OptInv: Software as a Service Solution for Inventory Optimization

Réka Győri, Tamás Imecs, Enikő Török, Károly Simon
Codespring LLC, Babeş-Bolyai University
gyori_reka92@yahoo.com
torok_eni@yahoo.com
tomy_imecs@yahoo.com
simon.karoly@codespring.ro

Abstract—**The presented OptInv software system provides support for inventory and sales optimization.**

Generally, small and medium enterprises cannot afford expensive Enterprise Resource Planning (ERP) systems, with effective supply management modules. Supplies are often managed based only on former experience. Inconvenient situations can occur when an order cannot be accomplished because the needed product is missing, or when the money invested in some products becomes a waste. Using the right methods and tools, these problems can be eliminated.

The aim of the OptInv project is to provide a Software as a Service (SaaS) solution for this kind of problems. OptInv imports data from accounting software systems and provides statistics about products, providers and sales. Products can be categorized, the value of the stock can be estimated, the consumption can be predicted and in this way the inventory can be optimized.

I. Introduction

The presented OptInv project provides support in inventory optimization for companies working with commerce. Keeps count of the inventory stock, provides different statistics and reports, which are shown on a web user interface.

Generally, ERP systems provide effective solutions for maintaining and managing the inventory. However, not every software provides efficient supply optimization. Small and medium enterprises cannot afford expensive ERP systems, so the inventory is managed based only on former experiences. This is a major problem, which can lead to unpleasant situations: an order cannot be accomplished because of the lack of the desired product, or money is wasted in huge amounts of unwanted products. By using the right methods and tools these problems could be avoided, money and inventory space could be saved and used more wisely.

By using the OptInv software users can inspect statistics related to their stocks. Statistics are made based on data imported from accounting software databases (e.g. HamorSoft - hMARFA).

There is possibility to import data related to products, sales and providers from different accounting software systems. Based on these data, the system generates different reports. In this way, the user can easily see, if there are on stock large quantities of products with small average consumption. Frequently required products can also be observed. In this way, orders can be made effectively and costumer demands can be easily satisfied.

Using the application products can be easily categorized. This could be a great advantage, because recurrent, sporadic and seasonal products have to be managed in different ways, different prediction methods have to be used for these categories.

II. Technologies

OptInv is built on the Java Enterprise Edition (JEE) platform, taking advantage of the platform services, such as security and transaction management. On server side Enterprise Java Bean (EJB) components are used. The persistence layer is implemented using EclipseLink, the reference implementation of the Java Persistence API (JPA). The EclipseLink multi-tenancy support is also used. Beyond the above mentioned technologies, the software's quality is also ensured by other enterprise development frameworks.

A. Java Enterprise Edition

Java Enterprise Edition (JEE) [1][2] supports the development of multilayer, distributed, multi-user, scalable and secure enterprise applications. Glassfish is the reference implementation of the JEE specification, the application server used by the OptInv project.

The JEE frameworks provide different services to facilitate the development process, such as resource management, dependency injection, persistence, transaction management, security, timer services, etc. In this way, developers can concentrate on the business logic and functionalities.

The server side JEE components have their own functionalities and they can communicate with each other. There are two main categories: web components, managed in the server's web container and EJB components, managed in the server's EJB container.

B. Enterprise JavaBeans

OptInv is using Enterprise Java Bean (EJB) [3] components on server side.

There are two categories of EJBs: Session Beans and Message-Driven Beans. There are two types of session beans: stateless session beans and stateful session beans. Stateless session beans do not preserve any state information between consecutive calls. Stateful beans are assigned to a client, and can preserve a conversational state through multiple calls. Message-driven beans are used for asynchronous message processing. Usually, they work as listeners for JMS (Java Message Service) messages; business logic procedures are executed as a reaction to these messages.

The communication between the components is realized through interfaces. Session beans can implement three types of interfaces: Local, Remote and Endpoint interfaces. The methods inside a Local interface can be accessed only from components which are inside the container. Remote interfaces enable remote clients to use the component's services. Endpoint interfaces are used in SOAP protocol-based web services.

Within the OptInv system stateless session beans are used in the repository, service and data import layers. The communication between the beans and the web components is realized using Local interfaces.

