2B and we map the goals to the requirements and the domain assumption for the S2B. Then after the mapping we list some use cases and sequence diagrams representing the interaction of actors with the system and the workflow of some functionalities which are granted by the systems and can be used by customer, store manager or checkpoint controllers. At the end of the chapter we introduce some specific constraints for that system and we express the software attributes.

**Page 18: Inserted** **aa m** **19/12/2020 12:27:00**

The user interfaces presented are extremely minimal. They are then greatly expanded in the design document when the platforms have been decided.

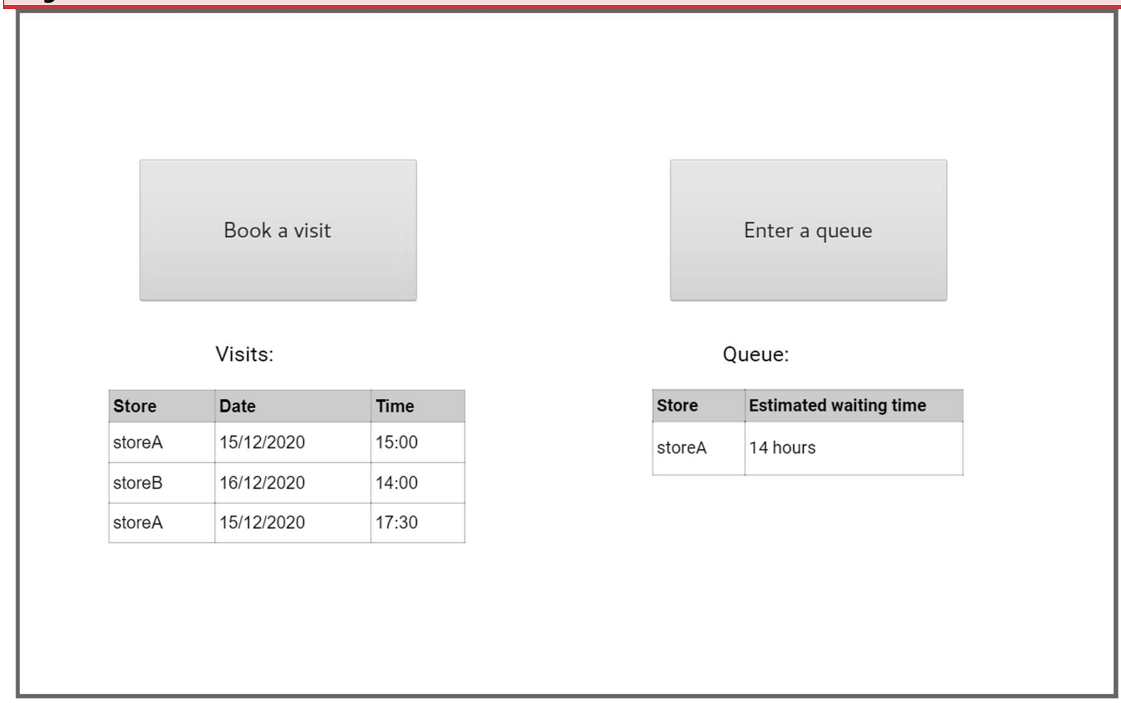
| Page 18: Formatted | Marco Petri | 21/12/2020 19:36:00 |
|---|---|---|

Normal, Space After: 0 pt, Line spacing: single, No bullets or numbering

| Page 18: Formatted | aa m | 19/12/2020 12:27:00 |
|---|---|---|

Font: Not Bold

| Page 18: Inserted | Marco Petri | 21/12/2020 19:38:00 |
|---|---|---|



Visits:

| Store | Date | Time |
|---|---|---|
| storeA | 15/12/2020 | 15:00 |
| storeB | 16/12/2020 | 14:00 |
| storeA | 15/12/2020 | 17:30 |

Queue:

| Store | Estimated waiting time |
|---|---|
| storeA | 14 hours |

| Page 18: Formatted | Marco Petri | 22/12/2020 18:05:00 |
|---|---|---|

List Paragraph, Bulleted + Level: 1 + Aligned at: 0.63 cm + Indent at: 1.27 cm

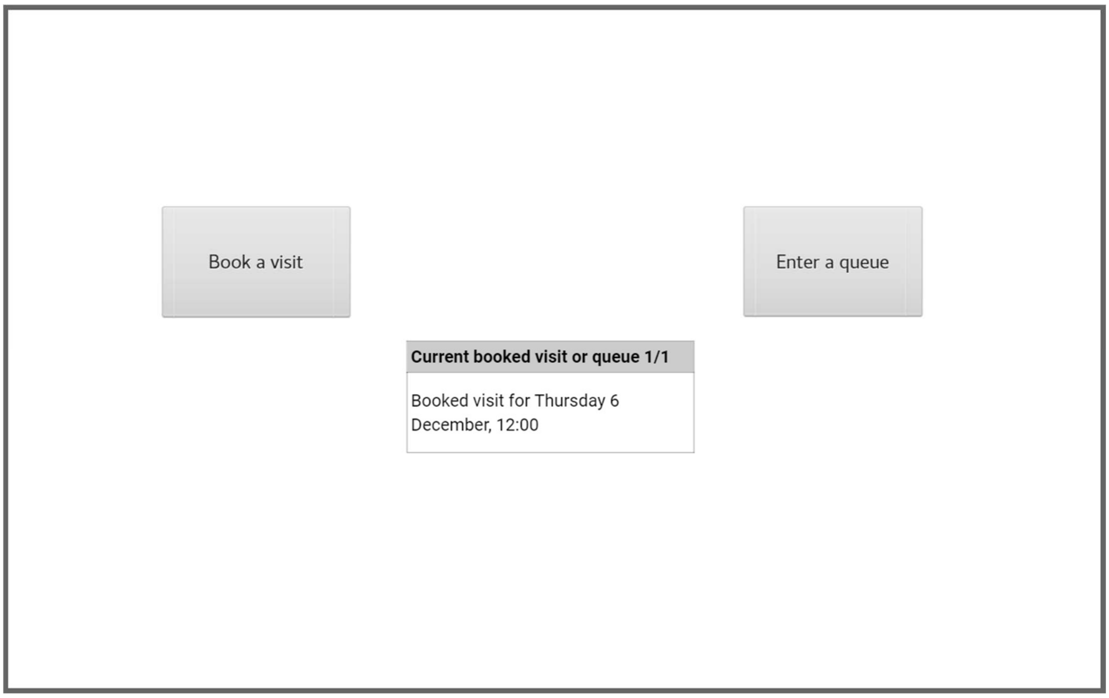
| Page 18: Inserted | aa m | 19/12/2020 12:24:00 |
|---|---|---|

On selecting a queue or a booking the customer will access the digital ticket and will be able to delete the booking or exit the queue.

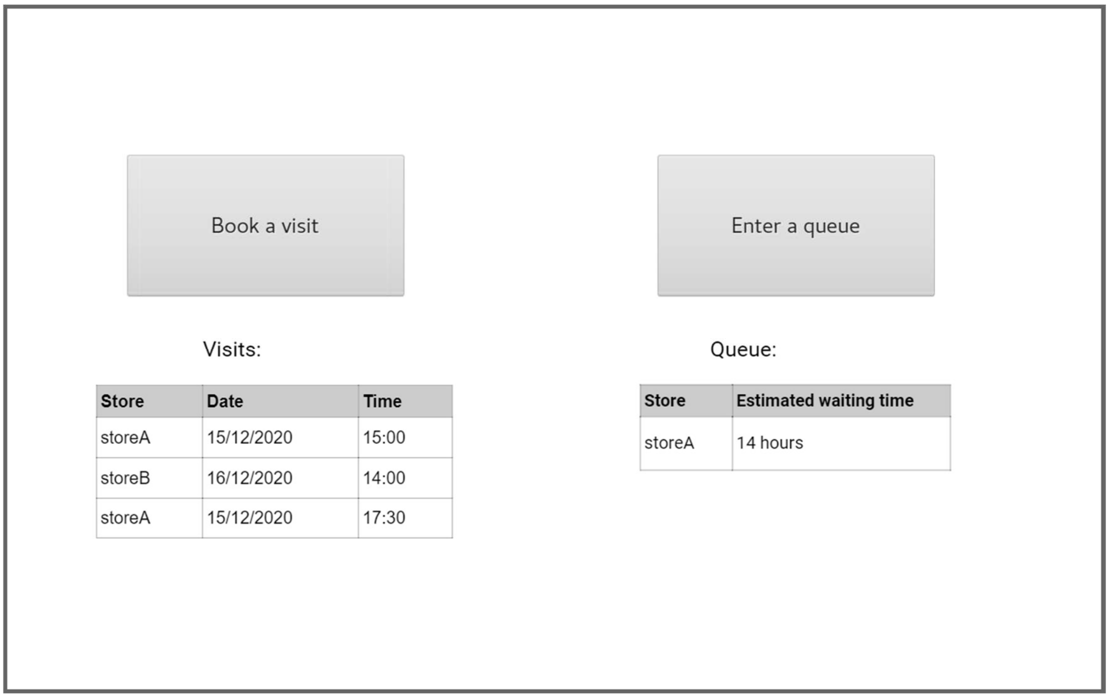
| Page 18: Inserted | Marco Petri | 21/12/2020 19:38:00 |
|---|---|---|

●

| Page 19: Deleted | Marco Petri | 22/12/2020 18:07:00 |
|---|---|---|



| Current booked visit or queue 1/1 |
|---|
| Booked visit for Thursday 6 December, 12:00 |



Visits:

| Store | Date | Time |
|---|---|---|
| storeA | 15/12/2020 | 15:00 |
| storeB | 16/12/2020 | 14:00 |
| storeA | 15/12/2020 | 17:30 |

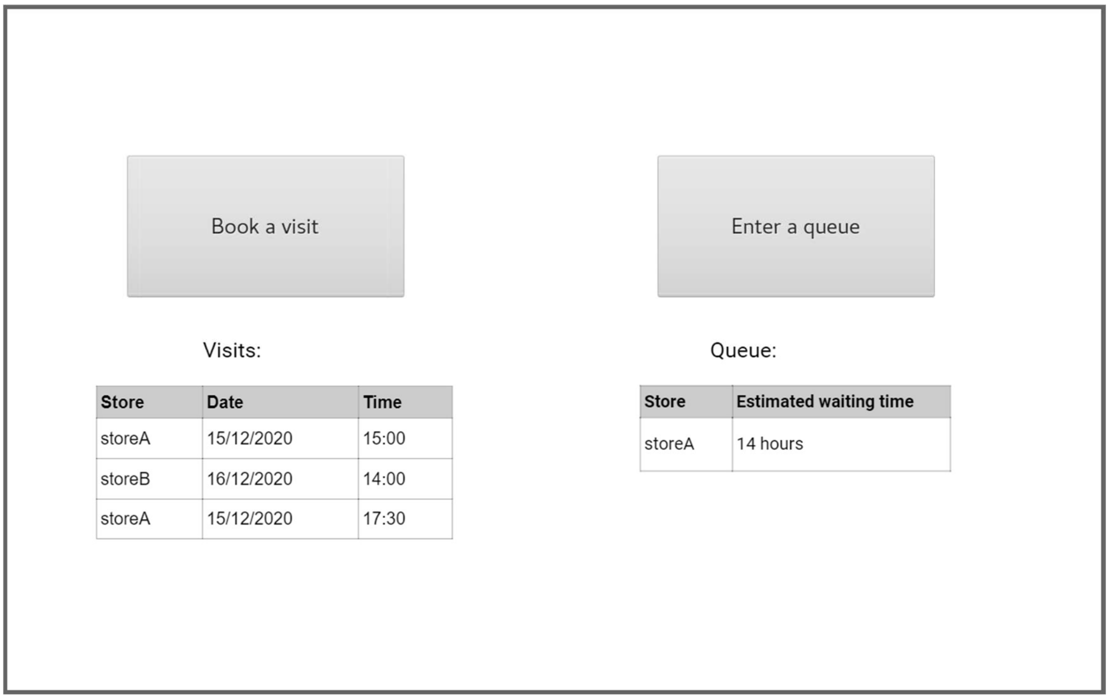
Queue:

| Store | Estimated waiting time |
|---|---|
| storeA | 14 hours |

**Page 19: Formatted** **Marco Petri** **22/12/2020 18:06:00**

Justified, Space After: 0 pt, Line spacing: Multiple 1.25 li, Bulleted + Level: 1 + Aligned at: 0.63 cm + Indent at: 1.27 cm

**Page 19: Inserted** **aa m** **19/12/2020 12:24:00**



Visits:

| Store | Date | Time |
|---|---|---|
| storeA | 15/12/2020 | 15:00 |
| storeB | 16/12/2020 | 14:00 |
| storeA | 15/12/2020 | 17:30 |

Queue:

| Store | Estimated waiting time |
|---|---|
| storeA | 14 hours |

**Page 19: Inserted** **Marco Petri** **22/12/2020 18:07:00**

**Page 19: Formatted** **Marco Petri** **22/12/2020 18:07:00**

Normal, No bullets or numbering

**Page 19: Formatted** **Marco Petri** **22/12/2020 18:06:00**

Justified, Space After: 0 pt, Line spacing: Multiple 1.25 li, Bulleted + Level: 1 + Aligned at: 0.63 cm + Indent at: 1.27 cm

**Page 20: Inserted** **Marco Petri** **22/12/2020 18:08:00**

**Page 20: Formatted** **Marco Petri** **22/12/2020 18:08:00**

Font: Not Bold

**Page 20: Formatted** **Marco Petri** **22/12/2020 18:08:00**

Normal, No bullets or numbering

**Page 20: Formatted** **Marco Petri** **22/12/2020 18:08:00**

Justified, Space After: 0 pt, Line spacing: Multiple 1.25 li, Bulleted + Level: 1 + Aligned at: 0.63 cm + Indent at: 1.27 cm

**Page 21: Deleted** **aa m** **18/12/2020 10:29:00**

**The application does not have any hardware interface. This is because all the hardware services used like the GPS or the camera are accessed indirectly through the operating system.**

**Page 21: Formatted** **Marco Petri** **22/12/2020 18:22:00**

Font: Bold

**Page 21: Formatted** **Marco Petri** **22/12/2020 18:08:00**

Bulleted + Level: 1 + Aligned at: 0.63 cm + Indent at: 1.27 cm

**Page 21: Inserted** **aa m** **18/12/2020 11:22:00**

- **Speaker**: it is used by the checkpoint controller at the entrance to call out loud the ticket's numbers.

**Queue display**: it is used to show the ticket's numbers called and about to be called.

•

**Page 21: Formatted** **Marco Petri** **22/12/2020 18:22:00**

Font: Bold

**Page 21: Formatted** **Marco Petri** **22/12/2020 18:08:00**

Justified, Space After: 0 pt, Line spacing: Multiple 1.25 li, Bulleted + Level: 1 + Aligned at: 0.63 cm + Indent at: 1.27 cm

**Page 21: Inserted** **aa m** **18/12/2020 11:24:00**

**Internet:** the application uses internet for remote communication with users

•

**Page 21: Formatted** **Marco Petri** **22/12/2020 18:09:00**

List Paragraph, Bulleted + Level: 1 + Aligned at: 0.63 cm + Indent at: 1.27 cm

**Page 21: Inserted** **Marco Petri** **22/12/2020 18:09:00**

-

**Page 21: Deleted** **aa m** **18/12/2020 10:29:00**

**HTTPS:** the application will use this protocol to safely communicate over the internet.

**Page 21: Formatted** **Marco Petri** **22/12/2020 18:08:00**

Normal, No bullets or numbering

**Page 21: Deleted** **Marco Petri** **22/12/2020 18:09:00**

**Page 21: Inserted** **aa m** **18/12/2020 11:24:00**

**Page 21: Deleted** **aa m** **18/12/2020 10:29:00**

**Wi-Fi:** the ticket machines might use it to communicate with the system, depending on the store's preference.

**Page 21: Formatted** **aa m** **18/12/2020 10:31:00**

Indent: Left: 1.25 cm

**Page 22: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 22: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 22: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 22: Deleted** **asus** **18/12/2020 11:04:00**

If a store was previously selected, c

**Page 22: Inserted** **asus** **18/12/2020 11:04:00**

C

**Page 22: Inserted** **asus** **18/12/2020 11:04:00**

for a store

**Page 22: Inserted** **asus** **18/12/2020 11:13:00**

(if the selected store is the same one for which the customer is already queuing, the request is simply denied)

**Page 22: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 22: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 22: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 22: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 22: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 22: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 22: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 22: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 23: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 23: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 23: Inserted** **asus** **18/12/2020 11:20:00**

. Finally, the visit must not overlap with any other visit booked by the customer.

