My Diet

Software Architecture Specification (SAS)

Version 1.0

7/10/13

Primary Authors:

Ana G. Hernandez Reynoso

Alba A. Magallanes García

Julio César Roa Gil

Diego R. Torres Lafuente

Victor M. Rodriguez Bahena

Approvals:

Role	Signature	Date
Project Manager	Victor M. Rodríguez	7/10/13
QA Manager	Ana G. Hernández R.	7/10/13

Revision History

Version	Date	Author's Full Name	Reason for Changes

Abstract

Software Architecture Specification Document is one of the most important files inside of a software project, it illustrates the big picture of the project and how the code is going to be structured and linked inside. The purpose of this document is to provide information about the code structure of My Diet application at a high level of abstraction. This SAS includes an outline description of the software architecture, including major software components and their interactions, an overview of the product with a brief business justification, a description of the hardware and software platforms on which the system is build and deployed and all the external components that interact with this architecture.

Table of Contents

1. Introduction

1.1 Purpose of this document

1.2 Document Scope

1.3 Terminology

1.3.1 Acronyms

1.4 Related Documents

2 Product Overview

2.1 Identification

2.2 Goals and Objectives

2.2.1 Extensibility

2.2.2 Portability and Platform Independence

2.2.3 Reuse

2.2.4 Reliability and Availability

2.2.5 Security

2.3 Product Family Roadmap

2.4 Product Environment

2.4.1 Software

2.4.2 Hardware

2.5 Assumption, dependencies and constraints

2.5.1 Product Cost / Schedule constraints

2.5.2 Compatibility constraints

2.5.3 Visual Design and User Interaction

2.5.4 Functionality

2.5.5 Performance and Stability

2.5.6 Tool availability constraints

2.5.6.1 Full Java IDE

2.5.6.2 On-device Developer Option

2.5.6.3 Graphic UI Builders

2.5.6.4 Develop on Hardware Devices

2.5.6.5 Develop on Virtual Devices

2.5.6.6 Debugging

2.5.6.7 Testing

2.5.6.8 Native Development

2.5.7 Environmental constraints

3 Architectural Specification

3.1 Organization
3.2 Syntax
3.3 The Architectural Model
3.3.1 Architectural concepts and representations
3.4 System Structural Overview
3.4.1 Class of Architectural Model
3.4.2 Main class diagram for Android Applications
3.4.3 Pictorial of Product's Structure
3.4.3.1Architectural Big Picture – My Diet
3.4.3.2Architectural Block Diagram – My Diet
3.5 Division into Architectural Components
3.5.1 View Tier
3.5.2 Control / Model Tier
3.5.3 Class Diagram – My Diet
3.5.4 Major Architectural Components
3.5.4.1ConnectionDB
3.5.4.2DAOIngredient
3.5.4.3DAOMeal
3.5.4.4DAORecord
3.5.5 List of Product Deliverables
3.5.5.1User Interfaces
3.5.5.2Programs without interfaces
3.5.5.3Internal Libraries
3.5.5.4Product Documentation
3.6 Theory of Operation
3.7 System-level Operations
3.7.1 Exception Handling
3.7.2 Memory Management
3.7.3 Instrumentation and Debug
3.7.4 Diagnostics and Error logging
3.7.5 Internationalization and Localization
3.8 Rationale for Selected Architecture
3.9 Summary evaluations of Alternative Architectures

3.9.1 Mainframe Architecture
3.9.2 Personal computer – Standalone Database
3.9.3 File sharing Architecture
3.9.4 Client / Server Architecture
3.9.5 Distributed Database Architecture
3.9.6 Pipe-Filter Architecture

Appendix A. Definitions

1. Introduction

1.1 Purpose of this Document

This document outlines the structure and major components that make up My Diet application. It identifies all external interfaces and each internal interface between the major system components.

The target audience includes design engineering and sustaining engineers. Anyone interested in understanding the high-level software structure of My Diet should read this document. After reading this document, you will be able to understand the architectural design used and why this architecture was selected.

1.2 Document Scope

The My Diet software architecture specification presents the structure and behavior of the whole system. This SAS only covers the first level of decomposition of the product into its major components. It also covers the interaction of the major components and describes how these components are used to meet the product goals and objectives.

This SAS is intended to cover version 1.0, only. Where needed, to ensure a complete understanding of the product, the SAS may reference other documents such as Software Requirements Specification Document and the Vision and Scope Document.

1.3 Terminology

1.3.1 Acronyms

Acronym	Meaning
SAS	Software Architecture Specification
SRS	Software Requirements Specification
TS	Test Strategy Document
LAN	Local Area Network
DBMS	Database Management System

UML	Unified Modeling Language
MVVM	Model-view-viewmodel
GUI	Graphic User Interface
XML	Extensible markup language
IPC	Inter-process Procedure Communication
AIDL	Android Interface Definition Language
MVC	Model-view-controller
UI	User interface
CRUD	Create, Remove, Update, Delete
OS	Operating System
DAO	Data Access Object.

For additional terminology, see “Appendix A: Definitions”.

1.4 Related Documents

The following documents are referenced by the Software Architecture Specification:

Document Name	Revision
Vision and Scope Document	2.0
Software Requirements Specification Document	2.0
Test Strategy Document	2.0

2. Product Overview

2.1 Identification

This project will consist about creating an Android application (app) called My Diet for tracking, controlling and supervising a user's diet or nutrition data based in the daily user's records. The project will be completed by November, 2013. Modules of the app will include a simple record form for meals, a way to report daily meals, adding food form, a user profile form, and caloric calculator.

2.2 Goals and Objectives

The goal of the app is to help people control their caloric intake with a points-earned reward system and help them in their quest to reduce overweight and obesity or avoid either one of those diseases. By achieving this, then, the company could sell the software license and obtain small revenue in return. Our vision for a year after the first release is to be ranked as one of the 25 best apps for calories track and diet control on the Android Market.

2.2.1 Extensibility

An extensible application refers to an application able to extend easily without modifying its original code base. You can enhance its functionality with new plug-ins or modules. Developers, software vendors, and even customers can add new functionality or application programming interfaces (APIs) by simply adding a new Java Archive (JAR) file into the application classpath or into an application-specific extension directory.

In this particular phase we intent to develop the core functionality to the application based on a modular structure which is very scalable. This type of architecture will easily adapt to the development of new features in the form of new modules.

2.2.2 Portability and Platform Independence

This application is going to be implemented for Android Jelly Bean 4.3 using Android Development Environment which will guarantee that with a few changes could be potentially exported to previous or even newer versions of this Operative System.

