Boardzilla
High-Level Architectural Design

Matthew Paulin	Hargun Bedi	Dylan Smith	Chenwei Song
paulinm	bedih	smithd35	songc12
400187147	400185463	001314410	400124879

Tianzheng Mai
mait6
400143042

August 31, 2021

Contents

1 Introduction

1.1 Purpose

High-level architectural design will provide an overview of the design of the entire system by identifying and delineating responsibilities of the system's components and all the interactions between them. The document's aim is to allow the stakeholders to not only understand the architectural design of the system, but also obtain a justification for why a particular design was chosen. The target audience includes the professor and teaching assistants who will inspect the document as well as the developers who may use this document as a reference.

1.2 System Description

The system that will be outlined in this document is called Boardzilla. Boardzilla is an online application that serves as a dashboard for a variety of information meant to serve as a daily briefing or hub. Each user will have their own protected account with a customizable dashboard screen containing the widgets of their choice. The users are able to select their desired amounts of a weather widget, a sticky notes widget, a calendar widget, a stock widget and a news widget. Widgets can be added or removed and can be repositioned on the users' dashboards. In addition, users will be able to input data into each widget and modify options unique to each widget.

1.3 Overview

After this introduction, the document will contain a detailed outline of the architectural design. It will begin with section two, an Analysis Class Diagram, describing how each of the system's components interact with each other. This diagram will also show the hierarchical relationship between components. Section three, Architectural Design, provides a description of the system's chosen architecture as well as a justification for this choice. It will also demonstrate how each component fits into the described architecture. The responsibilities of each of these components is then described in further detail in the fourth section, Class Responsibility Collaboration (CRC) Cards. Each card will detail the responsibilities of a single component and will name the collaborating components. The last section is a division of labour that details how the production of this document was split among group members.