C. *Java Persistence API and EclipseLink*

The Java Persistence API (JPA) [2] can be considered an abstraction layer over the Java Database Connectivity API (JDBC) and it is a standard for object-relational mapping (ORM) in Java. It specifies meta-information required for mapping, an object oriented query language (Java Persistence Query Language - JPQL) and a dynamically built query support, the Criteria Query API. The JPA 2.1 specification is part of the Java EE 7 platform and different JPA implementations are provided by JEE application servers (e.g. EclipseLink, Hibernate etc.). EclipseLink [4] is the reference implementation of the JPA specification. This framework is used in the OptInv project.

The data model is represented by JPA entities. Object-relational mapping can be done in two ways, by using annotations inside the entity classes or by xml deployment descriptors. In the OptInv project the annotation mechanism is used. The relations between the entities can be unidirectional or bidirectional and can be also categorized by cardinality (one to one, one to many, many to one, many to many). These relations are also defined by JPA annotations. JPQL is used for creating queries.

Data manipulation is realized by using a central service, the EntityManager interface. When persistence operations are performed, the entities are attached to an EntityManager and become managed and synchronized with data stored in the database. Entities can also be detached from the manager, and in this case they are no longer synchronized. In the OptInv project transaction-scoped entity managers are used, so the entities are managed only during transactions.

D. *Transaction management*

The Java Transaction API (JTA) specification is supported by JEE application servers. Transactions can be used either in declarative mode or by program commands. In the second way the beginning (begin transaction) and the ending (commit or rollback transaction) of the transaction has to be marked explicitly.

In the case of declarative transaction management, transactions are managed by the container. By given attributes the EJB container decides the beginning and the end of the transactions.

In the OptInv project there are container managed and component managed transactions. Component managed transactions are used by the data importer module.

E. *Java EE security and JAAS*

The JEE security mechanism is based on the Java Authentication and Authorization Service (JAAS). In the JAAS model users can have different principals, which are assigned to different roles and each role has certain rights. All these together form a security realm, which can be configured on the application server. Realm data (usernames, passwords, user groups) can be read from files, databases or from other sources. In the OptInv application a JDBC realm is used, the data is read from the system's database.

There are two ways to provide security: in a declarative way or controlled by the program. The declarative method is based on annotations. If the security is controlled by the program the caller's principal can be requested runtime from the session context.

Within the OptInv project the security layer is very important in order to protect the companies' data. On the component level the declarative model is used. The security of user requested data is controlled by the program. All service methods are intercepted and in the interceptor the caller's principal is retrieved from the session context. Based on the caller's company the multitenancy support [5] is enabled. In this way users can only access data owned by their company.

F. *Interceptors*

In JEE interceptors are used for reacting to component's lifecycle events or method calls. Similar to aspect oriented languages, they are used to implement cross-cutting functionalities.

Interceptors can be declared by annotations or in deployment descriptors, and they can be implemented either in the target class or in a separate class. The methods of an interceptor can be automatically triggered by instantiation, removal from the container, timeout or method call. There can be multiple interceptors assigned to a class or method, the execution order can be defined by a priority level.

In the OptInv project all service methods are intercepted, and before the execution of the methods the caller is identified. This is how the multi-tenancy of the application is realized.

G. *EclipseLink based multitenancy*

In order to support multiple companies the multitenancy principle is used. The OptInv application has a single database and can serve multiple clients at the same time due to its multitenancy support.

There are several frameworks supporting multitenancy, EclipseLink is one of them. The model can include tenant aware and non tenant aware entities in the same time. The tenant-id can be set either while configuring the persistence context, or dynamically when an EntityManager is created.

In the OptInv application the configuration for the multitenancy is made by annotations. A single-table strategy is used. Tenant ids are set dynamically by interceptor methods before each service call.

H. Other technologies

The OptInv web interfaces are created using Vaadin [5], a Java-based web development framework. On the model level data validation is solved using Hibernate Validator, an implementation for the Bean Validation specification. Restrictions are specified by annotations. This method is supported by the Vaadin framework, too, so it is also used for validation on the UI level.

