Finance Friends Documentation

Final Deployment

Finance Friends is now live and accessible! You can explore the platform and learn more about the project by **clicking here.**

Introduction

Finance Friends is a revolutionary platform designed to make learning about financial literacy a fun, interactive, and engaging experience for children. By leveraging advanced AI technology alongside a captivating narrative and interactive gameplay, Finance Friends introduces children to fundamental financial concepts and practical knowledge.

Overview

At its core, Finance Friends is a gamified learning platform where children embark on a journey through various thematic locations. Each location is meticulously designed to focus on different aspects of financial literacy, including financial mathematics, credits, savings, investments, taxes, and free conversation. This approach not only makes learning about finance more relatable but also ensures a comprehensive educational experience.

Target Audience

The primary audience for Finance Friends is children who are at the beginning of their educational journey. The platform is designed to be intuitive and user-friendly, making it accessible for children with varying degrees of familiarity with technology and finance.

Through this documentation, you will gain a comprehensive understanding of the platform, its features, and detailed guides on installation, configuration, and usage, along with the API documentation for further customization and integration.

Join us as we embark on this journey to make financial literacy an accessible, enjoyable, and essential part of every child's education.

Key Features

Finance Friends is not just a learning platform; it's an interactive, engaging, and comprehensive journey into financial literacy for children. Here are some of the standout features that make Finance Friends an essential tool for learning:

1. **Comprehensive User Management**

   - **Seamless Registration and Login:** Ensures an easy and secure entry into the world of financial learning.
   - **Personalized User Profiles:** Users have their own profiles, capturing their learning progress, preferences, and achievements, allowing for a tailored learning experience.

2. **Engaging Personalized Experiences**

   - **Custom Welcome Messages:** Users are welcomed with messages that make them feel recognized and valued, setting a positive tone for their learning journey.
   - **Interactive Learning Sessions:** Each session is an adventure, with content tailored to the user's pace and style, making complex financial concepts more understandable and engaging.

3. **Dynamic Learning Modules**

   - **Diverse Financial Topics:** A wide array of modules ensures a well-rounded understanding of financial literacy, from savings and investments to taxes and financial mathematics.
   - **Real-time Interactive Chatbot:** An intelligent chatbot accompanies users, ready to answer questions, clarify doubts, and provide guidance.

4. **Innovative Assessment Tools**

   - **Quizzes and Knowledge Checks:** Regular quizzes and assessments help reinforce learning, ensuring that key concepts are understood and retained.
   - **AI-Powered Quiz Evaluation:** Quizzes are evaluated instantly with AI, providing immediate feedback and explanations, enhancing the learning process.

5. **Motivation and Progress Tracking**

   - **Leaderboards:** Friendly competition is fostered through leaderboards, motivating users to engage more with the content and strive for improvement.
   - **Progress Tracking:** Users can track their learning journey, see their progress in different modules, and set personal learning goals.

Finance Friends combines these features to provide an educational experience that's not just informative but also engaging, interactive, and tailored to each user's needs. The platform ensures that learning about finance is a fun, rewarding, and enriching experience for every child.

Getting Started

This section guides you through setting up your development environment for both the backend and frontend components of Finance Friends.

Backend Setup

Prerequisites Before setting up the backend, ensure you have the following installed: - Node.js - npm (Node Package Manager) - MongoDB (local setup or MongoDB Atlas URI) - Python 3 (with Pip)

Installing Follow these steps to set up your backend development environment:

1. **Clone the Repository**: `git clone https://github.com/FinanceFriend/backend.git`
2. **Navigate to the Directory**: `cd backend`
3. **Install Dependencies**: `npm install`
4. **Set Up Environment Variables**:

- Create a `.env` file in the root of your project.
- Add your MongoDB URI and other necessary environment variables. Example:

```
MONGODB_URI=your_mongodb_uri
OPENAI_API_KEY=your_api_key
PORT=3001
```

5. **Start the Server**: `npm start`
6. **Verify Installation**:

- Open `http://localhost:3001` in your web browser. You should see a confirmation message indicating that the server is running.

Setting Up Python Environment To run the Python script with `langchain`, follow these steps:

1. **Create a Virtual Environment**: `python3 -m venv venv`

- This will create a directory called `venv` in your project folder.

2. **Activate the Virtual Environment**:

- On Windows:

```
venv\Scripts\activate
```

- On macOS and Linux:

```
source venv/bin/activate
```

- Your command prompt should now indicate that you are in a virtual environment.

3. **Install Python Dependencies**: `pip install -r requirements.txt`
4. **Run the Python Script**:

- With the dependencies installed, you can now run your Python script within the virtual environment.

