ChocAn

Design Document

Authors: Daniel Mendez, Alexander Salazar, Alexander Dupree,

Arman Alauizadeh, Kyle Zalewski, Dominique Moore

Table of Contents

1. Introduction

The purpose of this document is to describe in detail the design, layout, and format characteristics of the ChocAn software system. The software is built to allow easier and higher-quality recordkeeping throughout the network of Chocoholics Anonymous providers. This will be accomplished through carefully considered design, which will increase security and usability. The specifics of this architecture will be described and illustrated in the remaining sections of this document. The document is organized in sections of decreasing abstraction.

1.1 Purpose and Scope

This document will illustrate and explain high- to medium-level outlines for the ChocAn system structure. Low-level code and syntax is beyond the scope of this design document. The main goal is to provide overall explanation that is sufficient to convey the reasons for each design consideration, and the benefit of each to the overall project.

1.2 Target Audience

The intended audience for this document is the Chocoholics Anonymous administrative team. It will provide sufficient technical detail to these teams so that they are informed on the project's general structure and base functions. This information will facilitate useful discussion between the Chocoholics Anonymous team and our development team.

1.3 Terms and Definitions

ChocAn - Umbrella term for the software project/package itself.
API - Application Programming Interface

2. Design Considerations

Describe the purpose of this section and provide an outline of its subsections. Only a few sentences are expected here.

2.1 Constraints and Dependencies

2.1.1 - Constraints

The test plan for the project must be completed by November 18th, 2019. The entire project must be completed by December 6th, 2019. The software must be written in C++ version 17.

2.1.2 - Dependencies

The current data processing software is dependent on the development of the communications software, the design of the ChocAn provider's terminal, the software required by Acme Accounting Services, and the implementation of the electronic funds transfer component by a third party software developing company.

2.2 Methodology

2.2.1 - Model

Our design model will be following a layered approach for abstraction. We try to keep our dependencies on the domain layer so that we can make more on the fly changes to higher abstracted layers such as the terminal viewer if needed. We felt that this approach was better since the requirements of ChocAn are clearly defined and static. We don't benefit as much from a more dynamic approach.

2.2.2 - Development process.

This software will utilize an iterative development process like Agile. We talk about our design and code and potential changes on a regular basis. We all work on our own sections of the codebase and perform our own unit tests before trying to push into the master. We believe this is an effective way to develop since we all follow our own internal guidelines. This makes it so that we can all work on different aspects of the codebase and still be able to understand each others work for review.

3. System Overview

The software will consist of four layers:

