Network Software System Lab

**Food-Center**

**Winter 2012-2013**

**Project Book**

[**https://food-center.appspot.com**](https://food-center.appspot.com)

**Submitted by:**

Dror Baum 038748950 dror.baum@gmail

**Under supervision of:**

Genady Paikin,

Contents

[Introduction 4](#_Toc369386289)

[Development Environment 4](#_Toc369386290)

[Eclipse 4](#_Toc369386291)

[Google App-Engine - The Java Development Server 5](#_Toc369386292)

[Android Emulator 5](#_Toc369386293)

[Technologies 5](#_Toc369386294)

[Java 5](#_Toc369386295)

[Google App Engine 5](#_Toc369386296)

[Google Data Store 5](#_Toc369386297)

[Google Blobstore 5](#_Toc369386298)

[Google Image Service 6](#_Toc369386299)

[ORM - Object Rational Mapping 6](#_Toc369386300)

[Spatial Search – Javageomodel 6](#_Toc369386301)

[GWT - Google Web Toolkit 6](#_Toc369386302)

[RPC - Remote Procedure Call 6](#_Toc369386303)

[GCM - Google Cloud Messaging 7](#_Toc369386304)

[Google Channel API 7](#_Toc369386305)

[Google Users Service 7](#_Toc369386306)

[JUnit 7](#_Toc369386307)

[CSS 8](#_Toc369386308)

[Google Maps API and Google Maps Android API V2 8](#_Toc369386309)

[Simple Logging Facade for Java (SLF4J) 8](#_Toc369386310)

[Project Scope 8](#_Toc369386311)

[Purpose 8](#_Toc369386312)

[Requirements 8](#_Toc369386313)

[Administrator Requirements 8](#_Toc369386314)

[Restaurant Administrator Requirements 9](#_Toc369386315)

[Restaurant Branch Administrator Requirements 9](#_Toc369386316)

[Restaurant Branch Chef Requirements 9](#_Toc369386317)

[Restaurant Branch Waiter Requirements 9](#_Toc369386318)

[Company Administrator Requirements 10](#_Toc369386319)

[Company Branch Administrator Requirements 10](#_Toc369386320)

[System Overview 10](#_Toc369386321)

[System High level design 10](#_Toc369386322)

[Web Service 11](#_Toc369386323)

[Web Client 11](#_Toc369386324)

[Android Client 12](#_Toc369386325)

[Use Case Diagram 13](#_Toc369386326)

[Sequences Diagrams 14](#_Toc369386327)

[Common Sequences 14](#_Toc369386328)

[Employee Sequences 15](#_Toc369386329)

[Company Admin Sequences 20](#_Toc369386330)

[Company Branch Admin Sequences 21](#_Toc369386331)

[Restaurant Admin Sequences 23](#_Toc369386332)

[Restaurant Branch Admin Sequences 24](#_Toc369386333)

[Restaurant Chef sequences 25](#_Toc369386334)

[Restaurant Waiter Sequences 27](#_Toc369386335)

[RPC 28](#_Toc369386336)

[Considered RPC framework 28](#_Toc369386337)

[RequestFactory implementation in the system 30](#_Toc369386338)

[Data Base 30](#_Toc369386339)

[Datastore vs. Rational Database vs. Blobstore 30](#_Toc369386340)

[PMF and DbHandler 31](#_Toc369386341)

[JDO and RequestFactory integration 32](#_Toc369386342)

[Android Client 32](#_Toc369386343)

[Authentication 32](#_Toc369386344)

[Image Downloading 33](#_Toc369386345)

[GCM 33](#_Toc369386346)

[Special Design Patterns 33](#_Toc369386347)

[Users Guide 33](#_Toc369386348)

[Android User Guide 33](#_Toc369386349)

[Web User Guide 37](#_Toc369386350)

[Summery 40](#_Toc369386351)

[Bibliography 40](#_Toc369386352)

# Introduction

Now days most of the employers use food order services such as CIBUS or 10BIS in order to give food services for their employees. Using such a service simplified the money handling: employees don't need to keep receipts and deliver them at the end of the day/week/month in order to get their money; also restaurant and companies can have a report at the end of the month of all the orders involved and which user payment check this money needs to be taken from.

On the other hand, most of these services don't utilize the fact that most of the people today are using smartphones. The services that they provide are web-services such as http web client which at the worst scenario only shows you the available restaurants (CIBUS), or on the best scenario allow clients to make take-away/delivery orders (10BIS) which don't fit the mobiles screen properly, using a web page on the smartphone won't give the best user experience, and defiantly can't utilize all smartphone features. When employees want to make an order they must do it from their office where they have a computer.

With that in mind, the system we built enables restaurants (with multiple restaurant branches) to give services to companies' (with multiple company branches) employees. These services include not only take-away/ delivery options but also the option to make a table reservation for a group of employees without the need to make a phone call (for restaurants that permit it). The system also provide accurate reports for each restaurant on the money that each of the companies needs to pay at the end of the month, and for each company on the money that needs to be paid to the restaurants. It also provides the companies the payment information that needs to be calculated for each of its employees for tax. Employees will enjoy the benefits of using such mobile application, and the restaurants will get more orders because of great user experience.

# Development Environment

## Eclipse

Eclipse Java-EE is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system.

It is written mostly in Java and can be used to develop applications in Java and, by means of various plug-ins, other programming languages including JavaScript and CSS.

The Google Plugin for Eclipse handles most App-Engine and GWT related tasks in the IDE, including creating projects, invoking the GWT compiler, creating GWT launch configurations, validations, syntax highlighting, etc.

The Android plugin for Eclipse handles most of Android related tasks in the IDE, including creating projects and signing the application.

Released under the terms of the Eclipse Public License, Eclipse is free and open source software.

## Google App-Engine - The Java Development Server

The App Engine Java SDK includes a development web server for testing your application on your computer. The development web server simulates the App Engine Java runtime environment and all of its services, including the datastore. The [Google Plugin for Eclipse](https://developers.google.com/appengine/docs/java/tools/eclipse) can run the server in the Eclipse debugger. Development server can be run directly from eclipse or from the command line.

## Android Emulator

The Android SDK includes a mobile device emulator — a virtual mobile device that runs on your computer. The emulator lets you develop and test Android applications without using a physical device.

# Technologies

## Java

Java is a high-level programming language that was released in 1995 by Sun Microsystems. This is an object oriented language that is designed to be implementation loosely coupled. Java applications are compiled into class files (Byte code) that can run on any Java Virtual Machine (JVM) regardless of computer architecture.

In this system JDK 1.7 is used for the server, and JDK 1.6 is used for Android.

## Google App Engine

Google App Engine lets you run web applications on Google's infrastructure. App Engine applications are easy to build, easy to maintain, and easy to scale as your traffic and data storage needs grow. With App Engine, there are no servers to maintain: You just upload your application, and it's ready to serve your users.

## Google Data Store

The Datastore is one of several App Engine services offering a choice of standards-based or low-level APIs. App Engine Datastore is a schemaless object datastore providing robust, scalable storage for your web application, with the following features: No planned downtime, Atomic transactions, High availability of reads and writes, Strong consistency for reads and ancestor queries, Eventual consistency for all other queries.

## Google Blobstore

The Blobstore serve data objects, called *blobs,* that are much larger than the size allowed for objects in the Datastore service. Blobs are useful for serving large files, such as video or image files, and for allowing users to upload large data files. Blobs are created by uploading a file through an HTTP request. Blobstore creates a blob from the file's contents and returns an opaque reference to the blob, called a *blob key,* which can be later use to serve the blob. The application can serve the complete blob value in response to a user request, or it can read the value directly using a streaming file-like interface.

## Google Image Service

The Images service can resize, rotate, flip, and crop images; it can composite multiple images into a single image; and it can convert image data between several formats. It can also enhance photographs using a predefined algorithm. The API can also provide information about an image, such as its format, width, height, and a histogram of color values.

## ORM - Object Rational Mapping

ORM is a programming technique for converting data between incompatible type systems in object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language.

**Java Data Objects** (JDO) is an ORM implantation and specification of Java object persistence. One of its features is a transparency of the persistent services to the domain model. JDO persistent objects are ordinary Java programming language classes (POJOs); there's no requirement for them to implement certain interfaces or extend from special classes.

Object persistence is defined in the external XML metafiles or using Annotations, which may have vendor-specific extensions.

JDO enhanced classes are portable across different vendors' implementation. Once enhanced, a Java class can be used with any vendor's JDO product.

In our system we use DataNucleus 3.0 which is supplied with the GAE SDK.

## Spatial Search – Javageomodel

Javageomodel is a library which provides generalized solution for performing basic indexing and querying of geospatial data in Google App Engine.

## GWT - Google Web Toolkit

GWT is a development toolkit for building and optimizing complex browser-based applications. GWT is used by many products at Google, including [Google AdWords](http://google.com/adwords) and [Google Wallet](http://wallet.google.com). It's open source, completely free, and used by thousands of developers around the world.

GWT emphasizes reusable, efficientapproaches to common web development tasks, namely asynchronous remote procedure calls, history management, bookmarking, UI abstraction, internationalization, and cross-browser portability.

Using GWT, developers can develop and debug Ajax applications in the Java language using the Java development tools of their choice. When the application is deployed, the GWT cross-compiler translates the Java application to standalone JavaScript files that are optionally obfuscated and deeply optimized. When needed, JavaScript can also be embedded directly into Java code, using JNSI.

## RPC - Remote Procedure Call

Remote procedure call (RPC) is an inter-process communication that allows a computer's program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction. That is, the programmer writes essentially the same code whether the subroutine is local to the executing program, or remote. When the software in question uses object-oriented principles, RPC is called remote invocation or remote method invocation.

In this project we use **RequestFactory** (RF) as RPC is an alternative to GWT-RPC for creating data-oriented services. RequestFactory and its related interfaces (RequestContext and EntityProxy) make it easy to build data-oriented (CRUD) apps with an ORM-like interface on the client. It is designed to be used with an ORM layer like JDO or JPA on the server, although this is not required. It allows us to use a single RPC implementation to connect between the web-service the android client and the web-client.

## GCM - Google Cloud Messaging

Google Cloud Messaging for Android (GCM) is a service that allows you to send data from your server to your users' Android-powered device (Server-PUSH), and also to receive messages from devices on the same connection. The GCM service handles all aspects of queuing of messages and delivery to the target Android application running on the target device. GCM is completely free no matter how big your messaging needs are, and there are no quotas.

