James Heikkinen

Project Documentation:

Warren Buffy – Financial Assistant

A cat sitting on a table

Description automatically generated

My cat Buffy who provides financial advice.

# Executive Summary

Project Warren Buffy is a robust financial application that will enable users to utilize various tools to enhance their financial status in life. The average person does not receive a proper education on how to handle their finances and set themselves up to successfully increase their wealth. Project Warren Buffy seeks to repair this knowledge gap by providing the necessary tools to improve the average user’s life. The tools provided by Warren Buffy will be the following:

* Investment calculator to identify a portfolio’s growth over a specified period.
* Investment calculator for investment sizing for portfolio risk management.
* Calculator to determine the mortgage payment and display a chart to pay off the mortgage.
* Platform for the user to develop a budget based on their net income.

This application is beneficial for all users that range from novices to experienced professionals that are looking towards improving their finances. The design of the application will be able to provide the tools to the users but will also pass on the knowledge of various financial concepts that are critical for personal financial success.

During the use of the application, users will be able to identify if they can afford a specific house by using the mortgage calculator and comparing it to the budget they have created. The budget platform will give users the ability to save their income and their expenses to determine if they have a budget surplus or deficit. Which will provide them with insight as to how they can change their spending habits so they can maintain an affordable lifestyle and have the ability to work on building up their wealth. As the user’s lifestyle may change such spending habits or change in income, they will be able to adjust their current budget to see what their financial status looks like. Overall, the Warren Buffy project will provide a software platform for people who decide to improve their financial well-being through the use of our various wealth enhancing tools.

# Budget Functionality | Program Usage Description

The budget functionality will display the user’s current budget information that is stored in the database. To update the budget information, the user will need to click on the edit budget button to fill out the budget form.

A screenshot of a computer

Description automatically generated

# Portfolio Growth Calculator | Program Usage Description

The portfolio growth calculator will require the user to input information to calculate their projected growth for their portfolio. The information required is the portfolio amount, number of years for the investment to grow, the monthly contribution to the portfolio, and the interest rate that he portfolio grows. Once the fields are filled out, the user will need to click on the Calculate button to get the results.

A purple and white calculator

Description automatically generated

# Investment Risk Calculator | Program Usage Description

The investment risk calculator will require the user to input information to calculate the position size based on their risk tolerance. The fields required to calculate the risk are the portfolio amount, unit price at purchase, price to get out of investment if a loss were to occur, and the risk willing to take on a portfolio. Once the fields are completed, the user needs to hit the calculate button to get results.

A screenshot of a calculator

Description automatically generated

# Mortgage Payment Calculator | Program Usage Description

The mortgage payment calculator is a tool that the user can utilize to find out their potential mortgage payment. The field that need to be filled in are the property value, property tax, insurance for the home, any down payment to reduce the principal, interest rate of the loan, and the time to pay off the mortgage. Once the fields are completed, the user can select the calculate button to get results.

A purple screen shot of a calculator

Description automatically generated

# System Architecture

## 

## Source Code Structure

Source code structure introduction. The following is a summary of the source code directories and their contents:

|  |  |
| --- | --- |
| **Code Directory** | |
| **Directory** | **Usage** |
| C:\Users\{WindowsID}\WarrenBuffy | manage.py is the main file that the Python interpreter will run to execute the whole server functionality for the web application.  Contains the directory for the WarrenBuffy application and the myenv environment directory to run the settings of the application. |
| C:\Users\{WindowsID}\WarrenBuffy\ WarrenBuffy | Contains the directories for the migrations, models, static files, and templates.  admin.py is a file that requires registering models to be used via the admin feature of the Django app.  settings.py is where you can configure the application’s settings such as identifying where specific files are and connecting to a database via username and password.  urls.py is used to develop all the URLs needed for each web page view.  views.py is a file that contains functions that will run requests to pull data from the database and render it with the applicable html document. |
| C:\Users\{WindowsID}\WarrenBuffy\ WarrenBuffy\migrations | The files in this directory are important for integrating models with the database. It is the function that utilizes the Object-Relational Mapper (ORM) to add the necessary tables and columns to allow the application to function. Basically, building the database for you based on the models created. |
| C:\Users\{WindowsID}\WarrenBuffy\ WarrenBuffy\models | Each file in this directory play a function as a form or a model that will write and update data in the database for the user. |
| C:\Users\{WindowsID}\WarrenBuffy\ WarrenBuffy\static | This directory contains the files for the CSS, JavaScript, and pictures that are necessary for the application’s look and functionality. |
| C:\Users\{WindowsID}\WarrenBuffy\ WarrenBuffy\templates | This directory contains the files for all the html documents that are needed to run the Warren Buffy application. |
|  | |

# Executables

Describe the executables that are a part of this project. If there are multiple executables break out each one into its own heading and give its name and a description of what function it plays in the system.