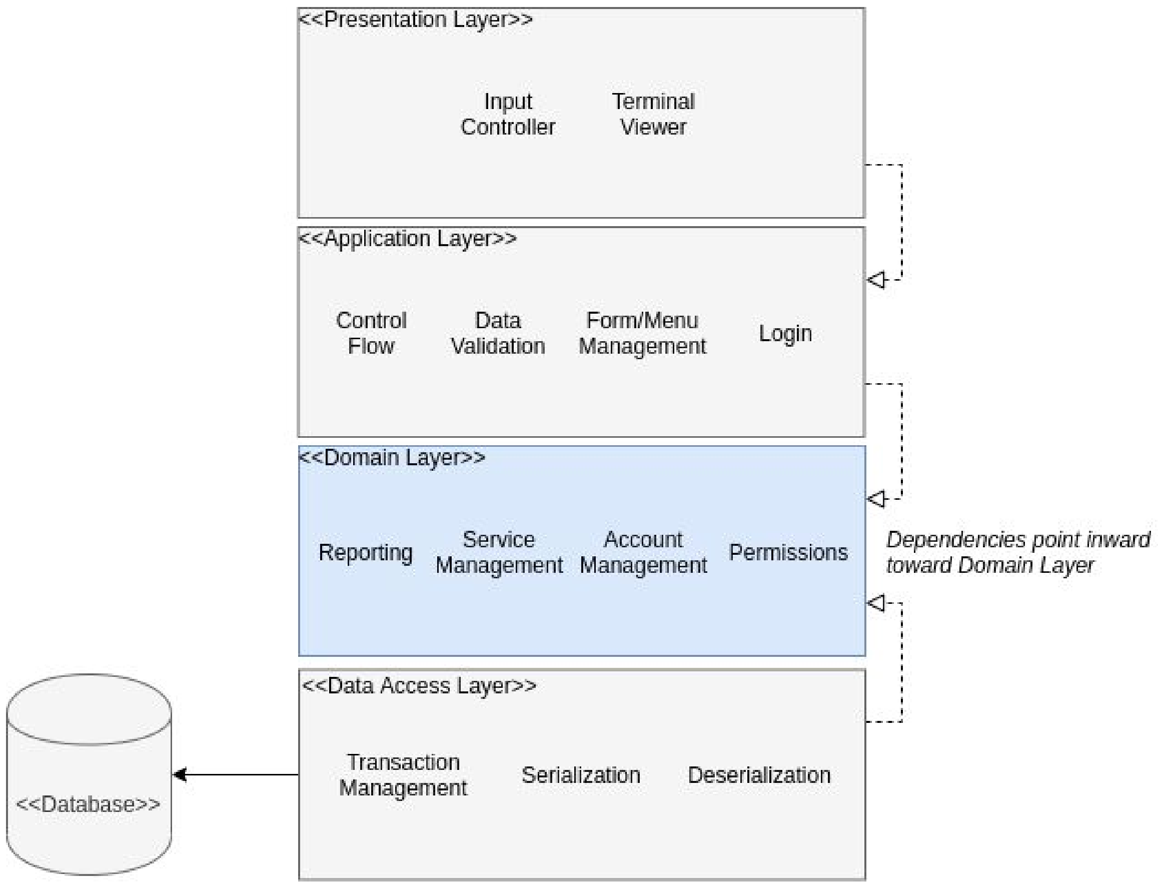
- The presentation layer
- The application layer
- The domain layer
- The data access layer

The presentation layer consists of the terminal viewer and input controller. The terminal viewer is a text file that lists the options the user can select. The input controller directs the user to the corresponding menu in the application layer. The type of account the user logs in on gives the user the corresponding permissions, and access to different functionalities of the software.

The application layer is where login, data validation, control flow, and form/menu management is managed. Depending on what the user selects in the terminal viewer, they will be sent to a corresponding menu for login. There are three types of menus that the user can potentially login to: member, provider, and manager. After the user enters their login information, the data get checked to see if it's valid. If the data the user entered successfully goes through validation they will sent to one of the three menus previously mentioned. Each of the three menus has different functionalities.

The domain layer handles reporting, service management, account management, and permissions. There are three different account types: member, provider, and manager. Each of the three account types has access to different functionalities of the software. This is where the permissions come in, they ensure that the user can only use the functions that they're allowed. The data that a user can access from the data access layer also depends on the type of account the user has logged in on.

The data access layer is where the data on the members, providers, and managers are held. Each account has an ID number that tells the system what kind of account the user has logged in on. The data is stored and maintained in an ordered list. When the user wants to access information on an account, the data access layer finds the information and sends it back to the domain layer.



4. System Architecture

The ChocAn system is a layered architecture, with each layer corresponding to a different class of functionality and responsibility. Each layer module implements their needed functionality independent of the other modules. Interfaces will be used to manage cross layer communication depencies, as a means of enforcing their respective independence and system modularity.

4.1 Data Access Layer

The primary responsibility of the *Data Access Layer* is to facilitate the retrieval and/or creation of data within the ChocAn database systems. This responsibility is carried out in two ways. First, by performing object serialization into data that can be stored into a database. Second, by performing data de-serialization into domain objects that the ChocAn application can use. The Data Access layer will implement a well defined interface provided by the *Domain Layer*.

4.2 Domain Layer

The *Domain Layer* is the core of the ChocAn application and handles the 'business logic' of the ChocAn domain. The *Domain Laye*r consist of objects and interfaces used to represent domain specific types of data and tasks. Such types will include Accounts (both member and provider), ChocAn ID generation, Service Transactions, Report Generation, and Authorization and Authentication. This domain logic will be largely dependent on ChocAn data services. As such, the *Domain Layer* will define a uniform interface that the *Data Access Layer* will implement. Inverting the dependency in this fashion keeps the *Domain Layer* independent and allows us to swap out different *Data Access* layer implementations.