5. **Deactivate the Virtual Environment**:

- Once you're done, you can deactivate the virtual environment by running:

```
deactivate
```

Frontend Setup

This is a Next.js project bootstrapped with `create-next-app`.

Getting Started

1. **Set Up Environment Variables**:

- Rename the .env.example file to .env in the root of your project.
- dd your backend API URI and other necessary environment variables. Example:

```
NEXT_PUBLIC_API_URL=your_api_uri
```

2. **Install Necessary npm Dependencies**: `npm install --force`
3. **Run the Development Server**: `npm run dev` or `yarn dev`
4. **Verify Installation**:

- Open `http://localhost:3000` with your browser to see the result.

Follow these steps to get your backend and frontend up and running. If you encounter any issues, refer to the FAQs or reach out to the support team.

API Documentation

This section provides a comprehensive guide to the API of Finance Friends, detailing the available endpoints, their functionalities, and how to interact with them.

Overview of the API

The Finance Friends API is designed to provide a seamless and secure way to interact with the platform, enabling users to access and manage resources programmatically. It is built with RESTful principles in mind, ensuring a predictable and consistent way to access the platform's data and services.

Backend endpoints

1. **User Registration**

- **Endpoint**: `/api/register`
- **Method**: `POST`
- **Description**: This endpoint is used to create a new user account.
- **Request Body**:
  - `username`: String (required) - The desired username of the user.
  - `email`: String (required) - The user's email address.
  - `password`: String (required) - The user's password.
  - `dateOfBirth`: Date (required) - The user's date of birth.
  - `countryOfOrigin`: String (required) - The user's country of origin.
  - `preferredLanguage`: String (required) - The user's preferred language.
- **Response**:
  - `success`: Boolean - Indicates if the operation was successful.
  - `message`: String - A message describing the outcome.
  - `user`: Object - Contains user information (excluding password).
- **Error Handling**:
  - Returns `400 Bad Request` if the username already exists, the email already exists, or the email format is incorrect.
  - Returns `500 Internal Server Error` for any server-side errors.
- **Security Notes**:
  - Passwords are hashed before being stored.
  - Email validation is performed to check the format.

2. **User Login**

- **Endpoint**: `/api/login`
- **Method**: `POST`
- **Description**: This endpoint is used for user authentication.
- **Request Body**:
  - `login`: String (required) - The user's username or email address.
  - `password`: String (required) - The user's password.
- **Response**:
  - `success`: Boolean - Indicates if the operation was successful.
  - `message`: String - A message describing the outcome.
  - `user`: Object - Contains user information (username and email).
- **Error Handling**:
  - Returns `400 Bad Request` if either field is missing or if the user is not found.
  - Returns `401 Unauthorized` if the password does not match.
  - Returns `500 Internal Server Error` for any server-side errors.
- **Security Notes**:
  - Password verification is performed using bcrypt.

3. **Fetch User Profile**

- **Endpoint**: `/api/user/:username`
- **Method**: `GET`
- **Description**: Fetches the profile details of a specific user by their username.
- **URL Parameters**:
  - `username`: String (required) - The username of the user whose profile is being requested.
- **Response**:
  - `success`: Boolean - Indicates if the operation was successful.
  - `user`: Object - Contains the requested user's information (excluding password).
- **Error Handling**:
  - Returns `404 Not Found` if the user does not exist.
  - Returns `500 Internal Server Error` for any server-side errors.

4. **Fetch All Users**

- **Endpoint**: `/api/users`
- **Method**: `GET`
- **Description**: Retrieves a list of all users. This endpoint is intended for admin use.
- **Response**:
  - `success`: Boolean - Indicates if the operation was successful.
  - `users`: Array - A list of user objects, each containing user information (excluding passwords).
- **Error Handling**:
  - Returns `401 Unauthorized` if the user is not authorized (non-admin).
  - Returns `500 Internal Server Error` for any server-side errors.

5. **Update User Information**

- **Endpoint**: `/api/user/:username`
- **Method**: `PUT`
- **Description**: Allows users to update their account information.
- **URL Parameters**:
  - `username`: String (required) - The current username of the user whose profile is being updated.
- **Request Body** (Any or all of the following):
  - `newUsername`: String (optional) - The new username for the user.
  - `email`: String (optional) - The new email address for the user.
  - `dateOfBirth`: Date (optional) - The new date of birth for the user.
  - `countryOfOrigin`: String (optional) - The new country of origin for the user.
- **Response**:

  - `success`: Boolean - Indicates if the operation was successful.
  - `message`: String - A message describing the outcome.
  - `user`: Object - Contains the updated user's information.
- **Error Handling**:
  - Returns `400 Bad Request` if the new username or email already exists, or if the email format is incorrect.
  - Returns `404 Not Found` if the original user is not found.
  - Returns `500 Internal Server Error` for any server-side errors.
- **Security Notes**:
  - Ensure this endpoint is accessible only to the authenticated user or users with admin privileges.
  - Validate the email format before processing updates.