### Python (python.exe)

This is the only executable that is needed to run the entire Django platform and the Warren Buffy application. Essentially, this is the Python interpreter that will execute all the python code that is necessary for the application to run. This includes the manage.py, settings.py, models, views, and many more functions of the Django platform.

# Code Architecture

Explain how the system is put together. Provide an introduction and then dig into the database or data store design and then explore the internals of the code.

Django platform develops database design using the Object-Relational Mapper (ORM) functionality that works with the models that are created by the developer (Django Software Foundation, n.d.). There are three main models that are used for the database. The auth\_user table is based off of the Django platform’s default user model. The userInfo table is a model that extends the user model and adds contact information to the user’s profile. Then the third table to mention is called budgetItems and this table contains all the columns that are specific budget item amounts to support the budget functionality. Overall, each user will have their own budget to work with and update as they see fit.

For the Warren Buffy application, the PostgreSQL database was used with the Django platform. Based on how Warren Buffy is configured, it can use other databases as well including the default (SQLite) one that comes with Django (Django Software Foundation, n.d.). Which leaves it with many options for developers to choose from.

## Database or Data Store

Introduction to the database or data store.

A screenshot of a computer

Description automatically generated

Programming/Coding Language

Provide details about what language the project is written in. – Be sure to identify any components, references and/or external DLLS.

**Overview –** The project is written in several different languages that provide support for the whole application. The different languages that this project consists of are HTML, CSS, JavaScript, Python, and SQL. With all these languages, this application is considered full stack based on having front-end and back-end functions.

**HTML –** This language is used to develop the pages of the Warren Buffy application. There are a total of 15 HTML documents called templates that are used to display the application’s content. They provide an interface for login, user registration, user account settings, budget functionality, and calculator functionalities. There is a base.html file that provides reusable code for the remaining HTML files. It essentially covers the HTML document essentials like header and footer. The rest of the HTML files are extended versions of the base.html.

**CSS –** CSS is code that makes the application look “pretty” and adds some elements of usability for the user. Essentially, it applies to all HTML documents throughout the application.

**JavaScript –** JavaScript is used for the calculator functionalities for the investment risk, mortgage, and portfolio growth calculators. These files contain functions that are reusable on the web page in case the user needs to make changes to the input to get different results. Each file contains calculations that manipulate the user’s inputs and provides a result onto the page for display. When the user recalculates, the results will update based on the changed inputs.

**Python –** Python is the major player in this application. The Django framework functions off of the Python programming language. This language provides support for models, Object-Relational Mappers (ORM), user functionality, views, and many more. Models are classes that provide the functionality for the structure of the database and tables. The variables of the model in general will be used as columns for defining the table. Based on the way the models are structured, the Django platform will use its ORM functionality to take that model and develop the tables necessary for the application’s database structure. Django also uses python to support the user account functionality where it contains the basic user table information for a developer to get started. In addition, there is an admin interface where users can be managed. The last area that Python is heavily used in this project is the views for rendering webpages via HTML documents with the data pulled from the models. These views render the page as functions and are critical for displaying the right content on the right page.

**SQL –** Since Django handles most of the database development through its ORM functionality, SQL is generally used to run queries and make any changes to the database to support the project’s development.

Project Classes

Classes within the project are used to abstract re-usable pieces of code. Classes are also used to group related values, known as properties. The project utilizes these classes:

### Class budgetItems | userInfo.py

The budgetItems class is a model that is used to display and alter budget information for the application’s budget functionality. This class also includes a function that will calculate the surplus or deficit of the user’s budget. Based on the class name and the variables in it, Django will create a database table called budgetItems with the applicable columns.

### Class userInfo | userInfo.py

The userInfo class is a model that extends onto the user model that Django uses. It adds more contact information such as phone number, address, city, state, and zip code. Which develops into a table based on how this model is set up.