JDBC is used by the data importer module. The OptInv system communicates with the external databases (accounting software databases) using the JDBC API. In the case of Hamor databases a JDBC-ODBC bridge is used for importing .dbf files.

Logging is implemented using the slf4j API and the log4j framework.

III. Development Tools and Methods

The OptInv project has been developed based on Agile principles, using Scrum methods and Continuous Integration practices.

Mercurial serves as an open source, distributed source code management tool, RhodeCode is used for central repository management. Project management and issue tracking is supported by Redmine. The specifications and other documentations are shared between developers using the XWiki platform.

OptInv uses Apache Maven as build- and dependency management system, and Artifactory for repository management.

Code quality is ensured using SonarQube, a freely available, open-source code analyzer tool. Its responsibility is to search for duplicates, calculate test-coverage and complexity measures, check for common bad-practices and coding conventions.

Jenkins is a continuous integration tool. Using the version control system, it has the ability to build maven projects, and to run automated tests.

OptInv uses Glassfish application server and MySQL database management system.

IV. The OptInv project

In this section the main functionalities of the system are summarized, the application's data model and architecture is presented and the main aspects of implementation are briefly described.

A. Basic functionalities

The system is designed as a Software as a Service (SaaS) solution (Figure 1).

In the center there is an application server. Database files are uploaded to this server and the data is imported into the internal database. The system can be configured by administrators. Based on the imported data statistics are calculated and results can be visualized on a web interface. The design also includes a public API for communicating with third party ERP systems and a notification system for mobile client applications, but these parts are not yet implemented in the current version.

The project in its current state is a prototype, which contains the OptInv Backend, the OptInv Web and the OptInv Server subsystems.

The OptInv Backend provides the repository and the business logic layers. Includes the data importer module, which imports data from uploaded files. The data analyzer component is also provided by the Backend subsystem.

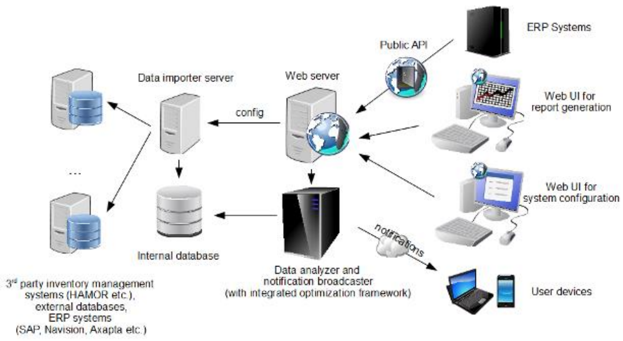

ERP Systems
Public API
Data importer server
Web server
Web UI for report generation
config
...
Web UI for system configuration
Internal database
notifications
Data analyzer and notification broadcaster (with integrated optimization framework)
User devices
3rd party inventory management systems (HAMOR etc.), external databases, ERP systems (SAP, Navision, Axapta etc.)

Figure 1. OptInv, Software as a Service

The OptInv Web is a web application, which communicates with the Backend. The data importer module can be configured for each company separately. The Web module also provides a view for uploading files. Reports and statistics generated by the data analyzer module can be visualized on other views.

B. The model

Entities are organized in a hierarchy. The AbstractModel is the abstract base class, which defines a universally unique identifier for each entity. It is extended by the BaseEntity abstract class, which defines a unique identifier corresponding to the primary keys and ensures serialization support for the entities. Entities are derived from this BaseEntity class.

The main entities of the OptInv project are: Product, Sale, ProductInfo, StockInfo, AcquisitionDetails, Provider and Invoice. Using these data different values can be calculated: the price of a product, the amount on stock etc. Operations and parameters are stored in Calculation objects, and their results are also cached in Result objects.

C. Architecture

The system has a multilayer architecture, the model is shared between these layers (Figure 2). The data access layer communicates with the MySQL database management system. The data importer module saves the data into the central database through the repository layer. The current version of the system has a Hamor-specific importer, but it can be easily extended with other importer modules.

Above the repository layer, there is a business logic layer. The web layer communicates with the backend via local interfaces published by the business components.

D. The main aspects of backend implementation

The OptInv system is based on the JEE platform, the data importer, repository and service layers use EJB components, more precisely stateless session beans. The dependency injection (DI) design pattern is implemented using EJB and CDI (Context and Dependency Injection) DI mechanisms.