4.3 Application Layer

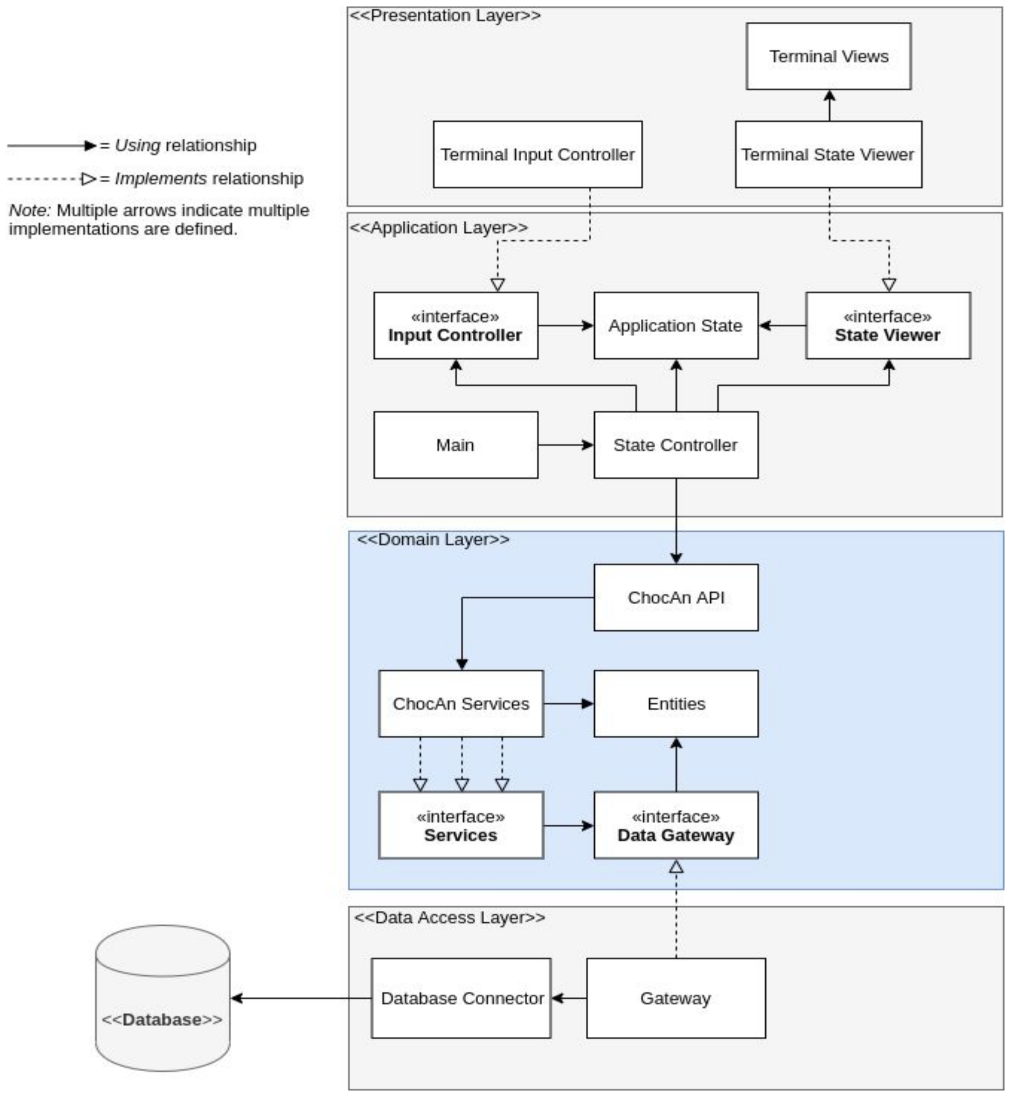
The *Application Layer* is handles user interaction with the application by being responsible for the application control flow. As the user interacts with the application, the internal state of the application will change. Furthermore, the set of actions we can perform at each state varies and depends on the internal state. As such, we will model user interaction by implementing the *Application Layer* as a Finite State Machine. This model allows us to explicitly manage control flow while giving us the ability to formally verify a systems correctness by mapping out paths through the State Machine. Since we do not want to tightly couple state representation with control logic, the *Application Layer* will define a State Viewer and Input Controller interface that the *Presentation Layer* will implement. This will allow us greater freedom over testing and presentation.