**Page 23: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 23: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 23: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 23: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R19

**Page 23: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R18

**Page 23: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 23: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R20

**Page 23: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R19

**Page 23: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 23: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R21

**Page 23: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R20

**Page 23: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 23: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R22

**Page 23: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R21

**Page 23: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 23: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R23

**Page 23: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R22

**Page 23: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 23: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R24

**Page 23: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R23

**Page 23: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 24: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R25

**Page 24: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R24

**Page 24: Formatted** **Marco Petri** **22/12/2020 22:04:00**

English (United Kingdom)

**Page 24: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R19

**Page 24: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R18

**Page 24: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R20

**Page 24: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R19

**Page 24: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R21

**Page 24: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R20

**Page 24: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R22

**Page 24: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R21

**Page 24: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R23

**Page 24: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R22

**Page 24: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R24

**Page 24: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R23

**Page 24: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R25

**Page 24: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R24

**Page 25: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R24

**Page 25: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R23

**Page 25: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R25

**Page 25: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R24

**Page 25: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R19

**Page 25: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R18

**Page 25: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R21

**Page 25: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R20

**Page 25: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R25

**Page 25: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R24

**Page 25: Deleted** **Marco Petri** **22/12/2020 18:01:00**

, R18

**Page 25: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R23

**Page 25: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R22

**Page 25: Deleted** **Marco Petri** **22/12/2020 18:01:00**

, R18

**Page 25: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R21

**Page 25: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R20

**Page 25: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R19

**Page 25: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R18

**Page 25: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R20

**Page 25: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R19

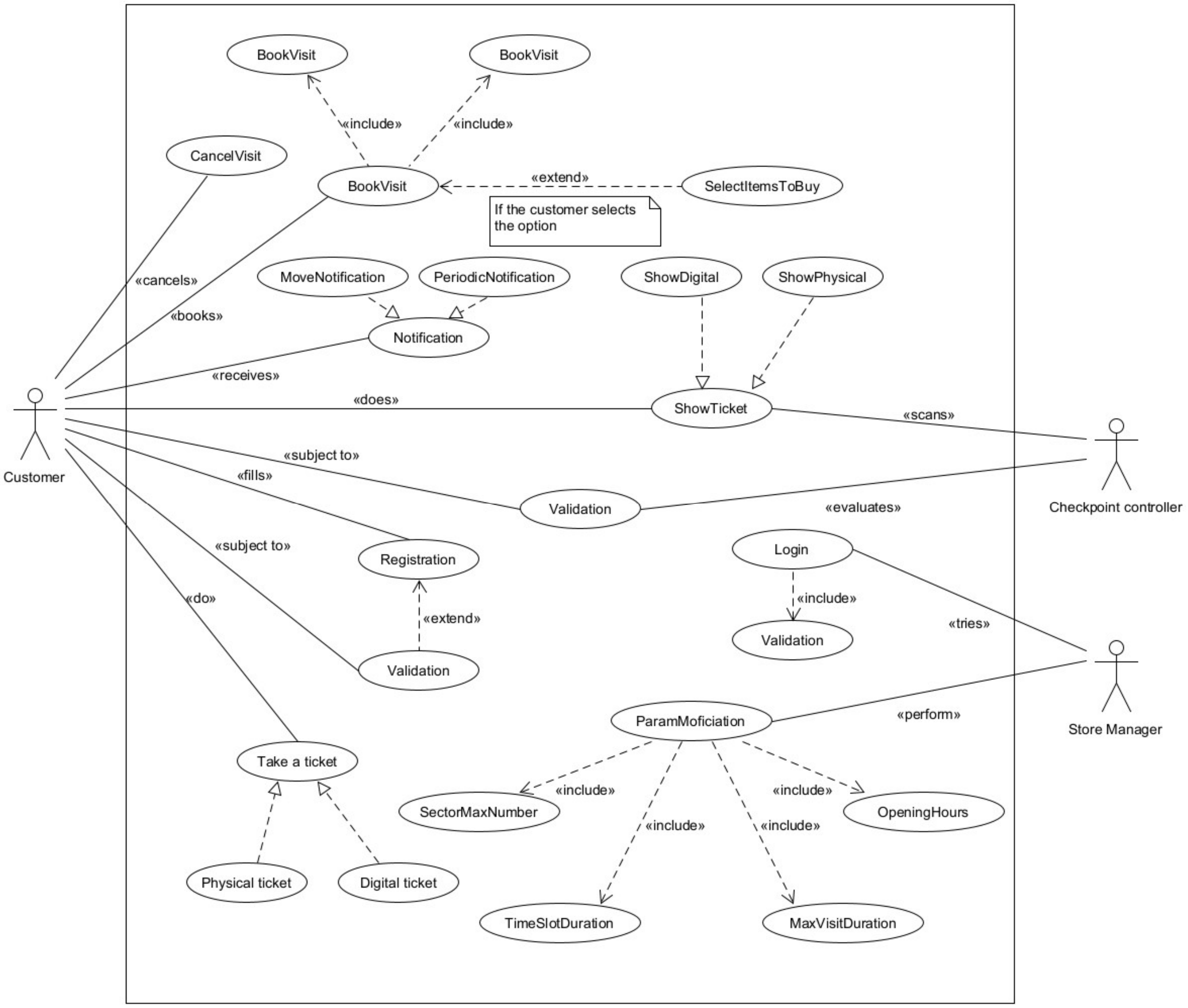
**Page 25: Deleted** **Marco Petri** **22/12/2020 18:02:00**

R22

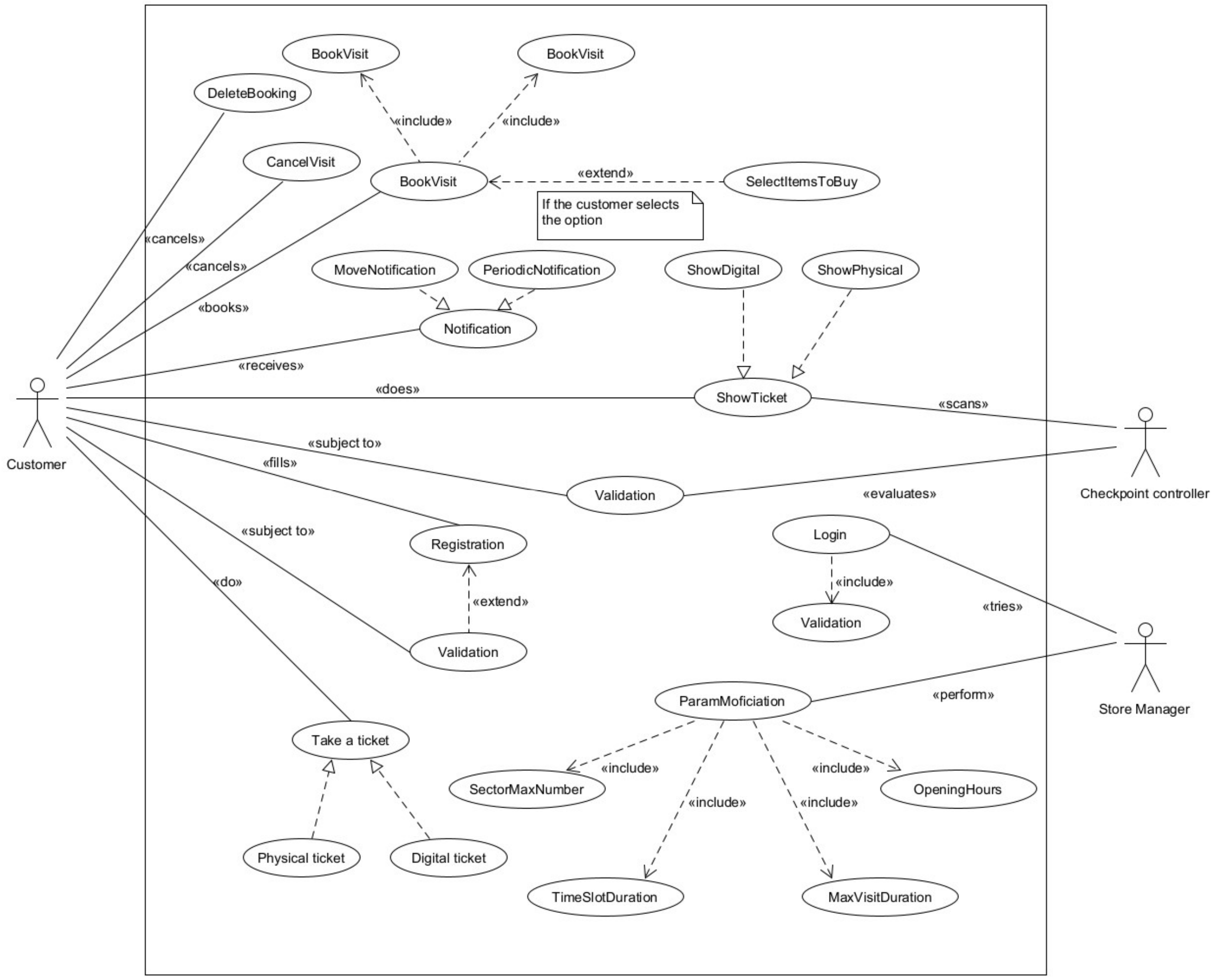
**Page 25: Inserted** **Marco Petri** **22/12/2020 18:02:00**

R21

**Page 26: Deleted** **Marco Petri** **22/12/2020 18:12:00**



**Page 26: Inserted** **Marco Petri** **22/12/2020 18:13:00**



**Page 26: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 26: Inserted** **asus** **18/12/2020 11:08:00**

1. .

**Page 26: Deleted** **asus** **18/12/2020 11:08:00**

: full name, birth date, username and a password.

**Page 26: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 26: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 27: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 27: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 27: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 27: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 27: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 27: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 27: Inserted** **asus** **18/12/2020 10:34:00**

1.

The system displays an overview of the current parameters of the selected store.

1.

**Page 27: Inserted** **asus** **18/12/2020 10:35:00**

Subsets of such parameters can be modified as well.

1.

**Page 27: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 27: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 27: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 28: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 28: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 28: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 28: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 28: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 28: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 28: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 28: Inserted** **asus** **18/12/2020 18:03:00**

1. Customer inserts their own data

**Page 28: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 28: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 28: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 28: Formatted** **Marco Petri** **22/12/2020 22:05:00**

English (United Kingdom)

**Page 29: Deleted** **Unknown**

**Page 29: Inserted** **asus** **18/12/2020 11:10:00**

1.

4.a

**Page 29: Deleted** **asus** **18/12/2020 11:09:00**

4.a

**Page 29: Inserted** **asus** **18/12/2020 10:37:00**

for another store (not for the same store)

**Page 29: Formatted** **Marco Petri** **22/12/2020 22:06:00**

English (United Kingdom)

**Page 29: Formatted** **Marco Petri** **22/12/2020 22:06:00**

English (United Kingdom)

**Page 29: Inserted** **asus** **18/12/2020 10:40:00**

Logged Customer already has a digital ticket for the selected store's queue. The request is denied and an error message is displayed..

**Page 29: Formatted** **Marco Petri** **22/12/2020 22:06:00**

English (United Kingdom)

**Page 29: Formatted** **Marco Petri** **22/12/2020 22:06:00**

English (United Kingdom)

**Page 29: Formatted** **Marco Petri** **22/12/2020 22:06:00**

English (United Kingdom)

**Page 30: Formatted** **Marco Petri** **22/12/2020 22:06:00**

English (United Kingdom)

**Page 30: Deleted** **asus** **18/12/2020 10:42:00**

selects a store from a digital map.

1.

**Page 30: Inserted** **asus** **18/12/2020 10:42:00**

1. selects the Book a Visit functionality.

A map containing all the available stores is shown.

1.

**Page 30: Deleted** **asus** **18/12/2020 10:43:00**

After selecting the store, Customer clicks the “Book a visit” button.

1.

**Page 30: Inserted** **asus** **18/12/2020 10:43:00**

Customer clicks on one of the stores.

1.

**Page 30: Inserted** **asus** **18/12/2020 10:44:00**

for the selected store

1.

**Page 30: Formatted** **Marco Petri** **22/12/2020 22:06:00**

English (United Kingdom)

**Page 30: Formatted** **Marco Petri** **22/12/2020 22:06:00**

English (United Kingdom)

**Page 30: Inserted** **asus** **18/12/2020 10:44:00**

Customer’s selected time slots overlap with a Visit that they have already booked. The Booking operation is denied and an error message is displayed.

**Page 30: Formatted** **Marco Petri** **22/12/2020 22:06:00**

English (United Kingdom)

**Page 30: Formatted** **Marco Petri** **22/12/2020 22:06:00**

English (United Kingdom)

**Page 30: Formatted** **Marco Petri** **22/12/2020 22:06:00**

English (United Kingdom)

**Page 30: Formatted** **Marco Petri** **22/12/2020 22:06:00**

English (United Kingdom)

**Page 30: Formatted** **Marco Petri** **22/12/2020 22:06:00**

English (United Kingdom)

**Page 30: Formatted** **Marco Petri** **22/12/2020 22:06:00**

English (United Kingdom)

**Page 31: Formatted** **Marco Petri** **22/12/2020 22:06:00**

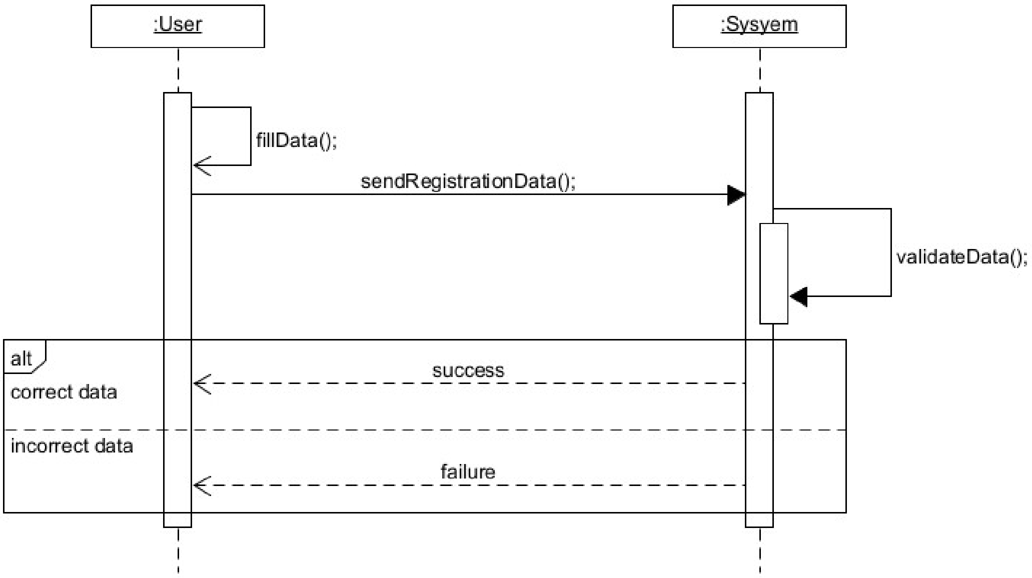
English (United Kingdom)