Our Scope and Vision document states that we will develop this application just for the Android platform.

2.2.3 Reuse

Since our intention is to build an application as modular as possible, there is a very high probability that the coded modules could be reused in the development of another application. For example, the code to communicate with the light MySQL database is highly reusable since a large portion, if not all the applications, use a database to store and retrieve information.

2.2.4 Reliability and Availability

In order to develop this application, we are going to use Agile Development and Continuous Integration paradigms which are going to contribute on the early detection of bugs, several builds of working software and a reliable application for the customer.

2.2.5 Security

For this first release of the application, we are not considering implementing any kind of external communication (Internet connection, Bluetooth communication, etc.). Because that, the only security concern at this point is data integrity, meaning that we will keep the front end isolated from the database using an interface. This will prevent the user to directly or accidentally manipulate data on the database.

2.3 Product Family Roadmap

Since this application is part of a school project, we do not have the intention to develop any other applications besides of My Diet as part of a family.

2.4 Product Environment

2.4.1 Software

- The application will be developed for Android Jelly Bean 4.3.
- Developers will be using the Android Development Environment.
- It will use a light MySQL database to store and gather all the needed information.

- It does not require the development of any additional libraries or drivers.

2.4.2 Hardware

There will be plenty of devices that will support My Diet application. Due to the backward compatibility of Android we will be able to support devices running Android 2.3 and up. More detailed information can be found on section 4 (Hardware Interfaces) of the Software Requirements Specification Document.

2.5 Assumptions, Dependencies and Constraints

2.5.1 Product Cost/Schedule Constraints

Deliverable	Phase	Type of Deliverable	Cost (Time)
Vision and scope and document	1	Documentation	15 hrs
SRS	1	Documentation	15 hrs
Test Strategy	1	Documentation	20hrs
Vision and Scope Document	2	Documentation	10hrs
SRS	2	Documentation	10hrs
Demo • Top priority feature requirements • Top priority feature requirements	1	Demo	120hrs

tests			
Demo ● Low priority feature requirements. ● Low priority feature requirements.	2	Demo	120hrs
Final Demo	3	Demo	120hrs

2.5.2 Compatibility Constraints

App quality directly influences the long-term success of your app in terms of installs, user rating and reviews, engagement, and user retention. Android users expect high-quality apps, even more so if they've spent money on them.

We are going to base our design on the Android external standard and constrains described on the Core App Quality Guidelines. This document helps you assess basic aspects of quality in your app through a compact set of core app quality criteria and associated tests. All Android apps should meet these criteria.

2.5.3 Visual Design and User Interaction

These criteria ensure that your app provides standard Android visual design and interaction patterns where appropriate, for a consistent and intuitive user experience.

Area	Description
Standard design	App follows Android Design guidelines and uses common UI patterns and icons: App does not redefine the expected function of a system icon (such as the Back button). App does not replace a system icon with a completely different icon if it triggers the standard UI behavior. If the app provides a customized version of a standard system icon, the icon strongly resembles the system icon and triggers the standard system behavior. App does not redefine or misuse Android UI patterns, such that icons or behaviors could be misleading or confusing to users.
Navigation	App supports standard system Back button navigation and does not make use of any custom, on-screen "Back button" prompts.
	All dialogs are dismissible using the Back button.
	Pressing the Home button at any point navigates to the Home screen of the device.
Notifications	Notifications follow Android Design guidelines. In particular: Multiple notifications are stacked into a single notification object, where possible. Notifications are persistent only if related to ongoing events (such as music playback or a phone call). Notifications do not contain advertising or content unrelated to the core function of the app, unless the

	user has opted in.
	App uses notifications only to: Indicate a change in context relating to the user personally (such as an incoming message), or Expose information/controls relating to an ongoing event (such as music playback or a phone call).

2.5.4 Functionality

These criteria ensure that your app provides expected functional behavior with the appropriate level of permissions.

Area	Description
Permissions	App requests only the absolute minimum permissions that it needs to support core functionality.
	App does not request permissions to access sensitive data (such as Contacts or the System Log) or services that can cost the user money (such as the Dialer or SMS), unless related to a core capability of the app.
Install location	App functions normally when installed on SD card (if supported by app). Supporting installation to SD card is recommended for most large apps (10MB+). See the App Install Location developer guide for information about which types of apps should support installation to SD card.
Audio	Audio does not play when the screen is off, unless

	this is a core feature (for example, the app is a music player).
	Audio does not play behind the lock screen, unless this is a core feature.
	Audio does not play on the home screen or over another app, unless this is a core feature.
	Audio resumes when the app returns to the foreground. It will indicate to the user that playback is in a paused state.
UI and Graphics	App supports both landscape and portrait orientations (if possible). Orientations expose largely the same features and actions and preserve functional parity. Minor changes in content or views are acceptable.
	App uses the whole screen in both orientations and does not letterbox to account for orientation changes. Minor letterboxing to compensate for small variations in screen geometry is acceptable.
	App correctly handles rapid transitions between display orientations without rendering problems.
User/app state	App should not leave any services running when the app is in the background, unless related to a core capability of the app. For example, the app should not leave services running to maintain a network connection for notifications, to maintain a Bluetooth connection, or to keep the GPS powered-on.

	App correctly preserves and restores user or app state. App preserves user or app state when leaving the foreground and prevents accidental data loss due to back-navigation and other state changes. When returning to the foreground, the app must restore the preserved state and any significant transaction that was pending, such as changes to editable fields, game progress, menus, videos, and other sections of the app or game. When the app is resumed from the Recent app switcher, the app returns the user to the exact state in which it was last used. When the app is resumed after the device wakes from sleep (locked) state, the app returns the user to the exact state in which it was last used. When the app is re-launched from Home or All Apps, the app restores the app state as closely as possible to the previous state. On Back key press, the app gives the user the option of saving any app or user state that would otherwise be lost on back-navigation.

2.5.5 Performance and Stability

To ensure a high user rating, your app needs to perform well and stay responsive on all of the devices and form factors and screens that it is targeting. These criteria ensure that the app provides the basic performance, stability, and responsiveness expected by users.

Area	Description
Stability	App does not crash, force close, freeze, or otherwise

	function abnormally on any targeted device.
Performance	App loads quickly or provides onscreen feedback to the user (a progress indicator or similar cue) if the app takes longer than two seconds to load.
	With StrictMode enabled (see StrictMode Testing, below), no red flashes (performance warnings from StrictMode) are visible when exercising the app, including during game play, animations and UI transitions, and any other part of the app.
Media	Music and video playback is smooth, without crackle, stutter, or other artifacts, during normal app usage and load.
Visual quality	App displays graphics, text, images, and other UI elements without noticeable distortion, blurring, or pixelation. App provides high-quality graphics for all targeted screen sizes and form factors, including for larger-screen devices such as tablets. No aliasing at the edges of menus, buttons, and other UI elements is visible.
	App displays text and text blocks in an acceptable manner. Composition is acceptable in all supported form factors, including for larger-screen devices such as tablets. No cut-off letters or words are visible. No improper word wraps within buttons or icons are visible. Sufficient spacing between text and surrounding elements.

