Netfood: A Software System for Food Ordering and Delivery

Cristina-Edina Domokos
Babes-Bolyai University
Cluj-Napoca, Romania
kitty.domokos@yahoo.com

Barna Séra
Babes-Bolyai University
Cluj-Napoca, Romania
serabarna1996@gmail.com

Károly Simon
Babes-Bolyai University
Cluj-Napoca, Romania
simon.karoly@codespring.ro

Lajos Kovács
Codespring
Odorheiu Secuiesc, Romania
kovacs.lajos@codespring.ro

Tas-Béla Szakács
Codespring
Odorheiu Secuiesc, Romania
szakacs.tas@codespring.ro

***Abstract*—Netfood is an order management software for food delivery companies. It is a delivery-oriented system that allows clients to order from multiple restaurants at the same time, and provides the possibility to order individually or in a group.**

Orders can be placed by users through the web interface. The data related to restaurants, foods and orders is managed by administrators. A mobile application is used by the delivery personnel. Both client applications are served with data by a central server.

The article presents the architecture and the implementation of the software system. The technologies, tools and methods used during the development process are also described.

I. Introduction

The Netfood project aims to develop a delivery-oriented order management system that allows users to order from multiple restaurants simultaneously and helps the work of the delivery personnel in tracking the orders.

There are many applications for food ordering from local restaurants in particular cities. Some systems support group orders too, but these are usually restaurant-oriented: an order can only contain items requested from a single restaurant. This model works well where the delivery can be solved separately by the restaurants, but there are settlements (e.g. small towns) where multiple restaurants are working together with the same delivery company.

As a concrete example, Odorheiu Secuiesc can be mentioned, where most of the local restaurants are in a partnership with a single delivery company. Currently, if someone wants to order, the client needs to call the company, which is not the most convenient solution. A delivery-oriented order management software could greatly improve the situation.

Netfood provides a web interface for users to place orders, to view restaurants and menus. The interface also provides the possibility of creating individual and group orders. Contrary to other systems, Netfood allows orders, which can contain items from all of the restaurants associated with the delivery company, no separate order is required for each restaurant.

The system also has an administration interface where members of the delivery company can track and manage the received orders: they can create new restaurants, menus, and modify existing ones.

The work of the delivery personnel is helped by a mobile application. This allows them to view the received orders, accept these orders, and receive all the information they need for successful delivery.

II. The Netfood Project

A. Functionalities

The server has two main tasks: to serve the client requests and to communicate with the database. It is connected to a web and a mobile client application. The communication between the peers is implemented by a RESTful API. The data is stored in a relational database.

The users and the administrators can log in to the system through the web interface, with a Facebook account or using a registered username.

On the opening page of the web application the daily menus are displayed for the current date. The date can be changed, and other food categories can also be accessed from the menu. The food items can be added to the shopping cart. Users have the possibility to remove the previously selected foods from the cart and to modify their quantities. After specifying the delivery address and the phone number an order can be placed. The ordering form is filled with default values from the user profile if available.

Group orders can be also initiated by inviting friends for the current order. The whole content of the shopping cart will be visible for each participant. Anyone has the possibility to invite new users to join the order. State indicators are displayed and updated when the members finish their orders. When all the participants finished, the order can be placed.

Additional features are also available for the clients: they can view the orders that are already submitted, the restaurants related to the delivery company, they have the option to edit their profile, and become friends with other users with whom they can start group orders later.

System administrators can view the orders received on a selected date. They can manage (add, delete, modify) foods, restaurants, categories, user roles (the client role can be changed to delivery or administrator).

