Software System for Newspaper Distribution

Iulia-Kinga Marton*, Zoltán Szabó**, Károly Simon***, Norbert Kandó ****
* Babeş-Bolyai University, Cluj-Napoca, Romania
** Babeş-Bolyai University, Cluj-Napoca, Romania
*** Codespring LLC, Babeş-Bolyai University, Cluj-Napoca, Romania
**** Codespring LLC, Cluj-Napoca, Romania
kinga_marton@yahoo.com
szabozoltanbors@gmail.com
simon.karoly@codespring.ro
kando.norbert@codespring.ro

***Abstract*—The paper presents a software system for newspaper distribution. The system contains a central server, an Android client application and a web application for data administration.**

Application data (customer database, distributor profiles, distributions, etc.) is managed by the system administrators using the web application. The application also generates statistics and provides opportunity to create checklists for verification.

The mobile client application is used by the distributors. Using this application, they can view their task list, the nearest addresses where they have to deliver newspapers and they can report feedback related to deliveries.

I. INTRODUCTION

The article presents the features, architecture and implementation of the "Newspaper Distribution" (NPD) software system. The main purpose of the software is to facilitate the work of newspaper publishers and distributors.

Without using proper software solutions for supporting the distribution process, related data management is not an easy task. Sometimes the data is stored in files (e.g. Excel files) and it is handed over to the distributors in a printed format. Using this method, it is hard to keep the data organized and on the other hand, the printed version is easy to lose, and searching for unknown addresses implies a great effort. The lack of feedback regarding the deliveries could be a problem as well.

The aim of the NPD system is to eliminate the mentioned problems.

The software includes a central server, a mobile application and a web-based administration interface. The data can be managed by the system administrators using the UI provided by the web application. The distributors can check their tasks using the mobile application. They can see the nearest addresses where they have to deliver newspapers, and they have the possibility to send feedback about the delivery process. Based on these feedback statistics can be visualized on the administration web user interface. The system also provides support for generating checklists in order to verify the distributors.

The first section of the article presents the project itself with its requirements and architecture. This section is followed by the description of the used technologies. In the third section some details are described about the implementation, including the main challenges encountered during the development process. The next section presents the used tools and methods through the project followed by the usage of the software. At the end of the article the conclusions and possibilities of feature enhancement are presented.

II. THE NPD PROJECT

A. Requirements

The main features available through the UI provided by the web application are the following:

- list, sort and filter the customers' data;
- manage the customers' data;
- manage the distributors' data;
- assign streets to distributors;
- manage the distributions: create, start and stop distributions;
- list, sort and filter distribution-related data (the list of the customers assigned to different distributors);
- view distribution-related statistics;
- import the customers' data from Excel file;
- generate checklists for distributor verification, save feedback and ratings received from the customers.

The main features provided by the mobile client application are:

- After logging into the application the task list is synchronized (the distributor receives the list of his assigned tasks).
- Automatic address suggestion: the nearest address is automatically selected by the application based on the current position of the distributor.
- Sending feedback: the distributor can send feedback regarding the deliveries. There is a possibility to choose feedback messages from a

predefined list (e.g. "*successful delivery"* or "*did not open the door"*) and custom comments can be formulated too.

- GPS coordinate refinement: the coordinates assigned to an address can be modified by the distributors based on their current position;
- Local data storage: the data has to be stored locally on client side and synchronized later with the server. This mechanism could be useful when internet connection problems are encountered during the distribution process.

B. *Architecture*

The NPD system can be divided into two main parts: the server and the mobile client application.

The server includes further subsystems. It provides the data model, communicates with the relational database management system and it contains the Repository Layer for the management of the persistent data. It also provides a Service Layer, which contains methods related to the business logic of the application. These methods can be accessed through service interfaces.

The web components are situated in the Web Application module. The responsibility of these components is to provide the web UI for data administration.

Some features are published as RESTful web services. These services are implemented by the REST service module. The Android application communicates with the server through this RESTful API [1].

The project architecture, including the relations between the subsystems, is shown in Figure 1.

