Names	Lai Yongquan	Yeow Kai Yao	Jessica Ho Chek Seen	Joseph Goh Seow Meng
Matric No	A0080909Y	A0086826R	A0085082E	A0087072A

CS3213 Assignment 3 - Team 11

Code repository URL: ssh://cs3213@cs3213.joeir.net:7294/~
Password: cs3213!!

Introduction

Our Visual IDE is intended to teach users (primarily young children) basic coding skills.

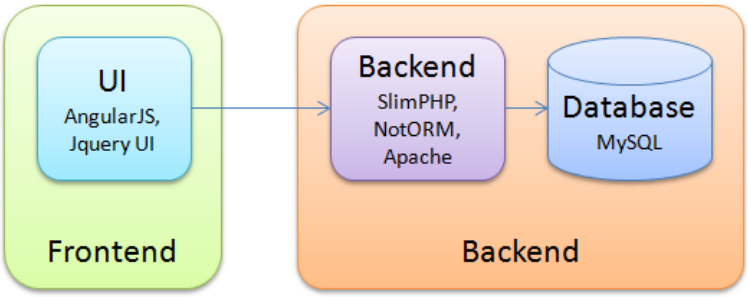
The IDE contains 3 main parts, with the menu on the left, display panel in the centre and draggable code blocks on the right. Basic instructions, mission objectives, and a help section can be found in the menu. The display panel shows the output of the user's code blocks, which are assembled on the right. In order for the program to run, one can drag instructions labeled under "actions" into the grey box under "workspace", then press play to run.

Software and Tools used

Client-side framework: Angular.js (https://angularjs.org/)
Client-side libraries: JQuery UI (http://jqueryui.com/)
Server-side framework: Slim PHP Microframework (http://www.slimframework.com/)
Data Persistence: MySQL (http://www.mysql.com/)
Data Access Middleware: NotORM (http://www.notorm.com/)
User Identification: Google OAuth 2 via Opauth Authentication Framework (http://opauth.org/)

Architecture

VisualIDE is a single-page web app, having a frontend and backend as shown below:



Frontend

Angular encourages a component-based MVC architecture for client-side development. Therefore, our client side is also divided into components. Each of the components in turn follows an MVC architecture. Some of the more important components are described below.

Authentication

Authentication is based on the OAuth 2.0 protocol, with Google as the OAuth provider. Implementation is done using the Opauth framework, which includes pre-defined strategies for authenticating via various OAuth providers.

Upon invocation, users are directed via a GET call to Google's login screen. On successful authentication, Google returns a code to request access to our users' account information via another

HTTPS GET call. The Opauth framework abstracts the collection of data from Google and places it in the global PHP $_SESSION variable where we can access it throughout our application thereafter.

For VisualIDE, only profile and contact information are requested. The users' Google account's unique identifier is stored locally and used as reference for all the user's saved programs.

Character and Background

The character component is an example of one of many components implemented following a Template View pattern, which is encouraged by Angular. A template (located in templates/character.php) contains placeholders e.g. {{ current.name }}, which are filled in with data from characterService (the model). In order to move the character, for example, the controller will modify the position in the model, which in turn causes Angular to re-render the template and update the view.

The background is handled in a similar way. There is a reference to a background image and a scale factor in the model, and the template is re-rendered when the model changes.

Commands

The drag-and-drop functionality is implemented with JQuery UI. A view holds all the elements required for the commands area: the "toolbox" (where users can see the blocks available and drag blocks out of), the "workspace" (where users assemble the blocks) and a trash can.

After each block is dropped or value changed, the controller updates the underlying variable (the model), which holds a Javascript array of objects representing the blocks in the workspace. We chose to have our model as a single source of truth, thus avoiding having multiple instance of data which might be accidentally referred to. The implementation was done by placing all our data within the $rootScope so that it can be accessed via calls from other controllers. A Publisher-Subscriber design within controllers was then implemented via $scope.$watch so that when the model changes, the blocks in the workspace will be updated to reflect this change.

Command Execution

For execution of the user's code, our design resembles the Command Pattern. The blocks that the users drag and drop into the workspace are translated into command objects. These command objects are then passed to the Command Processor which interprets and schedules when the commands are to be executed. This eases the implementation of more complex commands such as "repeat".

Backend

API Endpoints

We have chosen the open-source Slim Micro-Framework for the backend because of its minimalist implementation as opposed to more complete frameworks such as CodeIgnitor and Yii which would have made our codebase unnecessarily large. Being minimal in nature, Slim also enforces less structure, allowing us greater flexibility in defining our directory hierarchy and routes.

Since our application is single paged, the use of the Slim micro-framework is primarily as a front controller, defining paths to API endpoints which our user interface communicates with to perform CRUD operations.

The use of Slim also reduced the amount of code we had to write by bootstrapping our routing mechanisms. It also enabled features like custom Page Not Found routes, preventing unnecessary disclosure of the back-end infrastructure.

Page Templating

Slim also has a flexible inbuilt Template View implementation which we used as our page-controller by injecting base PHP code into calls to our API such as conditional inclusion of data access objects, user login status verifiers and other conditional functions, in line with the DRY principle.

Data Persistence

MySQL is used to provide data persistence, for storing users' saved programs.

Data Access Middleware

NotORM is used as an abstraction layer for us to build upon rather than using the default *mysqli* PHP object, as it hides complexities such as having to properly escape SQL, as well as manual handling of MySQL errors and proper formatting of data retrieved from the database.

Benefits

Low Coupling

Applying MVC with the subcomponents has allowed us to easily re-organize our user interface with minimal code changes. This allowed us to easily experiment with component placement while we tried to find a layout that a user would be comfortable with. Low coupling also gave us the ability to properly define our roles in development, allowing us to write code independently of each other after we have defined common interfaces.

Separation of Concerns (SoC)

In line with the SoC principle, AngularJS separates front-end code into the familiar Model-View-Controller pattern with it's concepts of Services, Directives and Controllers respectively.

SoC also promotes lower coupling in our code. As mentioned earlier, commandProcessor only contains code involved in executing user input, while characterService is only focused on providing support for the manipulation of the character's position, sprite and movement. SoC also has also allowed work to be distributed more easily to different team members as each member only has to focus on different aspect of the application, such that different members can work simultaneously.

This also facilitates testing as each component can be tested largely independently of each other. For example, the commandProcessor only needs to generate the correct schedule of when each block is to be executed and run it, while the characterService can be tested with hard-coded stubs to ensure that the functions are working.

Highly Future Proof

Since Angular imposes modularity, the sub-components of our system have been modularized based on their role and function. This would facilitate future addition of more functions by simply adding more modules.

On top of that, using the Command Pattern has allowed us for increased flexibility within the Command Processor itself. This allows for expansion and additional functionality (commands) with minimal effort.

Limitations

Cross-platform compatibility & feasibility on mobile devices

Due to client-side differences and time constraints, there are some issues with drag-and-drop behaviours on mobile devices. The small screen size of phones also makes user experience highly questionable.