4.4 Presentation Layer

The *Presentation Layer* is responsible for everything Input/Output. This includes input collection via the Provider or Manager terminal as well as formatted printing of the Application states representation. The *Presentation Layer* implements all interfaces defined in the *Application Layer*.

5. Detailed System Design

The following is a structural representation of the ChocAn system. At the component level, the software relies heavily on abstraction to keep each layer loosely coupled through well defined interfaces. It is important to note that all dependencies that cross a layer boundary point inward toward the domain layer. This allows us to make radical changes to each layer of the system without creating regressions for other layers.



5.1 Data Access Layer

5.1.1 - Database Connector

This component is responsible for defining the logic needed to connect to a database management system. The database connector shall be the only component in the system with concrete knowledge over how information is stored and retrieved from the database.

5.1.2 - Gateway

The Gateway implements the *Data Gateway* interface defined in the domain layer. The responsibility of the gateway is to serialize domain entities into data the *Database Connector* can store, and to de-serialize data into domain entities that the ChocAn service can use.

5.2 Domain Layer

5.2.1 - Entities

The Entity component is comprised of multiple classes that represent the business logic of the ChocAn domain. These entities include member accounts, provider accounts, manager accounts, and transaction objects as well as the domain logic that defines how these entities interact with each other.

5.2.2 - Data Gateway Interface

This component provides a well defined interface for ChocAn services to use. This interface requires that concrete implementations define methods such as account creation, deletion, and retrieval.

5.2.3 - Base Service

Each *ChocAn Service* is dependent on a data gateway to perform the service activities. The base service will perform any needed database connection and validation before a ChocAn service can be constructed.

5.2.4 - ChocAn Services

ChocAn services carry out the many activities a user may need in order to interact with ChocAn entities. Such services include report generation, account creation and deletion, and login authentication/authorization. Each service inherits from the base service to ensure that a data gateway is present and defined in the system.

5.2.5 - ChocAn API

The *ChocAn API* consolidates the individual services into a singular API that the application layer can use. The ChocAn API handles the setup and teardown of the micro services and connects each service to the proper data gateway.

5.3 Application Layer

5.3.1 - Application State

This component is a discriminated union of several other different, but fixed, types. This union is leveraged in the *State Controller* to define which set of actions a user can take at a certain point in the application. For example, on application startup the current state of the Application will be the ‘Login State’, and performing a successful login action will transition the *Application State* to a different state, where we can perform different actions.

5.3.2 - State Controller

The *State Controller* uses *Application State,* the *Input Controller,* and the *State Viewer* to provide an interactive experience for the user. First, the *State Viewer* renders the current state, then the *Input Controller* collects input from the user which is then passed to the *Application State* where the input is evaluated and a new state is returned.

5.3.3 - Input Controller

This component is an interface that defines methods for input collection. This interface allows the application, and other layers, to be agnostic of the details of input collection, which in turn allows greater freedom for testing and dependency management.

5.3.3 - State Viewer

This component defines an interface for state representation. This interface enables states to be concerned only with their behavior and not with how they are presented to the user. This separation of logic from presentation enables greater control over testing of the application.

5.3.4 - Main

The Main component is the entry point of the application and instantiates all objects needed by the application. Main will also manage the main control loop.

5.4 Presentation Layer

5.4.1 - Terminal Input Controller

This component implements the *Input Controller* interface by reading input from STDIN and sanitizing it for use in the application layer.

5.4.2 - Terminal State Viewer

This component implements the *State Viewer* interface by display a formatted representation of a states *view* to STDOUT. The terminal state viewer will also contain viewer commands that the views can call which will allow the views to be dynamic.

5.4.3 - Terminal Views

Each application state is associated with a specific state view. These views are stored as plain text files, which the *Terminal State Viewer* reads and displays to the console. Views will be composable, I.E. the view ‘header.txt’ can be included in a different view by using a viewer render command in the view file.