**Page 31: Formatted** **Marco Petri** **22/12/2020 20:12:00**

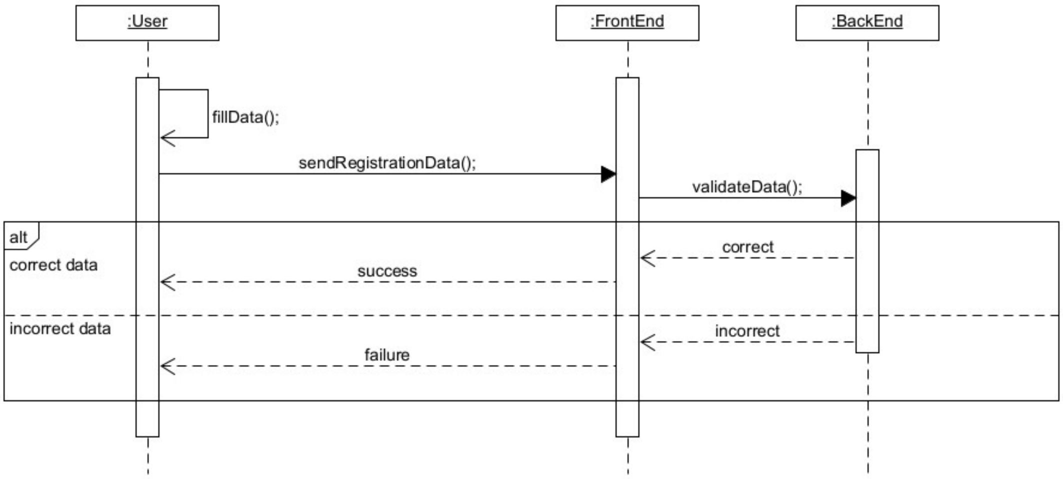
Justified

**Page 31: Deleted** **Marco Petri** **22/12/2020 20:12:00**

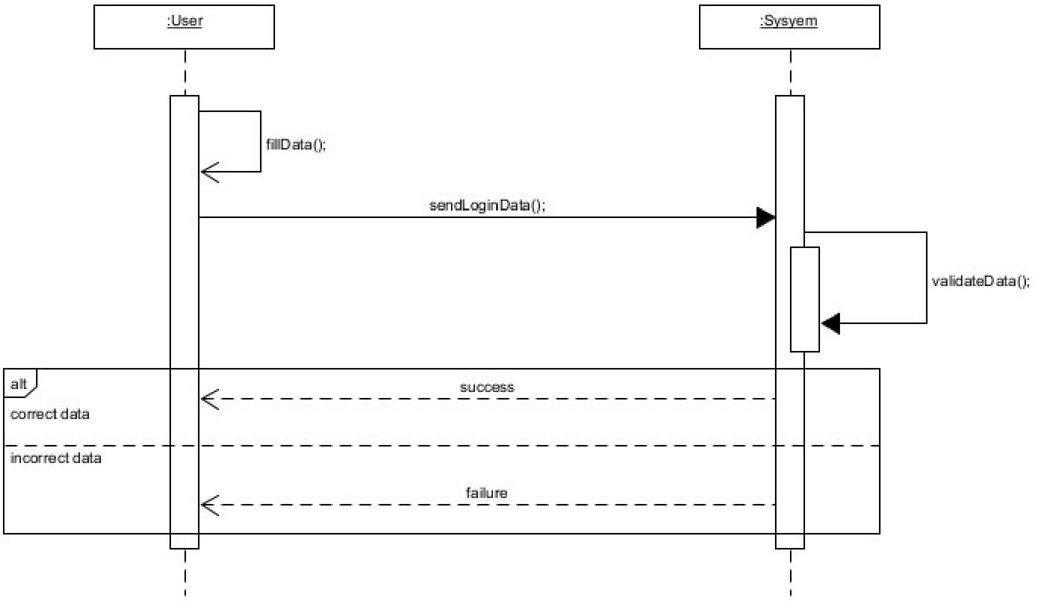
**Page 31: Inserted** **Marco Petri** **22/12/2020 21:28:00**



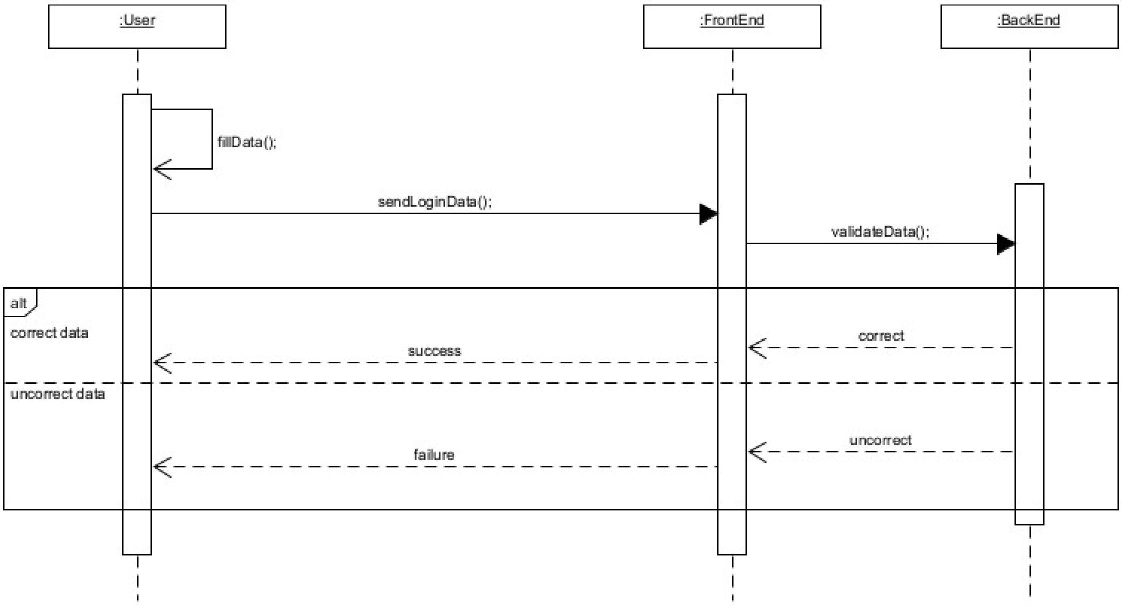
**Page 31: Deleted** **Marco Petri** **22/12/2020 20:12:00**



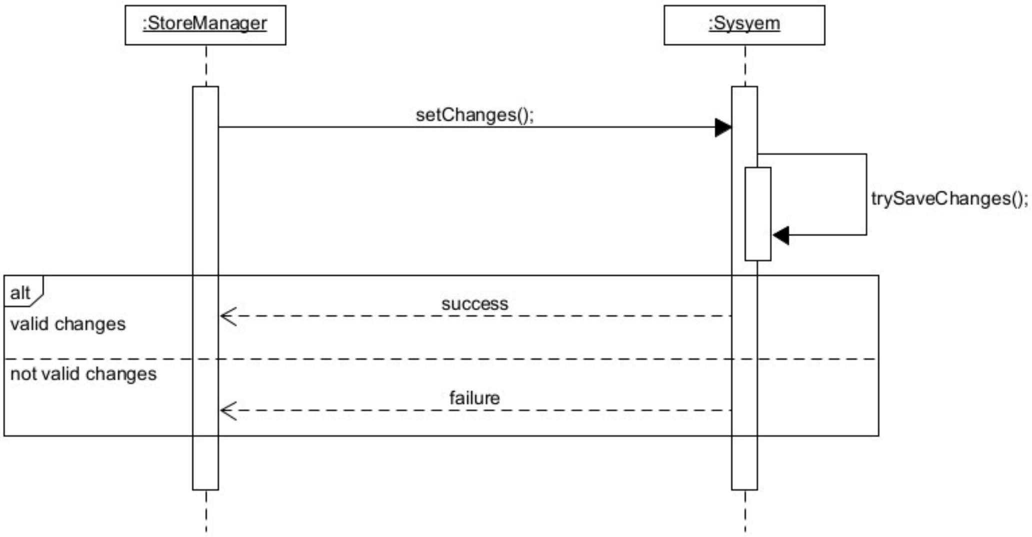
**Page 32: Inserted** **Marco Petri** **22/12/2020 21:28:00**



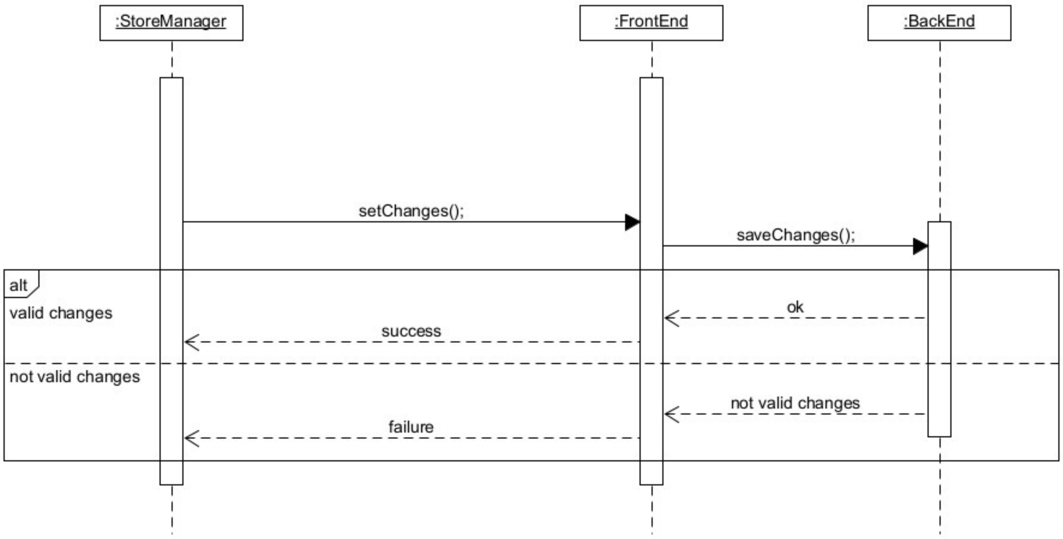
**Page 32: Deleted** Marco Petri 22/12/2020 20:12:00