2.5.6 Tool Availability Constraints

Based on the Architecture we chose we have plenty of development, testing and debug tools: First of all the system we are using for develop the Android tool is the Android Developer Tool. The Android Developer Tools (ADT) plug-in for Eclipse provides a professional-grade development environment for building Android apps. It's a full Java IDE with advanced features to help you build, test, debug, and package your Android apps.

2.5.6.1 Full Java IDE

- Android-specific refactoring, quick fixes, integrated navigation between Java and XML resources.
- Enhanced XML editors for Android XML resources.
- Static analysis tools to catch performance, usability, and correctness problems.
- Build support for complex projects, command-line support for CI through Ant. Includes ProGuard and app-signing.
- Template-based wizard to create standard Android projects and components.

2.5.6.2 On-device Developer Option

- Enable debugging over USB.
- Quickly capture bug reports onto the device.
- Show CPU usage on screen.
- Draw debugging information on screen such as layout bounds, updates on GPU views and hardware layers, and other information.
- Plus many more options to simulate app stresses or enable debugging options.

2.5.6.3 Graphical UI Builders

- Build rich Android UI with drag and drop.
- Visualize your UI on tablets, phones, and other devices. Switch themes, locales, even platform versions instantly, without building.
- Visual refactoring lets you extracts layout for inclusion, convert layouts, extract styles.
- Editor support for working with custom UI components.

2.5.6.4 Develop on Hardware Devices

- Use any commercial Android hardware device or multiple devices.
- Deploy your app to connected devices directly from the IDE.
- Live, on-device debugging, testing, and profiling.

2.5.6.5 Develop on Virtual Devices

- Emulate any device. Use custom screen sizes, keyboards, and other hardware components.
- Advanced hardware emulation, including camera, sensors, multi-touch and telephone.
- Develop and test for broad device compatibility

2.5.6.6 Debugging

- Full Java debugger with on-device debugging and Android-specific tools.
- Built-in memory analysis, performance/CPU profiling, OpenGL ES tracing.
- Graphical tools for debugging and optimizing UI, runtime inspection of UI structure and performance.

2.5.6.7 Testing

- Fully instrumented, scriptable test environment.
- Integrated reports using standard test UI.
- Create and run unit tests on hardware devices or emulator.

2.5.6.8 Native Development

- Support for compiling and packaging existing code written in C or C++.
- Support for packaging multiple architectures in a single binary, for broad compatibility.

2.5.7 Environmental Constraints

Chipset	ARM-based/X86 based	For the first release, Android is primarily targeted towards mobile handsets and portions of the platform, such as Dalvik VM graphics processing, currently assume an ARM or X86 architecture.
Memory	128 MB RAM; 256 MB Flash External	Android can boot and run in configurations with less memory, but it isn't recommended.
Storage	Mini or Micro SD	Not necessary for basic bring up, but recommended.
Primary Display	QVGA TFT LCD or larger, 16-bit color or better	The current Android interface targets a touch-based HVGA resolution display with a touch-interface no smaller than 2.8 inches in size. However, smaller displays will suffice for initial porting.
Navigation Keys	5-way navigation with 5 application keys, power, camera and volume controls	

3. Architectural Specification

3.1 Organization

In this section we are going to describe the architecture designing for My Diet application. The first part explains the architectural model and this concepts (section 3.3). Second, we explain the system structural, class of architecture model and the 'Big picture' representation (section 3.4). All of these should be read for the all stakeholders.

The following sections contains in more detail and technical vocabulary the architectural specification, it should be read for technical staff. Then, the document shows the division into architectural components from major to minor components, list of product deliverables, exception handling, memory management, logging and other operation's components (sections 3.5, 3.7). Finally it shows other architectural alternatives that they were evaluate in order to choose the architecture model (sections 3.8, 3.9).

3.2 Syntax

This document use UML (Unified Modeling Language) notation (http://www.uml.org/) for some of the diagrams. The others diagrams is easy to read and interpret for any stakeholders because use a combination of natural language with some UML notation.

3.3 The Architectural Model

Android's applications use an n-layer or multi-tier architectural model which it provides a model by which developers can create flexible and reusable applications. By segregating an application into tiers, developers acquire the option of modifying or adding a specific layer, instead of reworking the entire application.

3.3.1 Architectural Concepts and Representations

It uses UML (Unified Modeling Language) notation (http://www.uml.org/) for class diagrams. Also, we use a diagram blocks technique (http://en.wikipedia.org/wiki/Block_diagram) to explain in a graphical way the main architecture for My Diet application.

3.4 System Structural Overview

In this section, we are describing the android architecture and how we use to build My Diet application.

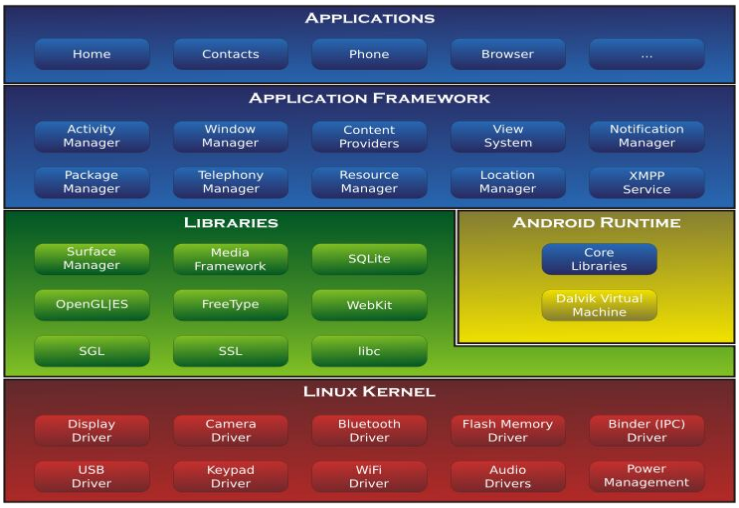
My Diet application is going to build in the application's tier as part of the android architecture. My Diet is divided in two main tier: View or Front-End and Control/Model tier.

The view tier is compose for all the user interfaces and the control/model tier is managing the business rules and communication for the local database. Also manage the messages between tier's or layers.