6. Delete User

- **Endpoint**: `/api/user/:username`
- **Method**: `DELETE`
- **Description**: Deletes a specific user's account.
- **URL Parameters**:
  - `username`: String (required) - The username of the user to be deleted.
- **Response**:
  - `success`: Boolean - Indicates if the operation was successful.
  - `message`: String - A message describing the outcome.
- **Error Handling**:
  - Returns `404 Not Found` if the user does not exist.
  - Returns `500 Internal Server Error` for any server-side errors.

7. Fetch User Stats

- **Endpoint**: `/api/stats/:username`
- **Method**: `GET`
- **Description**: Retrieves the statistics associated with a specific user.
- **URL Parameters**:
  - `username`: String (required) - The username of the user whose stats are being requested.
- **Response**:
  - `success`: Boolean - Indicates if the operation was successful.
  - `data`: Object - Contains the user's statistics.
    - `username`: String - The username of the user.
    - `completionPercentages`: Array of Numbers - An array of completion percentages.
    - `points`: Array of Numbers - An array of points.
    - `correctAnswers`: Number - The count of correct answers.
    - `incorrectAnswers`: Number - The count of incorrect answers.
    - `totalCompletion`: Number - The average of all completion percentages.

        * `totalPoints`: Number - The sum of all points.
        * `correctAnswersPercentage`: Number - The percentage of correct answers.
        * `progress`: Array of Objects - Each object contains `blockId` and `minilessonId` representing the user's progress.
- **Error Handling**:
  - Returns `404 Not Found` if the stats for the given username are not found.
  - Returns `500 Internal Server Error` for any server-side errors.

8. Update User Stats

- **Endpoint**: `/api/stats/:username`
- **Method**: `PUT`
- **Description**: Updates the statistics associated with a specific user.
- **URL Parameters**:
  - `username`: String (required) - The username of the user whose stats are to be updated.
- **Request Body** (Any or all of the following):
  - `newPoints`: Number (optional) - The new points to be added to the user's total on the index of `locationId`.
  - `locationId`: Number (optional) - Identifier of location where `newPoints` are to be increased.
  - `correctAnswers`: Number (optional) - The amount to increase total correctAnswers of a user.
  - `incorrectAnswers`: Number (optional) - The amount to increase total incorrectAnswers of a user.
  - `progress`: Object (optional) - An object containing the progress update. When this field is updated, the completion percentages are automatically recalculated and updated. The object should have the following structure:
    * `locationName`: String (required for progress update) - The name of the location.
    * `locationId`: Number (required for progress update) - The index in the progress array to update.
    * `lessonId`: Number (required for progress update) - The new lesson ID to set.
    * `minilessonId`: Number (required for progress update) - The new minilesson ID to set.
    * `blockId`: Number (required for progress update) - The new block ID to set.
- **Response**:
  - `success`: Boolean - Indicates if the operation was successful.
  - `message`: String - A message describing the outcome.
  - `data`: Object - Contains the updated stats for the user.
- **Error Handling**:

- Returns `400 Bad Request` if the provided data is invalid or if required fields are missing.
- Returns `404 Not Found` if no stats are found for the given username.
- Returns `500 Internal Server Error` for any server-side errors.

9. General Leaderboard

- **Endpoint**: `/api/leaderboard`
- **Method**: `GET`
- **Description**: This method provides a leaderboard of the first 100 users, sorted by their total points. It supports optional filtering based on the user's age and country of origin. Each user's entry includes their username, country, age, total points, and rank.
- **Query Parameters**:
  - `age` (optional): Integer - Specifies the age to filter the leaderboard.
  - `country` (optional): String - Specifies the country to filter the leaderboard.
- **Response**:
  - `success`: Boolean - Indicates if the operation was successful.
  - `leaderboard`: Array of Objects - List of user rankings, each containing:
    - `username`: String - The user's username.
    - `countryOfOrigin`: String - The user's country of origin.
    - `age`: Number - The user's age, calculated from their date of birth.
    - `totalPoints`: Number - The total points accumulated by the user.
    - `rank`: Number - The user's rank in the leaderboard. Users with the same number of points share the same rank.
- **Error Handling**:
  - On server-side errors, returns `500 Internal Server Error` with an error message.