**Page 32: Inserted** Marco Petri 22/12/2020 21:28:00

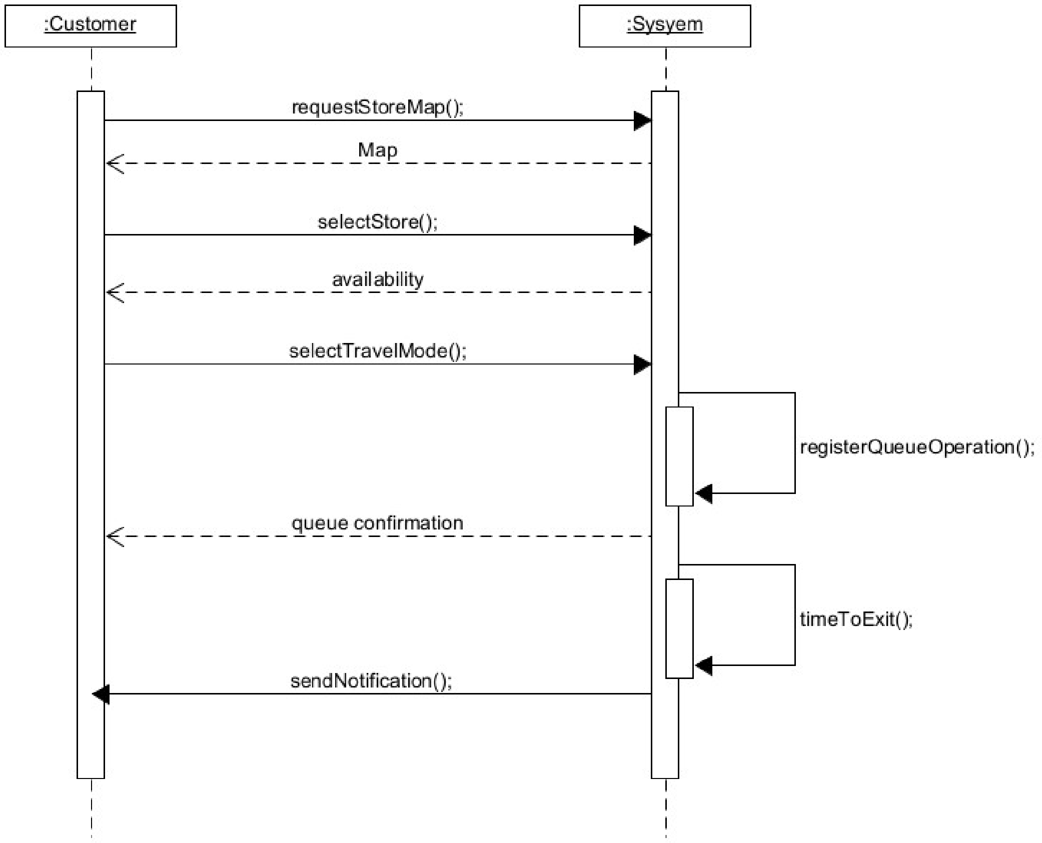


**Page 32: Deleted** **Marco Petri** **22/12/2020 20:12:00**

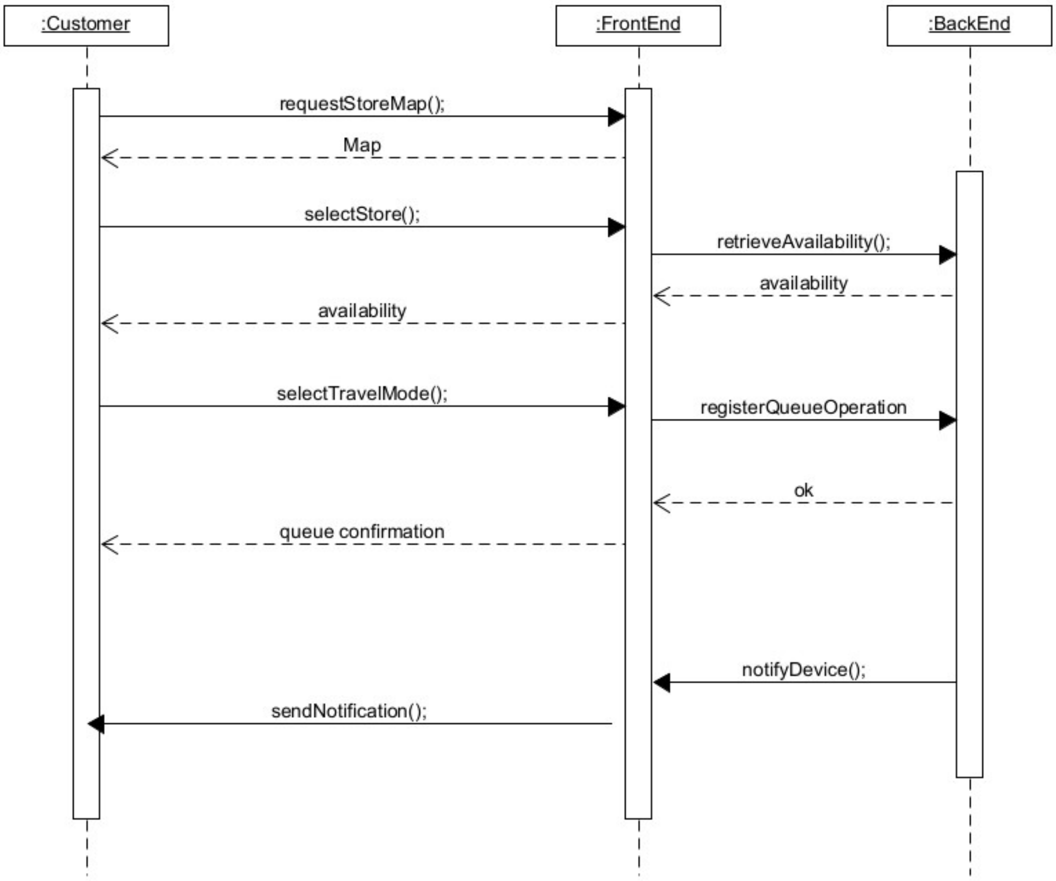


**Page 32: Inserted** **Marco Petri** **22/12/2020 21:35:00**

**Page 33: Inserted** **Marco Petri** **22/12/2020 21:29:00**

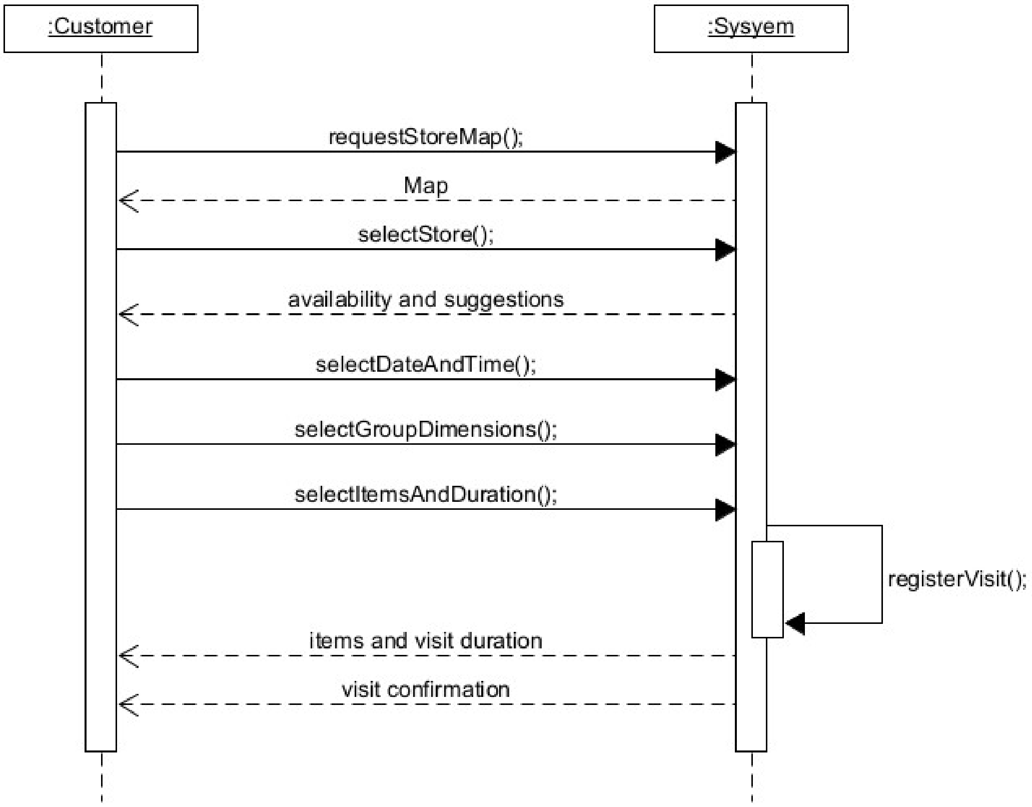


**Page 33: Deleted** **Marco Petri** **22/12/2020 20:12:00**

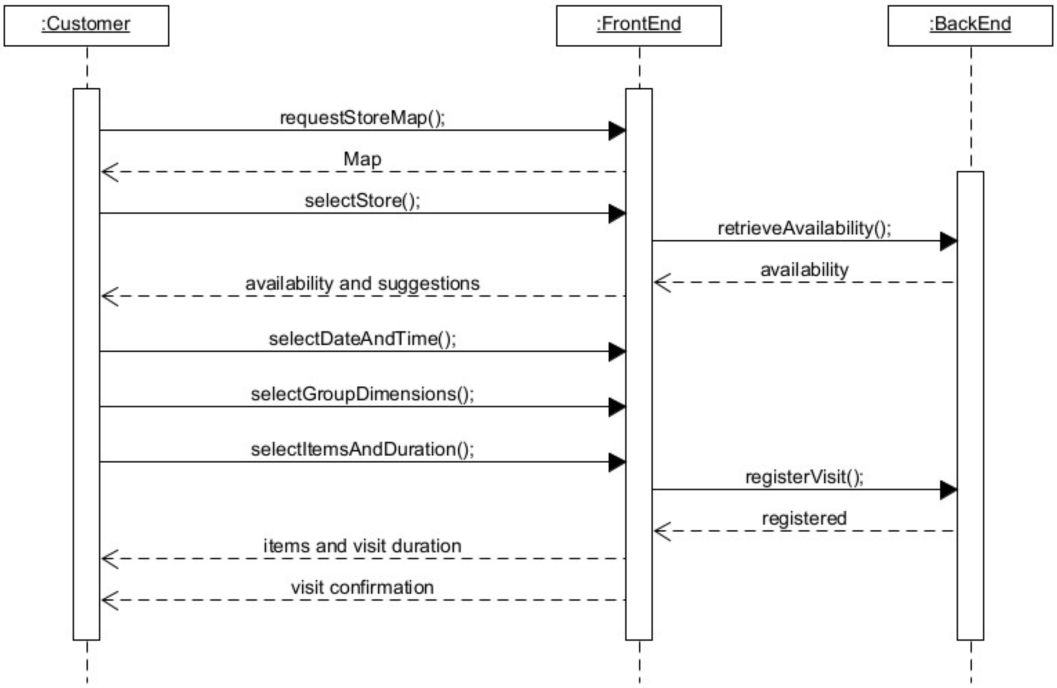


**Page 33: Inserted** **Marco Petri** **22/12/2020 21:35:00**

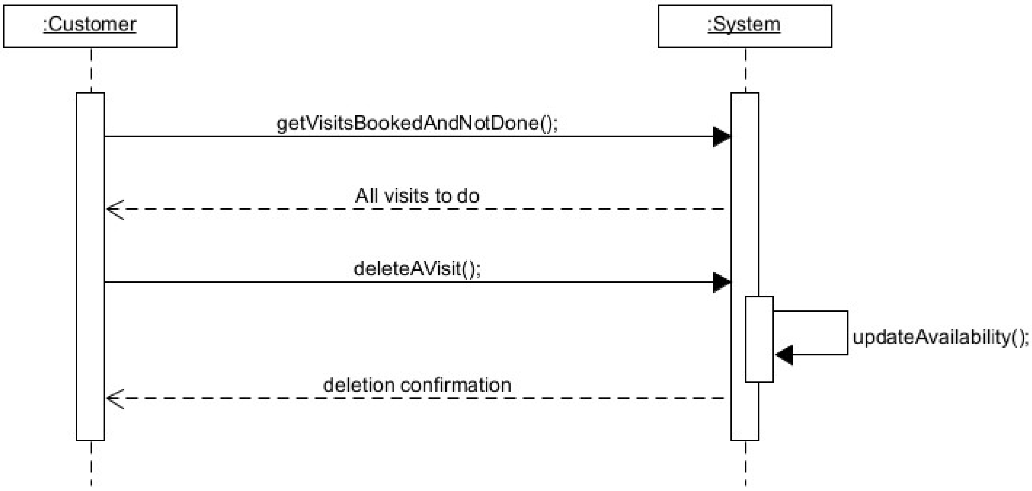
**Page 35: Inserted** **Marco Petri** **22/12/2020 21:29:00**



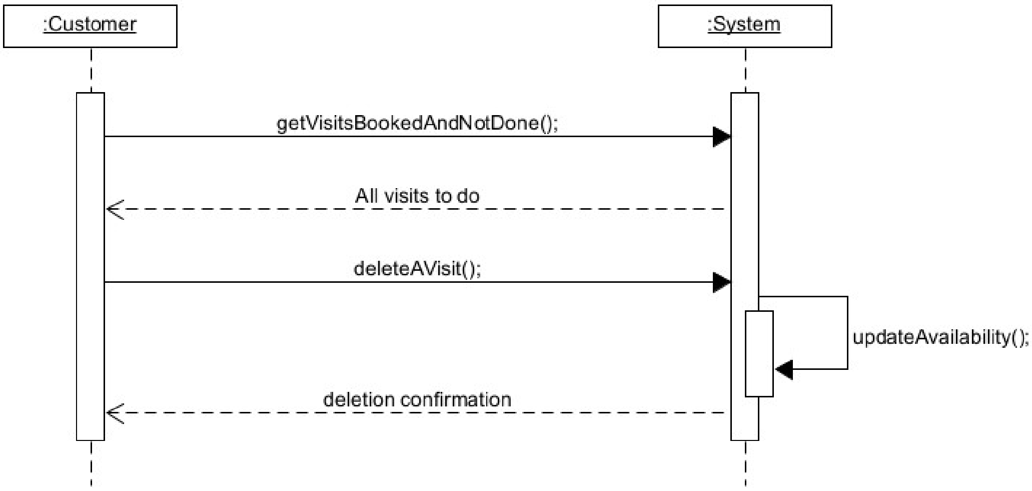
**Page 35: Deleted** **Marco Petri** **22/12/2020 20:12:00**



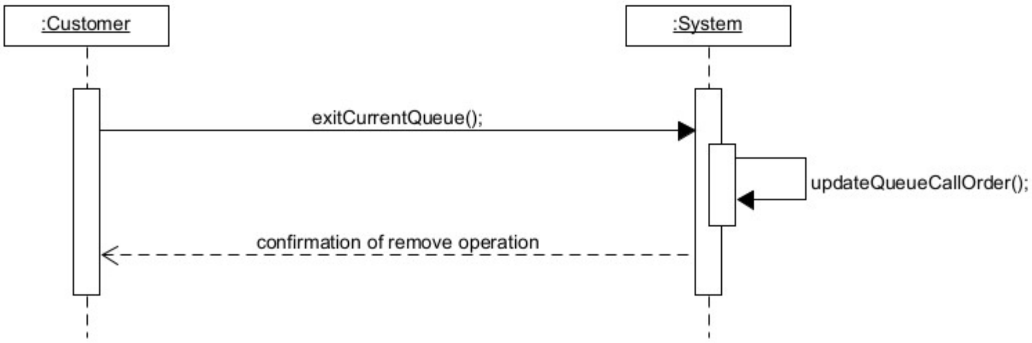
**Page 35: Deleted** **Marco Petri** **22/12/2020 21:29:00**



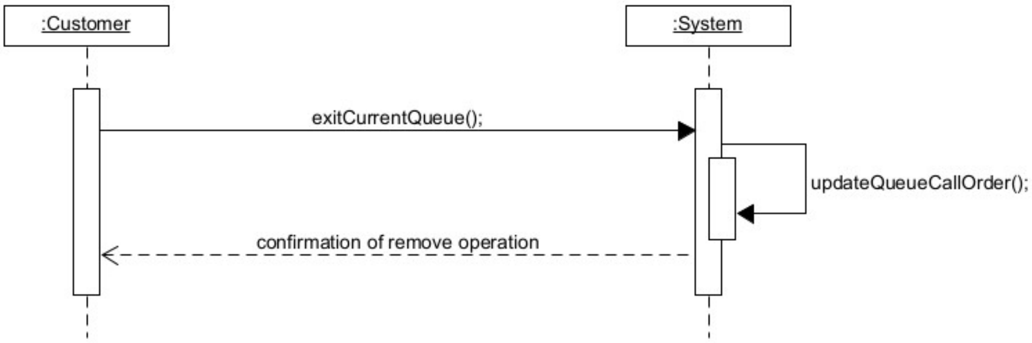
**Page 35: Inserted** **Marco Petri** **22/12/2020 21:29:00**



**Page 36: Deleted** **Marco Petri** **22/12/2020 21:29:00**



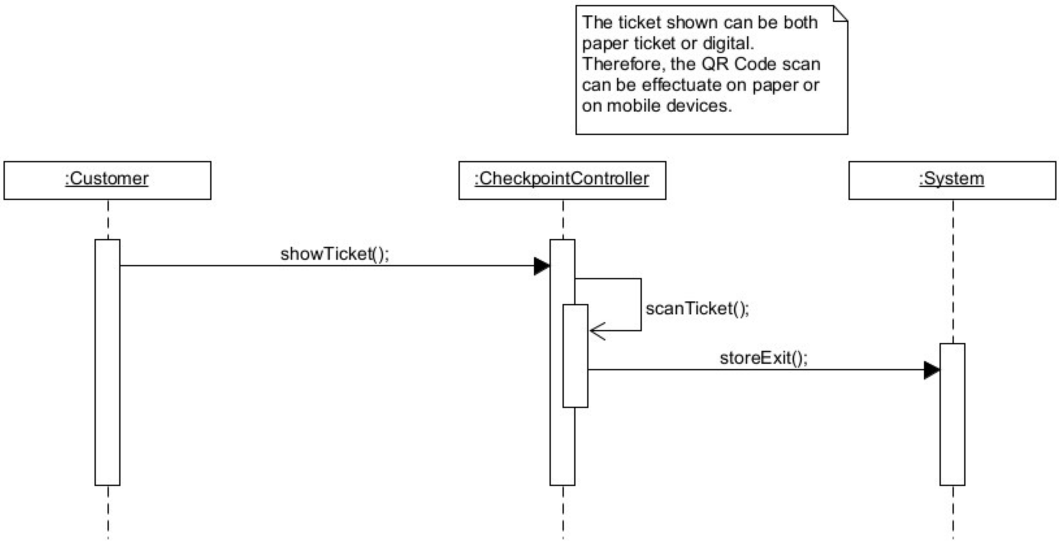
**Page 36: Inserted** **Marco Petri** **22/12/2020 21:30:00**



**Page 36: Inserted** **Marco Petri** **22/12/2020 18:15:00**

#### 1.1.1.1 Scanning at exit a customer

A checkpoint controller may scan a ticket of a customer who is about to exit the store.


The ticket shown can be both paper ticket or digital. Therefore, the QR Code scan can be effectuate on paper or on mobile devices.
:Customer
:CheckpointController
:System
showTicket();
scanTicket();
storeExit();

**Page 39: Deleted** **asus** **19/12/2020 12:26:00**

automatically

**Page 39: Inserted** **asus** **19/12/2020 12:26:00**

(such limit is set by the Store Manager)

**Page 40: Deleted** **asus** **19/12/2020 12:44:00**

reason

**Page 40: Inserted** **asus** **19/12/2020 12:44:00**

reason,

**Page 40: Deleted** **asus** **19/12/2020 12:27:00**

Furthermore

**Page 40: Inserted** **asus** **19/12/2020 12:27:00**

Furthermore,

**Page 40: Inserted** **asus** **19/12/2020 12:28:00**

**Page 40: Deleted** **asus** **19/12/2020 12:28:00**

**Page 41: Inserted** **asus** **22/12/2020 21:36:00**

**A**

**Page 41: Deleted** asus 22/12/2020 21:36:00

**a**

**Page 41: Deleted** Marco Petri 22/12/2020 19:59:00

Introductory text to the chapter (how it is subdivided and what are we going to say in the chapter)

**Page 41: Inserted** Marco Petri 22/12/2020 19:59:00

With

**Page 41: Inserted** asus 22/12/2020 21:36:00

A

**Page 41: Deleted** asus 22/12/2020 21:36:00

a

**Page 41: Inserted** Marco Petri 22/12/2020 19:59:00

alloy we want to verify some properties of the system. We introduce facts representing some assumptions on the system's configuration and we use time to denote the system's evolution

**Page 41: Inserted** asus 22/12/2020 21:44:00

i

**Page 41: Deleted** asus 22/12/2020 21:44:00

o

**Page 41: Inserted** Marco Petri 22/12/2020 20:04:00

on time. We want

**Page 41: Deleted** asus 22/12/2020 21:39:00

We want

**Page 41: Inserted** asus 22/12/2020 21:39:00

Our goal is

**Page 41: Inserted** Marco Petri 22/12/2020 20:01:00

to show

**Page 41: Inserted** asus 22/12/2020 21:39:00

that

**Page 41: Inserted** Marco Petri 22/12/2020 20:01:00

the system's properties are preserved by the operations

**Page 41: Inserted** asus 22/12/2020 21:45:00

that we defined in our predicates. Such operations include

**Page 41: Deleted** **asus** **22/12/2020 21:45:00**

of

**Page 41: Inserted** **Marco Petri** **22/12/2020 20:02:00**

of

**Page 41: Inserted** **asus** **22/12/2020 21:45:00**

:

**Page 41: Deleted** **asus** **22/12/2020 21:45:00**

**Page 41: Inserted** **Marco Petri** **22/12/2020 20:02:00**

getting in and out

**Page 41: Inserted** **asus** **22/12/2020 21:39:00**

of

**Page 41: Deleted** **asus** **22/12/2020 21:39:00**

from

**Page 41: Inserted** **Marco Petri** **22/12/2020 20:02:00**

from a queue

**Page 41: Inserted** **asus** **22/12/2020 21:45:00**

, entering or leaving

**Page 41: Deleted** **asus** **22/12/2020 21:45:00**

and

**Page 41: Inserted** **Marco Petri** **22/12/2020 20:02:00**

and a store

**Page 41: Inserted** **asus** **22/12/2020 21:38:00**

, booking or deleting visits and printing tickets

**Page 41: Inserted** **Marco Petri** **22/12/2020 20:02:00**

. Getting in and out

**Page 41: Deleted** **asus** **22/12/2020 21:40:00**

Getting in and out

**Page 41: Inserted** **asus** **22/12/2020 21:46:00**

We aim to prove that these operations

**Page 41: Inserted** **Marco Petri** **22/12/2020 20:03:00**

must

**Page 41: Deleted** **asus** **22/12/2020 21:46:00**

must

**Page 41: Inserted** **asus** **22/12/2020 21:46:00**

do

**Page 41: Inserted** **Marco Petri** **22/12/2020 20:03:00**

not violate some invariants of the system (e.g. customers controlled at the exit are always equal or less

**Page 41: Deleted** **asus** **22/12/2020 21:46:00**

equal or less

**Page 41: Inserted** **asus** **22/12/2020 21:46:00**

less or equal

**Page 41: Inserted** **Marco Petri** **22/12/2020 20:04:00**

than the one

**Page 41: Inserted** **asus** **22/12/2020 21:46:00**

s

**Page 41: Inserted** **Marco Petri** **22/12/2020 20:04:00**

controlled at the entrance). To do so we use assertions to verify that

**Page 41: Inserted** **asus** **22/12/2020 21:50:00**

,

**Page 41: Inserted** **Marco Petri** **22/12/2020 20:03:00**

given the facts and the predicate

**Page 41: Inserted** **asus** **22/12/2020 21:50:00**

,

**Page 41: Deleted** **asus** **22/12/2020 21:49:00**

**Page 41: Inserted** **Marco Petri** **22/12/2020 20:03:00**

**Page 41: Inserted** **asus** **22/12/2020 21:48:00**

performing an operation on a consistent state leads to another consistent state.