The developer just need to coding in this two tiers and do not mix with the rest of android's architecture.

3.4.1 Class of Architectural Model

Android's architecture was built as multi-tier architecture in which different components, packages, libraries and so on are logically separated in order to get a better reusability.



Android application architecture is framework-based application architecture as opposite to a free-style application architecture. This framework-based application is based on an existing framework and a developer extends certain classes or implements interfaces provided by the framework to build an application. Different parts of Android application can invoke each other and communicate between them only in an explicit way; there is no shared state between them in-memory.

Android application architecture follows very modern Model-View-ViewModel (MVVM) architectural pattern in how it handles relationships between GUI and a logic supporting the GUI. MVVM (http://en.wikipedia.org/wiki/Model_View_ViewModel) architecture responsibilities:

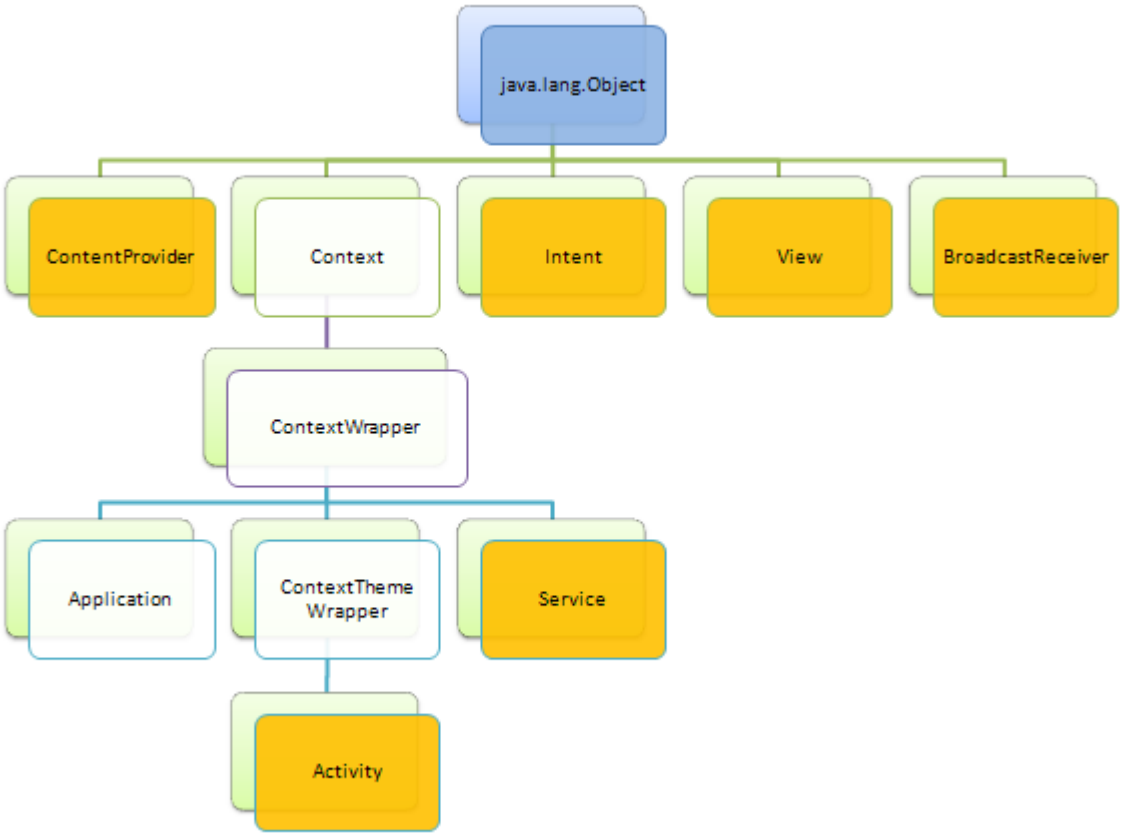
- GUI development is done by a GUI designer in technology more or less natural for this task (XML).
- Logic behind the GUI is implemented by a developer as ViewModel (which actually means Model of View) component
- Functional relationships between GUI and ViewModel are implemented through bindings that essentially define rules. Bindings can be written in the code or defined in a declarative way (Android uses both types of bindings).

Android application framework uses several communication patterns:

- Message passing which involves Intent class
- Publish/subscribe or pub/sub which also involves Intent class and BroadcastReceiver class
- Late binding and method calls that are used for accessing ContentProviders and local (in-process) Services
- Late binding and Inter-process Procedure Communication (IPC) for invoking remote (AIDL) Services

My Diet model is representing in a way that presentation, application processing, and data management functions are logically separated.

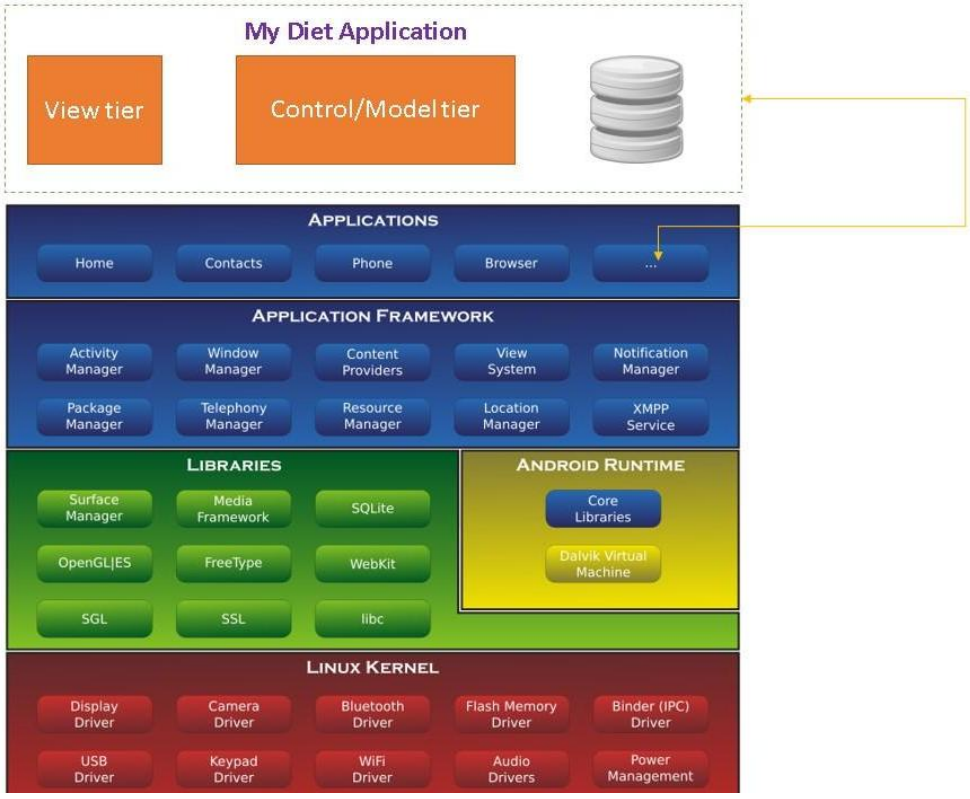
3.4.2 Main class diagram for Android Applications