The mobile application helps the delivery personnel to manage the incoming orders. Only users with delivery role can use the application. After a successful login, the orders are displayed for the current date. Both individual and group orders can be accepted partially, by selecting individual items from the corresponding list. Orders that have been already accepted can be viewed in a separate list, where the information required for the delivery is also displayed, including the customer's phone number, which can be called directly

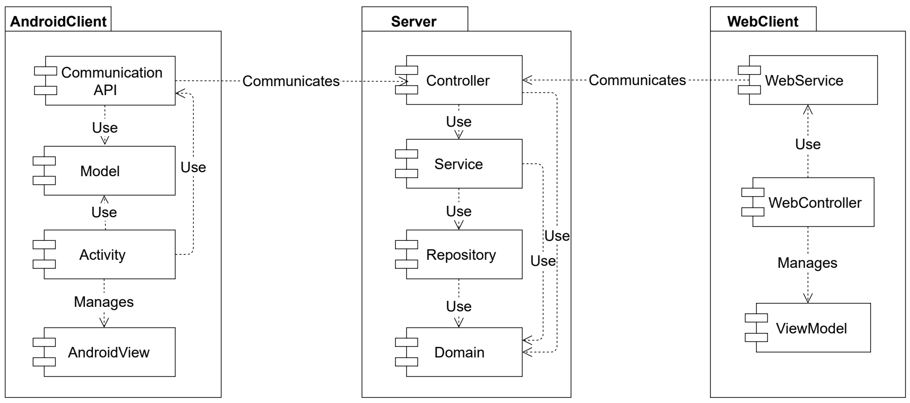


Fig. 1: The architecture of the system.

from the application. The system can be notified after the completion of an accepted order.

B. Architecture

The system consists of three main components: a Java-based server, an Angular 4 web and an Android client application (see Fig. 1).

The server has a multilayer architecture. The Repository module is responsible for the communication with the database and for any data manipulation task. The data is represented by model classes that can be found in the Domain module. These POJO classes are used by all the server components. The incoming REST requests are received by resources from the Controller module, and the responses are created using the appropriate components from the Service layer. The Service layer contains the business logic, using the Repository layer to manipulate the data required to perform the requested operations.

The Android client application follows the Model-View-Controller (MVC) principle. The Model layer contains classes representing the data. The data is displayed on the phone screen by views from the View Layer, the controllers are Activity classes. The application uses the Communication API to send REST requests, and to receive responses from the server.

The web client communicates with the server through the WebService layer, and the services are used by the WebController layer. This module is also responsible for controlling the ViewModel, which displays the user interface.

III. The server

The server is implemented using the Spring Java framework [1]. It provides an Inversion of Control (IoC) container, which is used for component management and configuration. It has support for the Dependency Injection (DI) pattern, used for managing the dependencies between components.

A. Spring Boot

Spring Boot [2] simplifies the process of creating Spring applications, speeding up the development by using default configurations for different application types. It also provides built-in webservers: Tomcat is used within the Netfood project.

B. Data Model

The model classes (see Fig. 2) are represented by Java Persistence API (JPA) entities, having private attributes with public getters and setters, and a public constructor without arguments. The classes are annotated with @*Entity* and @*Table*, each entity corresponds to a table from the database. They are stored in the *edu.codespring.netfood.domain* package.

One of the most important entities is the *Food* class, which is connected to the *Category* class with aggregation (n:1 relation). The *Choice* class is connected with aggregation (n:1 relation) to the *Food* and *Basket* classes. Another important entity is the *NetFoodUser* which stores the user data (name, email, password, etc.). The addresses are represented by the *Address* class, which is connected to the *NetFoodUser* class with aggregation (n:1 relation).

C. Data Access Layer

The data access layer creates a connection with the database and manipulates the data. A MySQL database is used for data storage. The Hibernate Object-Relational Mapping (ORM) framework is used as a JPA implementation by the server, and an abstraction layer is created over this framework using the Spring Data JPA module. The entities are annotated with JPA annotations and the Repository interfaces are derived from the JpaRepository interface. These interfaces declare methods for data manipulation, using method names based on specific conventions. The framework is able to generate the correct queries based on these method names. More complex queries can be created by using the @*Query* annotation followed by a JPQL command.

The NetFood repository interfaces are stored in the *edu.codespring.netfood.repository* package.

