**Timetable Manager**

Project documentation

Student(s): Racz Adrian-Catalin

**Group: 30431**

**Contents**

[I Project specification](#_heading=h.gjdgxs) **3**

[1.1 Domain Model Diagram](#_heading=h.30j0zll) 3

[II Use-Case model](#_heading=h.1fob9te) **3**

[2.1 Users and stakeholders](#_heading=h.f2irg1azp7rm) 4

[2.2 Use-Case identification](#_heading=h.3znysh7) 4

[2.3 UML Use-Case diagram](#_heading=h.2et92p0) 4

[III Architectural design](#_heading=h.tyjcwt) **5**

[3.1 Conceptual architecture](#_heading=h.3dy6vkm) 5

[3.2 Package diagram](#_heading=h.1t3h5sf) 5

[3.3 Class diagram](#_heading=h.4d34og8) 5

[3.4 Database (E-R/Data model) diagram](#_heading=h.2s8eyo1) 6

[3.5 Sequence diagram](#_heading=h.17dp8vu) 6

[3.6 Activity diagram](#_heading=h.3rdcrjn) 6

[IV Supplementary specifications](#_heading=h.26in1rg) **6**

[4.1 Non-functional requirements](#_heading=h.lnxbz9) 6

[4.2 Design constraints](#_heading=h.35nkun2) 6

[V Testing](#_heading=h.1ksv4uv) **7**

[5.1 Testing methods/frameworks](#_heading=h.44sinio) 7

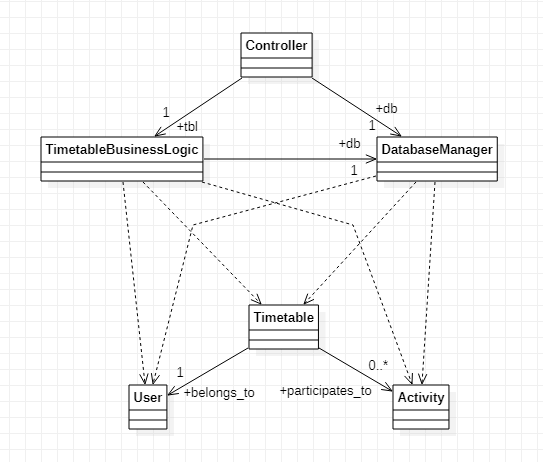
[5.2 Future improvements](#_heading=h.2jxsxqh) 7

[VI Bibliography](#_heading=h.z337ya) **7**

# I Project specification

Timetable Manager is an application that creates possible timetables based on user data. It focuses on timetables where people don’t have a constant work program (Eg. not everyone works from 8:00 AM to 16:00 PM) or people have to synchronize their work hours with others (if they have to work on a project with other people having their separate timetables or if a meeting has to take place when everyone is available). There are 2 things a user must choose, their available work hours and the activities they want to take part in. The administrator then generates the possible time tables and chooses the best ones for each of the user. The activities, added by the administrator, have their own requirements, like a minimum number of persons, a qualification or a set interval of time. In the end, all users get their own timetables with all their activities, constructed by following the conditions of the users and the activities themselves, and verified and approved by the administrator.

## 1.1 Domain Model Diagram



# II Use-Case model

There are users and administrators. The users can specify their available hours and choose the activities they take. They can also see their own timetable once the administrator chose it for them.

The administrator can generate valid timetables formed using the input of the clients and then choose the official ones. Also, he can create new activities with their specific requirements, that can later be picked by the users.

## 2.1 Users and stakeholders

* The users (clients/employees)
* The administrator

Stakeholders:

* the developer (me)
* irregular work intervals or project-based companies
* managers/employees of such companies
* educational institutions
* rival applications (like Planfy, Time Schedule App, Timetable maker, etc.)