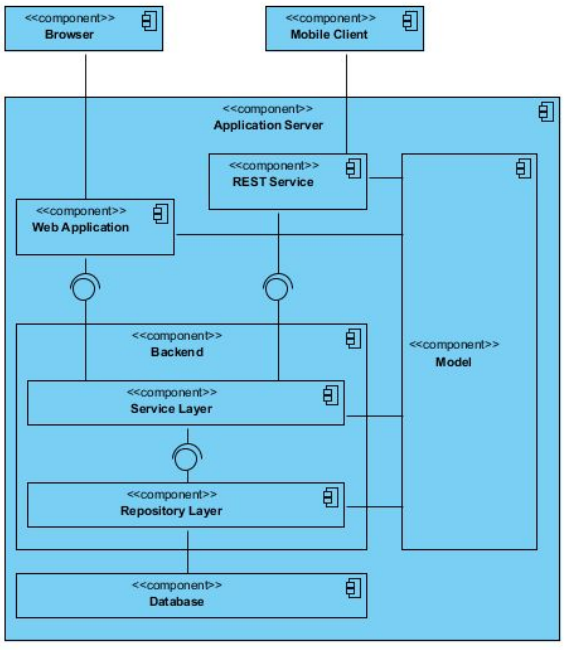

Figure 1: Architecture of NPD

III. USED TECHNOLOGIES

The NPD project is developed in Java and JavaSrcipt programming languages, using advanced Java technologies.

The server side is based on the Java Enterprise Edition (Java EE) [2] specification. The Glassfish application server is used, as the reference implementation of this specification. The components are Enterprise Java Beans (EJB) [3] in the repository, service and API layers. Specifically, stateless session beans are used, communicating through local interfaces. The full life-cycle of these components is managed by an EJB container provided by the Java EE application server, according to the Inversion of Control (IoC) [4] design pattern. The dependencies between the components are managed based on the Dependency Injection (DI) design pattern.

The model classes representing the main entities within the system are implemented based on the Java Persistence API (JPA) [5] specification and are validated based on the BeanValidation specification. EclipseLink [6] is used as a persistence framework, which is the reference implementation of the JPA specification.

The newspaper distribution processes are managed separately in different regions. The region-specific data (customers, distributors, etc.) has to be isolated: each user has only rights for accessing data related to a single region. This isolation is implemented by using the multi-tenancy mechanism. The multi-tenancy support is provided by an EclipseLink-dependent implementation, based on EJB interceptors.

The web interfaces are created with the Vaadin framework [7].

The client-server communication is based on RESTful web services and it is solved using the Jersey framework, which is the reference implementation of the Java API for RESTful Services (JAX-RS API) specification.

The system security relies on two different mechanisms. At the level of the REST API a token based authentication is used. Within the Web Layer user credentials are used for authentication and authorization, according to the Java Authentication and Authorization Service (JAAS) [8] specification.

As mobile development tool Android SDK [9] has been used. Within the mobile application the map is displayed using the Google Maps API. The Retrofit library is used for sending HTTP requests, and for making conversions between the Java Data Transfer Objects and their JSON representations. Database operations are supported by the OrmLite persistence framework.

IV. IMPLEMENTING THE PROJECT

A. *NPD Backend*

The NPD Backend module provides the model package containing the main entities of the system, the data access layer and the business logic components accessible through the service layer.

The model package contains the central entities, managed according to the JPA specification. The Java Persistence API (JPA) is a Java specification for relational data management in Java applications. The model classes are JPA entities, with private attributes

accessible through public getter and setter methods. Most of the entities extend the BaseEntity class, which implements the Serializable interface and extends the AbstractModel class. The MultitenantEntity inherits from the BaseEntity class and it is used for realizing the multi-tenancy. All entities have to be unequivocally identified, so that the AbstractModel class contains a universally unique id (UUID). Furthermore, each entity has an id, which is generated by the database management system and it is inherited by the entity classes from the BaseEntity superclass. For separating the data related to different regions, a tenant id is provided by the MultitenantEntity superclass for each multi-tenant entity.

The repository package contains interfaces for specifying the data access operations, and a sub-package containing the implementations for these interfaces. The operations are implemented within managed components, corresponding to the EJB specification. Stateless session beans are used, communicating through local interfaces. The repository components are organized into a hierarchy. The core database operations (CRUD operations) are implemented within a superclass. This superclass is extended by the repository components. For each entity class a separate repository class is considered, providing specific data access operations for the given entity. Most of the operations are implemented using Java Persistence Query Language (JPQL) queries. There are also queries with a structure, which can be determined only at runtime (e.g. complex filtering conditions). These queries are created dynamically using the Criteria Query API included into the JPA specification. In case a data access operation fails, the exception is logged and a layer specific exception is propagated to the upper layers.