D. Service Layer

The service layer is responsible for the business logic of the project. The requests are routed here by the controllers and this is the place where the data is processed. The service

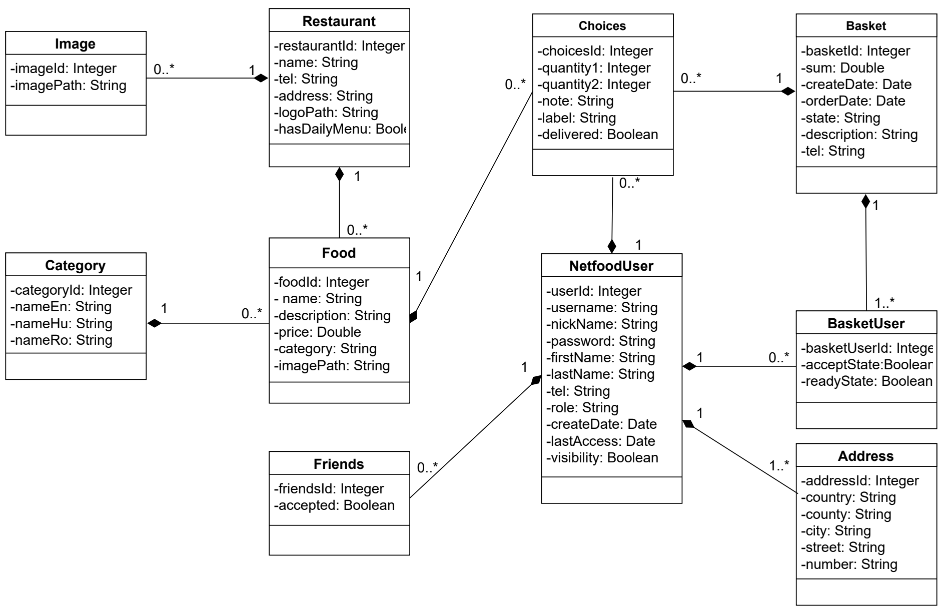


Fig. 2: Class diagram representing the model classes, their attributes and the relationships between them.

components are communicating with the data access layer for performing data manipulation.

The service classes are annotated with *@Service* and they are stored in the *edu.codespring.netfood.service* package.

E. RESTful API

REST (Representational State Transfer) is an architecture model used for client-server communication, typically allowing data exchange based on the HTTP protocol. The data and the operations are represented as resources, and these resources can be manipulated by simple operations, based on a HTTP analogy:

- POST: create resource
- DELETE: delete resource
- GET: get resource state
- PUT: change resource state

Various formats can be used for data transfer, e.g. XML, JSON, HTML, PDF, etc. The NetFood system uses the JSON format.

F. Spring Web MVC

The Web MVC framework can be used for creating MVC (Model-View-Controller) web applications and RESTful web services.

The framework is built around a DispatcherServlet, this receives all of the requests and forwards them to the handlers. The handler classes are annotated with *@RestController* and *@RequestMapping*.

The resources are associated with controller classes. The *@RequestMapping* annotation indicates the handler where the incoming request is forwarded. The requests are received by the methods from the controller classes, and after a preprocessing they are forwarded to the service layer.

G. Spring Security

The Spring Security framework is responsible for the application's security, primarily for user authentication and permission management. The framework is flexible and upgradable, therefore a token-based authentication and authorization can be easily implemented. In the Netfood project JWT (JSON Web Token) tokens are used.

JWT is an open standard that determines the safe transfer of information between communicating parties using JSON objects. The token consists of three strings separated by dots. The first one is the header, this part stores the token type and the hashing algorithm. The second part contains the payload and the third part is the signature.

If the user successfully authenticated himself, then a JWT will be generated, and the server sends it back to the client. The client stores the token and when a resource is required from the server, the token will be included into the authorization header of the request, in the following form: **Authorization: Bearer <token>**. The server checks the validity of the token and if the test passes, then the access will be granted to the requested resource.

H. Mail service

After a successful registration the system sends a confirmation e-mail to the user. The mail service is also used for password recovery. In both cases the e-mail contains a link for authentication. This notification system is implemented using the *JavaMailSender* interface provided by the Spring framework.

The link sent through the e-mail contains the JWT and the URL address to which the user will be redirected. The token generation is based on the e-mail address and a validity period is associated to the token.