## Google Channel API

The Channel API creates a persistent connection between your application and Google servers, allowing your application to send messages to JavaScript clients in real time without the use of polling. This is useful for applications designed to update users about new information immediately. Some example use-cases include collaborative applications, multi-player games, or chat rooms. In general, using the Channel API is a better choice than polling in situations where updates can't be predicted or scripted, such as when relaying information between human users or from events not generated systematically.

## Google Users Service

The Users Service provide authentication of users using any one of three methods: Google Accounts, accounts on your own Google Apps domains, or OpenID identifiers (experimental). An application can detect whether the current user has signed in, and can redirect the user to the appropriate sign-in page to sign in or, if your app uses Google Accounts authentication, create a new account. While a user is signed in to the application, the app can access the user's email address (or OpenID identifier if your app is using OpenID). The app can also detect whether the current user is an administrator, making it easy to implement admin-only areas of the app.

## JUnit

JUnit is a simple framework to write repeatable tests for Java. It is important in the development of test-driven development, and is one of a family of unit testing frameworks which is collectively known as xUnit that originated with SUnit.

In our project JUnit is used to validate Database calls and RPC services (GUI can't really be tested in automated tests).

App Engine provides testing utilities that use local implementations of datastore and other App Engine services

## CSS

Cascading Style Sheets (CSS) is a style sheet language used to describe the presentation semantics (the look and formatting) of a document written in a markup language. CSS is designed primarily to enable the separation of document content (written in HTML or a similar markup language) from document presentation, including elements such as the layout, colors, and fonts.

## Google Maps API and Google Maps Android API V2

The Google Maps Image APIs make it easy to embed a static Google Maps image or Street View panorama into a web page, with no need for JavaScript.

The Google Maps Android API v2 allows to offer interactive, feature-rich maps to users of your Android applications.

## Simple Logging Facade for Java (SLF4J)

The Simple Logging Facade for Java (SLF4J) serves as a simple facade or abstraction for various logging frameworks (e.g. java.util.logging, logback, log4j) allowing the end user to plug in the desired logging framework at *deployment* time.

Because of the way GAE parse the logs, Java Util Logging (JUL) framework is used in the server code.

# Project Scope

In this project we designed and developed a client-server application. The system implements web-service, web-client and android-client. The system provides services for restaurants, companies and companies' employees.

## Purpose

This project is designed to feasibility study the GAE framework, its available APIs, Android development and Android design patterns. It also designed to give service with minimal costs to companies (and their employees) and restaurants which applied.

## Requirements

Our system consists of a few actors: the companies' employees (make the food orders), companies' administrators, companies' branches administrators, restaurant administrators, and restaurant branch administrators. System administrators are configured out of the sandbox – using the GAE dashboard.

### Administrator Requirements

Administrator is defined outside the application sandbox, using the GAE dashboard.

1. Admin can add or delete a restaurant network
2. Admin can search for a restaurant network and do all the operations of the network admin on each of the networks
3. Admin can add or delete a company.
4. Admin can search for a company and do all the operations of the company admin on each of the companies.

### Restaurant Administrator Requirements

Restaurant Administrator is defined in the datastore.

1. Restaurant Admin can manage the restaurant menu.
2. Restaurant Admin can add default courses to the menu.
3. Restaurant Admin can delete default courses from the menu.
4. Restaurant Admin can set the default prices to courses.
5. Restaurant Admin can add a restaurant network branch.
6. Restaurant Admin can search for a restaurant branch and do all the operations of a Restaurant Branch Admin on each of the branches.
7. Restaurant Admin can set the restaurant name.

### Restaurant Branch Administrator Requirements

Restaurant Branch Administrator is defined in the datastore.

1. Restaurant Branch Administrator can manage branch options.
   1. Set the service types of this branch – deliver, take away and table reservation.
   2. Set the location of the branch
   3. Set the phone of the branch
2. Restaurant Branch Administrator can manage branch menu.
   1. Add branch courses to the menu.
   2. Add prices to courses for this branch.
   3. Delete courses from branch menu.
   4. Restore branch menu to default network menu.
3. Restaurant Branch Administrator can manage branch employees.
   1. Add a new waiter.
   2. Remove a waiter.
   3. Add new chef.
   4. Remove a chef.
4. Restaurant Branch Administrator can get the list of orders for a specified period.
5. Restaurant Branch Administrator can do everything that a chef and a waiter can do.

### Restaurant Branch Chef Requirements

Restaurant Branch Chef is defined in the datastore.

1. Chef can see the pending orders screen.
2. Chef can deliver or cancel a pending order.

### Restaurant Branch Waiter Requirements

Restaurant Branch Waiter is defined in the datastore.

1. Waiter can see the pending table reservations screen.
2. Waiter can confirm or decline a pending reservation.

### Company Administrator Requirements

Company Administrator is defined in the datastore.

1. Company Administrator can add or delete company branches.
2. Company Administrator can set / change the company name.
3. Company Administrator can search for a company branch and do all the operations of a company-branch-admin on each of the branches.

### Company Branch Administrator Requirements

Company Branch Administrator is defined in the datastore.

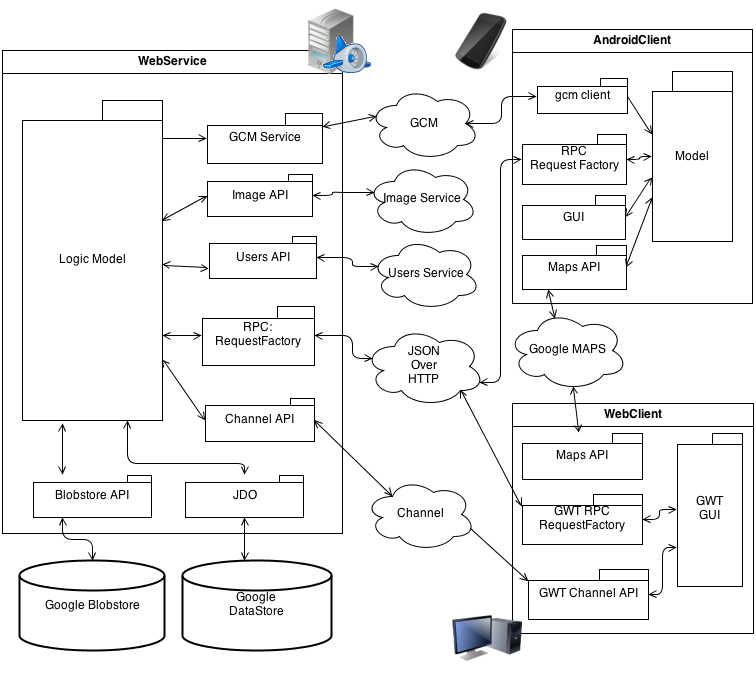
1. Company **B**ranch Administrator can set the base budget for all the company employees.
2. Company **B**ranch Administrator can set the type of applicable services which his company allows (delivery, take-away, order).
3. Company **B**ranch Administrator can manage his employees
   1. Add or remove an employ.
4. Company branch admin can get the list of all the orders made to each of the restaurants in order to check how much money he has to pay.
5. Company branch admin can get the list of all the orders made by each employ and print them.

# System Overview

## System High level design

The system is a client-server application. The components are written in Java.

The server is a servlet based server and was designed and implemented for GAE deployment. It also uses GWT in order to provide the web-client UI.



### Web Service

The web service is the core of the system: It has user authentication, it stores the data in the databases, it answers requests and push messages to the client, etc…; It provides the required services for the web-client and android-client. It is also a client for the Google services.

* Authentication is done using the Google Users Service.
* Databases used are the Google Datastore and Google Blobstore.
* DataNucleus JDO with Javageomodel are fetching and storing data in the datastore.
* Blob servlet is fetching and storing data in the blobstore.
* Channel API is used to push messages to the web client.
* GCM is used to push messages to the Android client.
* RequestFactory is the selected RPC.
* Image service is used to crop and resize images loaded to the system.

### Web Client

Web Client provides an Administration web-UI. It gives control to restaurant admins, restaurant branch admins, chefs, waiters, companies' administrators and companies' branch administrators.

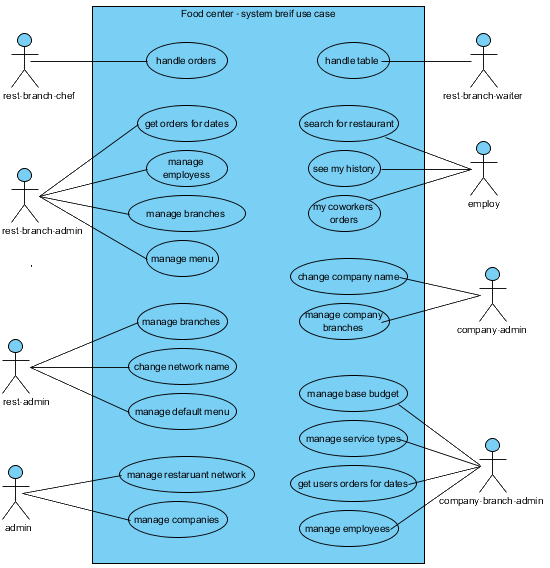
* Authentication is done using the Google Login screen.
* Client is completely written in Java (with css) and compiled to JavaScript using GWT.
* RequestFactory is the selected RPC.
* Channel API is used to get server-push messages from the web service.
* Google Maps is used to provide location of branches for proximity search.

### Android Client

Android Client is the employees' client, it provides them the ability to make food orders and table reservations, It also gives the employees the option to see their order history and receive notifications about orders which are delivered or canceled by the chef, and notifications regarding table reservations.

* Authentication is done using the Accounts on the phone (permission required).
* RequestFactory is the selected RPC.
* GCM is used to get pushed messages from the web service.
* Google Maps Android v2 is used to provide location of branches for proximity search.