### Class Meta | userInfo.py

Meta classes are used to reference the correct application and label the table for both the budgetItems and userInfo classes.

### Class UserContactForm | updateUserInfo.py

This class utilizes the User model from Django and develops a form for the user to input their information. This class works with the UserInfoForm class to put together a full form for the user to update their personal information. The functionality for this form is located in the application’s user profile settings.

### Class UserInfoForm | updateUserInfo.py

This class is the second part of the form that works with the UserContactForm class. It works off of the userInfo model and allows the user to update their contact info in the application’s user profile settings. The clean\_zipcode function is used to validate the zip code to ensure it is a number. The table column is set to varchar to retain the 0 at the beginning of the number.

### Class UpdatePassword | updateUserInfo.py

The UpdatePassword class is a separate form under the user profile settings section of the application. It will validate the old password and ensure that the user has the right password by having them submit it twice.

### Class Meta| updateUserInfo.py

Meta class is used to identify the applicable fields in the tables to change. Also, the label function allows the developer to label the form fields as they see fit. In addition, the Meta class is used to identify the model to work off of. This class is used for the UserContactForm and UserInfoFrom classes.

### Class registration | registration.py

Registration class works off of the User model to allow the user to input their information to sign up at the index.html page. This class works with the userInfoRegistration class to provide a registration form for the user.

### Class userInfoRegistration | registration.py

UserInfoRegistration class works with the registration class to allow the user to submit information into the registration form. This class works off of the userInfo model to get the extra contact information from the user.

### Class Meta | registration.py

The Meta classes are used to work off of the User and userInfo models to put together the registration form fields. They also pull the fields necessary to display on the form itself using the fields variable.

### Class budgetItemsForm | forms.py

The budgetItemsForm is a class that allows for the user to edit the budget item amounts for each line item of the budget functionality. It also contains a function that ensures that no negative values are entered into the form fields.

### Class Meta | forms.py

The Meta class for the budgetItemsForm model works off of the budgetItems model. Then it identifies the fields needed for the form and uses labels to make it more user friendly.

Project Modules

Modules are used for procedural based code that does not require state data like class modules do. Complete the introduction to modules.

### Views Module | views.py

This module contains fourteen functions that will render each page of the Warren Buffy application. Which these functions cover the registration, login, budget, calculators, and profile setting functionalities. Some of the views will pull in data from a model to apply it to the applicable HTML page.

### investmentRisk Module | investmentRisk.js

This module is a reusable function that will take user input and calculate the risk that should be taken on a specific investment. It will factor in the portfolio size and the price of the investment per unit and let the user know how many units to purchase based on the amount willing to lose on the investment.

### mortgageCalculator Module | mortgageCalculator.js

The mortgageCalculator module takes user input such as property value, taxes, interest rates, and insurance to display the mortgage payment that a homeowner would have to pay. This is a single function that can be used for the user to recalculate their payments anytime.

### portfoliogrowth Module| portfoliogrowth.js

This module is a basic calculation of potential portfolio growth and gives the user the option to add contributions to see how much their wealth can be impacted. This function can be reused at any time and will display the projected amount to the user.

Program Start and End Flow

The Warren Buffy application starts with entering in the URL in the web browser to access the index page. Then the user can go to the registration form to sign up for a user account and login if they have created a user account. Once the user logs in, they get to choose between the budget functionality card or the financial calculators card. The budget management page will display the user’s budget information and the user will have the option to click the edit button to edit the information to reflect their current situation. The financial calculators page will consist of the investment risk, portfolio growth, and mortgage loan calculators.

Once the user is logged in, each page will consist of a logout and user profile functionality. The logout button will redirect the user to the login page. The user account button will lead to the profile settings page. When the user gets to the profile settings page, they will be able to change their contact information and password as they see fit. To identify the logout and user profile functionality in the program diagram, I used the symbols \* to indicate logout and # to indicate the user profile settings page.