I. Liquibase

Liquibase is a database version tracker for managing the state changes of the database during development. Every change is logged in the *changelog* file, which can be in a XML, YML, JSON or SQL format. The NetFood project uses the XML format.

The database update operations are partially automated using Liquibase. The configuration file contains the database name and the information required for creating a connection. Liquibase connects to the database, and if the changelog file is modified then the new operations will be executed. The changes are grouped in *changesets*.

IV. The Web Interface

The web interface is a single page application (SPA). It is developed using the Angular 4 [3] front-end framework. The framework is responsible for managing the web components and it resolves the dependencies between them. It provides data binding (*one-way data binding* and *two-way data binding*) support, linking the data model to the view. It also provides Angular directives, which are used in the HTML pages, so the view is made up from these dynamic pages.

TypeScript is used as script language, because it supports the object-oriented programming paradigm and it provides the ability to use types and classes.

In order to make the web interface responsive, Bootstrap and ng-bootstrap UI elements are used in the views. Contrary to the Bootstrap framework, ng-bootstrap does not use jQuery or other JavaScript libraries. With the combination of these two frameworks, the web interface has a basic style and a uniform setting for the rendering units provided by different browsers. At the same time, it provides an optimal look: the interface adapts to the current device, the same content will be displayed differently for two screens with different resolutions. In this way the Netfood website can be conveniently browsed on mobile devices too.

Users have the possibility to choose the most appropriate language on the web page. Currently they can choose between English, Hungarian and Romanian. For internationalization and fast switching between languages ngx-translate is used. This library provides a *TranslateService* component, which loads the correct JSON file corresponding to the chosen language and changes all multilingual labels on the page. In the JSON files the texts are stored in **key: value** form, and the keys are used for specifying the content of the labels.

The web client communicates with the server using the Angular HttpClient [4] module. There are restricted resources within the Netfood project, accessible only from specific user roles. For accessing these restricted resources interceptors are used, provided by the HttpClient. Each HTTP request is processed by an interceptor, which adds the token, that was received after login, to the header of the request, then forwards the request to the server. In this way the user can be identified by the server and the access can be granted to the requested resource.

Similarly to resources, some services can be accessed only by users with proper permissions. The web interface supports two types of user roles: client and administrator. The paths to restricted interfaces are protected by route guards. A route guard will only allow a user to load a restricted view or to access a service, if certain conditions are met (e.g. the user is logged in, the user is in a certain role, etc.), otherwise the user is redirected to a default page. Such a route guard class must implement the *CanActivate* interface and the *canActivate* method in which the appropriate conditions are checked, and if the return value is *true*, then the user will be able to access the desired feature. These classes need to be assigned to the components' paths in order to allow only users having certain roles to access them. If there is no assigned route guard to a path, then it will be available for everybody.

V. The Android Application

The Android application is developed for order management and it is used by the delivery personnel.

Android [5] is a Linux-based operation system which is mainly used on smartphones and tablets. The current version is *8.1 Oreo*. The Android platform has a multi-layered architecture: Linux kernel, Hardware Abstraction Layer (HAL), Native C/C++ Libraries, Android Runtime, Java API framework and System Apps. The system app layer contains the applications (e-mail, SMS service, internet browsing, etc.). New applications are created using the Java API Framework.

The *Android Software Development Kit* (Android SDK) is required for Android application development. The SDK contains tools used during the development process: debugger, libraries, documentation, example code, emulator, etc. The primary programming language is Java, the views are defined in XML files.

A. Gradle build system

The Gradle [6] build and dependency management system is used to build the Android application. The build process is described in the **build.gradle** script file using a Groovy-based DSL (Domain Specific Language). Groovy is a dynamic language, it is used for script writing for the Java platform.

B. Retrofit

The communication between the server and client is based on REST requests and it is implemented using the Retrofit [7] framework. Retrofit is a REST client for creating, sending, receiving and processing requests. It also solves the serialization and deserialization of the objects.