**Page 41: Deleted** **asus** **22/12/2020 21:48:00**

we obtain a consistent system starting from a consistent one and by the application of a function. We use predicates to model the dynamics of the system.

**Page 41: Inserted** **Marco Petri** **22/12/2020 20:03:00**

we obtain a consistent system starting from a consistent one and by the application of a function. We use predicates to model the dynamics of the system.

Page 41: Deleted Marco Petri 22/12/2020 20:05:00

Page 41: Inserted asus 22/12/2020 21:29:00

Our code uses some built in libraries that deal with ordering and time. Such libraries require signatures to be **exact**, or else the predicates using them will not run properly. For this reasonreason, we used **run commands** in which we specify many exact signatures.

Page 41: Formatted Marco Petri 22/12/2020 21:35:00

Space Before: 12 pt

Page 41: Deleted asus 22/12/2020 21:36:00

reason

Page 41: Deleted Marco Petri 22/12/2020 21:35:00

Page 41: Formatted Marco Petri 22/12/2020 20:06:00

Heading 2, Adjust space between Latin and Asian text, Adjust space between Asian text and numbers

Page 41: Deleted Marco Petri 22/12/2020 20:06:00

```
sig Store {
```

Page 41: Inserted aa m 08/12/2020 17:34:00

```
sig Store {
```

1.1

Page 41: Formatted Marco Petri 22/12/2020 20:06:00

Font: (Default) Courier New, 11 pt

Page 41: Inserted Marco Petri 22/12/2020 19:46:00

## 1.1

```
open util/ordering[QueueTicket]
open util/time

abstract sig Person {}
abstract sig Ticket {
    ticketOwner: one Customer,
}

// general entities
sig DateTime {}

// entities which extend Person
sig StoreManager extends Person {}
sig Customer extends Person {}
sig CheckpointController extends Person {
    //tickets admitted into the store
    controllerCheckIns: dynamicSet[Ticket],
    //tickets scanned at the exit
    controllerCheckOuts: dynamicSet[Ticket],
}

// entities extending ticket
sig BookingTicket extends Ticket {}
sig QueueTicket extends Ticket {}

// store's signature
sig Store {
    storeManagers: some StoreManager,
    storeControllers: some CheckpointController,
    storeTicketMachines: some TicketMachine,
    storeProducts: some Item,
    //customers currently inside the store
    storeCustomersInStore: dynamicSet[Customer],
    //current queue for the store
    storeQueue: one Queue,
    //current bookings for the store
    storeBookings: dynamicSet[Booking],
    /* tickets that have been used to enter the store, but not necessarely
        to exit */
    storeUsedTickets: dynamicSet[Ticket],
    /* tickets that have not been used to enter the store, and are no more
         valid */
    storeNotUsedInvalidTickets: dynamicSet[Ticket]
}{
    #storeControllers > 0
    #storeTicketMachines > 0
}

sig TicketMachine {
    machinePrintedTickets: dynamicSet[QueueTicket]
}

sig Category {}

sig Item {
    itemCategory: one Category
}

// bookings and queues
sig Booking {
    bookingTicket: one BookingTicket,
    bookingDateTime: one DateTime,
    bookingItems: set Item,
    bookingCategories: set Category
}

sig Queue {
    queueTickets: dynamicSet[QueueTicket],
}

/************************
 *                      *
 *                      *
 *        FACTS         *
 *                      *
 *                      *
 ***********************/

/*-------------------------------------------------------------------------
```

```
    Ubiquity Facts: in this section we include facts stating that some
    entities cannot be shared by other entities.
    For instance: customers cannot be simoultaneously inside two stores
  ------------------------------------------------------------------------*/
/* No customer has superpowers that allow them to be in two stores at the
    same time */
fact noUbiquitySuperpowers{
    all t:Time | no disj s1,s2:Store |
        #(s1.storeCustomersInStore.t & s2.storeCustomersInStore.t) > 0
}

//No queue is owned by two stores
fact noSharedQueues{
    all  q:Queue | one s:Store | q in s.storeQueue
}

//No controller is owned by two stores
fact noSharedControllers{
    all  chk:CheckpointController | one s:Store | chk in s.storeControllers
}

//No ticket machine is shared by two stores
fact noSharedTicketMachine{
    all  tm:TicketMachine | one s:Store | tm in s.storeTicketMachines
}

//Each booking ticket is for one booking only
fact noSharedBookingTicket{
    all bt:BookingTicket | one b:Booking | b.bookingTicket = bt
}

//Every ticket is for only one store
fact ticketsAreForAtMostOneStore {
    all t: Ticket, time:Time | one s: Store |
        t in (s.storeUsedTickets.time + s.storeNotUsedInvalidTickets.time +
        s.storeBookings.time.bookingTicket + s.storeQueue.queueTickets.time)
}

//Every booking is for only one store
fact bookingsAreForAtMostOneStore {
    all b: Booking, t:Time| lone s: Store | b in s.storeBookings.t
}


fact ticketMustBeOnlyOfOneType {
    //tickets can either be:
    all t: Ticket, s: Store, time:Time |
        //used
        (t in s.storeUsedTickets.time implies
            t not in (s.storeNotUsedInvalidTickets.time +
            s.storeQueue.queueTickets.time +
            s.storeBookings.time.bookingTicket))
        and
        //not used and invalid
        (t in s.storeNotUsedInvalidTickets.time implies
            t not in (s.storeUsedTickets.time + s.storeQueue.queueTickets.time
            + s.storeBookings.time.bookingTicket))
        and
        //valid for booking
        (t in s.storeBookings.time.bookingTicket implies
            t not in (s.storeNotUsedInvalidTickets.time +
            s.storeQueue.queueTickets.time + s.storeUsedTickets.time))
        and
        //valid for queueing
        (t in s.storeQueue.queueTickets.time implies
            t not in (s.storeNotUsedInvalidTickets.time +
            s.storeUsedTickets.time + s.storeBookings.time.bookingTicket))
}

//Tickets may be printed at most once
fact ticketMayBePrintedByAtMostOneTicketMachine {
    all t: Ticket, time:Time| lone tm: TicketMachine |
        t in tm.machinePrintedTickets.time
```

```
}

/* A customer cannot be in a queue and at the same time in the store owning
    that queue */
fact queueCannotContainInStoreCustomer {
    all s: Store, t:Time | no c: Customer |
        c in s.storeCustomersInStore.t and
        c in s.storeQueue.queueTickets.t.ticketOwner
}

/*---------------------------------------------------------------------
    No random changes : the following facts state that no new entities
    appear in time unless an operation makes them appear.
    For instance, no new customer can appear in a store unless someone
    entered
----------------------------------------------------------------------*/
//A new customer appears iff someone enters or exits
fact CustomersDontMultiplyRandomly{
    all t:Time, s:Store | (some c:Customer |
        c in s.storeCustomersInStore.(t.next)
        and c not in s.storeCustomersInStore.(t) )
        iff
        ((some qt:QueueTicket |enterStoreQueue[s,qt,t]) or
        (some b:Booking | enterStoreBooking[s,b,t]))
}

//A customer disappers iff somedy left the store
fact CustomersDontVanishRandomly{
    all t:Time, s:Store | (some c:Customer | c in  s.storeCustomersInStore.(t)
        and c not in s.storeCustomersInStore.(t.next) )
        iff
        ( some tick:Ticket | leaveStore[s,tick,t])
}

//A ticket leaves a queue iff somebody left the queue or entered the store
fact queueticketsDontDisappearRandomly{
    all t:Time, s:Store |
        (some qt:QueueTicket |
            qt in  s.storeQueue.queueTickets.t and
            qt not in s.storeQueue.queueTickets.(t.next))
        iff
        (some qt:QueueTicket, c:Customer |
            leaveQueue[s.storeQueue,c, qt,t] or enterStoreQueue[s, qt,t ])
}

//A ticket appears in a queue iff somebody joined the queue
fact queueticketsDontAppearRandomly{
    all t:Time, s:Store |
        (some qt:QueueTicket |
            qt in  s.storeQueue.queueTickets.(t.next) and
            qt not in s.storeQueue.queueTickets.(t) )
                iff
            (some qt:QueueTicket, c:Customer | joinQueue[s.storeQueue,c,qt,t])
}

//Bookings appear iff a new visit is booked
fact bookingsDontAppearRandomly{
    all t:Time, s:Store |
        (some b:Booking|
            b in s.storeBookings.(t.next) and b not in s.storeBookings.(t) )
                iff
            ( some b:Booking, c:Customer | bookVisit[s,b,c,t])
}

/*Bookings disappear iff a visit is deleted or somebody enters the store
    by using a booking */
fact bookingsDontAppearRandomly{
    all t:Time, s:Store |
        (some b:Booking|
            b in  s.storeBookings.(t) and b not in s.storeBookings.(t.next))
        iff
        (some b:Booking, c:Customer |
            deleteAVisit[s,c,b,t] or enterStoreBooking[s,b,t])
```

```
}

//A new control appears iff somebody entered the store
fact ControlsDontMultiplyRandomly{
    all t:Time, s:Store |
        (some tick:Ticket |
            tick in  s.storeControllers.controllerCheckIns.(t.next) and
            tick not in s.storeControllers.controllerCheckIns.t )
        iff
        ( (some qt:QueueTicket |
            enterStoreQueue[s,qt,t]) or
            ( some b:Booking | enterStoreBooking[s,b,t]) )
}

/*--------------------------------------------------------------------
    No Useless Entities: the following facts state that if
    an entity exists, it must belong somewhere, i.e. we
    do not want isolted/useless entities. For instance, we can sureòy
    have unemployed Store Managers in real life, but they are not
    relevant for our analysis, therefore we include these facts to
    avoid their generation
----------------------------------------------------------------------*/
//Store managers manage at least one store
fact storeManagersManageAtLeastOneStore {
    all sm: StoreManager | some s: Store | sm in s.storeManagers
}

//Items are at least in one store
fact itemsAreAtLeastInOneStore {
    all i: Item | some s: Store | i in s.storeProducts
}

//Categories contain at least one item
fact categoriesContainsAtLeastOneItem {
    all c: Category | some i: Item | c = i.itemCategory
}


/* If tickets are printed, they are printed by a machine of the store they
    are for */
fact printedTicketsAreInQueueOrAreUsed {
    all t: Ticket , time:Time, s: Store |
        t in s.storeTicketMachines.machinePrintedTickets.time
        implies
        (t in s.storeQueue.queueTickets.time or
        t in s.storeNotUsedInvalidTickets.time
        or t in s.storeUsedTickets.time)
}

//All used tickets have been checked at entrance
fact usedTicketsHaveBeenScanned {
    all t: Ticket, s: Store , time:Time|
        t in s.storeUsedTickets.time implies
        t in s.storeControllers.controllerCheckIns.time

    all t: Ticket, time:Time, s: Store |
        t in s.storeControllers.controllerCheckIns.time
        implies
        (t in s.storeUsedTickets.time or
        t in s.storeNotUsedInvalidTickets.time or
        t in s.storeBookings.time.bookingTicket)
}

/************************
 *                      *
 *                      *
 *      PREDICATES      *
 *                      *
 *                      *
 ***********************/

//1. Join a queue
pred joinQueue[q:Queue, c:Customer, qt:QueueTicket, t:Time]{
    //preconditions
    #q.queueTickets.t > 0
    //c owns the ticket
    qt.ticketOwner = c
    //the ticket is not already in the queue
    qt not in q.queueTickets.t
    //customer is not already in the queue
    no tick:QueueTicket |
        tick in q.queueTickets.t and tick.ticketOwner = c
    //the ticket must not have been used
    no chk:CheckpointController |
        qt in (chk.controllerCheckIns.t + chk.controllerCheckOuts.t)

    //postconditions
    //the ticket is now in the queue
    all ticket:QueueTicket |
        (ticket in q.queueTickets.(t.next))
        iff
        (ticket in q.queueTickets.t or ticket = qt)
    //the new ticket is greater than any other ticket in the queue
    all ticket:QueueTicket |
        ticket in q.queueTickets.(t) => lt[ticket,qt]
}

//2. Leave a queue
pred leaveQueue[q:Queue, c:Customer, qt:QueueTicket, t:Time]{
    //preconditions
    qt.ticketOwner = c
    qt in q.queueTickets.t
    //the ticket must not have been used
    no chk:CheckpointController |
        qt in (chk.controllerCheckIns.t + chk.controllerCheckOuts.t)

    //postconditions
    //the ticket is no longer in the queue
    all ticket:QueueTicket |
        (ticket in q.queueTickets.(t.next))
        iff
        (ticket in q.queueTickets.t and ticket != qt)
}

//3. Book a visit
pred bookVisit[s:Store, b:Booking, c:Customer, t:Time]{
    //preconditions
    /*the customer must own the new booking*/
    b.bookingTicket.ticketOwner = c
    /*the new booking cannot overlap with any already existing booking for s*/
    no ovlpBooking: Booking |
        ovlpBooking in s.storeBookings.t and
        ovlpBooking.bookingDateTime = b.bookingDateTime and
        b.bookingTicket.ticketOwner = ovlpBooking.bookingTicket.ticketOwner
    /*the new booking cannot overlap with any already existing booking for
    any other store*/
    all s2:Store |
        s != s2 implies
        (no ovlpBooking: Booking |
            ovlpBooking.bookingTicket.ticketOwner = c and
            ovlpBooking.bookingDateTime = b.bookingDateTime and
            ovlpBooking in s2.storeBookings.t)

    //postconditions
    /*at time t.next, the store will only have the bookings that it had
        before + b*/
    all book:Booking |
        (book in s.storeBookings.(t.next))
        iff
        (book = b or book in s.storeBookings.t)
}

//4. Delete a visit
pred deleteAVisit[ s:Store, c:Customer, b:Booking, t:Time] {
    // preconditions
    /*b must be in the store before*/
    b in s.storeBookings.t
    /*b's customer must be c*/
    c = b.bookingTicket.ticketOwner

    // postconditions
    /*if a booking was in the store before and is different from b, it
    is in the store afterwards */
    all book:Booking |
        (book in s.storeBookings.t and book != b)
        iff
        book in s.storeBookings.(t.next)

}

//5.Enter a store from its queue
pred enterStoreQueue[s:Store, qt:QueueTicket, t:Time]{
    //preconditions
    /*entering customer is in the queue for s*/
    qt in s.storeQueue.queueTickets.t
    /*qt is the first of the queue*/
    all ticket:QueueTicket |
        (ticket in s.storeQueue.queueTickets.t and ticket != qt)
        implies
        lt[qt,ticket]
    /*the ticket must not have been used*/
    no chk:CheckpointController |
        qt in (chk.controllerCheckIns.t + chk.controllerCheckOuts.t)

    //postconditions
    /*the owner of the ticket is now in the store*/
    all c:Customer |
        c in s.storeCustomersInStore.(t.next)
        iff
        (c=qt.ticketOwner or c in s.storeCustomersInStore.t)
    /*the owner of the ticket is removed from the queue*/
    all ticket:QueueTicket |
        (ticket in s.storeQueue.queueTickets.(t.next))
        iff
        (ticket in s.storeQueue.queueTickets.t and ticket != qt)
    one chk:CheckpointController |
        qt in chk.controllerCheckIns.(t.next) and
        chk in s.storeControllers
    no chk:CheckpointController |
        qt in chk.controllerCheckIns.(t.next) and