2 Analysis Class Diagram

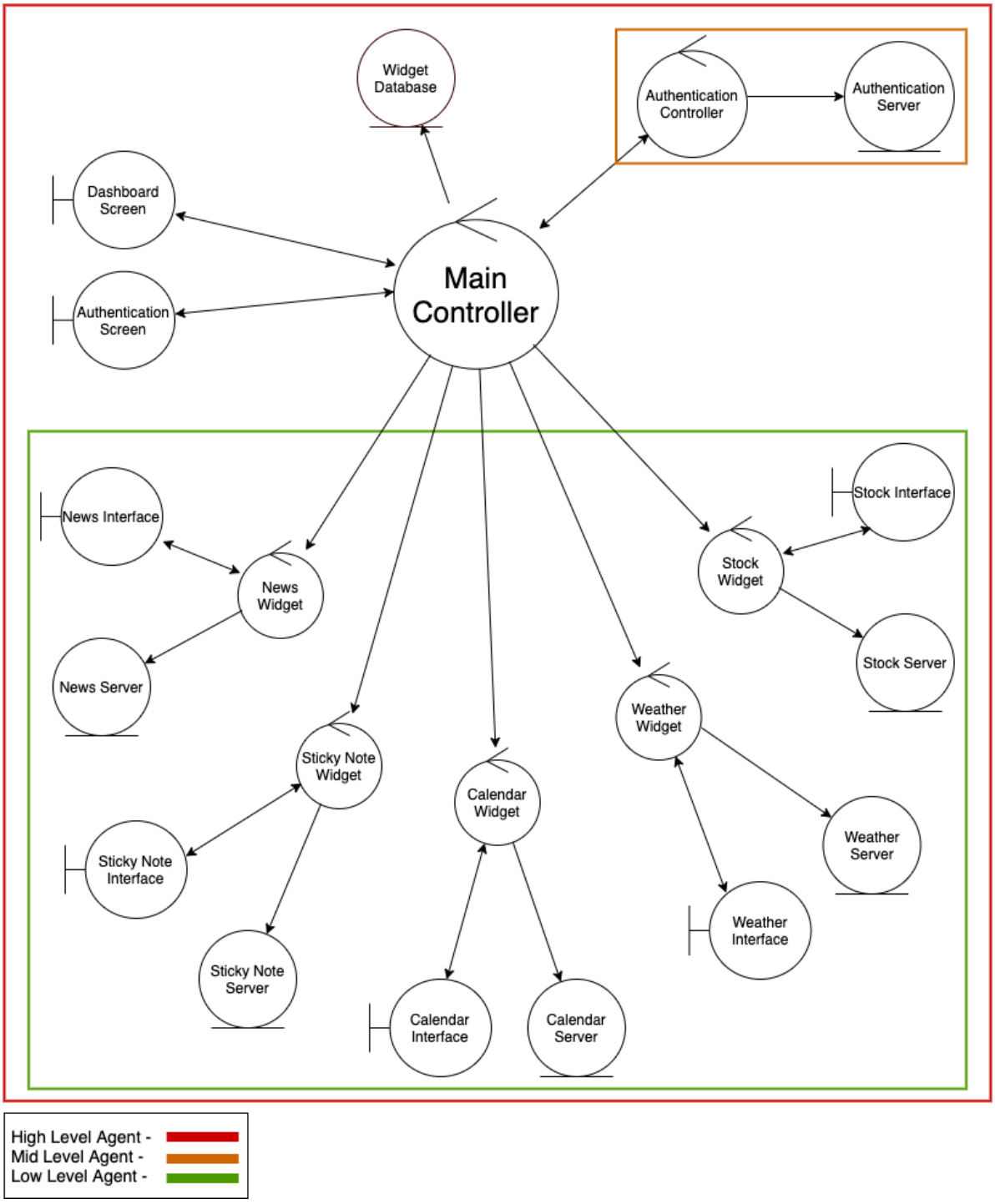

Figure 1: ACD Diagram

3 Architectural Design

3.1 System Architecture

The overall architecture for this system is the Presentation Abstraction Control (PAC) architecture. This architecture is a more complicated version of the Model View Controller (MVC) architecture. Each agent is separated into a triplet of a Presentation, an Abstraction, and a Control component. These agents are then organized hierarchically with communication between them being facilitated by the Control components of each triad. Similarly to the MVC architecture, the user interface elements are displayed solely through the Presentation components which obtain their information from the Abstraction. The Abstraction is like the Model in MVC, but may only contain some of the over-arching data structure of the system. It does not actively notify components or agents of changes to the data because the Control components handle this functionality. The Control components also process external events and adjust the stored data accordingly.

Our system represents the PAC architecture as a consequence of the hierarchy of the agents which are mostly comprised of PAC triplets. The system contains low level independent agents, the different widgets, that each provide different functionalities, have different interfaces and store their own data. This necessitates the need for every agent to have their own Presentation for the corresponding user interfaces, Abstraction for data storage, and Control to facilitate inter and intra-agent communication. Due to their differences, the widgets do not need to communicate with one another, thus all communication between high and low level agents is done through the main controller, the high level agent. All the widgets exist in parallel which allows users to view and interact with the interface of any of them at any given time. This architecture is ideal because we want the users to be able to view and interact with each of their widgets on the same screen. This parallelism is one of the major reasons we chose a PAC design over the MVC pattern. In addition, the PAC architecture's modular design makes it very easy to add different widgets without modifying any of the others, following the principal of information hiding. Additionally, the separation of the agents allows for some performance benefits due to the lack of a monolithic model.

3.2 Subsystems

The high level agent includes the Main Controller, the Authentication and Dashboard screens the Widget database. The controller corresponds to the Control component, the two screens constitute the Presentation of the agent, and the database represents the Abstraction. Interactions between the Abstraction and Presentation components is facilitated by the Main Controller. Connected to this agent are five low level widget agents as well as an authentication agent for managing authenticated sessions. Communication between these agents and the high level agent is also done through the Main Controller. The Dashboard Screen is the principal interface of this agent and displays all of the users widgets in the users' chosen layout. The addition, removal, or repositioning of the widget is communicated from the dashboard to the Widget Database through the controller. The dashboard will enable the user to logout, causing the controller to send data to the Authentication controller to invalidate a user's session. The secondary Presentation is the Authentication Screen and is responsible for displaying the user interface corresponding to the login, registration and password reset screens. User inputs are communicated to the Authentication Controller through the Main controller. The Main Controller also transfers control to the low level agents when necessary to update the displayed widgets or to enable user interaction with particular widgets.

The authentication agent is a unique subsystem in that it is the one agent that is not comprised of a PAC triplet. This is because there is no need for a display at this level in the hierarchy. Instead, the Presentation component is handled by the main controller, high level agent, which can more easily switch presentations between the Authentication Screen and the Dashboard Screen. The Authentication Controller is the Control component and the Authentication Server is the Abstraction. This agent is responsible for communicating user inputted authentication data to the Authentication Server as well as retrieving data from the server to validate and keep track of user sessions. The data from the server is then sent to the main controller, where the presentations can be adjusted accordingly.