Three elements are required for working with Retrofit. The first one is an interface with annotated methods (*@GET, @POST, @PUT, @DELETE*) for the REST requests. The second one is the Retrofit class which helps the implementation of the previously mentioned interface. In order to avoid code duplication the *ServiceGenerator* class is used. The interface is implemented with the use of the *createService()* method. The third element is a set of POJO classes. These classes represent the data model which is similar to the model created on server side.

C. User Interface

The UI is created using Activity classes, playing the controller role within the application. Each Activity has an associated XML view description file. Here are defined the elements of the view and their layout. Each element has an identifier. Using this identifier they can be reached and modified by the Activity class. Multiple fragments can be displayed in a single Activity. The fragments are usually responsible for a single feature and their life cycle depends on the Activity. Intents are used for switching between Activities.

The Netfood Android application has three main activities. The first one is the login interface. After a successful login the application switches to the main activity, which contains multiple fragments, where the orders can be managed by the delivery personnel. The third activity can be used for password recovery.

Internationalization

Similarly to the web client, the Android client interface is also available in multiple languages. The Android platform has built-in support for internationalization. The texts displayed on the views have to be specified as resources in the *res/values/strings.xml* file. When a text has to be displayed, it can be referenced in the layout descriptor XML file by its identifier. Inside the *res* library a file has to be created according to the desired language. In this XML file, the values of all resources have to be specified, in the desired language, based on the identifiers. The application takes the resources from the file which corresponds to the default language of the device. If the translation is missing, then the application will search for the default resource value in the *values/strings.xml* file.

VI. Development Tools and Methodologies

The development process of the Netfood project followed the Scrum agile software development method. Git was used for version control, GitLab as a project management tool, and also for managing the central repository and for assisting the continuous integration process. All functionalities were developed on separate branches. Once a functionality had been completed and approved, the corresponding branch was closed and merged into the central development branch.

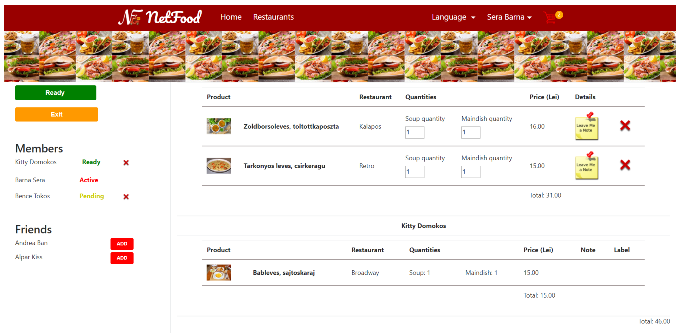
Three environments were used during the development: STS (Spring Tool Suite) for the server, Android Studio for the Android application and Visual Studio Code for the Web interface.

Maven was used as build automation and dependency management tool for the server, Gradle served this role in the case of the Android application, and the web client was managed with the npm (Node Package Manager) system.

VII. Using the Netfood System

A. The Web Interface

On the main page of the web interface the available foods are displayed. Users have the option to log in with a registered user or via Facebook by accessing the *Login* menu. They can



Kitty Domokos	Ready
Barna Sera	Active
Bence Tokos	Pending

Andrea Ban	ADD
Alpar Kiss	ADD

Product	Restaurant	Quantities		Price (Lei)	Details
Zoldborsoleves, toltottkaposzta	Kalapos	Soup quantity 1	Maindish quantity 1	16.00	
Tarkonyos leves, csirkeragu	Retro	Soup quantity 1	Maindish quantity 1	15.00	

Total: 31.00

Kitty Domokos

Product	Restaurant	Quantities		Price (Lei)	Note	Label
Bableves, sajtoskaraj	Broadway	Soup: 1	Maindish: 1	15.00		

Total: 15.00

Total: 46.00

Fig. 3: Shopping cart for a group order

also request password recovery, or they can navigate to the registration page.

Logged in users can view or edit their profile under the *My Profile* menu. They can view their submitted orders and manage their friends. The user database can be searched by names or usernames. Friend requests can be sent to users, and the received requests can be accepted. Orders can be placed by adding the desired foods to the shopping cart, and group orders can be initiated too.