## 2.2 Use-Case identification

**Use case name: Input available hours.**

**Level: Subfunction.**

**Main actor: User.**

**Main success scenario: The user gives his available hours and, if valid, the database entry of the user gets updated.**

**Extension: The input data is invalid, therefore the action fails, giving a pop up message and having nothing added to the database.**

**Use case name: Choose activities.**

**Level: Subfunction.**

**Main actor: User.**

**Main success scenario: The user selects the activity he wants to participate in and, if compatible, the activity and the user are bound together in the database.**

**Extension: The user selects a “meeting” activity that has a specific time (that is within the user’s available hours) and no qualification requirement, resulting in a successful message and the database updated.**

**Use case name: Generate timetables.**

**Level: User-Goal.**

**Main actor: Administrator.**

**Main success scenario: The logic layer of the program is accessed, where the database is consulted in order to create multiple solutions (timetables) and then displayed to the administrator.**

**Extension: The administrator clicks the button and the valid timetables are generated and displayed.**

**Use case name: Create activities.**

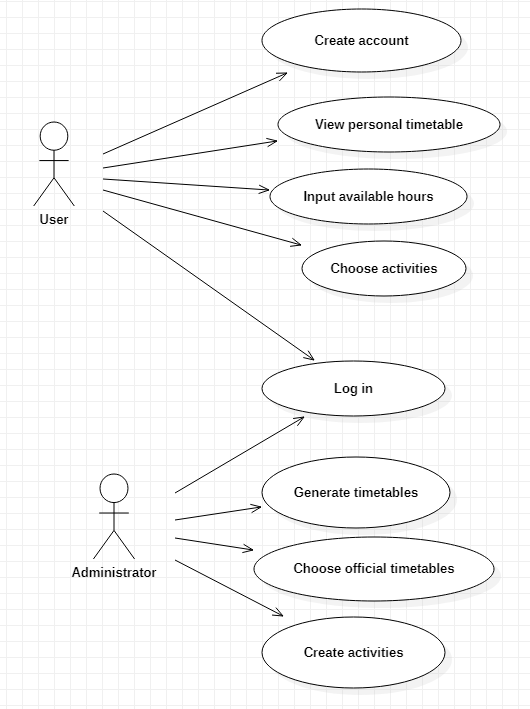
**Level: Subfunction.**

**Main actor: Administrator.**

**Main success scenario: A new activity is created in the database, specifying the requirements (time interval, required qualification, people needed, minimum time spent). It makes it available for the users to pick with choose activities.**

**Extension: The administrator inputs the name and the requirement parameters, successfully creating and adding it to the database, with a popup success message given.**