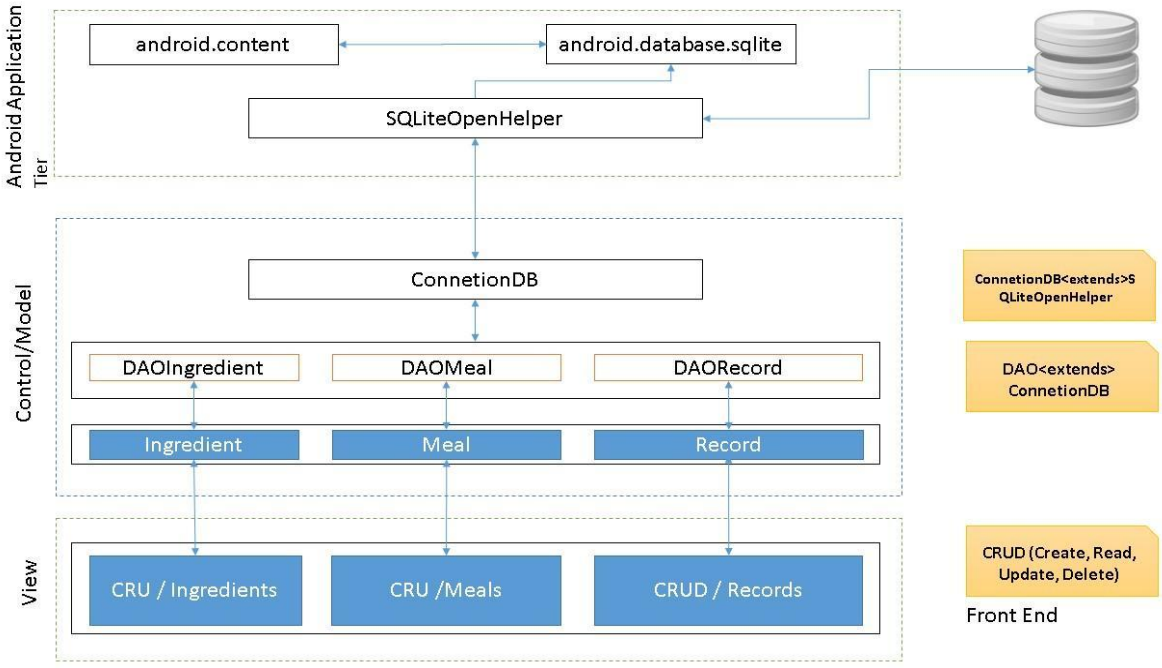
3.4.3 Pictorial of Product's Structure

My Diet architecture is in the application's tier from Android architecture. This figure shows up the main components on the MyDiet architecture's.

3.4.3.1 Architectural big picture - My Diet



3.4.3.2 Architectural block diagram - My Diet



3.5 Division into Architectural Components

My Diet Application is broken down in two main tiers, view and control/model. It follows and object-oriented decomposition (http://en.wikipedia.org/wiki/Decomposition_computer_science) as equal the android framework.

The main reason for this decomposition is to emulate the Model-View-Controller (MVC) in order to get a better cohesion between components a lower acoplation in the tiers because it offers a rapid and a quality development process.

3.5.1 View Tier

This tier manage the interaction between user interfaces (UI) or Front-End and the business rules. Also, this tier will render all the pages and interacting with the My Diet user's. It needs some components from Android Framework in order to control the events in the user interfaces.

In the view tier we can find the different views for ingredients, meals, and records. The main operations for those views are: create, read, update, delete, also known as CRUD operations.

In the ingredient and meals cases drop the delete operation as it defined in the requirements.

3.5.2 Control / Model Tier

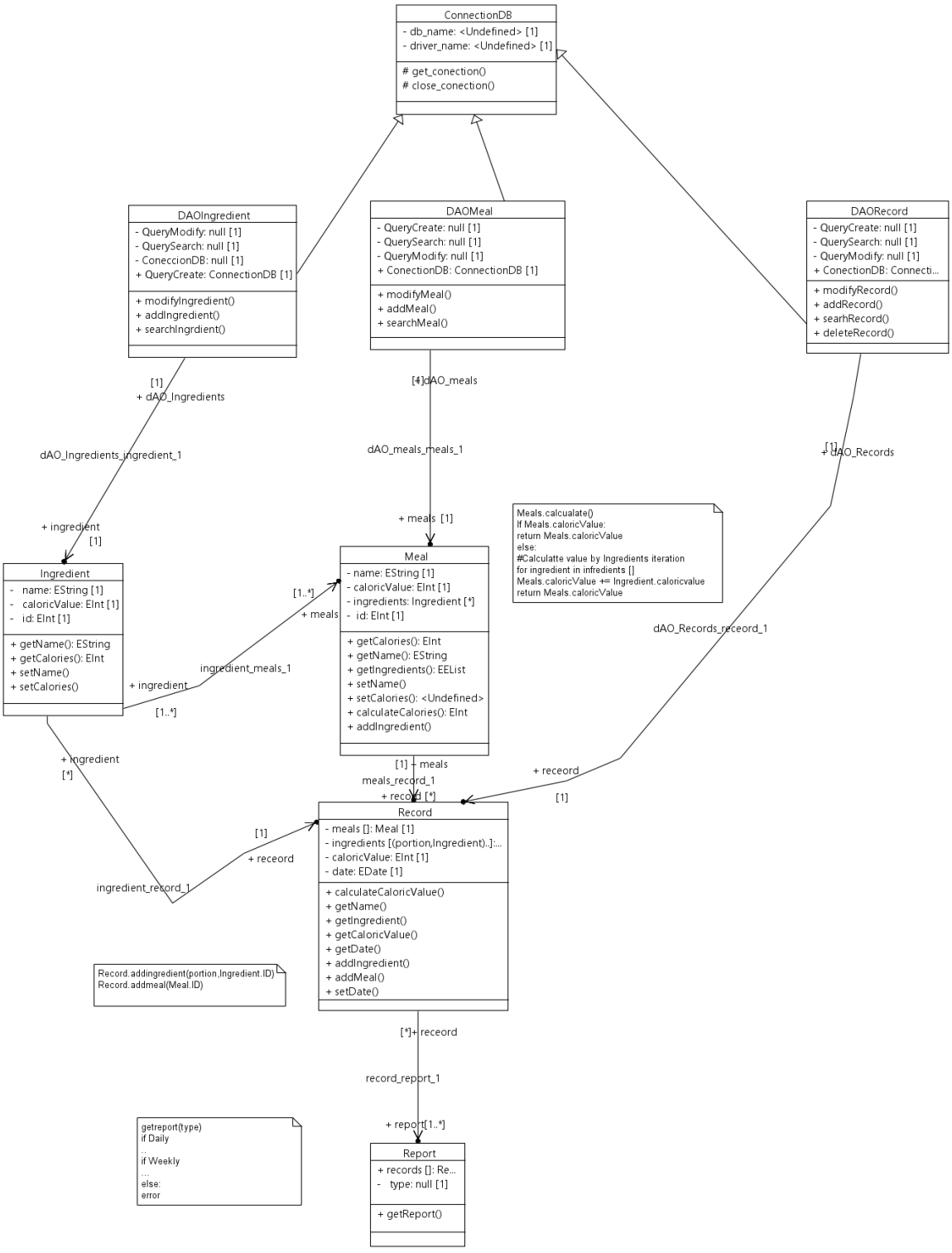
The main function for this tier is to manage the interaction between view tier and the business rules for My Diet. We can find all the business rules, calorie calculator, interaction with the local database and messages control flows with the view tier in order to respond in an optimal way all the requests generated by the user.