After initiating a group order (see Fig. 3), the friend list appears in a sidebar and the users can be invited to the group. The invited persons will receive a notification. The invitations can be accepted or rejected. Each participant has the possibility to exit the group or to invite additional users. Status indicators are displayed, and when everyone is ready, the order can be placed.

For the administrators the received orders will be displayed on the main page. They can add new restaurants, modify existing ones or delete them under the *Restaurants* menu. Under the *Menu* option they can manage daily menus and other food categories. The user roles can be modified under the *Users* menu.

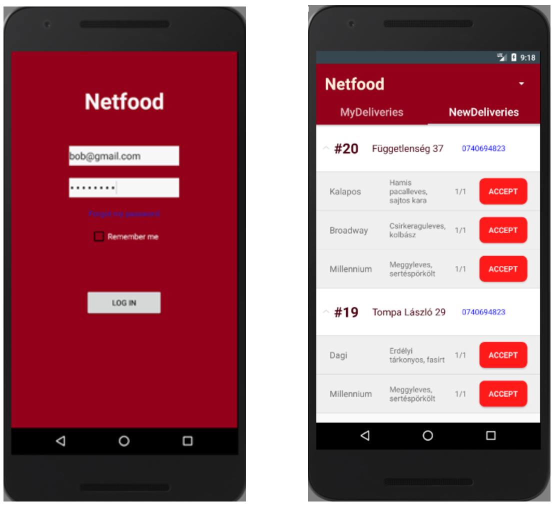
B. The Android Client Application

The Android application (see Fig. 4) can be used by the users with delivery role. When the application is launched, the *"Login"* and *Password Recovery* functions are available.

After a successful login, the accepted orders are displayed under the *My Deliveries* menu. If an order has been delivered, its state can be set to completed. For each order the shipping address will be displayed, and the customer's phone number can be called directly from the application. Orders that are not yet accepted can be viewed under the *New Deliveries* menu. These can be opened and accepted or partially accepted by the user.

VIII. Conclusions and further development

Within the Netfood project a software system has been developed for helping restaurants and food delivery companies. Users can create individual or group orders through the web interface. The menus, restaurants, users, and orders can be managed by the administrators. The delivery process is



bob@gmail.com

••••••••

#20	Függetlenség 37	0740694823	
Kalapos	Hamis pacalleves, sajtos kara	1/1	ACCEPT
Broadway	Csirkeraguleves, kolbász	1/1	ACCEPT
Millennium	Meggyleves, sertéspörkölt	1/1	ACCEPT
#19	Tompa László 29	0740694823	
Dagi	Erdélyi tárkonyos, fasírt	1/1	ACCEPT
Millennium	Meggyleves, sertéspörkölt	1/1	ACCEPT

(a) Login page (b) New Deliveries page.

Fig. 4: Android application

supported by the Android application: the couriers are imediately informed about new orders, they can accept deliveries receiving all the required information.

During the development process new functionalities emerged as further development possibilities:

- an interface for restaurant owners for directly managing their offers;
- the possibility for assembling daily menus from already uploaded foods;
- after placing an order the customer should receive a message with the estimated time of delivery;
- iOS version for the mobile application;
- Google Maps integration into the mobile application, to navigate the delivery personnel from the current location to the delivery address;
- a stand-alone mobile application used by customers for placing orders.

REFERENCES

[1] R. Johnson, *The Spring Framework - Reference Documentation*, [Online], Available: http://docs.spring.io/spring-framework/docs/2.0.x/reference/index.html.
[2] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch, A. Wilkinson, M. Overdijk, C. Dupuis, S. Deleuze, M. Simons, V. Pavi'c, J. Bryant, M. Bhave, *Spring Boot Reference Guide*, 2012-2018.
[3] Oswald Campesato, *ANGULAR 4 Pocket Primer*, 2017.
[4] *HttpClient*, [Online], Available: https://angular.io/guide/http
[5] *Android* [Online], Available: https://developer.android.com
[6] *Gradle official website*, [Online], Available: https://gradle.org
[7] *Retrofit - Square*, [Online], Available: http://square.github.io/retrofit/