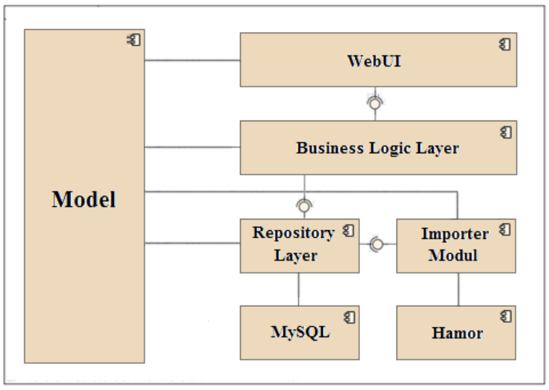


Figure 2. System architecture

The data model is represented by JPA entities, the persistence layer is implemented using EclipseLink. Object-relational mapping is based on JPA meta-information and it is configured using annotations. Repository components are organized into a hierarchy and the basic methods are inherited from a common base class. The Entity Manager is also injected into the base class. For validating data on the model level the Hibernate Validator implementation of the Bean Validation specification is used (by annotating entities).

Transaction management is used in a declarative way, the only exception is the data importer module, where transactions are managed by the components.

For authentication the application server's login mechanism is used with a JDBC security realm. Security related data (users and groups) is stored in the application's database.

By using EclipseLink, multitenancy is also supported. Single table strategy is used, because the same type of data is required for all the companies, so the database scheme can be the same for all the tenants. Not all the entities are tenant aware, only the product related entities. The required configuration is made by annotating the corresponding entities. The tenant id is set dynamically, service methods are intercepted and in the interceptor the user principal is retrieved from the session context.

E. Data import from Hamor systems

In order to provide statistics and reports with the OptInv system, data import from accounting software is necessary. This step is essential, because the data in accounting databases is in other format.

In the development process of the prototype, the Galfi Servco LLC provided a database with real data. They use HamorSoft accounting solutions. HamorSoft uses dbf files for data storage. In order to import data from this kind of files a JDBC-ODBC bridge is used.

Some data is used multiple times during the data importing process. For efficient processing a caching mechanism is implemented using hashmaps. The key in the hashmap is the data from the dbf file, and the value is the object saved in the internal database.

The main product related information is stored in a single dbf, also containing some data related to providers. Based on these data a Product object and partial ProductInfo and Provider objects can be created. The relation between the product and provider is represented by an AcquisitionDetails object. Furthermore, in this object will be saved the delivery price and packaging type, that are unknown at this part of the import. Other provider specific information is stored in a separate dbf, ProductInfo objects can be completed using this file.

The StockInfo object holds stock information related to the product such as quantity on stock, replenishment price etc. This object will be created in this step of the import, but only the Product attribute will be initialized now. The Product object contains an AcquisitionDetails and StockInfo data member, which will be initialized in this step.

A part of the data related to sales is kept in another dbf. Using this information an Invoice object can be created and saved into the internal database. By using a separate dbf Sales objects corresponding to these invoices can also be created. These objects are also added to the sales list referenced by Product objects.

The next step is to import the quantity on stock, but for this more dbf files need to be processed. A dbf stores the commercial transactions, and a column is indicating the state and validity of these transactions. Another dbf stores incoming and outgoing transports. Using these two files a StockInfo object can be initialized and the Product objects can be refreshed in the database and memory.

Some company-specific post processing could be also needed after the import process. These post processing operations can be configured and executed separately. For example, in the Galfi Servco database the same product appeared twice if two providers delivered it. In order to provide correct and unified statistics and reports, these products had to be merged. During post processing some invalid invoices had to be deleted, too.

F. Visualizing data

The system creates statements based on different calculations. Results are shown in table- or chart-based views.

On the user interface there is a table view, where the product information is shown in different columns, like: product id, product name, average consumption based on daily average sales in a selected period, replacement cost, current stock value and quantity on stock. There is a column indicating the availability of the product based on the average consumption. These columns can be sorted. For example, sorting the average consumption column in descendent order, frequently sold products can be easily identified. Sorting by the quantity on stock column, the user can easily identify products, which are unnecessary kept.