The multi-tenancy support is realized based on an EclipseLink-specific implementation, using EJB interceptors included into the interceptor package. All data access operations related to multi-tenant entities are intercepted by these interceptors. Based on the authenticated user the proper tenant id is always configured before resuming the intercepted method execution.

The components within the service package are responsible for the more complex business logic operations. The service layer components are also EJBs, and they are communicating with the repository layer through local interfaces. The operations provided by these components are also published via local interfaces. The DI design pattern is used for managing the dependencies between the components. The dependencies are injected by the EJB container, based on EJB or Context and Dependency Injection (CDI) annotations.

The security mechanism is based on the JAAS specification. The Java Authentication and Authorization Service is a Java specification for implementing a standard pluggable authentication module and information security framework. In this project the JAAS implementation is provided by the Glassfish application server. For server side user management, a Glassfish security realm is created, which contains a collection of users assigned to different groups. The groups are assigned to security roles. The authorization process is based on these roles. A JDBC security realm is used, the security-related data is stored within the application's database.

B. RESTful API

The communication between the mobile client and the server is based on RESTful web services. According to the REST architectural style the resources are accessed using Uniform Resource Locators (URL) and the REST commands are mapped to HTTP methods (GET, POST, PUT, DELETE, etc.).

The RESTful web services were implemented using the Jersey framework, which is the reference implementation of the Java Api for RESTful Services (JAX-RS API) standard. The module contains two packages: a resource and an assembler package. The resource package contains the Data Transfer Objects (DTO). These resource classes receive the REST requests and prepare and send the appropriate responses. The classes that make the conversion between DTOs and the entities are placed in the assembler package.

C. Web and Widgetset

The administration web application is based on the Vaadin framework, only the login page and the error pages are created using the Java Server Pages (JSP) technology.

Vaadin is an open source framework for rapid and effective web application development. On the server side it uses Java Servlet technologies. On the client side HTML and JavaScript is used. The UI is created using server side Java components and the Google Web Toolkit (GWT) technology is used in the background for creating the views displayed on the client side. The communication is based on Ajax techniques.

Within the NPD system the management of the web components is realized based on the Context and Dependency Injection pattern. The Vaadin CDI add-on is used for this purpose. The charts are created by using the dCharts Vaadin add-on.

The administration page can be accessed only by users with administrator rights. The authentication and authorization is done by the Glassfish server, according to the JAAS specification.

D. Mobile

The mobile client application communicates with the server via the provided RESTful API, so the requests are mapped to HTTP requests. Retrofit library is used for the implementation of the communication layer, including the conversion between the Java Data Transfer Objects and JSON objects. It collects the REST calls into Java interfaces, and the operations can be invoked after instantiating a corresponding RestAdapter.

Application settings are stored in the SharedPreferences, while the application data is stored in a local database. Local data manipulation is realized by the OrmLite framework. It is a lightweight persistence

framework that allows mapping Java objects to the database. Object-relational mapping can be specified by using Java annotations. The framework also supports transactional processing.

For implementing the map services the Google Maps API is used.

V. TOOLS AND METHODS

During the development process agile development principles [10] have been followed, using Scrum [11] methodologies. For project management a web based Kanban board has been used, provided by the Trello [12] project management tool.

Mercurial [13] has been used for version control, together with the TortoiseHg [14] client application and the RhodeCode central repository management system.

The build processes and the dependency management have been ensured by Apache Maven [15] and Gradle [16]. The Maven-Gradle plugin has been used, in this way the Gradle build for the Android application is also launched by Maven. Artifactory has been used as a repository management system for the external dependencies. Jenkins served as a Continuous Integration [17] system. XWiki is used for editing and sharing project-related information and documentation.

The code quality has been maintained using SonarQube, which covers eight quality aspects: comments, coding rules, potential bugs, complexity, unit tests, duplications and architecture.

VI. USING THE NPD

After a successful authentication the administrator can access data related to customers, distributors, distributions and statistics by using the web UI. Each administrator has access only to data corresponding to a given region. The administration page is composed of six main tabs: *Distribution, Distributors, Customers, Statistics, Verification*, and *Settings* (Figure 2).

