Module 3 Journal

Hai Nguyen

CS 499 Computer Science Capstone

Professor Brook Goggin

08 Nov 2023

## Part One:

ePortfolios are digital portfolios that can serve multiple purposes. In this context, they are a means to compile and present a body of work or evidence of someone's qualifications. In fact, ePortfolios offer a valuable avenue for self-promotion and marketing, playing a pivotal role in establishing one's capabilities, knowledge, and skills. These digital portfolios enable individuals to show their body of work in an organized and visually appealing manner, presenting concrete evidence of their qualifications. Whether aiming to facilitate a career change or secure new employment opportunities, ePortfolios provides a powerful platform to impress potential employers by highlighting achievements and competencies. A well-structured ePortfolio contributes to a professional online image, allowing individuals to effectively market themselves in today's digital age.

However, along with the benefits come risks, including content ownership, content accuracy, and the potential exposure of intellectual property and sensitive information to public consumption. To mitigate these risks while maximizing the marketing potential of ePortfolios, individuals must exercise caution. Careful content selection is crucial, with an emphasis on excluding sensitive, proprietary, or confidential materials. Privacy settings on ePortfolios platforms should be adjusted to control who can access the content, safeguarding privacy and sensitive data. Additionally, maintaining transparency in the presentation of qualifications and experiences is essential to avoid misrepresentation. The owners should regularly update and review their ePortfolios to ensure accuracy, remove outdated content, and consider the sensitivity and ownership of the materials they show. Furthermore, understanding the privacy and security features of the platform they use can help protect their data and reputation.

## Part Two:

### Status Checkpoints Artifact 1

* In the ClickedItemActivity.java, I aim to extend the functionality of the app by implementing innovative features or functionalities. This may involve using advanced mobile programming techniques, frameworks, or libraries to enhance the user experience or provide additional value to users. (Course Outcome 3)
* Practical Enhancements for ClickedItemActivity.java:
* Implement a feature that allows users to interact with the displayed item, such as adding comments or reviews.
* Enhance the user interface with more dynamic and interactive elements, like animations or gestures.
* Provide options for users to customize the displayed item's details or appearance based on their preferences. In this case, users can choose to display item’s text size (small, medium, or large).
* Original code:

ClickedItemActivity.java:

package com.zybooks.projecttwohainguyenui;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;

import android.os.Bundle;

import android.widget.EditText;

import android.widget.ImageView;

import android.widget.TextView;

public class ClickedItemActivity extends AppCompatActivity {

ImageView imageView;

TextView textView;

EditText editText;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_clicked\_item);

imageView = findViewById(R.id.imageView);

textView = findViewById(R.id.tvName);

editText = findViewById(R.id.inputText);

Intent intent = getIntent();

if (intent.getExtras() != null) {

String selectedName = intent.getStringExtra("name");

int selectedImage = intent.getIntExtra("image", 0);

textView.setText(selectedName);

imageView.setImageResource(selectedImage);

}

}

}

Modified code with 3 enhancements: ClickedItemActivityModified.java.

With 3 modifications, the ClickedItemActivityModified will now allow users to add comments, provide dynamic animations, and potentially customize the displayed item's details.

“package com.zybooks.projecttwohainguyenui;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;

import android.os.Bundle;

import android.view.View;

import android.widget.AdapterView;

import android.widget.ArrayAdapter;

import android.widget.EditText;

import android.widget.ImageView;

import android.widget.Spinner;

import android.widget.TextView;

public class ClickedItemActivity extends AppCompatActivity {

ImageView imageView;

TextView textView;

EditText editText;

Spinner textSizeSpinner; // Added Spinner for text size selection

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_clicked\_item);

imageView = findViewById(R.id.imageView);

textView = findViewById(R.id.tvName);

editText = findViewById(R.id.inputText);

textSizeSpinner = findViewById(R.id.textSizeSpinner); // Initialize spinner from layout

Intent intent = getIntent();

if (intent.getExtras() != null) {

String selectedName = intent.getStringExtra("name");

int selectedImage = intent.getIntExtra("image", 0);

textView.setText(selectedName);

imageView.setImageResource(selectedImage);

// Enhancement 1: Allowing users to add comments or reviews

editText.setOnFocusChangeListener(new View.OnFocusChangeListener() {

@Override

public void onFocusChange(View view, boolean hasFocus) {

if (!hasFocus) {

String userComment = editText.getText().toString();

// Save the comment to data structure or send it to a server

}

}

});

// Enhancement 2: Adding animations such as fade in to the ImageView

imageView.animate().alpha(1.0f).setDuration(1000);

// Enhancement 3: Allowing users to customize the displayed item's details

// Provide options to change the text size based on user preferences.