This tier uses the main objects such as: ingredient, meal and record. These objects are use to encapsulate the inputs from a user and pass these objects to respond some request. In that point, it uses the DAO classes to manage the request and interchange messages with the database.

The major components are: ConnectionDB class who is the database connections manager, DAO classes which manage the differents operations (CRUD) for each base object (ingredient, meal, record) with the database, and base classes which encapsulate the information.

3.5.3 Class Diagram – My Diet

In this diagram we can see the different relationships between the different components, the main methods and their attributes.



3.5.4 Major Architectural Components

My Diet application has two major components:

- A class which manage the connections with the database.
- Classes which manage the CRUD operations for base classes.

3.5.4.1 ConnectionDB

Component Type	Main class who is managing the connections between application and local database. This class uses the sqlite component from android SDK.
Purpose	Manage the communication between application and databases requests.
Functionality	Opening and closing connections between My Diet application and the local database.
Externally resources required	Not apply.
Externally visible attributes	Maintainability, compatibility, reliability..
External interfaces	None.
Internal interfaces	None
Dependencies and Inter-relationships with other major components	android.content.Context class: Require android.database.sqlite.SQLiteOpenHelper class: Extends
Requirements and constraints	Openning and closing database connections .
Development Verification/Validation/Analysis/Test strategies	Unit testing: Test case

3.5.4.2 DAOIngredient

Component Type	Class who is managing the ingredient requests between application and local database. This class uses ConnectionDB class.
Purpose	Manage the CRU operations for ingredients in the database.
Functionality	Respond requests (CRU operation) from the view tier and establish communication with the database. Passing the information and return the messages between the view and control/model tiers.
Externally resources required	None.
Externally visible attributes	Maintainability, extensibility, reliability.
External interfaces	None
Internal interfaces	None
Dependencies and Inter-relationships with other major components	ConnectionDB class: extends
Requirements and constraints	Add, modify and search ingredients.
Development Verification/Validation/Analysis/Test strategies	Unit testing: Test case

3.5.4.3 DAOMeal

Component Type	Class who is managing the meals requests between application and local database. This class uses ConnectionDB class.
Purpose	Manage the CRU operations for meals in the database.
Functionality	Respond requests (CRU operation) from the view tier and establish communication with the database. Passing the information and return the messages between the view and control/model tiers.
Externally resources required	None.
Externally visible attributes	Maintainability, extensibility, reliability.
External interfaces	None
Internal interfaces	None
Dependencies and Inter-relationships with other major components	ConnectionDB class: extends
Requirements and constraints	Add, modify and search meals.
Development Verification/Validation/Analysis/Test strategies	Unit testing: Test case

3.5.4.4 DAORecord

Component Type	Class who is managing the records requests between application and local database. This class uses ConnectionDB class.
Purpose	Manage the CRUD operations for records in the database.
Functionality	Respond requests (CRUD operation) from the view tier and establish communication with the database. Passing the information and return the messages between the view and control/model tiers.
Externally resources required	None.
Externally visible attributes	Maintainability, extensibility, reliability.
External interfaces	None
Internal interfaces	None
Dependencies and Inter-relationships with other major components	ConnectionDB class: extends
Requirements and constraints	Add, modify, delete and search records.
Development Verification/Validation/Analysis/Test strategies	Unit testing: Test case

3.5.5 List of Product Deliverables

Release	Implemented features
1.0	1. View daily nutritional summary reports. 2. Calorie counter or calculator.
1.1	1. Check the progress with a simple calorie tracker that shows how you're tracking against your daily calorie budget and diet. 2. Add food items easily with our database (foods and recipes).
1.2	1. Set reminders and notifications for meals and reports. 2. Customized goals and calorie counter based on your specific diet profile – age, gender, height, etc.

3.5.5.1 User Interfaces

Component Name	Brief Description
Load page	My diet app screen. Load page.
User profile	Profile view: The user can configure his profile.
Ingredients menu	Food/Ingredients view: The user can add, modify and delete food/ingredients.
Custom meals menu	Custom meals view: The user can add, modify, search and delete the custom meals.
Records menu	Records view: The user can add, modify and delete the records of the calories or meals consumed.
Meals menu	Meals view: The user can view a several meals with the calorie information. Add, modify and search meal.
Calorie indicator	Calorie indicator view: The user can

	view the quantity of calories consumed in the current day.
Reports	Reports view: The user can show the daily and weely report for the consumed calories.
Help	Help view: The user can find the user guide in order to involve with the application in a right way.
About	About view: The user can review the version information from the application, the authors, license and name company.

3.5.5.2 Programs without Interfaces

Component Name	**Brief Description**
ConnectionDB	Manage the communication between application and databases requests.
DAOIngredient	Manage the CRU operations for ingredients in the database.
DAOMeal	Manage the CRU operations for meals in the database.
DAORecord	Manage the CRUD operations for records in the database.