## 2.3 UML Use-Case diagram



# III Architectural design

## 3.1 Conceptual architecture

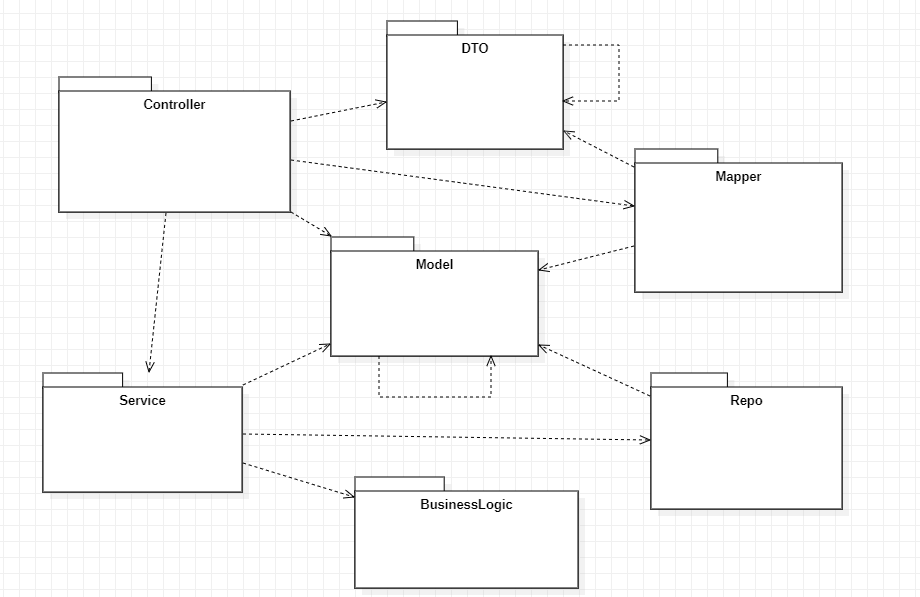
Type of application: Web

Architecture: Client-Server

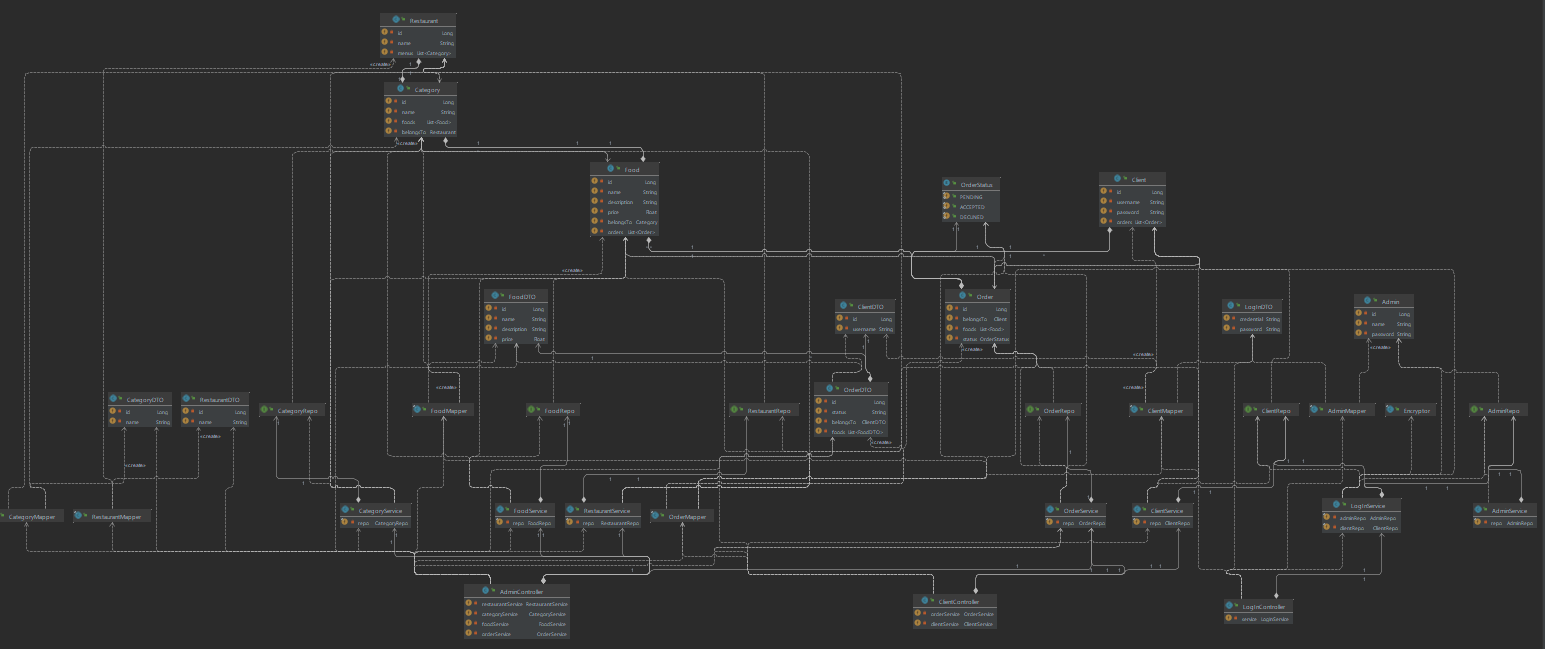
The database contains 3 main tables: user, activity and timetable. The user table stores the information about a user, like their name, qualification and username. The activity table stores all possible activities and their details. The timetable table contains pairs of users and activities, signifying the segment of a user’s timetable and the single activity executed in the chosen time interval of the given day. If you take all entries belonging to one user from this table, you should get the full timetable for this user. Additionally, some helper tables are present, like the day table and qualification table that are there just for good database practices. The activity\_day table is a many-to-many connection table.

The Client-Server architecture is useful in my implementation because there are a lot of users and only one administrator, taking the form of many clients, with one main server that ties them all together. On the client side, the users can provide input, while on the server side, the administrator can manage that data and send appropriate responses back.

## 3.2 Package diagram



## 3.3 Class diagram



## 3.4 Database (E-R/Data model) diagram

## 

## 3.5 Sequence diagram

## 

## 3.6 Activity diagram

# 

# IV Supplementary specifications

## 4.1 Non-functional requirements

Availability: Running continuously, until all administrators decide to shut down their parts.

Security: Simple log-in security that requires a registered account with a set username and password. Additionally, for the registration, a special key known and communicated by the user’s administrator may be required.

Scalability: Horizontal scalable for multiple administrators running their own separate clients (aka multiple companies using the application at the same time, without intermingling with each other)

Accessibility: Accessible on any device and operating system, as long as it can normally run internet browsers. Valid for both users and administrators.