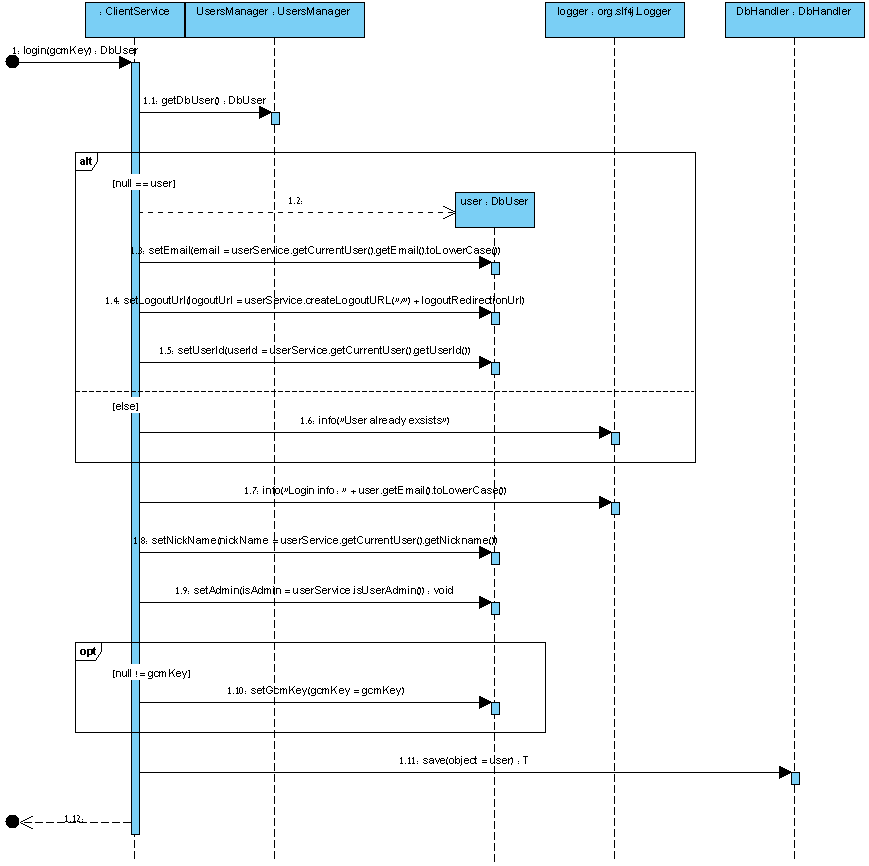
## Use Case Diagram



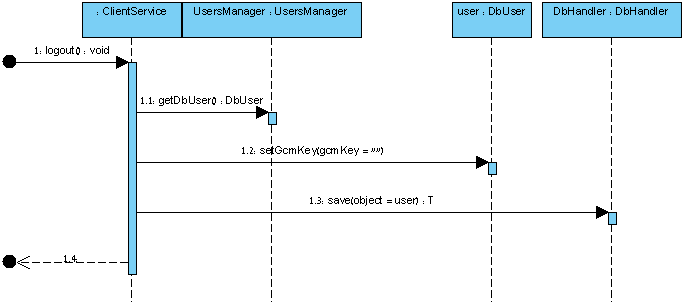
## Sequences Diagrams

### Common Sequences

#### Login sequence

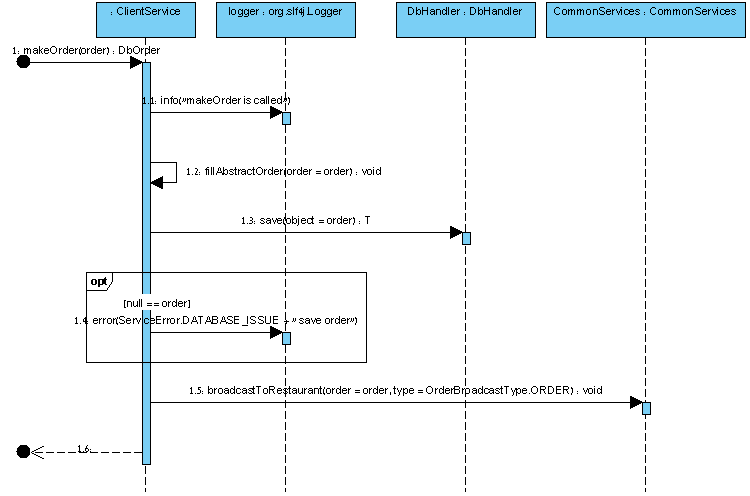


#### Logout Sequence

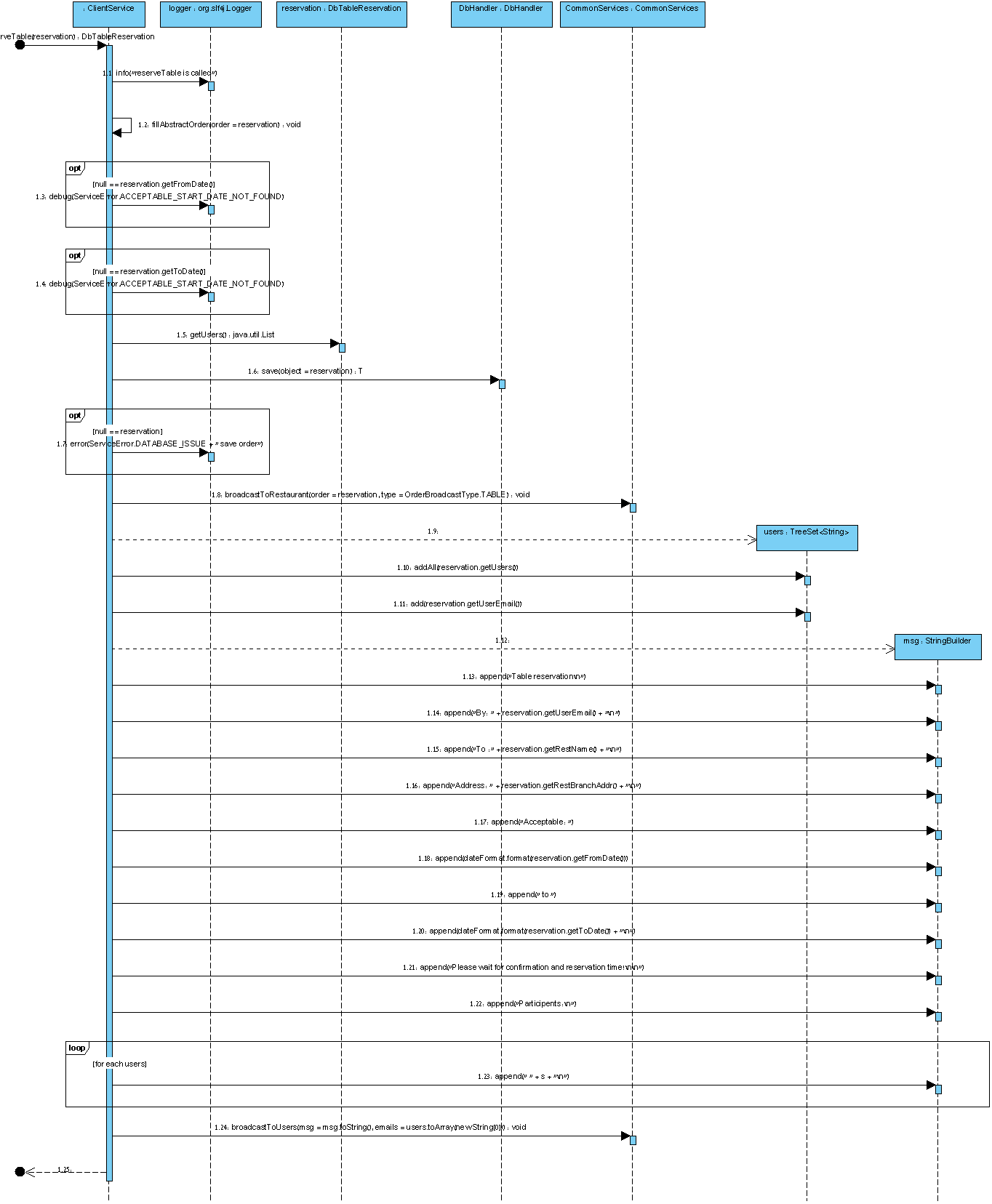


### Employee Sequences

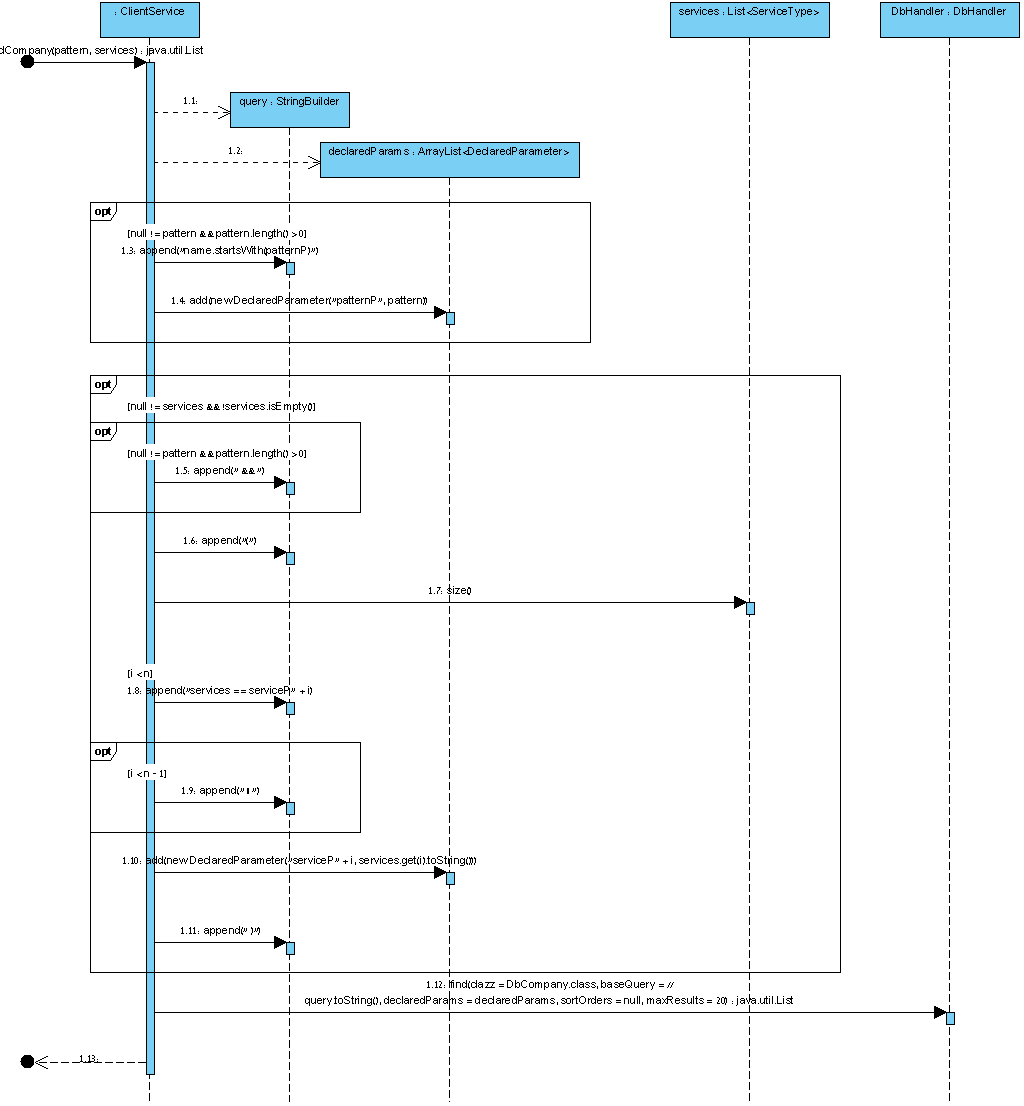
#### Make Order Sequence



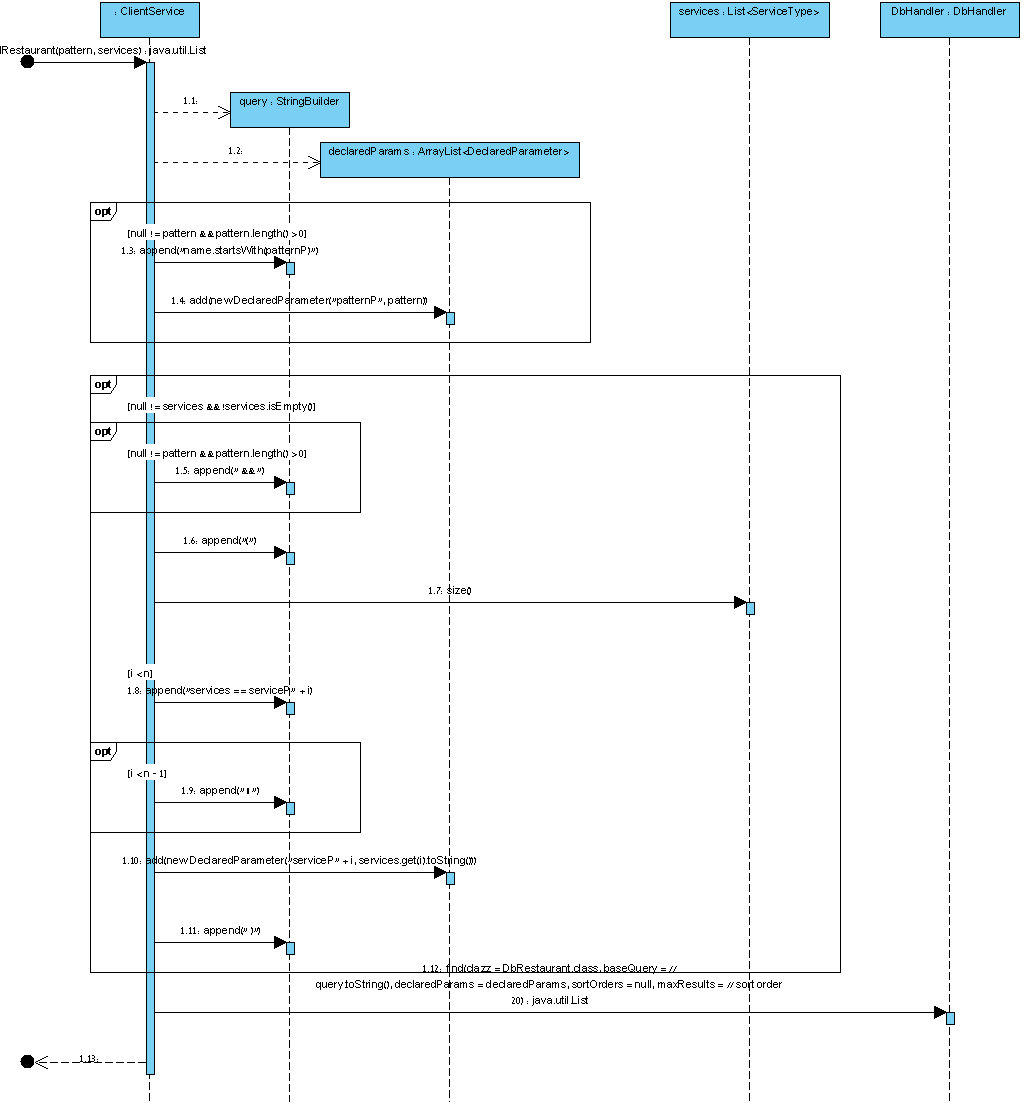
#### Reserve Table Sequence



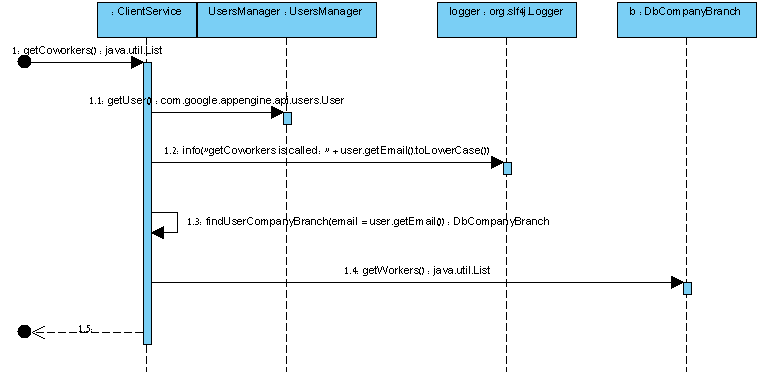
#### Find Company Sequence



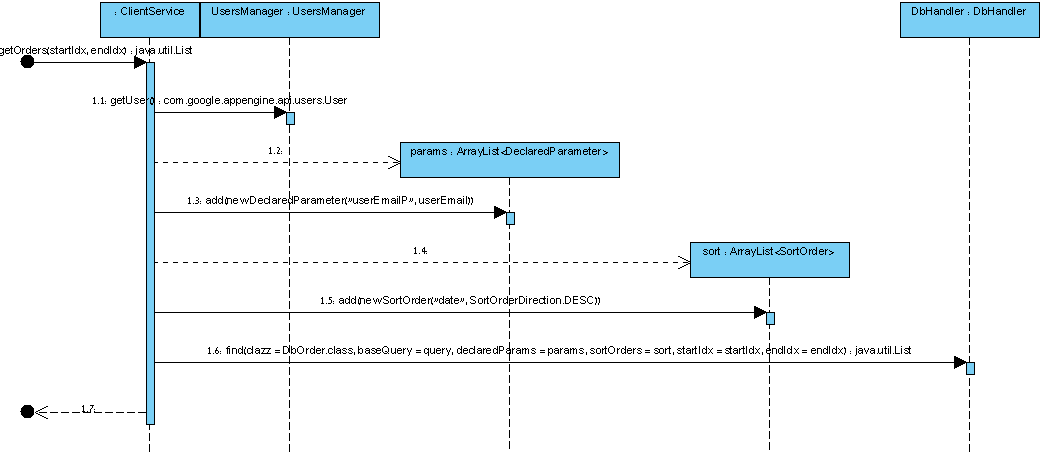
#### Find Restaurant Sequence



#### Get Coworkers Sequence

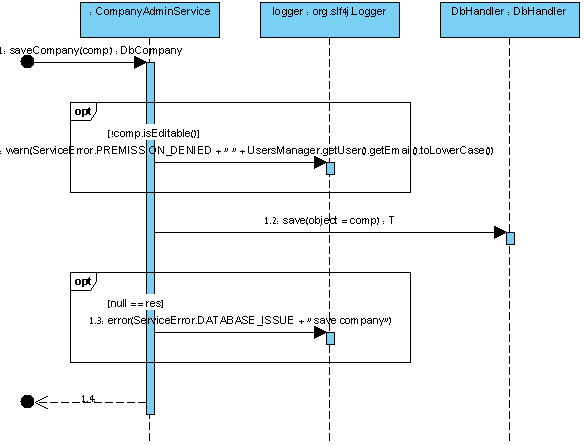


#### Get Orders History Sequence



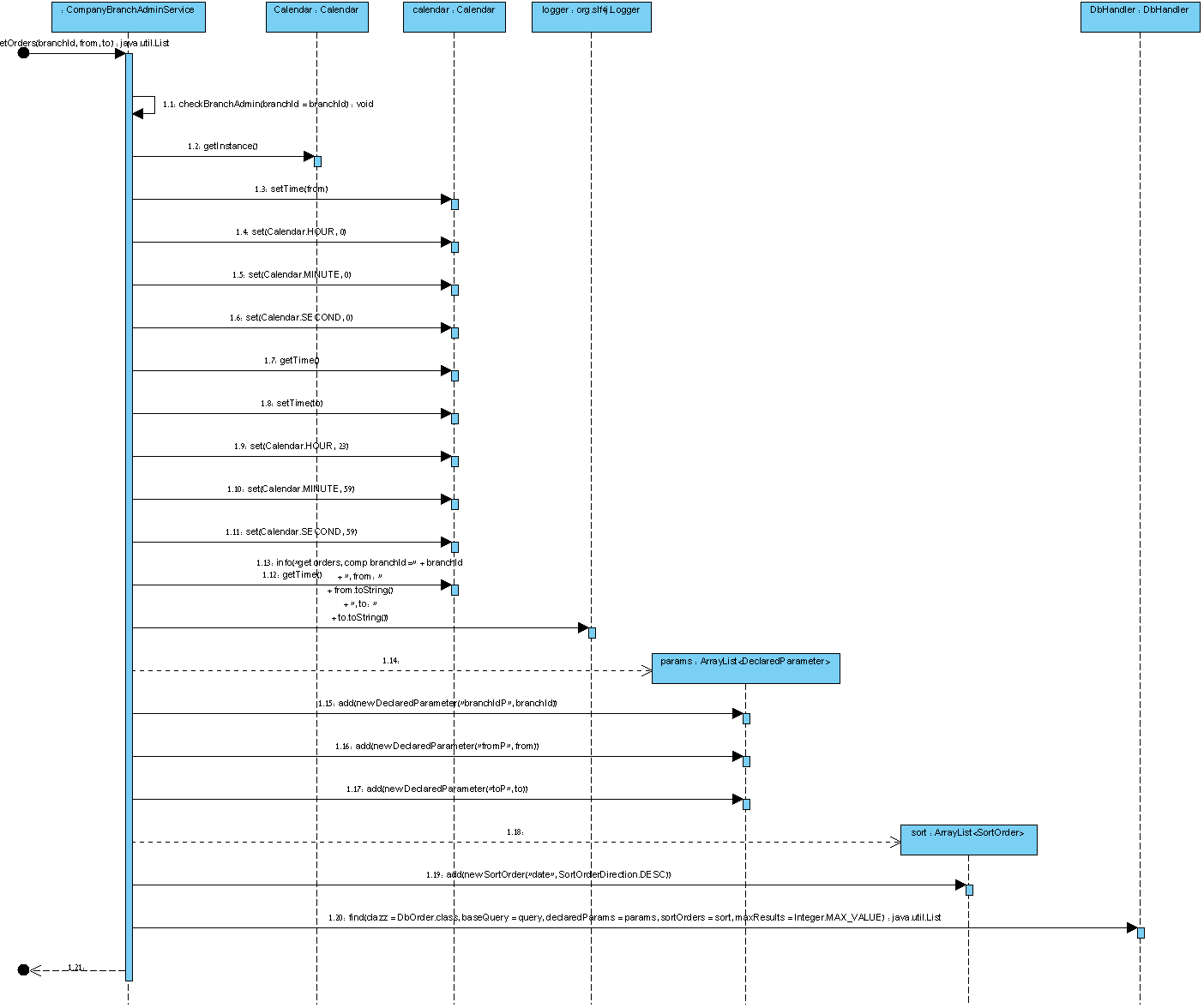
### Company Admin Sequences

#### Save Company

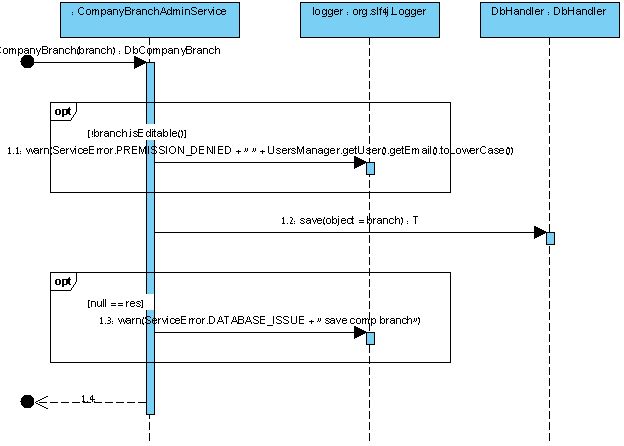


### Company Branch Admin Sequences

#### Get Orders History

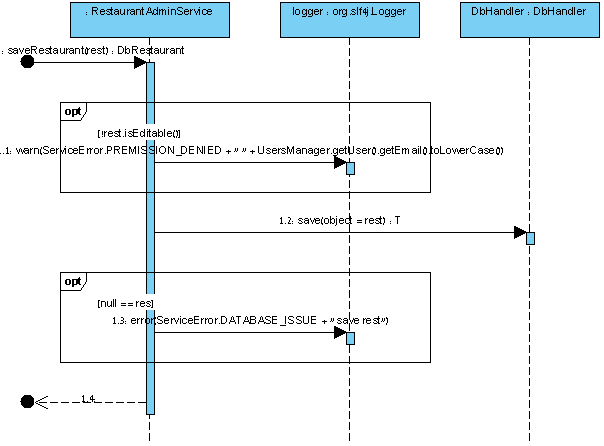


#### Save Company Branch



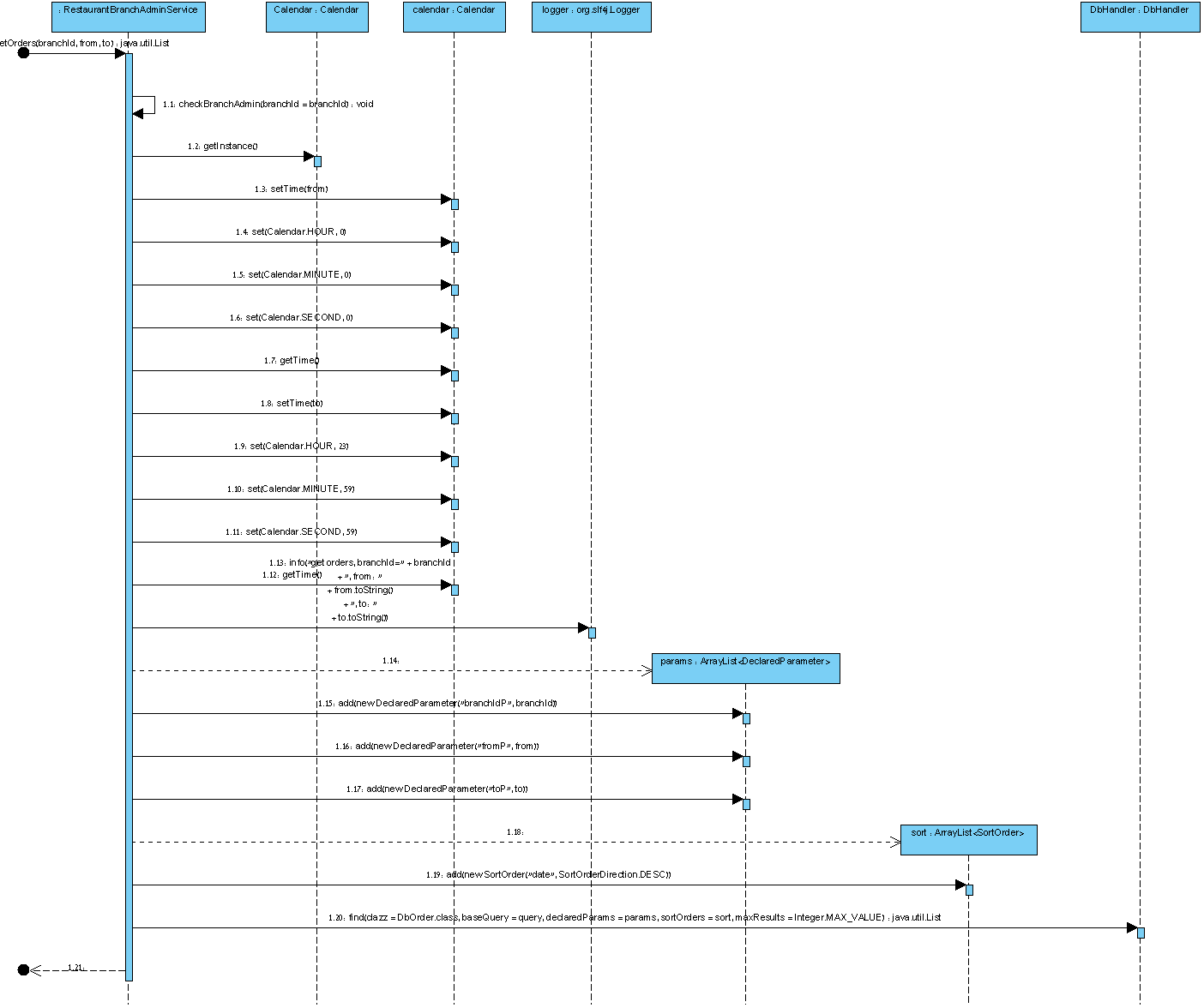
### Restaurant Admin Sequences

#### Save Restaurant Sequence



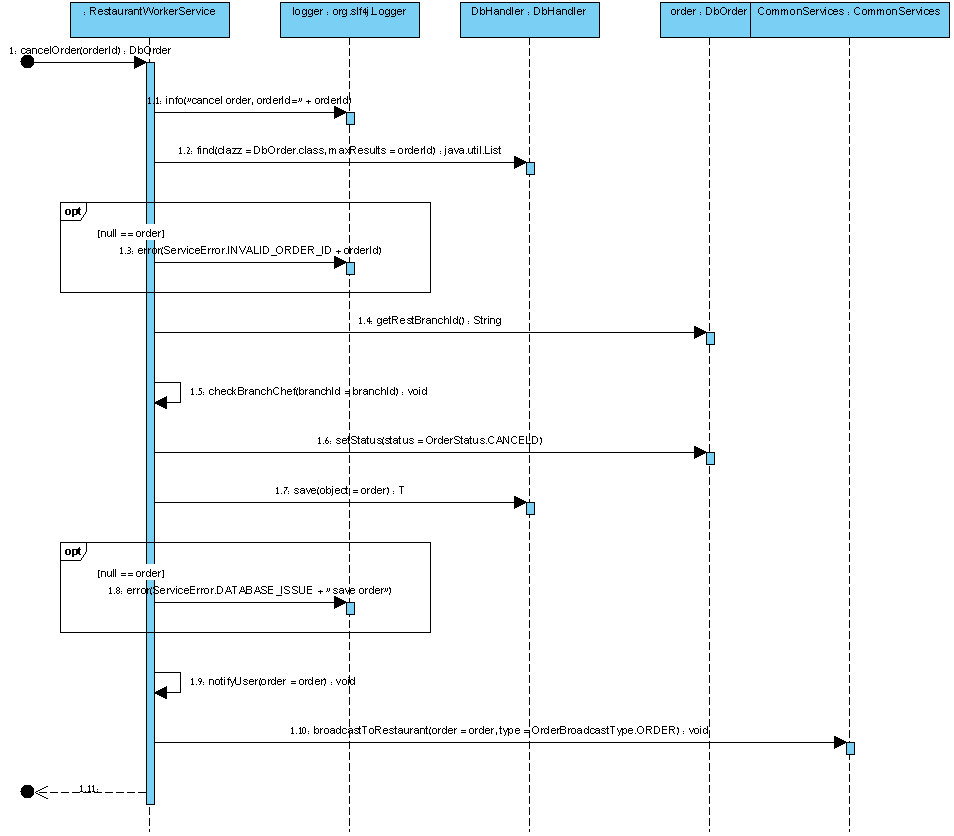