The low level agents are all PAC triplets corresponding to one of five different widget subsystems. Each

agent has a particular user interface so every Presentation component uniquely displays widget data. Since there can be multiple of each type of widget, at one time, there may be multiple Presentation components corresponding to the same agent. In addition, users can interact with the agents to modify the data displayed as well as other widget settings. Each widget will differ in what can be modified by the user. Relevant widget data is retrieved from the Abstraction, represented by different widget servers and is then sent to the Presentation through the Control. Similarly, all updated widget data is reflected in the Abstractions through communication from the presentation through the Control components. The low level agents do not communicate with one another at all, but receive communication from the high level agent when given control or asked to surrender it.

4 Class Responsibility Collaboration (CRC) Cards

Class Name: Main Controller	
Responsibility:	**Collaborators:**
• Interacts with Dashboard Screen to display the users' widgets in their locations when their session is validated • Uses the controllers of the widgets to display the widget information • Passes control to the controllers corresponding to the widget that the user interacts with (News Widget, Sticky Note Widget, Calendar Widget, Weather Widget, Stock Widget) • Interacts with the Widget Database to save widgets' positions • Update references to widgets in the Widget Database if they are added or deleted on the user's dashboard • Interact with Authentication Controller to invalidate the session upon user logout • Interact with Authentication Controller to validate a session upon user login • Interacts with Authentication Screen to get user login • Interacts with Authentication Screen to obtain user registration data • Interacts with the Authentication Controller to reset a user's password • Interacts with Authentication Screen to display an error message for invalid inputs or incorrect credentials • Interacts with Authentication Screen to display a success message upon password change or user registration	• Dashboard Screen • Authentication Screen • Widget Database • Authentication Controller • News Widget • Calendar Widget • Weather Widget • Stock Widget • Sticky Note Widget

Table 1: CRC for Main Controller

Class Name: Dashboard Screen	
Responsibility:	**Collaborators:**
• Allows users to add and delete widgets • Enable users to reposition widgets and minimize them • Send updated widget statuses to the Main Controller • Display widgets in accordance with the user's saved layout • Interacts with the Main controller to allow the users to log out	• Main Controller

Table 2: CRC for Dashboard Screen

Class Name: Authentication Screen	
Responsibility:	**Collaborators:**
• Allows users to log in to their account with valid credentials. • Allows users to register a new account by entering a valid username and password. • Allows users to reset password if they have an existing account. • Display an error message after user submits invalid credentials for login, registration, or password modification • Display a success message upon password change or registration	• Main Controller

Table 3: CRC for Authentication Screen

Class Name: Widget Database	
Responsibility:	**Collaborators:**
• Stores a reference to the identity of all widgets • Stores users' widget positioning data	N/A

Table 4: CRC for Widget Database

Class Name: Authentication Controller	
Responsibility:	**Collaborators:**
• Interact with Authentication Server to check if the session is authenticated • Interact with Main Controller to invalidate the session if the user attempts to log out • Interact with the Main Controller to validate the session upon login or registration • Interact with Main Controller to notify if the credentials entered are valid logins or registrations • Interact with the Authentication Server to store the registered credentials • Interact with Main Controller to notify if the account has registered successfully • Interact with the Authentication Server to verify login attempts • Interact with the Main Controller to redirect the user to the Dashboard Screen upon login or registration • Interact with Main Controller to notify if the password has been changed successfully • Interact with Authentication Server to store the reset credentials	• Main Controller • Authentication Server

Table 5: CRC for Authentication Controller

Class Name: Authentication Server	
Responsibility:	**Collaborators:**
• Stores login credentials • Stores references to valid user sessions	• Authentication Controller

Table 6: CRC for Authentication Server

Class Name: News Widget	
Responsibility:	**Collaborators:**
• Interact with the News Server to get updated news data in accordance with the widgets' settings • Interact with News Interface to display the news contents. • Interact with the News Interface to get user inputted widget settings • Interact with the News Server to store updated settings	• News Interface • News Server

Table 7: CRC for News Widget

Class Name: News Interface	
Responsibility:	**Collaborators:**
• Display news data in accordance with the widgets' settings • Allow users to modify widget settings • Enables users to choose news topics • Interact with News Widget to send updated settings	• News Widget

