Front-End (Presentation Layer) - Partially done with Claude

- Login page [Luke]
- User choices page: [Luke]
    - choosing up to 5 stocks (out of, say, 20 that we offer)
    - assigning percentages of user's funds allocation
        - E.g. investment in stock A is a% of total funds.
        - Rule: all percentages should add up to 100%
- Analysis page: [Nastia]
    - application outputs the analysis of the user's choice of **stocks** and **percentages**
        - Comparison with S&P 500 in real life
            - Needs real time year-to-date data on S&P 500
            - Needs real time year-to-date data on selected stocks
        - Comparison with inflation index
            - Needs real time year-to-date data on selected stocks
            - Inflation index is fixed
        - Evaluation of risk of the portfolio
            - Needs real time volatility value of S&P 500
            - Needs real time volatility value of selected stocks
    - Shows all of the above to the user
    - Shows the summary score of the analysis (either static or real time, TBD): formula
- Suggestion on the best percentage split [Luke]
    - Improve loss function (only stocks, not inflation or other metrics)
    - **Input:** list of **selected stock** by the user
    - **Output:** the **best way to split the funds** between the chosen stocks
    - [potentially] the final score on this split (i.e. the Analysis page, but now on the suggested percentages, not the user selected ones).

Backend (Application Layer) - Team Implements this with some AI help

- Stock List Processor (Software Development) [Andrew]
- Stock List Analyzer (Software Development) [Nastia]
- Stock List RNN ML Optimizer (Machine Learning) [Luke]

Data Access Layer - Partially done with Claude

- External API Integrations [Nastia]
    - Yahoo Finance: Free Real Time Stock APIS
        - Real Time APIs are either expensive or have low requests/day; will be massive bottleneck for CNN training and testing
        - Use aggressive caching or ML variations to generate many more combinations
- Stock Data Repository Pattern [Andrew]
    - Abstraction Layer between Data Access and Infrastructure

- Translation API to enable communication between the two

Infrastructure [Andrew]

- Git Branches (optional)
    - To ensure members only work on their component to reduce Merge Conflicts

Git Branches Divvied Responsibilities	**Instructions**

# Main branches 	
Main <- merge to here last
Develop <- testing branch
 	
# Andrew's branches 	
feature/infrastructure
feature/repositories
feature/api-client
feature/database
 	
# Nastia's branches 	
feature/analyzer
feature/analysis-page
 	
# Luke's branches 	
feature/cnn-optimizer
feature/ml-suggestions
	List local branches
	- git branch
	List remote branches
	- git branch -r
	
	Switch to a branch
	- git checkout branch_name
	
	Move changes in branch A to branch B
	- git stash
	- git switch correct-branch
	- git stash pop
	
	Merge two branches (feature into main)
	- git switch main
	- git merge feature-branch

# Shared branches feature/login-page feature/user-choices	Delete a branch after fully done - git switch main - git branch -d feature-login - git push origin --delete feature-login Rename a current branch (the name is not reflective) - git branch -m new-name Rename another branch - git branch -m old-name new-name

Essential Idea:

Different branches enable each team member to focus exclusively on their component.

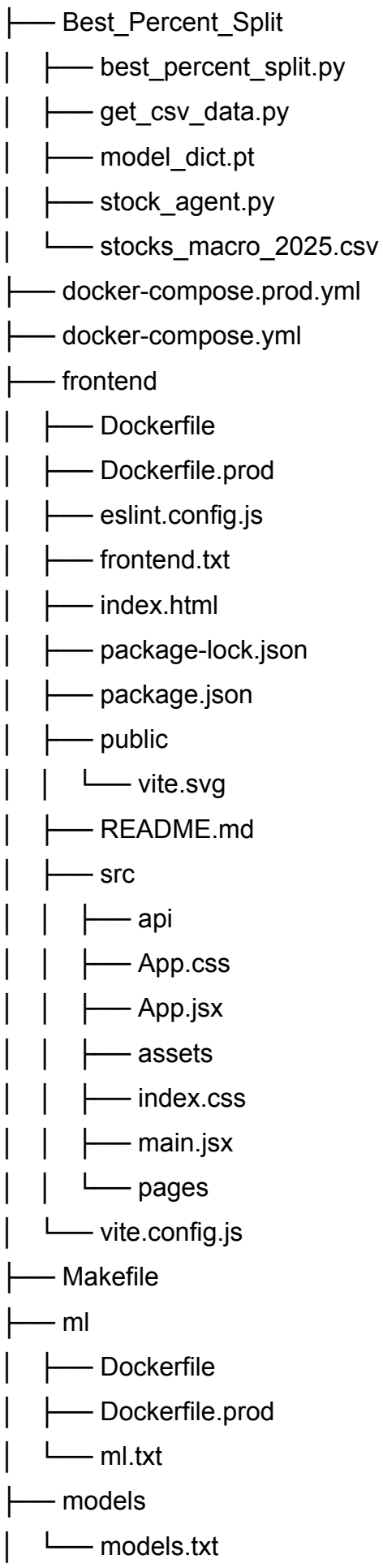
- Git tracks differences in each branch
- If a repo has a current state of K with files and folders, then a branch at that point makes two exact copies of that repo (including the original)
- Each branch commit, push, and pull remains independent of one another (person developing feature A does not need to deal with commits from features B, C, D, and E all coming in)
- Each branch maintains its own history from the point the branching occurred.

Basic Workflow:

- A feature needs to be implemented
- Create a new branch reflective on that feature to be implemented
- Implement the feature on that branch
- Merge the feature branch with the develop branch to add it to the overall system

Tree Project View

```
.
├── ai
│   └── ai.txt
├── backend
│   ├── app.py
│   ├── backend.txt
│   ├── Dockerfile
│   ├── Dockerfile.prod
│   └── requirements.txt
```


├── Best_Percent_Split
│ ├── best_percent_split.py
│ ├── get_csv_data.py
│ ├── model_dict.pt
│ ├── stock_agent.py
│ └── stocks_macro_2025.csv
├── docker-compose.prod.yml
├── docker-compose.yml
├── frontend
│ ├── Dockerfile
│ ├── Dockerfile.prod
│ ├── eslint.config.js
│ ├── frontend.txt
│ ├── index.html
│ ├── package-lock.json
│ ├── package.json
│ ├── public
│ │ └── vite.svg
│ ├── README.md
│ ├── src
│ │ ├── api
│ │ ├── App.css
│ │ ├── App.jsx
│ │ ├── assets
│ │ ├── index.css
│ │ ├── main.jsx
│ │ └── pages
│ └── vite.config.js
├── Makefile
├── ml
│ ├── Dockerfile
│ ├── Dockerfile.prod
│ └── ml.txt
├── models
│ └── models.txt

```
├── quick-start.sh
└── README.md
```