### Restaurant Branch Admin Sequences

#### Get Orders History Sequence

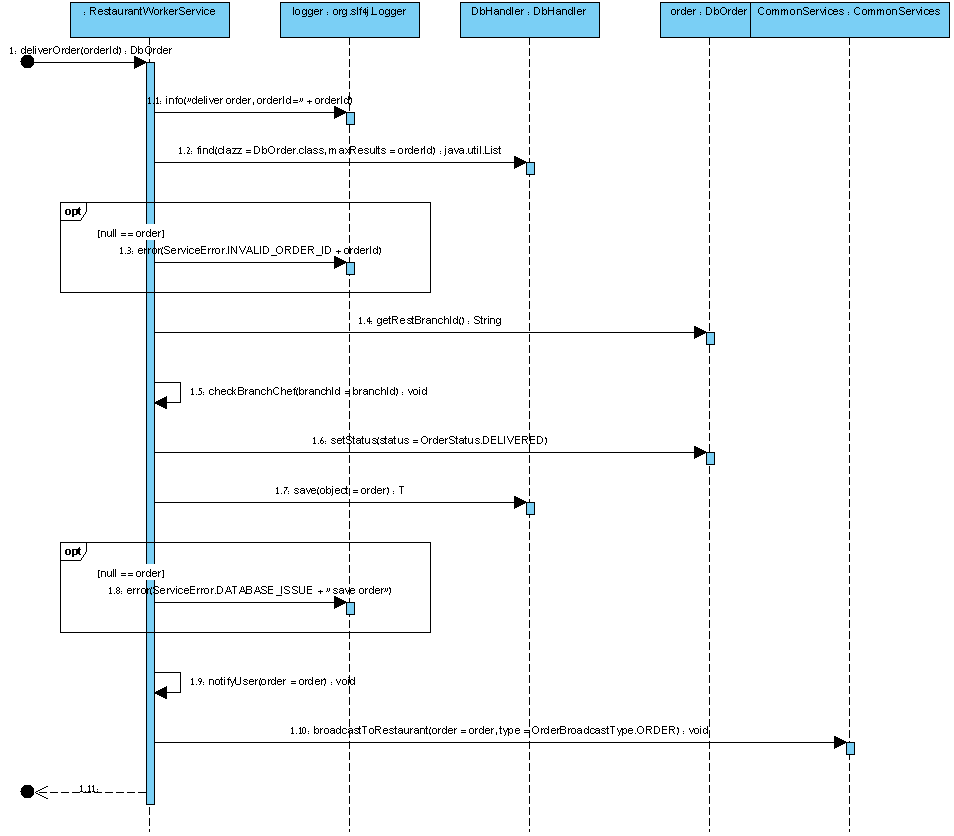


### Restaurant Chef sequences

#### Cancel Order

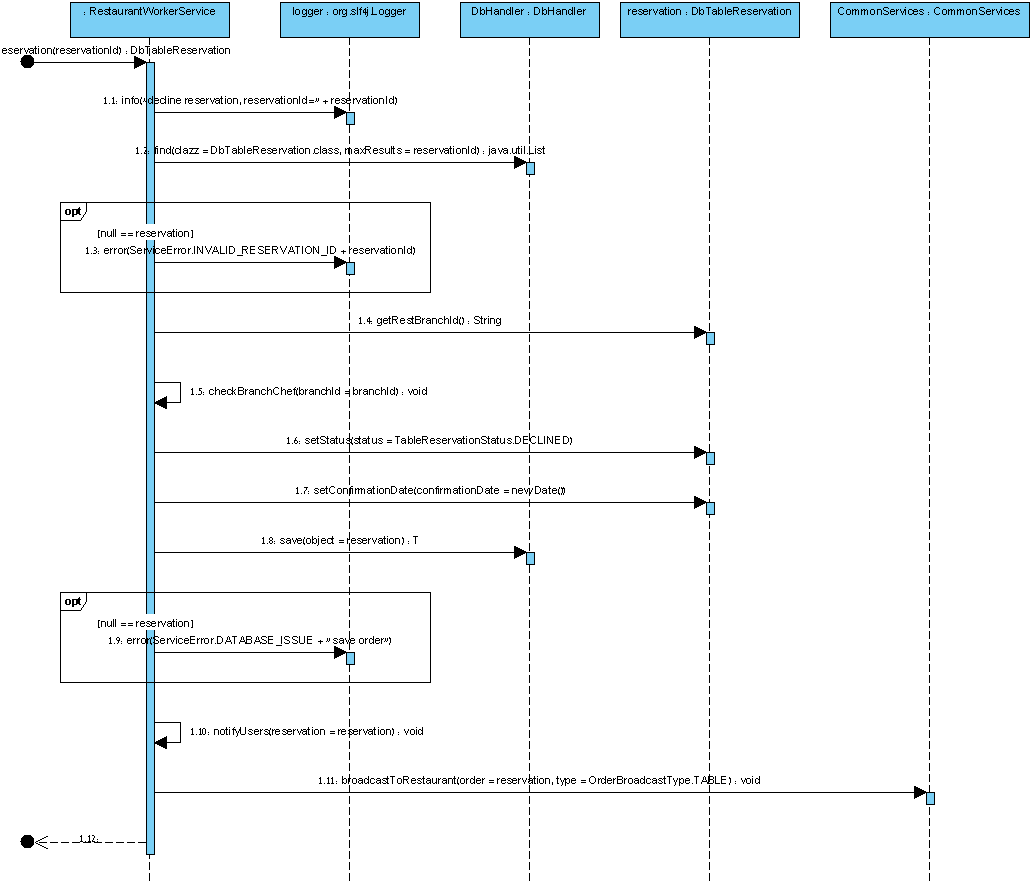


#### Deliver Order Sequece

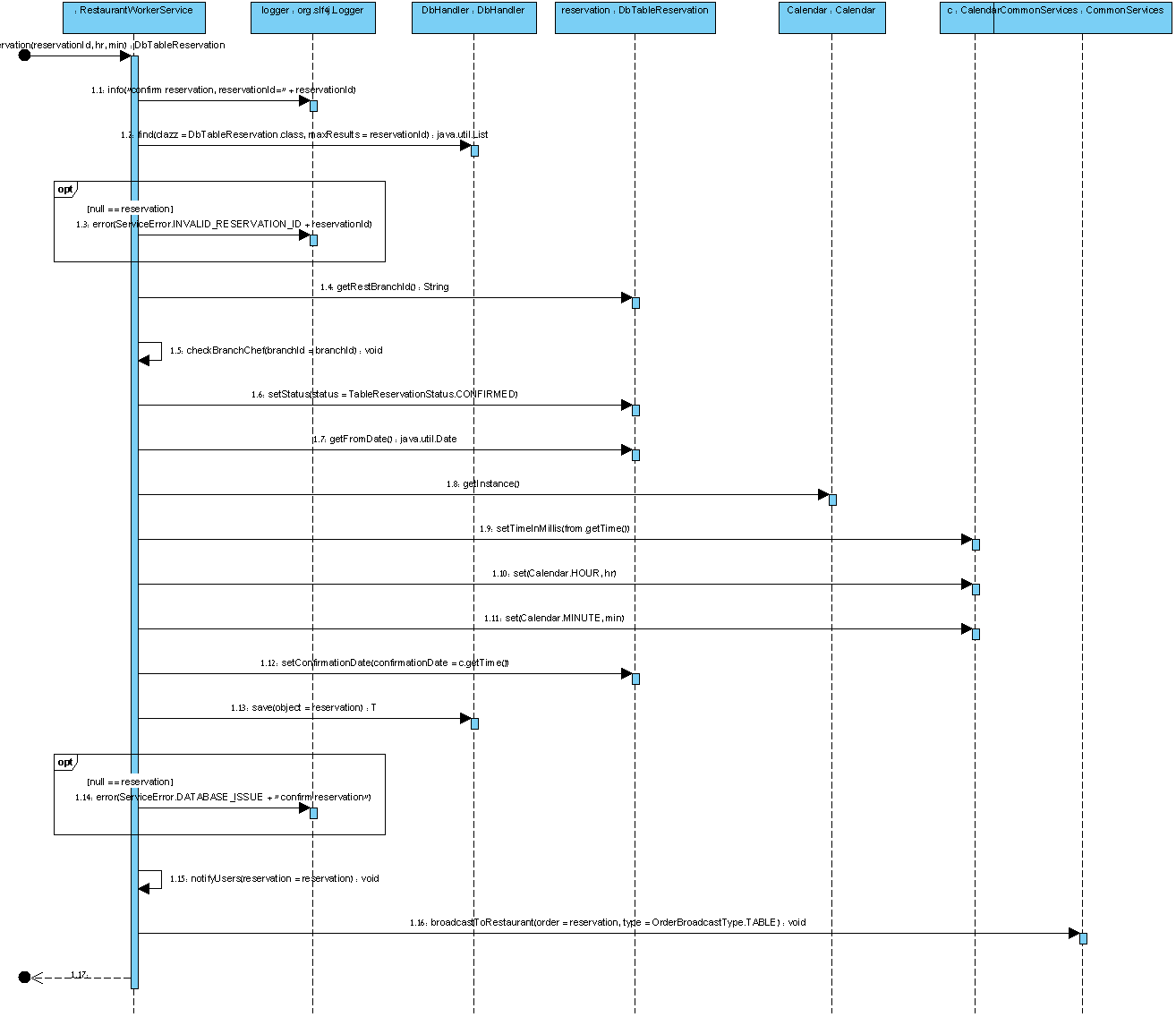


### Restaurant Waiter Sequences

#### Decline Reservation Sequence



#### Confirm Reservation



## RPC

The RPC used in the system is the RF (RequestFactory), after considering many RPC methods, it’s the most suitable RPC for our design, and we don't need to write the services twice - for Web-client and Android-client.

### Considered RPC framework

#### RequestFactory

Pros:

* Designed and implemented by Google.
* Request Factory is supported by GWT and Android; therefore a single RPC can be used in the whole system.
* It transfers JSON object (or only changes in the objects) - which is less expensive than XML objects in terms of network bandwidth.
* It is fairly documented.
* There is no need to run external tool to share the API between modules (just use shared folder).
* It supports Asynchronies calls.
* It shares Interface and not Classes

Cons:

* When full object is needed to be transferred, it’s the caller responsibility to call the "with" API.
* Since Google Endpoints is available, Google recommends using Endpoints for mobile devices (which is not GWT supported) because RF is not supported by other devices than Android
* It does not support the RESTFUL approach.
* Requires to Annotate Proxy and Request interfaces
* A "Loader" class needs to be implemented or alternatively ORM Objects needs to implement RF- non-ORM functions.

#### Google Endpoints

Pros:

* Transfer JSON objects
* Recommended by Google
* Supports RESTFUL approach

Cons:

* Was for beta testers only when we started the project.
* Not supported by GWT - a second RPC is required for web-client.
* There is a need to run external tool to create the API.
* When adding APIs in eclipse, old classes are overridden.

#### JAX-RS

Pros:

* Has many implementation, most known one is JBOSS Rest-Easy.
* Well documented, with plenty examples on the net.
* Supports the RESTFUL approach
* JSON objects are transferred.
* Has JavaScript/ Java client.
* No need to run external tool to generate the API classes.
* Share interfaces and not Classes.
* Configuration using annotations, no need to configure servlets - it is done automatically.

Cons:

* Pure support for Android! - The JBOSS Rest Easy library was cloned and integrated into Android with a huge stack overflow bug because of recursion on failures.

#### JAX-WS (or AXIS2) - WSDL

Pros:

* Probably the best documented, stable and used API framework.
* Simple annotations / XML configuration is used.
* WSDL show our interface to the world, any developer who wishes to use our services - can generate API interface from the WSDL.

Cons:

* XML is transferred
* Axis2 needs to run in order to generate API classes from the WSDL file after any change to the API functions.

### RequestFactory implementation in the system

#### Proxies

Proxy interfaces were declared for each of the objects we wanted to share. This Interface has only the methods we want to share, and use the *"@ProxyForName"* annotation in order to associate between the interface and the class name as a string (this allow us to use the same classes in the Android client which doesn't know of the DB ORM classes).