        chk not in s.storeControllers
    all chk:CheckpointController |
        chk.controllerCheckOuts.(t.next) = chk.controllerCheckOuts.(t)
    all chk:CheckpointController |
        qt not in chk.controllerCheckIns.(t.next)
        iff
        chk.controllerCheckIns.(t.next) = chk.controllerCheckIns.(t)
    all chk:CheckpointController |
        qt  in chk.controllerCheckIns.(t.next)
        iff
        chk.controllerCheckIns.(t.next) = chk.controllerCheckIns.(t) + qt

}

//6. Enter a store by booking
pred enterStoreBooking[s:Store, b:Booking, t:Time]{

    //preconditions
    /*is an active booking at the time of entering*/
    b in s.storeBookings.t
    /*the ticket related to b must not have been used*/
    no chk:CheckpointController |
        b.bookingTicket in (chk.controllerCheckIns.t +
            chk.controllerCheckOuts.t)

    //postconditions
    /*the booking is removed from the active bookings*/
    all book:Booking | book in s.storeBookings.(t.next)
        iff
        ( book != b and book in s.storeBookings.t)
    /*the owner of the ticket is now in the store*/
    all c:Customer |
        c in s.storeCustomersInStore.(t.next)
        iff
        (c=b.bookingTicket.ticketOwner or c in s.storeCustomersInStore.t)
    one chk:CheckpointController |
        b.bookingTicket in chk.controllerCheckIns.(t.next) and
        chk in s.storeControllers and
        chk in s.storeControllers
    no chk:CheckpointController |
        b.bookingTicket in chk.controllerCheckIns.(t.next) and
        chk not in s.storeControllers
    all chk:CheckpointController |
        chk.controllerCheckOuts.(t.next) = chk.controllerCheckOuts.(t)
    all chk:CheckpointController |
        b.bookingTicket not in chk.controllerCheckIns.(t.next)
        iff
        chk.controllerCheckIns.(t.next) = chk.controllerCheckIns.(t)
    all chk:CheckpointController |
        b.bookingTicket in chk.controllerCheckIns.(t.next)
        iff
        chk.controllerCheckIns.(t.next) = chk.controllerCheckIns.(t) +
                                          b.bookingTicket
}

//7.Leave a store
pred leaveStore[s:Store, tick:Ticket, t:Time]{
    //preconditions
    /*the owner of tick is in the store*/
    tick.ticketOwner in s.storeCustomersInStore.t
    tick in s.storeControllers.controllerCheckIns.t

    //postconditions
    /*the customer is no longer in the store*/
    all c:Customer |
        c in s.storeCustomersInStore.(t.next)
        iff
        ( c != tick.ticketOwner and c in s.storeCustomersInStore.(t))
    /*customer has been checked at the exit*/
    one chk:CheckpointController |
        tick in chk.controllerCheckOuts.(t.next) and
        chk in s.storeControllers
    no chk:CheckpointController |
        tick in chk.controllerCheckOuts.(t.next) and
        chk not in s.storeControllers
    all chk:CheckpointController |
        chk.controllerCheckIns.(t.next) = chk.controllerCheckIns.(t)
    all chk:CheckpointController |
        tick not in chk.controllerCheckOuts.(t.next)
        implies
        chk.controllerCheckOuts.(t.next) = chk.controllerCheckOuts.(t)
    all chk:CheckpointController |
        tick  in chk.controllerCheckOuts.(t.next)
        implies
        chk.controllerCheckOuts.(t.next) = chk.controllerCheckOuts.(t) + tick
}

//8. Print a ticket for a store
pred printTicket[s:Store, qt:QueueTicket ,c:Customer, t:Time]{
    //preconditions
    qt.ticketOwner = c
    c not in s.storeCustomersInStore.t
    qt not in s.storeTicketMachines.machinePrintedTickets.t
    qt not in s.storeQueue.queueTickets.t
    qt not in s.storeControllers.controllerCheckIns.t
    qt not in s.storeControllers.controllerCheckOuts.t

    //postconditions
    s.storeTicketMachines.machinePrintedTickets.(t.next) =
        s.storeTicketMachines.machinePrintedTickets .t + qt
    s.storeQueue.queueTickets.(t.next) = s.storeQueue.queueTickets.t + qt
    all qn1: QueueTicket |
        (qn1 in s.storeQueue.queueTickets.(t.next) and qn1 != qt )
        implies
        lt[qn1, qt]
}

/************************
 *                      *
 *                      *
 *      AUXILIARY       *
 *      PREDICATES      *
 *                      *
 *                      *
 ************************/

pred checkoutSubsetCheckin[s:Store, t:Time]{
     s.storeControllers.controllerCheckOuts.t
        in
    s.storeControllers.controllerCheckIns.t
}

pred storeOutSubsetIns[s:Store, t:Time]{
    s.storeControllers.controllerCheckOuts.t
        in
    s.storeControllers.controllerCheckIns.t
}

pred noCustomerAppearsTwice[q:Queue, t:Time]{
    no disj qt1, qt2: QueueTicket |
        qt1 in q.queueTickets.t and
        qt2 in q.queueTickets.t and
        qt1.ticketOwner = qt2.ticketOwner
}

pred disjointQueues[q1,q2:Queue, t:Time]{

    no c:Customer |
        c in q1.queueTickets.t.ticketOwner and
        c in q2.queueTickets.t.ticketOwner
}

pred allGuestsChecked[s:Store, t:Time]{
    all c:Customer |
        c in s.storeCustomersInStore.t
        implies
        (some chk:CheckpointController |
            c in chk.controllerCheckIns.t.ticketOwner)
}

pred hasOverlappingBooking[ s:Store, t:Time]{
    some disj b1,b2:Booking |
        b1 in s.storeBookings.t and
        b2 in s.storeBookings.t and
        b1.bookingDateTime = b2.bookingDateTime and
        b1.bookingTicket.ticketOwner = b2.bookingTicket.ticketOwner
}

pred haveOverlappingBooking[ s1,s2:Store, t:Time]{
    some disj b1,b2:Booking |
        b1 in s1.storeBookings.t and
        b2 in s2.storeBookings.t and
        b1.bookingDateTime = b2.bookingDateTime and
        b1.bookingTicket.ticketOwner = b2.bookingTicket.ticketOwner
}

pred checkedInButNotOut[c:Customer, s:Store, t:Time]{
    some tick:Ticket |
        tick.ticketOwner = c and
        tick in s.storeControllers.controllerCheckIns.t and
        tick not in s.storeControllers.controllerCheckOuts.t
}

/************************
 *                      *
 *                      *
 *      ASSERTIONS      *
 *                      *
 *                      *
 ************************/
/* Our assertions are aimed at proving that the operations that were defined
    in the predicates do not alter the consistency of a state.
    At first, we prove that the chosen property is respected in a basic case.
    Then, we show that if we are in a state that respects the property, and
    we perform some operation on the entity we are working with, the state in
    the following time will also respect such property.
    For more details, please check the reference section.
*/

/*This assertion proves that at any time and for every controller, the
    tickets that they checkedOut are a subset of those they checkedIn */
assert CheckoutSubsetCheckin{
    /*Base Case: if no one was controlled for a store S, the property holds*/
    all s:Store, t:Time |
        #s.storeControllers.controllerCheckOuts.t = 0 and
        #s.storeControllers.controllerCheckIns.t = 0
            implies
        storeOutSubsetIns[s,t]

    /*Inductive Steps: if we start from a consistent state, and some
    customer enters the Store or leaves it, the property still holds*/
    all s:Store, t:Time |
        storeOutSubsetIns[s,t] and
        (some qt:QueueTicket | enterStoreQueue[s,qt,t])
            implies
        storeOutSubsetIns[s,t.next]
    all s:Store, t:Time |
        storeOutSubsetIns[s,t] and
        (some b:Booking| enterStoreBooking[s,b,t])
            implies
        storeOutSubsetIns[s,t.next]
    all s:Store, t:Time |
        storeOutSubsetIns[s,t] and
        (some tick:Ticket | leaveStore[s,tick,t])
            implies
        storeOutSubsetIns[s,t.next]
}

//This assertion proves that no customer can appear twice in the same queue
assert NoCustomerTwiceSameQueue{
    /*Base Case: the property holds for an empty queue*/
    all q:Queue, t:Time |
        #q.queueTickets.t = 0
        implies
        noCustomerAppearsTwice[q,t]

    /*Inductive Steps: if we start from a consistent queue and some customer
    joins the queue or leaves it, the property still holds*/
    all q:Queue, t:Time |
        noCustomerAppearsTwice[q,t] and
        (some c:Customer, qt:QueueTicket | joinQueue[q,c,qt,t])
            implies
        noCustomerAppearsTwice[q,t.next]
    all q:Queue, t:Time |
        noCustomerAppearsTwice[q,t] and
        (some c:Customer, qt:QueueTicket | leaveQueue[q,c,qt,t])
            implies
        noCustomerAppearsTwice[q,t.next]
}

//This assertion proves that no customer can be twice in different queues
assert NoTwoQueuesShareCustomer{
    /*Base Case: the property holds for two empty queues*/
    all q1,q2:Queue , t:Time |
        #q1.queueTickets=0 and
        #q2.queueTickets=0
            implies
        disjointQueues[q1,q2,t]
    /*Inductive Steps: if we start from two consistent queue sand some
    customer joins on of the queues or leaves it, the property still holds*/
    all q1,q2:Queue , t:Time |
        disjointQueues[q1,q2,t] and
        (some c:Customer, qt:QueueTicket | joinQueue[q1,c,qt,t])
            implies
        disjointQueues[q1,q2,t.next]
    all q1,q2:Queue , t:Time |
        disjointQueues[q1,q2,t] and
        (some c:Customer, qt:QueueTicket | leaveQueue[q1,c,qt,t])
            implies
        disjointQueues[q1,q2,t.next]
}

/*This assertion proves that any customer that is inside the store was
    checkedIn at some point*/
assert NoUncheckedGuest{
    /*Base Case: the property holds for an empty store*/
    all s:Store, t:Time |
        #s.storeCustomersInStore = 0
        implies
        allGuestsChecked[s,t]

    /*Inductive Steps: if we start from a consistent state, and some
    customer enters the Store or leaves it, the property still holds*/
    all s:Store, t:Time |
        allGuestsChecked[s,t] and
        (some qt:QueueTicket | enterStoreQueue[s,qt,t])
            implies
        allGuestsChecked[s,t.next]
    all s:Store, t:Time |
        allGuestsChecked[s,t] and
        (some b:Booking| enterStoreBooking[s,b,t])
            implies
        allGuestsChecked[s,t.next]
    all s:Store, t:Time |
        allGuestsChecked[s,t] and
        (some tick:Ticket| leaveStore[s,tick,t])
            implies
        allGuestsChecked[s,t.next]
}

//This assertion proves that a store cannot have two overlapping bookings
assert NoOverlappingBookingSameStore{
    /*Base Case: the property holds for an Store without bookings*/
    all s:Store, t:Time |
        #s.storeBookings.t = 0
        implies
        !hasOverlappingBooking[s,t]

    /*Inductive Steps: if we start from a consistent state, and some visit
    is either booked or deleted, the property still holds*/
    all s:Store, t:Time |
        !hasOverlappingBooking[s,t] and
        (some c:Customer,b:Booking | bookVisit[s, b,c,t])
            implies
        !hasOverlappingBooking[s,t]
    all s:Store, t:Time |
        !hasOverlappingBooking[s,t] and
        (some c:Customer,b:Booking | deleteAVisit[s, c,b,t])
            implies
        !hasOverlappingBooking[s,t]
}

//This assertion proves that no two stores can have overlapping bookings
assert NoOverlappingBookingDiffStore{
    /*Base Case: the property holds for two stores both without a booking*/
    all s1,s2:Store, t:Time |
        #s1.storeBookings.t = 0 and
        #s2.storeBookings.t = 0
            implies
        !haveOverlappingBooking[s1,s2,t]
    /*Inductive Steps: if we start from a consistent state, and some visit
    is either booked or deleted, the property still holds*/
    all s1,s2:Store, t:Time |
        !haveOverlappingBooking[s1,s2,t] and
        (some c:Customer,b:Booking | bookVisit[s1, b,c,t])
            implies
        !haveOverlappingBooking[s1,s2,t]
    all s1,s2:Store, t:Time |
        !haveOverlappingBooking[s1,s2,t] and
        (some c:Customer,b:Booking | deleteAVisit[s1, c,b,t])
            implies
        !haveOverlappingBooking[s1,s2,t]
}

/*This assertion proves that if a customer is inside a store it has some
    ticket that was checked in but not checkedout*/
assert ControllersWorkProperly{
    /*Base Case: the property holds for an empty store*/
    all s:Store, t:Time |
        #s.storeCustomersInStore.t = 0
        implies
        ( all c:Customer |
            c in s.storeCustomersInStore.t
            implies
            checkedInButNotOut[c,s,t])

    /*Inductive Steps: if we start from a consistent state, and some
    customer enters the Store or leaves it, the property still holds*/
    all s:Store, t:Time |
        ( all c:Customer |
            c in s.storeCustomersInStore.t
            implies checkedInButNotOut[c,s,t]) and
        (some qt:QueueTicket | enterStoreQueue[s,qt,t])
            implies
        (all c:Customer |
            c in s.storeCustomersInStore.t
            implies checkedInButNotOut[c,s,t.next])

    all s:Store, t:Time |
        ( all c:Customer |
            c in s.storeCustomersInStore.(t.next)
            implies checkedInButNotOut[c,s,t]) and
        (some b:Booking|
            enterStoreBooking[s,b,t])
            implies
            (all c:Customer |
                c in s.storeCustomersInStore.(t.next)
                implies checkedInButNotOut[c,s,t.next])

    all s:Store, t:Time |
        ( all c:Customer |
            c in s.storeCustomersInStore.t
            implies checkedInButNotOut[c,s,t]) and
        (some tick:Ticket| leaveStore[s,tick,t])
            implies
        (all c:Customer |
            c in s.storeCustomersInStore.(t.next)
            implies checkedInButNotOut[c,s,t.next])
}

pred show{}
run show for 4 but exactly 2 Store, exactly 2 Queue


/*********************
*   Run Predicates  *
*********************/
run joinQueue for 7 but
    exactly 2 Store, exactly 2 Queue, exactly 6 QueueTicket,
    exactly 4 Time, exactly 2 CheckpointController
run leaveQueue for 7 but
    exactly 1 Store, exactly 1 Queue, exactly 6 QueueTicket,
    exactly 4 Time, exactly 2 CheckpointController
run bookVisit  for 7 but
    exactly 2 Store, exactly 2 Queue, exactly 6 QueueTicket,
    exactly 4 Time, exactly 2 CheckpointController, exactly 3 Booking,
    exactly 3 BookingTicket
run deleteAVisit  for 7 but
    exactly 1 Store, exactly 1 Queue, exactly 6 QueueTicket,
    exactly 5 Time, exactly 2 CheckpointController, exactly 3 Booking,
    exactly 3 BookingTicket
run enterStoreQueue for 7 but
    exactly 1 Store, exactly 1 Queue, exactly 6 QueueTicket,
    exactly 4 Time, exactly 2 CheckpointController, exactly 4 Customer
run enterStoreBooking for 7 but
    exactly 1 Store, exactly 1 Queue, exactly 6 QueueTicket,
    exactly 4 Time, exactly 2 CheckpointController, exactly 3 Booking,
    exactly 3 BookingTicket
run leaveStore for 7 but
    exactly 1 Store, exactly 1 Queue, exactly 6 QueueTicket,
    exactly 4 Time, exactly 2 CheckpointController, exactly 4 Customer
run printTicket for 7 but
    exactly 1 Store, exactly 1 Queue, exactly 6 QueueTicket,
    exactly 4 Time, exactly 2 CheckpointController, exactly 4 Customer

/********************
*   Run Assertions  *
*********************/
check CheckoutSubsetCheckin
check NoCustomerTwiceSameQueue
check NoTwoQueuesShareCustomer
check NoUncheckedGuest
check NoOverlappingBookingSameStore
check NoOverlappingBookingDiffStore
check ControllersWorkProperly
```