// Set up the spinner with predefined text size options

ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(

this,

R.array.text\_size\_options,

android.R.layout.simple\_spinner\_item

);

adapter.setDropDownViewResource(android.R.layout.simple\_spinner\_dropdown\_item);

textSizeSpinner.setAdapter(adapter);

// Handle spinner item selection to change text size

textSizeSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {

@Override

public void onItemSelected(AdapterView<?> adapterView, View view, int position, long id) {

// Get the selected text size from the spinner

String selectedTextSize = adapterView.getItemAtPosition(position).toString();

// Set the text size based on the selected option

switch (selectedTextSize) {

case "Small":

textView.setTextSize(14);

break;

case "Medium":

textView.setTextSize(18);

break;

case "Large":

textView.setTextSize(24);

break;

}

}

@Override

public void onNothingSelected(AdapterView<?> adapterView) {

// Do nothing if nothing is selected

}

});

} } }”

|  |  |  |
| --- | --- | --- |
| 1 | Artifact 1: Software Design & Engineering  Weigh Tracking App |  |
| 2 | Working on Initial Enhancements |  |
| 3 | Submitted, Awaiting Instructor Feedback |  |
| 4 | Working on Final Enhancements |  |
| 5 | Upload to ePortfolio |  |
| 6 | Finalized ePortfolio |  |

### Status Checkpoints Artifact 2

* Practical Enhancement: Further optimization and evaluation of sorting algorithms, particularly quicksort, to improve their efficiency and performance.
* Course Outcome Alignment: This enhancement aligns with Course Outcome 1, as it aims to demonstrate proficiency in sorting algorithms and their optimization, which can support diverse audiences in making informed decisions by enhancing computer system and application performance.
* Original code:

void quickSort(vector<Bid>& bids, int begin, int end) {

int mid = 0;

//if zero or one bid to sort, then done

if (begin >=end) {

return;

}

// partition bids into low and high parts

mid = partition(bids, begin, end);

//recursively call quicksort using midpoint value (begin to end)

quickSort(bids, begin, end);

//recursively call quicksort using midpoint value (mid + 1, end)

quickSort(bids, mid + 1, end);

}

* Modified code: Switch to a different sorting algorithm, insertion sort when the partition size becomes small (< 10). The idea is that insertion sort can be more efficient than quicksort for small datasets due to its lower overhead. Overall, this optimization aims to reduce the overhead of recursive calls in quicksort for small partitions, contributing to better performance in scenarios where quicksort may be less efficient due to the associated function call overhead.

void quickSort(vector<Bid>& bids, int begin, int end) {

int mid = 0;

// Optimization: Use insertion sort for small partitions

if (end - begin + 1 <= 10) {

// Insertion sort implementation

for (int i = begin + 1; i <= end; ++i) {

Bid key = bids[i];

int j = i - 1;

// Move elements of bids[begin..i-1] that are greater than key.title

// to one position ahead of their current position

while (j >= begin && bids[j].title.compare(key.title) > 0) {

bids[j + 1] = bids[j];

--j;

}

// Insert the key into the appropriate position

bids[j + 1] = key;

}

} else {

// Continue with quicksort for larger partitions

mid = partition(bids, begin, end);

quickSort(bids, begin, end);

quickSort(bids, mid + 1, end);

}

}

|  |  |  |
| --- | --- | --- |
| 1 | Artifact 2: Data Structures and Algorithms  Sorting Algorithms in VectorSorting.cpp |  |
| 2 | Working on Initial Enhancements |  |
| 3 | Submitted, Awaiting Instructor Feedback |  |
| 4 | Working on Final Enhancements |  |
| 5 | Upload to ePortfolio |  |
| 6 | Finalized ePortfolio |  |

### Status Checkpoints Artifact 3

* For the ProjectTwoDashboard.ipynb artifact, I can align practical enhancements and intended course outcome 3: Building a full-stack application using different programming languages (e.g., Node.js for the back-end), demonstrate the ability to work with multiple programming languages to create a full-stack application.
* Implement a backend using Node.js, Express, and MongoDB to handle data storage and retrieval.
* Create RESTful API endpoints in Node.js to communicate with MongoDB, allowing the Dash application to fetch and update data.
* Adjust the current MongoDB operations in your Python code to utilize the API endpoints provided by your Node.js backend.
* Ensure that the Dash application interacts with MongoDB through your Node.js server rather than directly.
* Set up communication between your Dash application and the Node.js backend. Dash can make HTTP requests to the Node.js server to fetch data or send updates.
* This outcome aligns well with the development of a data dashboard using Dash, which involves integrating Python (Dash) for the front-end with MongoDB as a NoSQL database on the back-end.

|  |  |  |
| --- | --- | --- |
| 1 | Artifact 3: Databases  International Animal Shelter Project  ProjectTwoDashboard.ipynb |  |
| 2 | Working on Initial Enhancements |  |
| 3 | Submitted, Awaiting Instructor Feedback |  |
| 4 | Working on Final Enhancements |  |
| 5 | Upload to ePortfolio |  |
| 6 | Finalized ePortfolio |  |