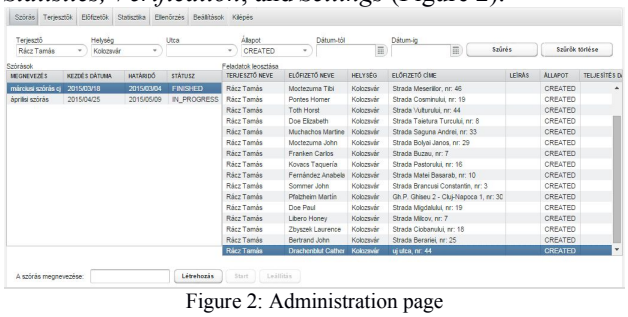

Figure 2: Administration page

The *Distribution* tab gives the possibility to control the distribution cycles (creating, starting and stopping distributions). It also provides information regarding the tasks from a selected distribution. These tasks can be filtered based on different criteria: distributor, street, status, date or any combination of these.

The *Distributors* page (Figure 3) is used for distributor data management, for registering new distributors, modifying existing profiles and assigning streets to distributors. Also, for each distributor an average rating is displayed based on customers` feedback.

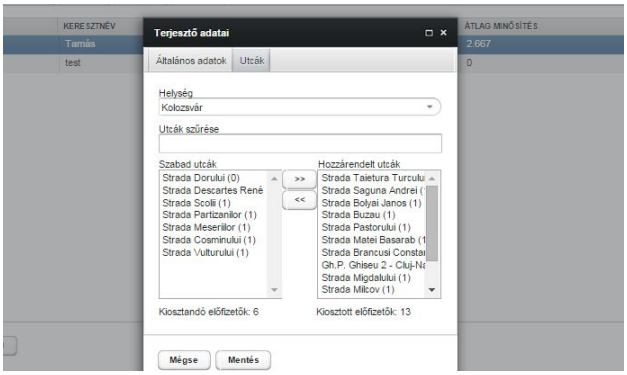

Figure 3: Distributor profile update on the administration page

The *Customer* tab is used for managing the customers' data. The list of customers can be filtered based on different criteria: name, street, status or any combinations of these.

The application generates different statistics based on distributor feedback. These statistics are presented under the *Statistics* tab (Figure 4). There are three types of statistics: a pie chart with the current state of a selected distribution, a pie chart displaying the performance of a selected distributor within a selected distribution, and a third chart showing this data in a daily breakdown.

The administrators can generate different checklists using the *Verification* tab. These checklists can be used to verify the distributors, by asking the selected customers about the quality of the service. The received customer feedback and rating is saved by using the web UI.

Customers' data can be imported/exported from/to Excel files using the *Settings* tab. The import process can be customized: the administrator has several options related to data processing (e.g. what happens if the data appears both in the database and the Excel file, or if the data appears in the database, but is missing from the Excel file).

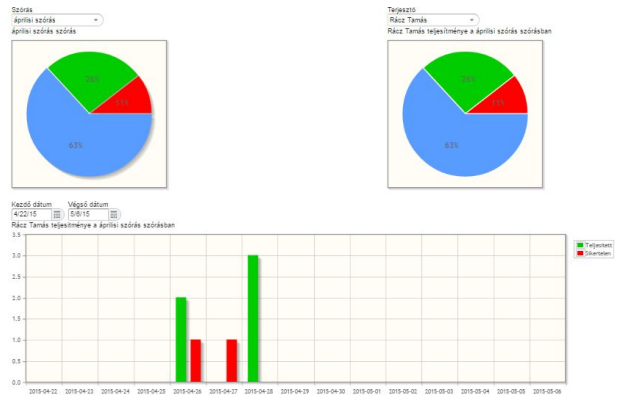

Figure 4: Statistics tab

The main goal of the mobile application is to provide a handy tool for the distributors to make the distribution process easier. They receive tasks related to a given distribution, get directions for each address and can send feedback about the deliveries. The application consists of

two main views: a map view providing directions and a list view, which contains the current tasks.