**Page 41: Formatted** **Marco Petri** **22/12/2020 22:07:00**

Font: 8 pt, Bold, Font color: Blue

**Page 41: Formatted** **Marco Petri** **22/12/2020 22:07:00**

Font: 8 pt, Font color: Blue

**Page 41: Formatted** **Marco Petri** **22/12/2020 22:07:00**

Font: 8 pt

**Page 41: Formatted** **Marco Petri** **22/12/2020 22:07:00**

Font: 8 pt, Bold, Font color: Blue

**Page 41: Formatted** **Marco Petri** **22/12/2020 22:07:00**

Font: 8 pt, Font color: Blue

**Page 41: Formatted** **Marco Petri** **22/12/2020 22:07:00**

Font: 8 pt

**Page 47: Formatted** **Marco Petri** **22/12/2020 22:07:00**

Font: 8 pt

**Page 52: Formatted** **Marco Petri** **22/12/2020 22:05:00**

Adjust space between Latin and Asian text, Adjust space between Asian text and numbers, Pattern: Clear (White)

**Page 52: Deleted** **Marco Petri** **22/12/2020 19:46:00**

```
       storeManagers: some StoreManager,
```

**Page 52: Inserted** **aa m** **08/12/2020 17:34:00**

```
        storeManagers: some StoreManager,
```

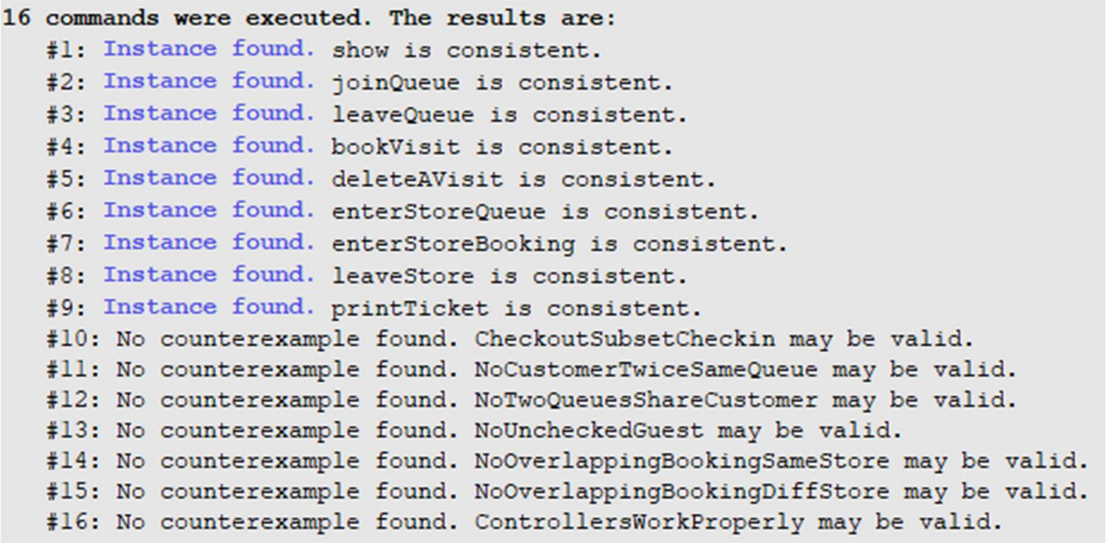
**Page 52: Formatted** **Marco Petri** **22/12/2020 20:06:00**

Font: 9 pt

**Page 52: Inserted** **Marco Petri** **22/12/2020 22:08:00**

## 1.1 Alloy analysis

Given the alloy code we run the assertion and found there are no errors. This is the result we obtain:



```
16 commands were executed. The results are:
   #1: Instance found. show is consistent.
   #2: Instance found. joinQueue is consistent.
   #3: Instance found. leaveQueue is consistent.
   #4: Instance found. bookVisit is consistent.
   #5: Instance found. deleteAVisit is consistent.
   #6: Instance found. enterStoreQueue is consistent.
   #7: Instance found. enterStoreBooking is consistent.
   #8: Instance found. leaveStore is consistent.
   #9: Instance found. printTicket is consistent.
   #10: No counterexample found. CheckoutSubsetCheckin may be valid.
   #11: No counterexample found. NoCustomerTwiceSameQueue may be valid.
   #12: No counterexample found. NoTwoQueuesShareCustomer may be valid.
   #13: No counterexample found. NoUncheckedGuest may be valid.
   #14: No counterexample found. NoOverlappingBookingSameStore may be valid.
   #15: No counterexample found. NoOverlappingBookingDiffStore may be valid.
   #16: No counterexample found. ControllersWorkProperly may be valid.
```

**Page 52: Formatted** **Marco Petri** **22/12/2020 22:05:00**

Font: 10 pt, Font color: Black, English (United Kingdom)

**Page 52: Formatted** **Marco Petri** **22/12/2020 22:09:00**

Justified, Line spacing: Multiple 1.25 li, Adjust space between Latin and Asian text, Adjust space between Asian text and numbers

**Page 52: Formatted** **Marco Petri** **22/12/2020 22:09:00**

Font: (Default) Garamond, 12 pt

**Page 52: Deleted** **Marco Petri** **22/12/2020 19:46:00**

```
        storeControllers: some CheckpointController,
        storeTicketMachines: some TicketMachine,
        storeProducts: some Item,
        //customers currently inside the store
        storeCustomersInStore: set Customer,
        //customers visits at the store
        //a visit is counted when the customer enters the store
        storeCustomersVisits: set Visit,
        //current queue for the store
        storeQueue: one Queue,
        //current bookings for the store
        storeBookings: set Booking,
        //tickets that have been used to enter the store, but not necessarely to exit
```

```
	storeUsedTickets: set Ticket,
	//tickets that have not been used to enter the store, and are no more valid
	storeNotUsedInvalidTickets: set Ticket
}{
	#storeControllers>0
	#storeTicketMachines>0
}


sig Visit{
	visitTicket: one Ticket,
	visitDate: one Date,
}

abstract sig Person {}

sig CheckpointController extends Person {
	controllerControlsEntrance: set Ticket,
	//rejected tickets
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
	controllerControlsExit: set Ticket,
}

sig StoreManager extends Person {}

sig Customer extends Person {}

sig TicketMachine {
	machinePrintedTickets: set QueueTicket
}

sig Category {}

sig Item {
	itemCategory: one Category
}

abstract sig Ticket {
	ticketOwner: one Customer,
	ticketQRCode: one QRCode,
}

sig Booking {
	bookingTicket: one BookingTicket,
	bookingDate: one Date,
	bookingItems: set Item,
	bookingCategories: set Category
}

sig BookingTicket extends Ticket {}

sig QueueTicket extends Ticket {}

sig Queue {
	queueNumbers: set QueueNumber,
	queueFirstNumberInLine: lone QueueNumber,
}



sig QueueNumber {
	queueNumberTicket: one QueueTicket,
	queueNumberNext: lone QueueNumber,
}
```

```
sig QRCode {}{
       //every TicketNumber corresponds to exactly one QueueTicket
       one q: Ticket | this = q.ticketQRCode
}

sig Date {}

//FACTS

fact  customersFacts{
       //customers in store have been checked in but not checked out
       all c: Customer | all s: Store | one t: Ticket |
       c in s.storeCustomersInStore implies
       (c = t.ticketOwner and
       t in s.storeControllers.controllerControlsEntrance and
       t not in s.storeControllers.controllerControlsExit)
       //customers have max one queueTicket  Fixare!!!!!!!!!!!!!!!!!!
       //tanti booking non si sovrappongo
       all c: Customer | lone t: Ticket | c = t.ticketOwner
}

fact queueFacts{//metti numbers!!!!!!!!!!!!!!!
       //a first ticket is not the next of any ticket
       all q: Queue | no qt: QueueNumber | qt.queueNumberNext = q.queueFirstNumberInLine
       //if there are at least than 1 tickets there must be a first ticket
       all q: Queue | #q.queueNumbers > 0 implies #q.queueFirstNumberInLine = 1
       //if there is a firstTicket it must belong to the queue tickets
       all q: Queue | q.queueFirstNumberInLine in q.queueNumbers
}

fact queueTicketsFacts{
       //tickets' next not reflexive
       no qt: QueueNumber | qt.queueNumberNext = qt
       //tickets' next not cyclic
       no qt: QueueNumber | qt in qt.^queueNumberNext
       //tickets are connected to the others in the same queue
       all qt: QueueNumber | one q: Queue | qt in q.queueFirstNumberInLine.*queueNumberNext
       and qt in q.queueNumbers
       //no shared next tickets
       all disj qt1,qt2,qt3: QueueNumber | qt1.queueNumberNext = qt3 implies qt2.queueNumberNext != qt3
       //queueTickets are assigned to one queue
       //all qt: QueueTicket | one q: Queue | qt in q.queueTickets
}


fact storeEntitiesAssignedToAStore{
       all sm: StoreManager | some s: Store | sm in s.storeManagers
       all cp: CheckpointController | one s: Store | cp in s.storeControllers
       all i: Item | some s: Store | i in s.storeProducts
       all c: Category | some i: Item | c = i.itemCategory
       all q: Queue | one s: Store | q in s.storeQueue
       all b: Booking | one s: Store | b in s.storeBookings
       all tm: TicketMachine | one s: Store | tm in s.storeTicketMachines
}


fact ticketsFacts{
       //tickets are assigned a store:
       all t: Ticket | one s: Store | t in (s.storeUsedTickets + s.storeNotUsedInvalidTickets +
       s.storeBookings.bookingTicket + s.storeQueue.queueNumbers.queueNumberTicket)

       //tickets can either be:
       all t: Ticket | all s: Store |
    //s.storeUsedTickets&(s.storeNotUsedInvalidTickets+
    //used                    FIXARE col + e magari migliorare
    (t in s.storeUsedTickets implies (t not in s.storeNotUsedInvalidTickets and
    t not in s.storeQueue.queueNumbers.queueNumberTicket and t not in s.storeBookings.bookingTicket)) and
    //not used and invalid
    (t in s.storeNotUsedInvalidTickets implies (t not in s.storeUsedTickets and
    t not in s.storeQueue.queueNumbers.queueNumberTicket and t not in s.storeBookings.bookingTicket)) and
    //valid for booking
    (t in s.storeBookings.bookingTicket implies (t not in s.storeNotUsedInvalidTickets and
    t not in s.storeQueue.queueNumbers.queueNumberTicket and t not in s.storeUsedTickets)) and
    //valid for queueing
    (t in s.storeQueue.queueNumbers.queueNumberTicket implies (t not in s.storeNotUsedInvalidTickets and
    t not in s.storeUsedTickets and t not in s.storeBookings.bookingTicket))

    //tickets have an owner
    all t: Ticket | one c: Customer | t.ticketOwner = c
    //if tickets are printed, they are printed by a machine of the store they are for
    all t: Ticket | all s: Store | t in s.storeTicketMachines.machinePrintedTickets
    implies (t in s.storeBookings.bookingTicket or t in s.storeQueue.queueNumbers.queueNumberTicket
    or t in s.storeNotUsedInvalidTickets or t in s.storeUsedTickets)
    //tickets can be controlled only once at the entrance
    all t: Ticket | lone c: CheckpointController | t in c.controllerControlsEntrance
    //tickets can be controlled only once at the exit
    all t: Ticket | lone c: CheckpointController | t in c.controllerControlsExit
    //tickets can only be printed once
    all t: Ticket | lone tm: TicketMachine | t in tm.machinePrintedTickets
    //all tickets checked at the exit have been checked at the entrance
    all t: Ticket | all s: Store |
    t in s.storeControllers.controllerControlsExit implies t in s.storeControllers.controllerControlsEntrance
    //all used tickets have been checked at entrance
    all t: Ticket | all s: Store |
    t in s.storeUsedTickets iff t in s.storeControllers.controllerControlsEntrance
}

fact visitsFacts{
    //all visits are assigned to one store
    all v: Visit |  one s: Store |  v in s.storeCustomersVisits
    //a visit was made if and only if its ticket was checked in by the controller
    all s: Store | all v: Visit |
    v.visitTicket in s.storeControllers.controllerControlsEntrance iff v in s.storeCustomersVisits
    //every time a controller checks in a ticket only one visit is counted
    all s: Store | all t: Ticket | one v: Visit |
    t in s.storeControllers.controllerControlsEntrance implies
    (v in s.storeCustomersVisits and v.visitTicket = t)
}

 //ASSERTIONS

//a ticket can only be used for max a single visit
assert ticketMaxOneVisit{
    all t: Ticket | lone v: Visit |
    t = v.visitTicket
}
check ticketMaxOneVisit for 6

//all used tickets of a store have been used for visiting that store
assert usedTicketForVisitOfSameStore{
       all s: Store | all v: Visit |
       v in s.storeCustomersVisits implies v.visitTicket in s.storeUsedTickets
}
check usedTicketForVisitOfSameStore for 6

//all customers in queue cannot enter before their ticket is called
assert customersRespectQueue{
       all s: Store | all qt: QueueTicket | all c: Customer |//RIGUARDARE
       (c = qt.ticketOwner and qt in s.storeQueue.queueFirstNumberInLine.*queueNumberNext.queueNumberTicket)
       implies c not in s.storeCustomersInStore
}
check customersRespectQueue for 6

//no customer has tickets for more than one queue
assert noCustomerInTwoQueues {
       all disj q1,q2: Queue | all c: Customer | c in q1.queueNumbers.queueNumberTicket.ticketOwner
       implies c not in q2.queueNumbers.queueNumberTicket.ticketOwner
}
check noCustomerInTwoQueues for 6

//no customer has more than one tickets for the same queue
assert noCustomerTwiceInQueue{
       all q: Queue | no disj t1, t2: QueueTicket | t1 in q.queueNumbers.queueNumberTicket
       and t2 in q.queueNumbers.queueNumberTicket
       and t1.ticketOwner = t2.ticketOwner
}
check noCustomerTwiceInQueue for 6



// PREDICATES

//visite non possono avere stesso date and time

//

//customers can use the ticket machine to visit the store they want
pred customersCanUseTicketMachineToVisit{
       #Store>0
       all s: Store | some v: Visit |
       v in s.storeCustomersVisits and
       v.visitTicket in s.storeTicketMachines.machinePrintedTickets
}
run customersCanUseTicketMachineToVisit for 6

//customers can use online queue to visit the store they want
pred customersCanUseOnlineQueueToVisit{
       #Store>0
       all s: Store | some v: Visit | some qt: QueueTicket |
       v in s.storeCustomersVisits and
       v.visitTicket = qt and
       v.visitTicket not in s.storeTicketMachines.machinePrintedTickets
}
run customersCanUseOnlineQueueToVisit for 6

//customers can use online booking to visit the store they want
pred customersCanUseOnlineBookingToVisit{
       #Store>0
       all s: Store | some v: Visit | some bt: BookingTicket |
       v in s.storeCustomersVisits and
       v.visitTicket = bt
}
run customersCanUseOnlineBookingToVisit for 6
```

```
pred show {}

/*run {
       #Store = 2
       #Date = 4
       #Item = 2
       #Category = 1
       #Queue = 2
       #Booking = 1
       #TicketMachine = 3
       #Customer = 3
       #QueueTicket = 2

} for 5*/


run show

//run show for exactly 2 Store, 4 Person, 2 TicketMachine, 2 Category,
//2 Item, 15 Ticket, 1 Booking, 5 Queue, 5 TicketNumber, 15 QRCode, 1 Date

//run show for 11 but 3 Date, 3 Item, 3 Customer, 4 CheckpointController, 3 Booking
```