3.5.5.3 Internal Libraries

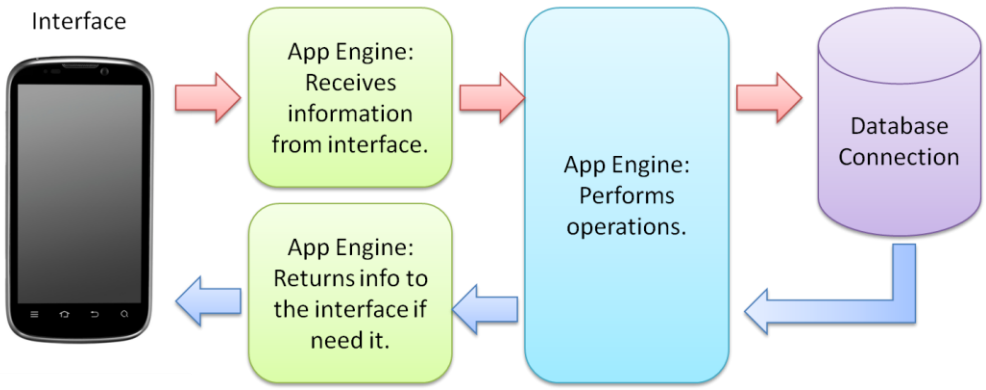
Component Name	**Brief Description**
Android framework.	A framework to develop android applications.

3.5.5.4 Product Documentation

Component Name	Brief Description
Vision and scope	Shows up the vision and scope for the application.
Software requirements	Shows up all the requirements and their constraints.
Test strategy	Shows up how to test the requirements.
Architecture	Shows up how is the architecture for the application.

3.6 Theory of Operation

The architectural model in which android is based and the applications can be build allows us to create simple modules that can be reused and handle most of the use cases as a simple data flow operation. See the image below:



The major components of the application are: Interface, Application Engine, DAO's and Database. Application Engine can be divided also in four major classes: Ingredient, meal, record and report.

3.7 System-Level Operations

3.7.1 Exception Handling

The Exceptions in the application are going to be "Application Exceptions", despite other approaches we are not going to handle or use the "System Exceptions", ins this paradigm we are not going to use the classic try and catch structure, exceptions are going to be handled by the Thread.setDefaultUncaughtExceptionHandler() method.

3.7.2 Memory Management

The memory management is going to be handled by the Android OS. Since the tools is not going to allocate or deallocate memory manually.

3.7.3 Instrumentation and Debug

>> Are we going to perform code coverage? my suggestion is No... but I dont know

3.7.4 Diagnostics and Error Logging

The error logs are going to be handled with the Log Class, the Android logging system provides a mechanism for collecting and viewing system debug output.

3.7.5 Internationalization and Localization

This is not applicable in the tool, since it does not support internationalization (the only language available is English) and localization.

3.8 Rationale for Selected Architecture

There are two layers added to the Android architecture, these layers are: view tier (user interface) and the control model tier (interaction).

The major components of the view tier layer are:

- View ingredients: gives a view of the ingredients that are in the database.
- View meals: gives a view of the meals that are in the database.
- View records: gives a view of the records that the user has added to the database.

All of these components in the layer support the usability requirement. This view will be the interaction that the user will have with the application and is important that the views are easy to navigate through and that each of these views has all the tools required.

The second layer added is the control model tier, and its major components are:

- Connection DB: Manage the communication between application and databases requests.
- DAO classes: this component manages all the CURD operations for the ingredients, meals and records.

These two major components manage the CRUD operations for base classes giving the application the cabaility to escalate, adapt and perform as required.

3.9 Summary Evaluations of Alternative Architectures

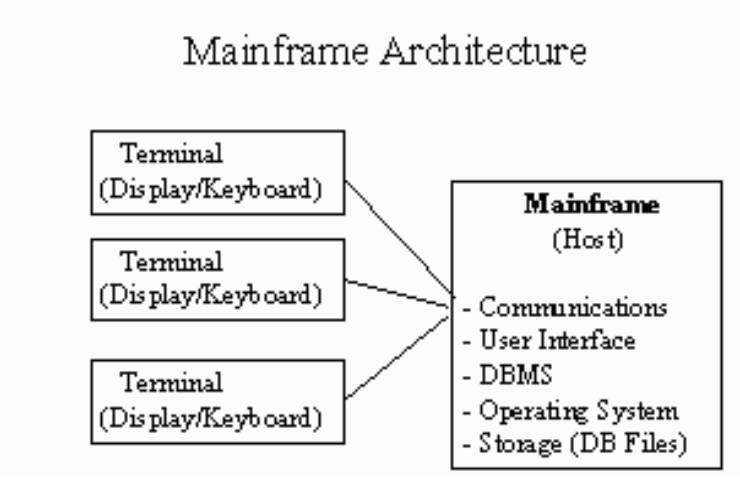
The architecture model for the database in the application is a entity-relationship model; the architecture chosen for the Android application itself is a n-layer model. The application is placed in the Android's application layer and it adds a layer for the interaction with the database and the business rules, and other that handles the user interface.

Other architectures for database handling and applications on Android are:

- Mainframe architecture
- Personal computer - Standalone database
- File sharing architecture
- Client/server architecture
- Distributed database architecture
- Pipe-filter architecture

3.9.1 Mainframe architecture

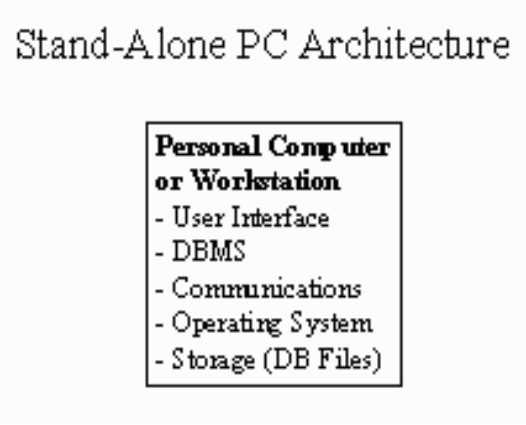
In this architecture the database is located in a mainframe computer and the applications are run on the same computer. Multiple users get access to the applications using simple terminals that don´t have power of their own. In this type of architecture, user interface is text-mode screens. Though, users can't manipulate data outside of the standard applications.



We chose entity relationship architecture over mainframe architecture because there is no need for several terminals to access the database. There will be only one database that will be accessed by the Android application so there is no point in enabling other terminals to access the database. Also, this type of architecture does not implement a graphical user interface which is important for My Diet.