A diagram of a company

Description automatically generated

# Test Plan

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Details** | **Results** | **Action to correct (If Applicable)** |
| All Input Fields | Test the input by submitting symbols, letters, and numbers. Verify the right data type is recived and the wrong ones are rejected.  Test the data boundaries if they are numeric or password related. Example can be ensuring that the budget items don't receive negative numbers. | Zip code field takes in letters.  Budget form accepts negative values.  Calculators accept negative numbers. Need to add exception handling for negative numbers. | Added exceptions handling for the zip code to be a number only.   Added error handling for the budget for m to reject negative numbers.  Added exceptions handling to the calculators to reject negative numbers |
| Verify User Registration | Fill out user form and verify data is added to database.   Verify user budget info is defaulted to 0 for all budget items.  Verify form will not submit when any form fields are empty. | duplicate key value violates unique constraint "budgetItems\_pkey" DETAIL: Key (id)=(1) already exists.  Error when hitting submit button.  After registering, just goes to thank you for registering page. Will need to lead user to login page.  Budget default info added 0 for all items. | Updated the admin information to prevent conflict. When resetting the migrations it was essentially starting over with the data and I did not remove some parts of the admin account information. Conflict with data won't happen anymore.  Added a redirect to go to the login page when the user registers the page. |
| Verify User Login | Verify that you are the correct user that is logged in by verifying the user accoun information.  Log out and log in as another user to verify the information is different and matches for that user. | Correct information has popped up for the logged in user.  Logged in as another user, information does not spill over. Each user had the correct information. |  |
| Change Profile Contact Information | Verify user contact information appears in the form.  Change data in the form fields and see if the fields are updated after submitting the form. | User contact information does appear.  Changed information in the form and the changes have updated. |  |
| Change Password | Change the password by using the form fields.   Verify password changed in the database. Then logout and test the changed password. | Changed the password.  Verified password changed. |  |
| Budget Management Page | Verify the the correct budget item amounts are showing for the user.  If all items are 0, then it will be required to edit the budget with some actual data. See Edit budget section. | Correct budget information appears, even after updating the information.  Need to send user back to budget management page after submitting form. | I was able to get the edit budget form to go back to the budget page by redirecting it in the update budget item view. |
| Edit Budget Form | Update the fields in the budget form.  Then verify that the updates took place in the budget management page. The goal is to have the form go back to the budget management page after submission. | I was able to pull data from the database and update the information after submitting the form. |  |
| Portfolio Growth Calculator | Results display for the calculator.   The calcuations are correct. Verify with other known sites to see that the calculation is correct. | Results displayed. However, the results displayed in the footer of the site. Will need to update the CSS code.  Calculations are correct. |  |
| Investment Risk Calculator | Results display for the calculator.   The calcuations are correct. Manually calculate the results. No known sites for comparison. | Results displayed.  Calculations are correct. |  |
| Mortgage Calculator | Results display for the calculator.   The calcuations are correct. Verify with other known sites to see that the calculation is correct. | Results displayed.  Calculations are correct. |  |

# Deployment Plan

A deployment plan that I would use for the Warren Buffy application is the Canary strategy. This strategy will start with a small portion of users testing the software before releasing it to the public (Tremel, 2017). The plan is to test on a select group of users prior to releasing it to the public. This strategy will identify any potential issues before it goes out to the public and by giving us the window necessary to fix these issues. Once the issues have been resolved and testing results are meeting expectations, then the software can be released on the internet for all users to access. Overall, this deployment plan will help prevent security and usability issues.

# Maintenance Plan

The maintenance plan for Warren Buffy will cover password resets, software updates, and inactive user handling. Password resets will consist of contacting the help desk via email to get the password reset. In another scope, the project will contain a more modern method to do it through the web application. But for now, password resets will have to be managed this way. Another maintenance item to consider is software updates where improvements and bug fixes are implemented to increase the quality of the software. Which will be implemented via patches to the backend of the web application and will require proper communication to report downtime. Then the last maintenance item will consist of removing users who have been inactive on the account after two years. No activity at this length of time indicates the user is not taking their financial situation seriously with our application. Therefore, it is critical to remove the users to prevent unnecessary data storage on the server.

# List of Remaining Items (% Complete)

I estimate that this project is about 90% complete. The reason is due to some bugs that need to be fixed and the need to enhance the app’s usability. The following items need to be addressed for the app’s completion:

* Input validation for the user model portion of the user contact information form.
* Write up content on the web application that can help the user make good financial decisions and how to use the tools.
* Add menu functionality for the app.
* Add “breadcrumbs” to the site.
* Enhance the look of the forms.
* Enhance the calculator results display.
* Enhance the look of the budget display.