**Page 52: Inserted** **aa m** **08/12/2020 17:34:00**

```
       storeControllers: some CheckpointController,
       storeTicketMachines: some TicketMachine,
       storeProducts: some Item,
       //customers currently inside the store
       storeCustomersInStore: set Customer,
       //customers visits at the store
       //a visit is counted when the customer enters the store
       storeCustomersVisits: set Visit,
       //current queue for the store
       storeQueue: one Queue,
       //current bookings for the store
       storeBookings: set Booking,
       //tickets that have been used to enter the store, but not necessarely to exit
       storeUsedTickets: set Ticket,
       //tickets that have not been used to enter the store, and are no more valid
       storeNotUsedInvalidTickets: set Ticket
}{
       #storeControllers>0
       #storeTicketMachines>0
}


sig Visit{
       visitTicket: one Ticket,
       visitDate: one Date,
}

abstract sig Person {}

sig CheckpointController extends Person {
       controllerControlsEntrance: set Ticket,
       //rejected tickets
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
       controllerControlsExit: set Ticket,
}

sig StoreManager extends Person {}
sig Customer extends Person {}

sig TicketMachine {
	machinePrintedTickets: set QueueTicket
}

sig Category {}

sig Item {
	itemCategory: one Category
}

abstract sig Ticket {
	ticketOwner: one Customer,
	ticketQRCode: one QRCode,
}

sig Booking {
	bookingTicket: one BookingTicket,
	bookingDate: one Date,
	bookingItems: set Item,
	bookingCategories: set Category
}

sig BookingTicket extends Ticket {}

sig QueueTicket extends Ticket {}

sig Queue {
	queueNumbers: set QueueNumber,
	queueFirstNumberInLine: lone QueueNumber,
}



sig QueueNumber {
	queueNumberTicket: one QueueTicket,
	queueNumberNext: lone QueueNumber,
}

sig QRCode {}{
	//every TicketNumber corresponds to exactly one QueueTicket
	one q: Ticket | this = q.ticketQRCode
}

sig Date {}

//FACTS

fact  customersFacts{
	//customers in store have been checked in but not checked out
	all c: Customer | all s: Store | one t: Ticket |
	c in s.storeCustomersInStore implies
	(c = t.ticketOwner and
	t in s.storeControllers.controllerControlsEntrance and
	t not in s.storeControllers.controllerControlsExit)
	//customers have max one queueTicket  Fixare!!!!!!!!!!!!!!!!!!!
	//tanti booking non si sovrappongo
	all c: Customer | lone t: Ticket | c = t.ticketOwner
}

fact queueFacts{//metti numbers!!!!!!!!!!!!!!!
	//a first ticket is not the next of any ticket
	all q: Queue | no qt: QueueNumber | qt.queueNumberNext = q.queueFirstNumberInLine
	//if there are at least than 1 tickets there must be a first ticket
	all q: Queue | #q.queueNumbers > 0 implies #q.queueFirstNumberInLine = 1
	//if there is a firstTicket it must belong to the queue tickets
	all q: Queue | q.queueFirstNumberInLine in q.queueNumbers
}

fact queueTicketsFacts{
	//tickets' next not reflexive
	no qt: QueueNumber | qt.queueNumberNext = qt
	//tickets' next not cyclic
	no qt: QueueNumber | qt in qt.^queueNumberNext
	//tickets are connected to the others in the same queue
	all qt: QueueNumber | one q: Queue | qt in q.queueFirstNumberInLine.*queueNumberNext
	and qt in q.queueNumbers
	//no shared next tickets
	all disj qt1,qt2,qt3: QueueNumber | qt1.queueNumberNext = qt3 implies qt2.queueNumberNext != qt3
	//queueTickets are assigned to one queue
	//all qt: QueueTicket | one q: Queue | qt in q.queueTickets
}


fact storeEntitiesAssignedToAStore{
	all sm: StoreManager | some s: Store | sm in s.storeManagers
	all cp: CheckpointController | one s: Store | cp in s.storeControllers
	all i: Item | some s: Store | i in s.storeProducts
	all c: Category | some i: Item | c = i.itemCategory
	all q: Queue | one s: Store | q in s.storeQueue
	all b: Booking | one s: Store | b in s.storeBookings
	all tm: TicketMachine | one s: Store | tm in s.storeTicketMachines
}


fact ticketsFacts{
	//tickets are assigned a store:
	all t: Ticket | one s: Store | t in (s.storeUsedTickets + s.storeNotUsedInvalidTickets +
	s.storeBookings.bookingTicket + s.storeQueue.queueNumbers.queueNumberTicket)

	//tickets can either be:
	all t: Ticket | all s: Store |
	//s.storeUsedTickets&(s.storeNotUsedInvalidTickets+
	//used                     FIXARE col + e magari migliorare
	(t in s.storeUsedTickets implies (t not in s.storeNotUsedInvalidTickets and
	t not in s.storeQueue.queueNumbers.queueNumberTicket and t not in s.storeBookings.bookingTicket)) and
	//not used and invalid
	(t in s.storeNotUsedInvalidTickets implies (t not in s.storeUsedTickets and
	t not in s.storeQueue.queueNumbers.queueNumberTicket and t not in s.storeBookings.bookingTicket)) and
	//valid for booking
	(t in s.storeBookings.bookingTicket implies (t not in s.storeNotUsedInvalidTickets and
	t not in s.storeQueue.queueNumbers.queueNumberTicket and t not in s.storeUsedTickets)) and
	//valid for queueing
	(t in s.storeQueue.queueNumbers.queueNumberTicket implies (t not in s.storeNotUsedInvalidTickets and
	t not in s.storeUsedTickets and t not in s.storeBookings.bookingTicket))

	//tickets have an owner
	all t: Ticket | one c: Customer | t.ticketOwner = c
	//if tickets are printed, they are printed by a machine of the store they are for
	all t: Ticket | all s: Store | t in s.storeTicketMachines.machinePrintedTickets
	implies (t in s.storeBookings.bookingTicket or t in s.storeQueue.queueNumbers.queueNumberTicket
	or t in s.storeNotUsedInvalidTickets or t in s.storeUsedTickets)
	//tickets can be controlled only once at the entrance
	all t: Ticket | lone c: CheckpointController | t in c.controllerControlsEntrance
	//tickets can be controlled only once at the exit
	all t: Ticket | lone c: CheckpointController | t in c.controllerControlsExit
	//tickets can only be printed once
	all t: Ticket | lone tm: TicketMachine | t in tm.machinePrintedTickets
	//all tickets checked at the exit have been checked at the entrance
	all t: Ticket | all s: Store |
	t in s.storeControllers.controllerControlsExit implies t in s.storeControllers.controllerControlsEntrance
	//all used tickets have been checked at entrance
	all t: Ticket | all s: Store |
	t in s.storeUsedTickets iff t in s.storeControllers.controllerControlsEntrance
}

fact visitsFacts{
	//all visits are assigned to one store
	all v: Visit |  one s: Store |  v in s.storeCustomersVisits
	//a visit was made if and only if its ticket was checked in by the controller
	all s: Store | all v: Visit |
	v.visitTicket in s.storeControllers.controllerControlsEntrance iff v in 
s.storeCustomersVisits
	//every time a controller checks in a ticket only one visit is counted
	all s: Store | all t: Ticket | one v: Visit |
	t in s.storeControllers.controllerControlsEntrance implies
	(v in s.storeCustomersVisits and v.visitTicket = t)
}

 //ASSERTIONS

//a ticket can only be used for max a single visit
assert ticketMaxOneVisit{
	all t: Ticket | lone v: Visit |
	t = v.visitTicket
}
check ticketMaxOneVisit for 6

//all used tickets of a store have been used for visiting that store
assert usedTicketForVisitOfSameStore{
	all s: Store | all v: Visit |
	v in s.storeCustomersVisits implies v.visitTicket in s.storeUsedTickets
}
check usedTicketForVisitOfSameStore for 6

//all customers in queue cannot enter before their ticket is called
assert customersRespectQueue{
	all s: Store | all qt: QueueTicket | all c: Customer |//RIGUARDARE
	(c = qt.ticketOwner and qt in s.storeQueue.queueFirstNumberInLine.*queueNumberNext.queueNumberTicket)
	implies c not in s.storeCustomersInStore
}
check customersRespectQueue for 6

//no customer has tickets for more than one queue
assert noCustomerInTwoQueues {
	all disj q1,q2: Queue | all c: Customer | c in q1.queueNumbers.queueNumberTicket.ticketOwner
	implies c not in q2.queueNumbers.queueNumberTicket.ticketOwner
}
check noCustomerInTwoQueues for 6

//no customer has more than one tickets for the same queue
```

```
assert noCustomerTwiceInQueue{
       all q: Queue | no disj t1, t2: QueueTicket | t1 in q.queueNumbers.queueNumberTicket
       and t2 in q.queueNumbers.queueNumberTicket
       and t1.ticketOwner = t2.ticketOwner
}
check noCustomerTwiceInQueue for 6



// PREDICATES

//visite non possono avere stesso date and time

//

//customers can use the ticket machine to visit the store they want
pred customersCanUseTicketMachineToVisit{
       #Store>0
       all s: Store | some v: Visit |
       v in s.storeCustomersVisits and
       v.visitTicket in s.storeTicketMachines.machinePrintedTickets
}
run customersCanUseTicketMachineToVisit for 6

//customers can use online queue to visit the store they want
pred customersCanUseOnlineQueueToVisit{
       #Store>0
       all s: Store | some v: Visit | some qt: QueueTicket |
       v in s.storeCustomersVisits and
       v.visitTicket = qt and
       v.visitTicket not in s.storeTicketMachines.machinePrintedTickets
}
run customersCanUseOnlineQueueToVisit for 6

//customers can use online booking to visit the store they want
pred customersCanUseOnlineBookingToVisit{
       #Store>0
       all s: Store | some v: Visit | some bt: BookingTicket |
       v in s.storeCustomersVisits and
       v.visitTicket = bt
}
run customersCanUseOnlineBookingToVisit for 6

pred show {}

/*run {
       #Store = 2
       #Date = 4
       #Item = 2
       #Category = 1
       #Queue = 2
       #Booking = 1
       #TicketMachine = 3
       #Customer = 3
       #QueueTicket = 2

} for 5*/


run show

//run show for exactly 2 Store, 4 Person, 2 TicketMachine, 2 Category,
//2 Item, 15 Ticket, 1 Booking, 5 Queue, 5 TicketNumber, 15 QRCode, 1 Date

//run show for 11 but 3 Date, 3 Item, 3 Customer, 4 CheckpointController, 3 Booking
```

**Page 52: Formatted Marco Petri 22/12/2020 20:06:00**

Font: 9 pt

**Page 52: Formatted Marco Petri 22/12/2020 19:46:00**

Left, Line spacing: single, Don't adjust space between Latin and Asian text, Don't adjust space between Asian text and numbers

**Page 53: Inserted asus 22/12/2020 20:44:00**

31

**Page 53: Deleted asus 22/12/2020 20:44:00**

5

**Page 53: Inserted Marco Petri 22/12/2020 18:55:00**

5

**Page 53: Deleted Marco Petri 22/12/2020 18:55:00**

1.5

**Page 53: Inserted asus 19/12/2020 12:38:00**

1.5

**Page 53: Deleted asus 19/12/2020 12:38:00**

0

**Page 53: Inserted asus 19/12/2020 12:37:00**

4

**Page 53: Deleted asus 19/12/2020 12:37:00**

1

**Page 53: Inserted Marco Petri 22/12/2020 18:55:00**

30

**Page 53: Deleted Marco Petri 22/12/2020 18:55:00**

2

**Page 53: Inserted Marco Petri 22/12/2020 18:54:00**

2.5

**Page 53: Deleted Marco Petri 22/12/2020 18:54:00**

10

**Page 53: Inserted aa m 22/12/2020 18:53:00**

1

**Page 53: Inserted Marco Petri 22/12/2020 18:55:00**

7

**Page 53: Deleted Marco Petri 22/12/2020 18:55:00**

.5

**Page 53: Inserted Marco Petri 22/12/2020 18:54:00**

2.5

**Page 53: Deleted** **Marco Petri** **22/12/2020 18:54:00**

2.5

**Page 53: Inserted** **Marco Petri** **22/12/2020 18:52:00**

20

**Page 53: Deleted** **Marco Petri** **22/12/2020 18:52:00**

14

**Page 53: Deleted** **aa m** **22/12/2020 18:52:00**

hrs

**Page 53: Deleted** **Marco Petri** **22/12/2020 18:54:00**

22

**Page 53: Inserted** **aa m** **22/12/2020 18:52:00**

22hrs

**Page 54: Deleted** **Marco Petri** **22/12/2020 18:56:00**

Introductory text to the chapter (how it is subdivided and what are we going to say in the chapter)

**Page 54: Inserted** **Marco Petri** **22/12/2020 18:56:00**

This section includes all the document and references used to produce this documentation. We include in this part of the document every website or document used to

**Page 54: Moved from page 7 (Move #1)** **Marco Petri** **22/12/2020 18:00:00**

1. http://dati.istat.it/Index.aspx?DataSetCode=DCCV_ICT: is an ISTAT statistic about the usage internet diffusion in Italy;
1. Oracle Directory Server Enterprise Edition Deployment Planning Guide: is a document about the operations needed in the process of designing a system [Chapter 12];
1. https://docs.oracle.com/cd/E20295_01/html/821-1217/fjdch.html#scrolltoc;
1. 29148-2018 - 29148-2018 - ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering - IEEE Standard: ISO/IEC/IEEE document about the building and designing of a system and RASD definition;
1. https://gdpr-info.eu/: contains the official PDF of the Regulation (EU) 2016/679 (General Data Protection Regulation).
1. https://www.iso.org/isoiec-27001-information-security.html: ISO/IEC 27001 international standard.

**Page 54: Inserted** **asus** **22/12/2020 21:31:00**

1. 
1. https://alloy.readthedocs.io/en/latest/modules/ordering.html details about the ordering module in Alloy
1. https://alloy.readthedocs.io/en/latest/modules/time.html details about the time module in Alloy

http://alloytools.org/tutorials/day-course/s4_dynamic.pdf paper from MIT about the use of Alloy to verify properties

1.

**Page 54: Inserted** **Marco Petri** **22/12/2020 18:00:00**

1.

## Header and footer changes

**Page 1: Inserted** **Marco Petri** **22/12/2020 22:17:00**

Purpose

**Page 1: Deleted** **Marco Petri** **22/12/2020 22:17:00**

Alloy analysis

**Page 1: Inserted** **Marco Petri** **22/12/2020 22:17:00**

Purpose

**Page 1: Deleted** **Marco Petri** **22/12/2020 22:17:00**

Alloy analysis

## Text Box changes

## Header and footer text box changes

## Footnote changes

## Endnote changes