## 4.2 Design constraints

Languages: Java, SQL/HSQL

Frameworks: Maven, Java Spring

Security measurements: username-password registration

Technologies: Hibernate

Database specifications: MySQL

# V Testing

## 5.1 Testing methods/frameworks

Mockito

## 5.2 Future improvements

Delete, update, let the admin pick for the users, more complex activities (like being mandatory or needing multiple people of different qualifications), remake the qualifications in a hierarchical manner or allow multiple ones, more CSS in the frontend to spare your eyes from torment.

# VI API documentation

1. POST/login/user

Request: body (JSON):

{ “username”: “uname”,

“password”: “pass123”

}

Responses: - 200 OK – user was found

* 404 Not Found – User not found
* 400 Bad Request – fields are blank

1. POST/login/admin

Request: body (JSON):

{ “username”: “uname”,

“password”: “pass123”

}

Responses: - 200 OK – admin was found

* 404 Not Found – Admin not found
* 400 Bad Request – fields are blank

1. POST/signup/user

Request: body (JSON):

{ “username”: “uname”,

“firstname”: “Marco”,

“ lastName”: “Polo”,

“ qualification”: “FULL\_STACK”,

“password”: “pass123”

}

Responses: - 200 OK – name available, user registered successfully

* 400 Bad Request – fields are blank or name exists

1. POST/signup/admin

Request: body (JSON):

{ “username”: “uname”,

“password”: “pass123”

}

Responses: - 200 OK – name available, admin registered successfully

* 400 Bad Request – fields are blank or name exists

1. GET/util/qualification

Request: body (JSON):

{

}

Responses: - 200 OK – qualifications returned successfully

* 400 Bad Request – something went wrong

1. GET/util/day

Request: body (JSON):

{

}

Responses: - 200 OK – days returned successfully

* 400 Bad Request – something went wrong

1. GET/admin/activity

Request: body (JSON):

{

}

Responses: - 200 OK – activities returned successfully

* 404 Not Found – activities not found

1. POST/admin/activity

Request: body (JSON):

{

“name”: “Meeting”,

“nbPeople”: 10,

“minTime”: “00:30:00”,

“intervalStart”: “01:02:03”,

“intervalEnd”: “10:20:30”,

“qualification”: “MANAGEMENT”,

“day”: “MONDAY”

}

Responses: - 200 OK – activity created

* 400 Bad Request – non-null fields are null, or unique fields repeat

1. POST/user/activity/get\_by\_qualification

Request: body (JSON):

{

“qualification”: “BACKEND”

}

Responses: - 200 OK – activities returned successfully

* 404 Not Found – no such activities found

1. POST/user/activity/get\_filtered

Request: body (JSON):

{

“name”: “Meeting”,

“day”: “MONDAY”,

“qualification”: “BACKEND”

}

Responses: - 200 OK – activities returned successfully

* 404 Not Found – no such activities found

1. POST/user/choose

Request: body (JSON):

{

“intervalStart”: “01:02:03”,

“intervalEnd”: “10:20:30”,

“user”: {“id”: 1},

“activity”: {“id”: 1}

}

Responses: - 200 OK – timetable created successfully

* 400 Bad Request – id of user or activity is missing, or timetable already exists
* 404 Not Found – activity or user not found

1. GET/user/{id}/chosen\_timetables

Request: body (JSON):

{

}

Responses: - 200 OK – timetables returned successfully

* 400 Bad Request – invalid id
* 404 Not Found – no timetables, or the user, found

1. GET/user/{id}/official\_timetable

Request: body (JSON):

{

}

Responses: - 200 OK – timetables returned successfully

* 400 Bad Request – invalid id
* 404 Not Found – no timetables, or the user, found

1. POST/admin/timetables/generate

Request: body (JSON):

{

}

Responses: - 200 OK – timetables and groups generated successfully

* 400 Bad Request – something went wrong

1. GET/admin/timetables

Request: body (JSON):

{

}

Responses: - 200 OK – timetables returned successfully

* 404 Not Found – no timetables found

1. POST/admin/timetables/pick/{id}

Request: body (JSON):

{

}

Responses: - 200 OK – official timetable changed successfully

* 400 Bad Request – invalid id
* 404 Not Found – timetable group not found

# VII Bibliography

<https://stackoverflow.com/> (If I wrote all links I would reach 50 pages)

<https://docs.oracle.com/javase/8/> (Documentation for various things)

Previous assignments