Summary

This documentation goes over the design of the Warren Buffy application. It contains the system architecture to see where the stored code is located via directories. In addition, it provides the description of the classes and modules and states their purpose for the web application. An ERD is provided for the database that the application uses to store the budget item and user profile information. Which also includes a description of the database functionality. The documentation includes a section to convey the program’s functionality based on how the user interacts with the application. After completion of the software, the testing plan indicates the functionalities that were tested. Once satisfied with testing, deciding the method for deployment is critical for getting the product out to the user. As this product is in production, there will be required maintenance for the life cycle of the Warren Buffy application. Maintenance plays an important role in ensuring the product meets the user’s expectations and that the quality of the software is adhered to. Overall, this documentation is critical for understanding the purpose behind the design of the Warren Buffy application so it can be maintained and improved upon without causing severe issues with the software.

# APPENDIX B (BUILD AND RELEASE PROCESS)

Describe the build & release process required to implement an update.

The build and release implementation will require a process that ensures the integrity of the software while being able to improve it. This process will require a developer, stage, and production environment to implement the stages of the update. The developer environment will be an experimental build of the planned updates for the application. Here the developers can have room for mistakes and have the ability to test out the new functionalities. For the stage environment, the functionality has been generally tested but it needs to be assessed with a copy of the production environment software. In this phase, the developers can test the application to get a more realistic scenario of what that functionality is going to do to the software. Once the testing has been satisfied on the stage environment, the production environment will be updated via patch. To make this happen, the user will need to be notified of the time that the web server will be down for maintenance. Once the process for updating the production server starts, the developers will make a back up copy of the current production environment. Once the copy is complete, the developers will install the patch and run testing to ensure that the patch had installed correctly. After everything passes testing, the web server will go back online for users to access the updated content. Overall, this process will reduce risk for any major issues from occurring.

# APPENDIX C (CLIENT INSTALLATION INSTRUCTIONS)

Detail how a client machine or device is prepared to utilize the project.

Considering this is a web application, the client installation is not applicable. The user would simply type in the URL into a web browser and will need a network connection if the Django platform is not installed locally (context dependent). For the users to access the application via a network, a web server will need to be set up. Which means the web server will need to be configured appropriately based on the domain environment and that will determine the URL that the user has to type into the browser. For local installations, refer to the developer setup instructions. The developer will type in the browser “127.0.0.1:8000” to access the web application. If a user wants to use this for personal means, they can choose to do the local installation.

# APPENDIX D (DEVELOPER SETUP INSTRUCTIONS)

Detail how a developer must setup their environment in order to work on the code.

# Python

1. Download the latest version of Python. (3.12.1) Windows installer (64-bit)
2. Run python executable.
3. Select “Add python.exe to PATH”
4. Click install now via run as administrator.

A screen shot of a computer

Description automatically generated

1. Close window when installation is completed.

A screenshot of a computer

Description automatically generated

# Database

1. Run postgresql-16.2-1-windows-x64.exe as administrator.
2. Select all defaults until superuser password section.
3. Type in your desired password. Make sure you have a complex password for security.
4. Select all defaults and install database software.
5. Type in “pip install psycopg2-binary” in command prompt.

# Django Install

1. Create virtual environment by typing “python -m venv myenv” into command prompt. “myenv” can be named whatever you want.
2. Activate your environment by typing “myenv\Scripts\activate”.
3. Install Django by typing in “py -m pip install Django”.
4. Verify install by typing “django-admin --version”.
5. Type “django-admin startproject myproject” to create project.
6. Change directory by typing “cd myproject”.
7. Create app by typing “python manage.py startapp myapp”

# Configure PostgreSQL in Django

1. Go to pgAdmin 4 on desktop apps.
2. Go to object explorer on the left hand side and right click PostgreSQL 16.
3. Go to create and select database.
4. Name your database and accept the defaults and hit save.
5. Open “settings.py” via text editor.
6. Update the database as follows:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'your\_database\_name',

'USER': 'postgres',

'PASSWORD': 'your\_postgres\_password',

'HOST': 'localhost',

'PORT': '5432',

}

}

**Key Commands Via Command Prompt**

1. “python manage.py makemigrations WarrenBuffy” : used to integrate the models with database.
2. “python manage.py migrate”: implement the migration.
3. “python -m venv myenv”: create environment for application.
4. “myenv\Scripts\activate”: activate the environment to run the application.
5. “python manage.py runserver”: Run the application.
6. “python manage.py collectstatic”: run everytime a new static file such as css, js, or image has been added.