#### Requests

These are interfaces for our requests which are not part of the object itself. Each *Service* class got its own *Request* interface; those *Request*s are shared with the client the same way *Proxies* are shared.

When the service Handles ORM objects, the "Magic" of RF transfers them to *Proxies* which are served by the *Request*.

## Data Base

### Datastore vs. Rational Database vs. Blobstore

Unlike traditional relational databases, the Datastore uses a distributed architecture to automatically manage scaling to very large data sets. While the Datastore interface has many of the same features as traditional databases, it differs from them in the way it describes relationships between data objects. Entities of the same kind can have different properties, and different entities can have properties with the same name but different value types.

These unique characteristics imply a different way of designing and managing data to take advantage of the ability to scale automatically. In particular, the Datastore differs from a traditional relational database in the following important ways:

* The Datastore is designed to scale, allowing applications to maintain high performance as they receive more traffic:
  + Datastore writes scale by automatically distributing data as necessary.
  + Datastore reads scale because the only queries supported are those whose performance scales with the size of the result set (as opposed to the data set). This means that a query whose result set contains 100 entities performs the same whether it searches over a hundred entities or a million. This property is the key reason some types of queries are not supported.
* Because all queries are served by pre-built indexes, the types of queries that can be executed are more restrictive than those allowed on a relational database with SQL. In particular, the following are not supported:
  + Join operations
  + Inequality filtering on multiple properties
  + Filtering of data based on results of a subquery
* Unlike traditional relational databases, the Datastore doesn't require entities of the same kind to have a consistent property set (although you can choose to enforce such a requirement in your own application code).

With that in mind - Google gives free quota for the datastore, and Google Cloud SQL costs from the 1st read/write.

The Blobstore is used to serve data objects, called blobs, that are much larger than the size allowed for objects in the Datastore service.

### PMF and DbHandler

In JDO, you interact with the Datastore via a *PersistenceManager* instance, both to persist and update objects, and to fetch and query over them. The *PersistenceManager* instance is provided by a *PersistenceManagerFactory* object. Since the *PersistenceManagerFactory* is time-consuming to create, typically the Singleton pattern is used to ensure that only one is created per application. In our application, the *PMF* class serves this purpose.

*PMF* class also used to implement the "Open Session In View" which is describe later in this document.

Since as much as the DataNucleus is high level, it is still acting as low level API when requires to use GQL queries which must hold the class field names. In order to simplify things:

* *DbHandler* class was created to deal with all the database transactions.
* *AbstractDbObject* class was created - each ORM class inherits this abstract class. This allows the *DbHandler* to define simple methods which gets the abstract class.
* A simplified wrapper to the Query API was implemented and instead of using 2 fields for parameters and values, the *DeclaredParam* class was created, using an array of those, the *DbHandler* parse and convert the input into the 2 arrays which are used in the Query API.
* Loading objects is done lazily, with that in mind, the lazy loading is done in a different transaction, this is done because the datastore limit us for the number of objects type in a single transaction (cross group transactions)
* Storing the data is done using a single transaction! This is because of the fact that each servlet is called from a different thread - and we want to prevent thread race when storing the data, if it is not full success - than it will not be committed.

### JDO and RequestFactory integration

#### DbObjectLocator

By default RequestFactory was designed to transfer objects and each object should have implemented its own service API inside the ORM class, but in our server, we want to separate between the objects and the services. This is done by adding the *"locator*" to the annotation described before. The Locator class must implements the methods required by the RF such as *getId()* and *getVersion()*.

#### Open session in view

Open session in view is Hibernate (JPA implementation) term speaking for solving the following problem:

In a typical web-application after the main logic of the action has been completed, and therefore, the Hibernate Session has already been closed and the database transaction has ended. If you access detached objects that have been loaded in the Session inside your JSP (or any other view rendering mechanism), you might hit an unloaded collection or a proxy that isn't initialized.

The fact is that we never face this issue because the JDO is configured to create transactions when needed, but we face another issue which its solution is the same:

After RF uses the *PersistenceManager* and the *Loader* to load the object it uses the same algorithm to fetch children of this object (If the "*with*" API is called) and this is fine, but updating an *Object* which was loaded with different *PersistenceManager* is impossible.

The "Open session in view" design was implemented here as "Single PM for Thread": This is done by using a *ServletFilter* and *ThreadLocal<PersistenceManager>* in *PMF* for each thread, and this is because in a servlet based server, each *RF* "fire" call is processed in a different servlet thread (RF call can actually hold multiple calls when "fire" is called - and they will all run on the same servlet thread).

## Android Client

### Authentication

Authentication is done using the Accounts in the device. Authentication token is generated from the Accounts manager, than using these token a cookie is requested from the GAE auth service, once the cookie is received, it is stored and added to all of the server requests.

Using Android*RequestTransport* which implements *RequestTransport* we add this cookie to the RF calls.

Using *AuthCookieImageDownloader* the cookie is added to the ImageDownloader.

### Image Downloading

The Universal-Image-Loader library is used to Asynchrony load images from the server. A temporary image is shown until the image is loaded from the server.

### GCM

GCM library is used to get push messages from the server. The messages are translated to Notifications available to the user. Such messages are received upon a delivery of an order, order cancel, table reservation by user, table reservation acceptance and the decline of table reservation.

### Special Design Patterns

The following design patterns are used in the Android app:

* Android Action Bar: The action bar is a window feature that identifies the user location, and provides user actions and navigation modes. Using the action bar offers your users a familiar interface across applications that the system gracefully adapts for different screen configurations.
* Android Navigation Drawer: The navigation drawer is a panel that transitions in from the left edge of the screen and displays the app’s main navigation options.
* Action Bar Pull To Refresh: An easy way to add a modern version of the pull-to-refresh interaction to the application.

# Users Guide

## Android User Guide

|  |  |
| --- | --- |
| The main screen the user sees is the login screen.  C:\workspace\food-center\food-center-wiki\android_imgs\1_android_login (Custom).png | C:\workspace\food-center\food-center-wiki\android_imgs\2_main (Custom).pngOnce the user logged in the restaurants are loaded from the server. This page has a drawer on the left side. |

|  |  |
| --- | --- |
| Pressing one of the restaurants will open the restaurant, which will allow the user to select the branch.  C:\workspace\food-center\food-center-wiki\android_imgs\4_rest (Custom).png | After selecting the branch, the branch main page is shown. In this page the user can select courses (by tapping / sliding), reserve a table and open the drawer for branch info.  C:\workspace\food-center\food-center-wiki\android_imgs\5_branch (Custom).png |
| Openning the drawer the map of the branch is shown.  C:\workspace\food-center\food-center-wiki\android_imgs\6_branch_info (Custom).png | Selecting courses will change the Action Bar to order mode, and allow the user to make a delivery/ take away - depends on what the branch permited.  C:\workspace\food-center\food-center-wiki\android_imgs\7_menu_slide (Custom).png |

|  |  |
| --- | --- |
| Pressing the delivery or take away button chage to confirmation screen.  C:\workspace\food-center\food-center-wiki\android_imgs\8_confirm order (Custom).png | On the Branch Screen pressing the table reservation button (before selecting courses)  Open the coworkers screen, user can select coworkers and make reservation.  C:\workspace\food-center\food-center-wiki\android_imgs\10_reserve_table (Custom).png |
| After pressing the calendar button, user has to select propriate time to arrive to restaurant, restaurant admin/waiter will approve/decline with exact arrival time.  C:\workspace\food-center\food-center-wiki\android_imgs\11_1_reserve_time (Custom).png | On the main screen the user can open the drawer, and select to logout, see his order history or his co-workers.  C:\workspace\food-center\food-center-wiki\android_imgs\3_drawer (Custom).png |

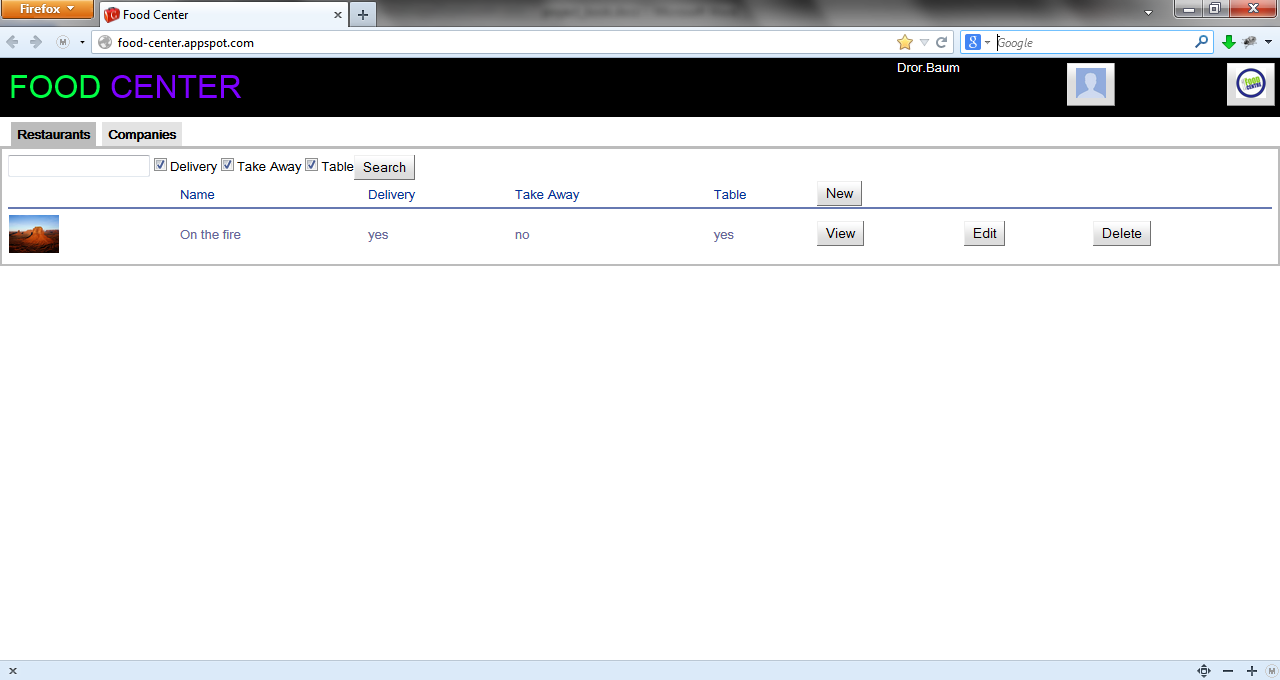
|  |  |
| --- | --- |
| Selecting the History - opens a new screen with the orders history of this client. You can slide down to get more!  C:\workspace\food-center\food-center-wiki\android_imgs\9_history (Custom).png | ­­­ |

## Web User Guide

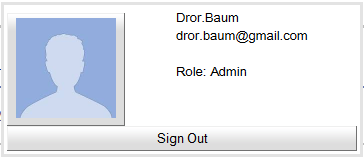
After signing in, the main screen will hold all the restaurants.

