# Draft of the structure of the MyThaiStar specification

## Basic ideas

The goal of the specification guide and the restaurant specification is to create a document that is:

* Aligned to the implementation: Structure, concepts and nomenclature should be the same in the code. Since this is an angular client, the whole process flow is part of the client: The server will present REST interfaces for necessary server interaction. These small operations will be documented as use cases and use case functions. The client modules and routes will be documented as part of the dialogue.
* Allow for short design documentation: Much of the structuring of components and data should be presented here and not need to be formally repeated in another documentation. The split of the application into components presented here should be present in the code.
* Slim: It should contain only the strictly needed information and should be very pragmatic.
* Easy to maintain: It should be versioned together with the code. A developer should be able to change it using only a text editor.

Because of these goals, it is very important for me to have a hard discussion about the structure of the components, the data, the use cases and the dialogue features. I am no angular expert and would welcome this greatly.

## Structure of the document

Roughly, the document shall contain the following chapters:

* Introduction and fundamental ideas
* Application components, use cases and use case functions
* Data model
* Dialogue, Dialogue modules and screens
* System interfaces
* Appendix

I do not think we need to specify printouts or non-functional requirements. Otherwise, they would get additional chapters.

The following list of contents is a current very first draft and will probably change in the future.

## Structure for the application components and use cases

### ACO\_Booking\_Ordering

This application component handles the management of bookings (whole reservations) and orderings (food and drink for one person).

**UC\_Book\_Table**: This use case will get the relevant data for the booking of a table (date, time, email, numbers, friends, invitation text) and create a booking (if possible) for it. If no tables are available, it will create an appropriate message. The invitation of friends could be delegated to UF\_Invite\_friends.

**UC\_Order\_Meal**: This use case will get the relevant data for a meal (dishes, drinks, additions, comments), validate the data and store it for one guest of a booking.

**UF\_Invite\_friends**: This use case function will create unique references for each invited friend and send emails containing an invitation text, a link to directly place the order for the meal, a link to accept the invitation without ordering and a link to refuse. The handling of the link to directly order the meal will be part of a dialogue route, the others will be handled by the following UC.

**UC\_Handle\_invitation\_feedback**: This use case will handle refusals and acceptances for invitation. The UC will validate the data and update the booking. In case of invalid data, it will present an appropriate response.

### ACO\_Menu\_Management

This application component is responsible for managing the data about dishes and drinks the restaurant can offer. The presentation of the menu and the selection of the dishes is done in the dialogue.

**UF\_Manage\_Menu-Items**: This use case allows the creation, update and deletion of dishes, drinks, additions, Categories of dishes and drinks, and twitter data (hashtags).

**UF\_Search\_Menu-Items**: This use case allows to search or filter menu items based on different search parameters, e.g. names, categories or hashtags. A hit list will be returned, the entries of which can then (including images) be read in full.

### ACO\_Twitter\_Integration

This application component will typically contain mostly configuration data and maybe a cache of some sort. It is responsible to encapsulate the handling of the twitter API.

**UF\_Get\_Twitter\_Feedback**: This use case function uses the configured hashtag of a dish or drink to get the twitter feedback information (number of likes, last comments).

**UC\_Rate\_Dish\_Drink**: This use case allows to either “like” a dish or drink, or to enter a comment for the dish or drink. Both actions will probably create a tweet for the corresponding hashtag.

## Structure of the data model

Components encapsulate the data they are responsible for. Therefore, the cutting of components depends upon the data that needs to be managed, and the cutting of data models depends on the components.

The data model is therefore cut into two model components: MCO\_Bookings\_Orders and MCO\_Menu.

The following chapters contain a rough draft of the data model. It especially does not contain all the relevant data.

### MCO\_Bookings\_Orders



Figure 1: Data model structure for booking

The main part is that every person (friends and host) form a booked person. Even if no email addresses (and only numbers of guests) were given, links to order for all participants could be sent to the host.

The order encapsulates the dishes and drinks one participant has ordered:



Figure 2: Data model structure of an order

The main distinction here is that an ordered dish may contain a comment and a number of additions, while a drink is simply ordered without any supplementary information.

### MCO\_Menu

The structure of the menu data could be as follows:



Figure 3: The structure of the menu data

Main decisions here were e.g. that a drink is not specific for a course, and that the categories for drinks and dishes are separated.

## Structure of the dialogue

I am no expert on angular. I know the dialogue is split into modules, and the dialogue flow is managed as routes. I am not sure on the best way to structure the dialogue specification to allow for a good alignment to the implementation. One way could be a split into the following “modules” (instead of screens):

**DIM\_MyThaiStar\_Main**: This module includes the welcome page and any additional page (imprint etc.) needed.

**DIM\_Table\_Booking**: This module includes the Booking of a table and the entering of e-mail addresses for friends, but not yet the ordering of dishes and drinks.

**DIM\_Dishes\_Drinks\_Ordering**: This module is perhaps the biggest and contains the presentation of the menu, the search and filter functionality, the twitter integration, the selection and commenting of dishes and drinks, the selection of additions, and the completion of the order.

**DIM\_Twitter\_Rating**: This small modules encapsulates the leaving of twitter comments for dishes and drinks. It is perhaps too small and too closely related to the Dishes/Drinks-Ordering Module, and should be integrated into this one.

## Structure of the system interfaces

Since the only really external interface is the twitter API and the mail API, no specification seems to be needed here.

We could present a vision of the internal REST APIs, but this is best left to the developer, and it is not needed for the client split in one application of the current size: I would omit it here.