**Create Admin Via Command Prompt**

**Note:** Remember to store them somewhere safe and that you can access when you forget them.

1. python manage.py createsuperuser
2. Create Username
3. Create Password

**Password Encryption Packages**

1. Pip install bcrypt
2. Pip install django[argon2]

**References Used for Project**

Django Software Foundation. (n.d.). How to install Django. *How to install Django | Django documentation | Django*.

Retrieved April 21, 2024, from https://docs.djangoproject.com/en/5.0/topics/install/#database-installation

Django Software Foundation. (n.d.). Making queries. *Making queries | Django documentation | Django*. Retrieved April

21, 2024, from <https://docs.djangoproject.com/en/5.0/topics/db/queries/>

Django Software Foundation. (n.d.). Quick install guide. *Quick install guide | Django documentation | Django*.

Retrieved April 21, 2024, from <https://docs.djangoproject.com/en/5.0/intro/install/>

Django Software Foundation. (n.d.). Writing your first Django app, part 2. *Writing your first Django app, part 2 | Django*

*documentation | Django*. Retrieved April 21, 2024, from <https://docs.djangoproject.com/en/5.0/intro/tutorial02/>

EDB.com. (n.d.). Download PostgreSQL. *Community DL Page*. Retrieved April 21, 2024,

from <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>

Gregorio, F. D., Varrazzo, D., The Psycopg Team, . (n.d.). Installation. *Installation — Psycopg 2.9.9 documentation*.

Retrieved April 21, 2024, from <https://www.psycopg.org/docs/install.html>

Portilla, J. (n.d.). Python and Django Full Stack Web Developer Bootcamp. *Course: Python and Django Full Stack Web*

*Developer Bootcamp | Udemy*. Retrieved April 21, 2024, from <https://www.udemy.com/course/python-and-django-full-stack-web-developer-bootcamp/learn/lecture/6660210#overview>

Portilla, J. (n.d.). Python and Django Full Stack Web Developer Bootcamp. *Course: Python and Django Full Stack Web*

*Developer Bootcamp | Udemy*. Retrieved April 21, 2024, from <https://www.udemy.com/course/python-and->django-full-stack-web-developer-bootcamp/learn/lecture/6660586#overview

Portilla, J. (n.d.). Python and Django Full Stack Web Developer Bootcamp. *Course: Python and Django Full Stack Web*

*Developer Bootcamp | Udemy*. Retrieved April 21, 2024, from <https://www.udemy.com/course/python-and-django-full-stack-web-developer-bootcamp/learn/lecture/6660916#overview>

Tremel, E. (2017, November 21). Six Strategies for Application Deployment. Six Strategies for Application

Deployment - The New Stack. Retrieved April 1, 2024, from https://thenewstack.io/deployment-strategies/

**Images – Creative Commons License**

Youngson, N. (n.d.). Creative Commons Chalkboard Budget Finance Image. *Budget Finance - Free of Charge Creative*

*Commons Chalkboard image*. Retrieved April 21, 2024, from <https://www.picpedia.org/chalkboard/b/budget-finance.html>

Verch, M. (n.d.). Calculator/Calculator. *Calculator - Creative Commons Images*. Retrieved April 21, 2024,

from <https://foto.wuestenigel.com/taschenrechner-calculator/>

svgsilh.com. (n.d.). SVG > mortgage home credit percent. *SVG > mortgage home credit percent - Free SVG Image & Icon.*

*| SVG Silh*. Retrieved April 21, 2024, from <https://svgsilh.com/image/149874.html>

Quoteinspector.com, . (n.d.). Coin stacks and pen on market returns. *Page for individual images • Quoteinspector.com*.

Retrieved April 21, 2024, from <https://www.quoteinspector.com/images/investing/risk-tolerance-writing/>

Quoteinspector.com, . (n.d.). Coin stacks and pen on market returns. *Page for individual images • Quoteinspector.com*.

Retrieved April 21, 2024, from <https://www.quoteinspector.com/images/investing/coin-investment-returns/>