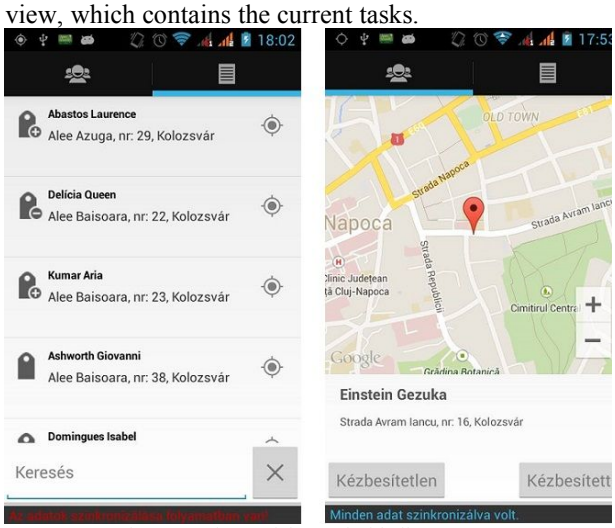

Figure 5: Views of the Android client

The map view (Figure 5) includes the "delivered" and "undelivered" buttons for sending feedback about deliveries. After pressing the "undelivered" button a pop-up window appears where the distributor can select from predefined messages indicating the reason of the failure and there is also a possibility to add custom comments. After sending a feedback, the application searches the next address based on the associated GPS coordinates. Within the list view the tasks are ordered by street names and house numbers and there are filtering possibilities by street name and status. A GPS icon is displayed for each address, in order to specify more accurate GPS coordinates.

The mobile application supports a caching mechanism, which ensures that the data is stored locally, even if there is a temporary Internet connection problem. The data is saved into a local storage and the application always checks if there is unsynchronized data, which has to be sent to the server. Both views have a status bar at the bottom of the screen, which indicates the status of the data (synchronized or not synchronized). After a logout operation all locally stored data are deleted. Before this data removal the user receives a warning message if the storage still contains unsynchronized data.

Users have several configuration possibilities within the application available under the Settings menu.

VII. Conclusions and Further Development

NPD is a software system for facilitating the work of newspaper distributors, providing monitoring and control tools for distribution managers. The system can be extended with several further features. The following are a few possibilities for extending the administration page:

- customers shown on a map, clustering support and eventually a heatmap based visualization;
- automatic validity check for addresses;
- elimination of the "one street - one distributor" constraint (in the current version only one distributor can be assigned to addresses within a given street);
- more algorithms for generating verification lists.

Some further features for the mobile application:

- notifications about new distributions;
- route proposals for a given address;
- different reports regarding previous and current distributions.

Moreover a central administration interface could be developed for managing the regions and regional system administrators and for providing global statistics.

Acknowledgment

The infrastructure required for the development process has been provided by Codespring LLC, and therefore the authors would like to express their special thanks for this support.

References

[1] Official web page of JAX-RS. [Online] [Cited: May 25, 2015.] https://jax-rs-spec.java.net/.

[2] Official web page of the Java Enterprise Edition. [Online] [Cited: May 25, 2015.] http://www.oracle.com/technetwork/java/javaee/overview/index.html.

[3] Andrew Lee Rubinger and Bill Burke, *Enterprise JavaBeans 3.1, 6th Edition;* O'Reilly Media, 2010.

[4] Martin Fowler, *Inversion of Control Containers and the Dependency Injection pattern;* 2004.

[5] Official web page of the Java Persistence API related resources. [Online] [Cited: May 26, 2015.] http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html.

[6] Official documentation of EclipseLink 2.5. [Online] [Cited: May 7, 2015.] http://www.eclipse.org/eclipselink/documentation/2.5/eclipselink_otlcg.pdf.

[7] Marko Grönroos, *The Book of Vaadin: Vaadin 7 Edition - 4th Revision;* Vaadin, Ltd., 2014.

[8] JAAS Reference Guide. [Online] [Cited: May 17, 2015.] http://docs.oracle.com/javase/7/docs/technotes/guides/security/jaas/JAASRefGuide.html.

[9] Official web site of Android. [Online] [Cited: May 17, 2015.] https://developer.android.com.

[10] Robert C. Martin, *Agile Software Development, Principles, Patterns and Practices;* Pearson Education, Prentice Hall, 2002.

[11] Kin h. Pries, John M. Quingley, *Scrum Project Management*; Taylor an Francis Group,LLC., 2011.

[12] Official web page of the Trello. [Online] [Cited: May 25, 2015.] http:// trello.com.

[13] Official web page of Mercurial. [Online] [Cited: May 25, 2015.] http://mercurial.selenic.com.

[14] Official web page of TortoiseHg. [Online] [Cited: May 25, 2015.] http://tortoisehg.bitbucket.org.

[15] Official web page of Apache Maven. [Online] [Cited: May 25, 2015.] http://maven.apache.org.

[16] Official web page of Gradle. [Online] [Cited: May 25, 2015.] http://gradle.org.

[17] Jez Humble, David Farley, *Continuous Delivery: reliable software releases through build, test, and deployment automation*; Pearson Education, Inc., 2011.