The companies tab shows the companies is nearly identical to the restaurants and will not be described in this manual.

There are 2 view options - View and Edit screens which look almost the same, this guide will only describe the edit screen, the view screen is described where there is a major difference from the edit screen.



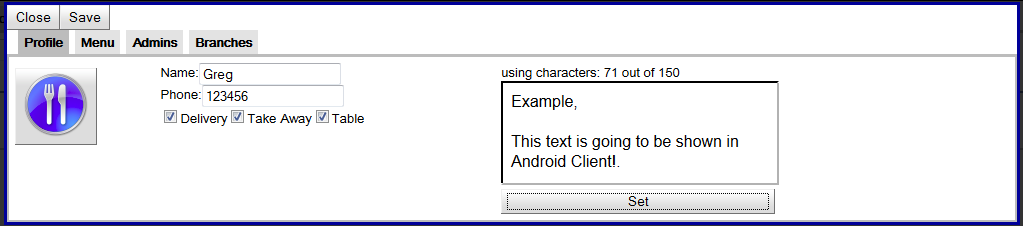
By pressing the profile image, the profile panel is opened.



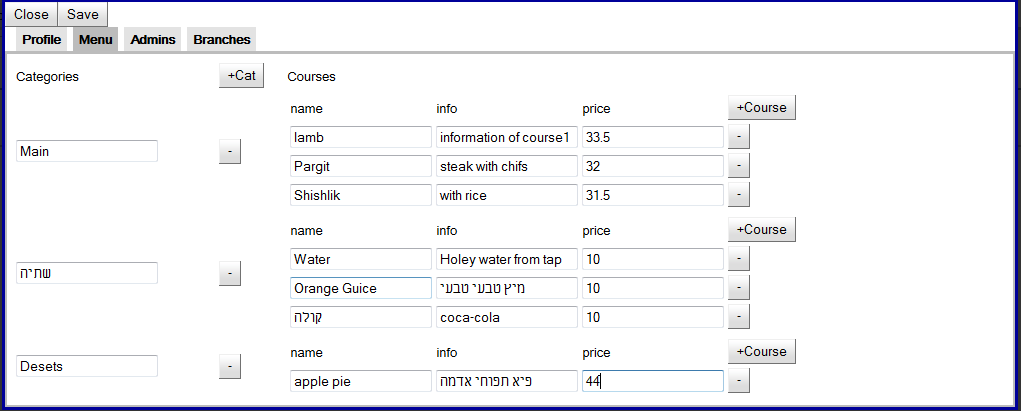
User can see his role, change his profile image or sign out.

By pressing the new button on the main screen (only Administrator can do it) a new restaurant will be created, after edit is completed pressing the save button, will save it to the database.

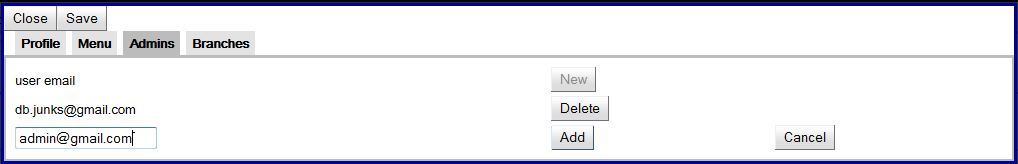
The restaurant profile tab



The restaurant Menu tab: categories, courses and their info and price can be added/ edited in the menu tab.



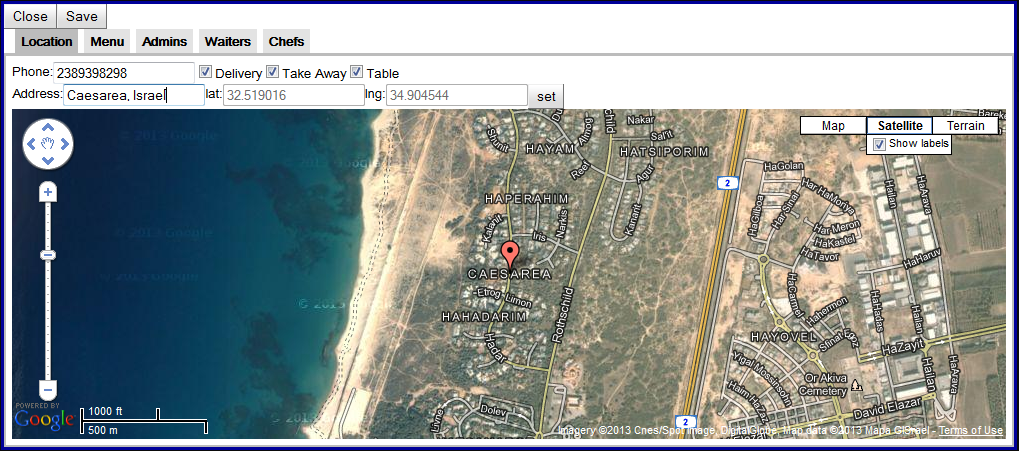
Admins can be added/ deleted in the admins tab



In the branches tab, new branches can be added and already existing branches can be deleted or edited.



After pressing the Add branch the added branch will have the location and profile panel. The location can be set using the mouse, or typing an address

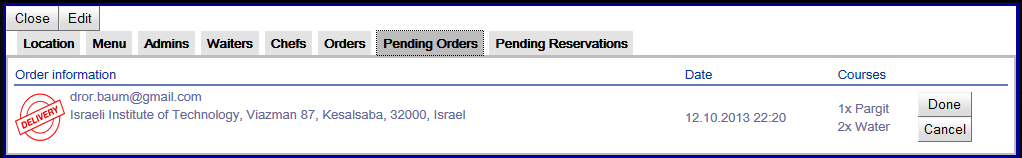


The menu tab is identical to the restaurant menu tab, and will not be shown here.

The Admins, Waiters and Chefs tab are identical to the Restaurant Admins tab and will not be shown here.

Once everything is saved the employee which was associated to one of the companies, can order food/ make table reservations.

In Restaurant Branch view mode - pending orders and pending table reservation tabs are shown to the administrator, pending orders tab is shown to chefs, and pending table reservations tab is shown to waiter.





When a new order/ table reservation is made, it will be pushed by the server and will be shown on this screen.

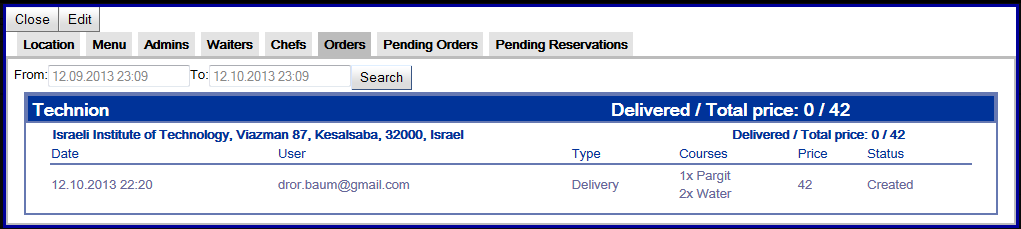
When order is done or canceled from this screen a notification is sent to the client.

When table reservation is accepted or declined from this screen a notification is sent to all the clients!

Restaurant Branch Admin can also see orders history on the Orders tab. He can select a range of dates and get the list of orders.

The orders are organized by Company name, and then by company branch addresses.

Each branch may have multiple orders.



# Summery

Looking back, it’s hard to tell how much I’ve learned. I entered this project with a lot of knowledge of Java and web programming and with some knowledge of Android development, but without any clue on Google App Engine, GWT or most of the Google services.

I learned many new APIs and frameworks such as RF, and although I was familiar with ORM, JPA and SQL, this was my 1st time using JDO, and Google datastore GQL. It was hard to design and implement the ORM classes because of the limitations of the datastore.

Possibly the biggest challenge I faced was learning new technologies and integrating them to work. Countless hours of debugging and frustration ultimately brought me a deeper understanding of the importance of JUNIT testing. I also learned that good documentation is never enough, and thanks to [Jeff Atwood](http://en.wikipedia.org/wiki/Jeff_Atwood) and [Joel Spolsky](http://en.wikipedia.org/wiki/Joel_Spolsky) for creating stackoverflow.com.

I also learned that there are many stages to building an application – that writing the code is only a small portion of the work. Designing the project and Integration of known components requires a lot of research.

# Bibliography

Food-Center: <https://food-center.appspot.com>

Google App-Engine: <https://developers.google.com/appengine/docs/whatisgoogleappengine>

GAE - Development Server: <https://developers.google.com/appengine/docs/java/tools/devserver>

Android Emulator: <http://developer.android.com/tools/help/emulator.html>

ORM: <http://en.wikipedia.org/wiki/Object-relational_mapping>

DataNucleus JDO: <http://www.datanucleus.org/index.html>

RPC: <http://en.wikipedia.org/wiki/Remote_procedure_call>

RequestFactory: <http://www.gwtproject.org/doc/latest/DevGuideRequestFactory.html>

GCM: <http://developer.android.com/google/gcm/index.html>

Google Channel API: <https://developers.google.com/appengine/docs/java/channel/>

Google Datastore: <https://developers.google.com/appengine/docs/java/datastore/>

Google Blobstore: <https://developers.google.com/appengine/docs/java/blobstore/>

Google Images Service: <https://developers.google.com/appengine/docs/java/images/>

Google Users Service: <https://developers.google.com/appengine/docs/java/users/>

Google Maps API: <https://developers.google.com/maps/>

Google Maps Android API v2: <https://developers.google.com/maps/documentation/android/>

Spatial search - Javageomodel: <http://code.google.com/p/javageomodel/>

SLF4J: <http://www.slf4j.org/>

JUNIT 4.0: <http://junit.org/>

ActionBar-PullToRefresh: <https://github.com/chrisbanes/ActionBar-PullToRefresh>

Android-Universal-Image-Loader: <https://github.com/nostra13/Android-Universal-Image-Loader>

Hibernate Open Session In View: <https://community.jboss.org/wiki/OpenSessionInView?_sscc=t#jive_content_id_The_problem>

Android Navigation Drawer: <http://developer.android.com/design/patterns/navigation-drawer.html>

Android ActionBar: <http://developer.android.com/guide/topics/ui/actionbar.html>

Action Bar Pull To Refresh: <https://github.com/chrisbanes/ActionBar-PullToRefresh>