Table 8: CRC for News Interface

Class Name: News Server	
Responsibility:	**Collaborators:**
• Store News data • Acquires updated News data • Stores widget settings	N/A

Table 9: CRC for News Server

Class Name: Calendar Widget	
Responsibility:	**Collaborators:**
• Acquires Calendar information from the Calendar Server in accordance with the widgets' settings • Interact with Calendar Interface to display the widget contents. • Interact with the Calendar Interface to get user inputted widget settings • Interact with the Calendar Interface to get user input • Interact with the Calendar Server to store changes to calendar • Interact with the Calendar Server to store widget settings	• Calendar Interface • Calendar Server

Table 10: CRC for Calendar Widget

Class Name: Calendar Interface	
Responsibility:	**Collaborators:**
• Display calendar data in accordance with the widgets' settings • Allow users to modify widget settings • Allow user to add events to their calendar • Allow user to toggle between calendar views • Interact with Calendar Widget to send updated settings	• Calendar Widget

Table 11: CRC for Calendar Interface

Class Name: Calendar Server	
Responsibility:	**Collaborators:**
• Store Calendar data • Acquires updated Calendar data • Stores widget settings	N/A

Table 12: CRC for Calendar Server

Class Name: Weather Widget	
Responsibility:	**Collaborators:**
• Interact with the Weather Server to get updated weather data in corresponding to the widgets' settings • Interact with Weather Interface to display the widget contents. • Interact with the Weather Interface to get user inputted widget settings • Interact with the Weather Server to store updated settings	• Weather Interface • Weather Server

Table 13: CRC for Weather Widget

Class Name: Weather Interface	
Responsibility:	**Collaborators:**
• Display weather data corresponding to the widgets' settings • Allow users to modify weather location • Allows users to toggle between various weather data • Interact with Weather Widget to send updated settings	• Weather Widget

Table 14: CRC for Weather Interface

Class Name: Weather Server	
Responsibility:	**Collaborators:**
• Store weather data. • Acquires updated weather data • Stores widget settings	N/A

Table 15: CRC for Weather Server

Class Name: Stock Widget	
Responsibility:	**Collaborators:**
• Interact with the Stock Server to get updated stock data in accordance with the widget settings • Interact with Stock Interface to display the stock data. • Interact with the Stock Interface to get user inputted stock settings • Interact with the Stock Server to store users' updated stock information	• Stock Interface • Stock Server

Table 16: CRC for Stock Widget

Class Name: Stock Interface	
Responsibility:	**Collaborators:**
• Display stock data corresponding to the widgets' settings • Allow users to select which stocks they are following • Allow users to select different views that contain varying stock information • Interact with Stock Widget to send updated settings	• Stock Widget

Table 17: CRC for Stock Interface

Class Name: Stock Server	
Responsibility:	**Collaborators:**
• Store Stock data. • Acquires updated stock information • Stores user stock preferences	N/A

Table 18: CRC for Stock Server

Class Name: Sticky Note Widget	
Responsibility:	**Collaborators:**
• Interact with the Sticky Note Server to retrieve sticky note data and settings • Interact with Sticky Note Interface to display users' sticky notes. • Interact with the Sticky Notes Interface to get user inputted settings • Interact with the Sticky Note Server to store updated notes	• Sticky Note Interface • Sticky Note Server

Table 19: CRC for Sticky Note Widget

Class Name: Sticky Note Interface	
Responsibility:	**Collaborators:**
• Display user created sticky notes with their corresponding settings • Allow users to modify the text on sticky notes • Interact with Sticky Note Widget to save updated sticky notes	• Sticky Note Widget

Table 20: CRC for Sticky Note Interface

Class Name: Sticky Note Server	
Responsibility:	**Collaborators:**
• Stores sticky note data • Stores sticky note settings	N/A

Table 21: CRC for Sticky Note Server

A Division of Labour

This document was created through a collaborative effort by the whole group, exchanging ideas and filling in sections together. Meetings were held regularly to iterate on this document until an acceptable final version was achieved. By signing this document, the group members certify that this division of labor is fair and accurate.

A.1 Signatures

Matthew Paulin	Date
Hargun Bedi	Date
Dylan Smith	Date
Chenwei Song	Date
Tianzheng Mai	Date