On a different view statistical reports are presented. For example, monthly sales are visualized using a table and a chart-based view. The chart-based view is created using the dCharts Vaadin add-on.

G. *User interface*

The web user interface is created using the Vaadin web development framework. It is based on the ServerUI class implementing the UI interface. The views use the VaadinTabSheet component. Based on users' role different views are shown for users with different rights. For creating forms BeanFieldGroup components are used, bounded together with the BeanValidation technology.

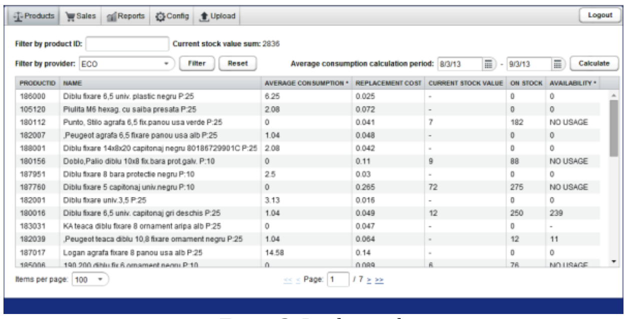
Logged in as an administrator, the admin view is shown, where companies and users can be managed. Logged in as a user multiple views are shown: products, sales, configuration and file upload views.

There are views with tables displaying data about a large number of entities. A paging mechanism is implemented in the system. A lazy loading method is used, data queries are created dynamically to achieve a better performance. All views provide filtering and sorting possibilities.

V. Using the OptInv System

After the login, the administrator can manage company profiles and users belonging to companies.

If a user belongs to a company, after login he can see a view, which displays a table with information about products: product id, name, daily average consumption calculated in a selected period, the minimal price, current stock value, quantity on stock and the availability, based on product quantity and average consumption (Figure 3.). Products can be filtered, ordering support is provided for each table column and a paging mechanism also can be used.



Filter by product ID: Current stock value sum: 2836

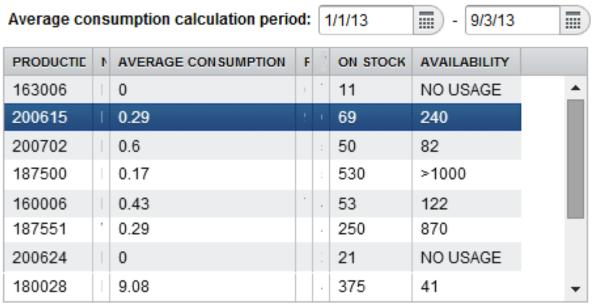
Filter by provider: ECO Filter Reset Average consumption calculation period: 8/3/13 - 9/3/13 Calculate

PRODUCTID	NAME	AVERAGE CONSUMPTION *	REPLACEMENT COST	CURRENT STOCK VALUE	ON STOCK	AVAILABILITY *
186000	Diblu fixare 6,5 univ. plastic negru P:25	6.25	0.025	-	0	0
105120	Piulita M6 hexag. cu saiba presata P:25	2.08	0.072	-	0	0
180112	Punto, Stilo agrafa 6,5 fix.panou usa verde P:25	0	0.041	7	182	NO USAGE
182007	,Peugeot agrafa 6,5 fixare panou usa alb P:25	1.04	0.048	-	0	0
188001	Diblu fixare 14x8x20 capitonaj negru 80186729901C P:25	2.08	0.042	-	0	0
180156	Doblo,Palio diblu 10x8 fix.bara prot.galv. P:10	0	0.11	9	88	NO USAGE
187951	Diblu fixare 8 bara protectie negru P:10	2.5	0.03	-	0	0
187760	Diblu fixare 5 capitonaj univ.negru P:10	0	0.265	72	275	NO USAGE
182001	Diblu fixare univ.3,5 P:25	3.13	0.016	-	0	0
180016	Diblu fixare 6,5 univ. capitonaj gri deschis P:25	1.04	0.049	12	250	239
183031	KA teaca diblu fixare 8 ornament aripa alb P:25	0	0.047	-	0	-
182039	,Peugeot teaca diblu 10,8 fixare ornament negru P:25	1.04	0.064	-	12	11
187017	Logan agrafa fixare 8 panou usa alb P:25	14.58	0.14	-	0	0
185006	190 200 diblu fix 6 ornament negru P:10	0	0.089	6	76	NO USAGE