3.9.2 Personal Computer – Standalone Database

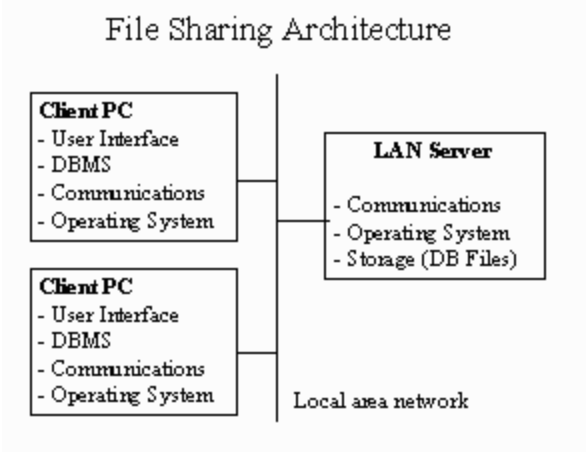
The database is located in the hard disk of a PC and applications run in the same PC and have direct access to the database. The database can be accessed only by a single user. This architecture can implement a graphical user interface.



This architecture might have been useful for MyDiet, but the type of access that the app needs to the database can be solved by the entity relationship in a much simpler way.

3.9.3 File sharing architecture

In the architecture for file sharing the computers are connected to a LAN, there is only one file server that stores a copy of the files in the database. The application runs on each computer and access the same set of files on the server. The application serves as a DBMS. Each computer on the LAN has a copy of the same application. There is a limit on the number of users that can share the data.



Since the database in MyDiet needs to be accessed by a single user, we don't need to implement file sharing with other computers.

3.9.4 Client / Server Architecture

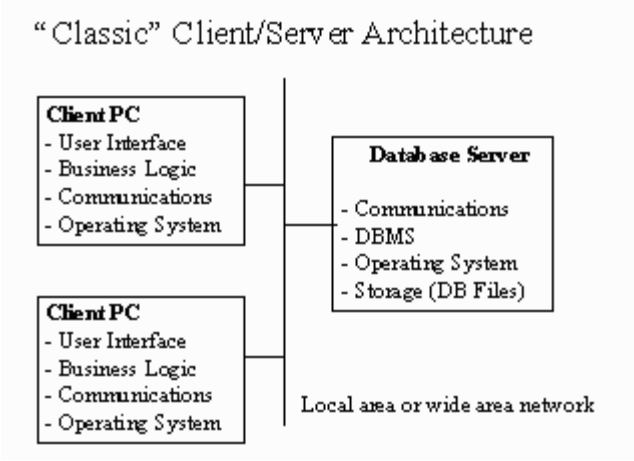
There are three main points in this architecture:

- Server machines: they run a copy of an operating system and a DBMS manages the database. It must provide a listening state to accept notifications from the clients.

- Client machines: they run a copy of an operating system and run the applications using their own memory. The clients connect to the DBMS in a running server through the middleware.

- Middleware: is a small portion of software that sits between client and server. It makes the connection between them passing commands.

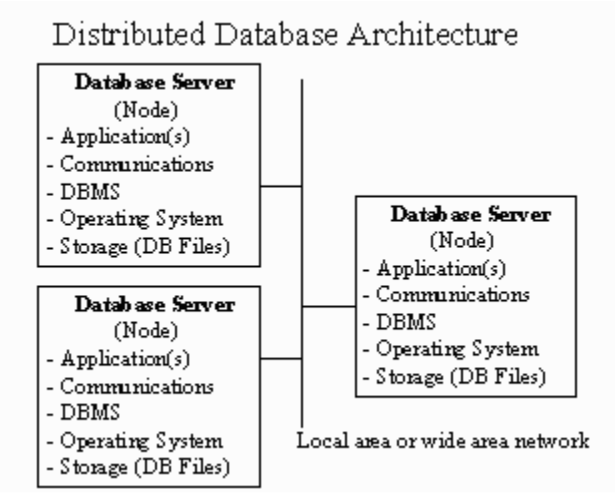
The implementation of this architecture is somewhat complex due to the middleware and the network must be suited for this type or architecture.



The database in MyDiet is a local database accessed by only one user. Even though client/server model might work, the cost for the implementation of the middleware is too high to implement when not all of the capability of this model will be implemented.

3.9.5 Distributed Database Architecture

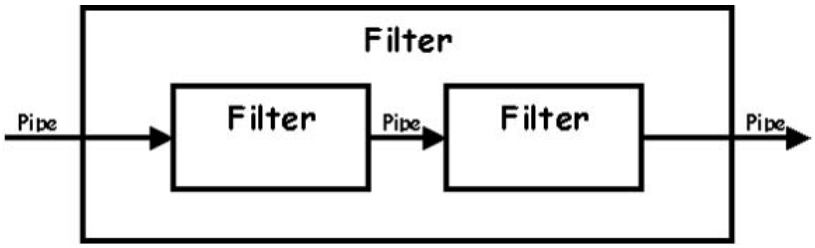
In this type of architecture, multiple DBMS run on multiple servers connected through a network. It requires data partitioning where the data can be split to improve performance. Data may also be replicated across multiple sites to improve performance by moving a cpu of data closer to the users and improve reliability and prevent data loss when site fails.



Distributed architectures are useful when the database must be shared with several users, or the data is too large to be processed by a single unit, but since the database for MyDiet is simple, the cost for implementing a distributed system would be too high and not all of its capabilities would be used; wasting resources.

3.9.6 Pipe-Filter architecture

This is a typical architecture for stream processing. A filter defines a process step and the data flows through a sequential chain of filters that represent a whole system. The filters are the set of inputs and outputs of said streams and imply local transformations. The pipes facilitate data flow and are the connectors between the filters. This type or architecture has poor performance and is not appropriate for interaction.



The architecture for Android application already uses a n-layer architecture, and it's simpler to add layers to the application layer rather than implementing other types of architectures.

Appendix A. Definitions

Term	Definition
Architecture	It is a blueprint for the system and the project software development.
Deliverables	Tangible or intangible object produced as a result of the project.
Exception handling	Process of responding to an exception during computation. The exceptions are anomalous or exceptional events requiring special considerations.
Memory management	Process of controlling and coordinating computer memory, assigning portions called blocks to various running programs to optimize.
Android architecture	Architecture composed by applications written in java, framework services and libraries executed in a virtual machine, native libraries, daemons and services written in C or C++. And the Linux Kernel which includes drivers for hardware, networking, file-system access and inter-process communication.
Object Oriented	It is a programming paradigm that represents concepts as objects that have attributes that describe the object and associated procedures as methods. Objects are used to interact with other objects to design applications and computer programs.
Class diagram	Type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations or methods and the relationships among objects.