Items per page: 100 << < Page: 1 / 7 > >>

Figure 3. Products tab

For example, Figure 4. presents a partial report about products. For the selected product (id: 200615) the daily average consumption in the given period (1/1/2013-9/3/2013) is 0.29 and 69 pieces are available in stock, so this quantity is enough for approximately 240 days. There are products with minimal consumption, or absolutely no sales, and a large amount on stock, which is unnecessary.



Average consumption calculation period: 1/1/13 - 9/3/13

PRODUCTIL	N	AVERAGE CONSUMPTION	F	ON STOCK	AVAILABILITY
163006		0		11	NO USAGE
200615		0.29		69	240
200702		0.6		50	82
187500		0.17		530	>1000
160006		0.43		53	122
187551		0.29		250	870
200624		0		21	NO USAGE
180028		9.08		375	41

Figure 4. Products information

The second tab displays information about sales: product id and name, sold quantity, price, invoice and date. Filtering and ordering is also supported.

In a separate tab different reports can be visualized. Currently two report types are supported: a table showing the daily average consumption for each month in the last year, and a graph showing this data for a selected product. This graph can be useful for isolating products with recurrent, sporadic and seasonal usage. For example, the product in Figure 5. can be sporadic, because there are months without any sales. The system can be easily extended with new report types.

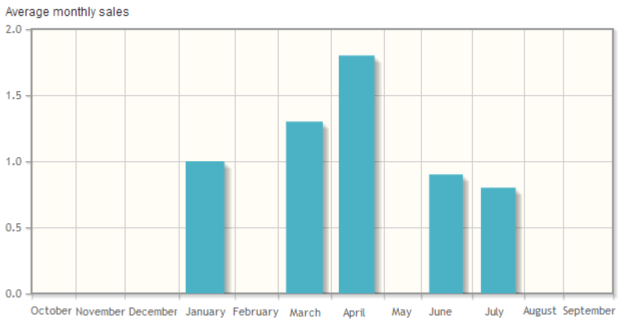


Figure 5. A sporadic product

Data import and reimport can be done using the Config tab. The Upload tab can be used to upload files required for import. According to the company's database type, it also indicates which files are required from the accounting software.

VI. Conclusions and Further Development

In its current state Optinv is a prototype, but it already provides help for companies in inventory and sales optimization. During the development the provider of the test database continuously tested the application and checked the results. Based on the feedback, the results are correct, the system can be used, and offers information, which is not provided in this form by the accounting software. Some further development possibilities:

- supporting data import from other accounting software systems;
- automated and scheduled import;
- order management sub-system;
- mobile application, automatic notifications for major inventory changes;
- additional statistics and reports;
- automatic product categorization and consumption forecasting;
- optimization of orders (e.g. grouping products).

Acknowledgment

The authors would like to thank Codespring LLC for supporting the development process and ensuring the infrastructure. Thanks to Zoltán Pálhegyi, CEO at AVE Harghita for his professional advices. Thanks for the data and support provided by the GalfiServco team. In addition the authors would like to thank HamorSoft LLC for the possibility of collaboration. Also thanks to Lóránd Tompa, project manager at Codespring LLC for his help in the first phase of development.

REFERENCES

[1] Arun Gupta, Java EE 7 Essentials, O'Reilly Media, 2013.

[2] ***, (2014) Java Enterprise Edition reference documentation [Online]. Available: http://docs.oracle.com/javaee/7/tutorial/doc

[3] Richard Monson-Haefel, Bill Burke, Enterprise JavaBeans 3.0 5th Edition, O'Reilly Media, 2006.

[4] ***, (2014) EclipseLink Wiki [Online]. Available: http://wiki.eclipse.org/EclipseLink/Development/Indigo/Multi-Tenancy.

[5] Marko Grönroos, Book of Vaadin: Vaadin 7 Edition, revision 2nd ed., Vaadin Ltd, 2014.