10. Get Leaderboard By User

- **Endpoint**: `/api/leaderboard/:username`
- **Method**: `GET`
- **Description**: Retrieves the ranking information of a specific user across different leaderboards (general, age-based, and country-based). This method finds the user's rank in each of these categories. If the user is not found, a 404 error is returned.
- **Path Parameters**:
  - `username`: String - The username of the user for whom to retrieve leaderboard data.
- **Response**:
  - `success`: Boolean - Indicates if the operation was successful.

  - `userData`: Object - Contains the ranking information of the user in different leaderboards. It includes:
    * `username`: String - The user's username.
    * `age`: Number - The user's age, calculated from their date of birth.
    * `country`: String - The user's country of origin.
    * `generalRank`: Number - The user's rank in the general leaderboard.
    * `ageRank`: Number - The user's rank in the age-specific leaderboard.
    * `countryRank`: Number - The user's rank in the country-specific leaderboard.
- **Error Handling**:
  - If the user is not found, returns `404 Not Found` with an appropriate error message.
  - On server-side errors, returns `500 Internal Server Error` with an error message.

Langchain endpoints

1. Get Welcome Message

- **Endpoint**: `api/langchain/welcome`
- **Method**: `POST`
- **Description**: Generates a personalized welcome message for a new or returning user. For new users, the message introduces the module, the user's guide (friend), and provides both a child-friendly and a parent-focused description of the module. For returning users, it includes a motivational message, highlighting their progress and the next steps in their learning journey. This message is generated using the OpenAI language model.
- **Request Body**:

  The request body is a JSON object with the following structure: `json { "currentBlock": 0, "currentLesson": 0, "currentMinilesson": 0, "land": { "id": 0, "name": "string", "friendName": "string", "friendType": "string", "moduleName": "string", "moduleDecriptionKids": "string", "moduleDescriptionParents": "string" }, "progress": 0, "user": { "username": "string", "dateOfBirth": "string", "preferredLanguage": "string" } }`
- **Response**:
  - `success`: Boolean - Indicates if the operation was successful.
  - `message`: String - The customized welcome message.

- **Error Handling**:
  - Returns `500 Internal Server Error` for any server-side errors.

2. **Get Lesson Message**
   - **Endpoint**: `/api/langchain/lessonMessage`
   - **Method**: `POST`
   - **Description**: Generates a message for the current mini-lesson or a quiz based on the user's progress. For lesson messages, it includes a theoretical explanation and a playful scenario. For quiz messages, it generates a 5-question quiz based on the mini-lesson content, formatted as JSON objects with question details. This approach is aimed at enhancing the learning experience in a fun, interactive way, using the OpenAI language model. This call automatically updates user progress.
   - **Request Body**:

     The request body is a JSON object with the following structure:

```
{
  "currentBlock": 0,
  "currentLesson": 0,
  "currentMinilesson": 0,
  "land": {
    "id": 0,
    "name": "string",
    "friendName": "string",
    "friendType": "string",
    "moduleName": "string"
  },
  "user": {
    "username": "string",
    "dateOfBirth": "string",
    "preferredLanguage": "string"
  }
}
```

   - **Response**:
     - `success`: Boolean - Indicates if the operation was successful.
     - `message`: String - The lesson message.
     - `nextIds`: Object - Next `lessonId`, `minilessonId` and `blockId` that await for user's completion. All fields are null if location is completely done.
   - **Error Handling**:
     - Returns `500 Internal Server Error` for any server-side errors.

3. **Get AI Answer From User Message**

   - **Endpoint**: `/api/langchain/userMessage`
   - **Method**: `POST`
   - **Description**: Save message that user have sent and respond with AI-generated message in chat-like conversation.
   - **Request Body**:

     The request body is a JSON object with the following structure:

```
{
  "currentLesson": 0,
  "currentMinilesson": 0,
  "land": {
      "id": 0,
      "name": "string",
      "friendName": "string",
      "friendType": "string",
      "moduleName": "string"

  },
  "user": {
      "username": "string",
      "dateOfBirth": "string",
      "preferredLanguage": "string"

  },
  "message": "string"
}
```

   - **Response**:
     - `success`: Boolean - Indicates if the operation was successful.
     - `message`: String - A message we want.
   - **Error Handling**:
     - Returns `500 Internal Server Error` for any server-side errors.

4. **Get Freeform Chat Message**

   - **Endpoint**: `/api/langchain/freeformUserMessage`
   - **Method**: `POST`
   - **Description**: This endpoint enables a freeform chat experience in the "Imagination Jungle" module, where users interact with Cleo the Chameleon. It supports both text-based conversations and image generation based on user input. For text chats, Cleo offers educational

and respectful responses. Inappropriate or offensive content triggers a response emphasizing positive communication. If the user opts for image generation, the endpoint returns an image URL based on the provided message.

- **Request Body**:

  The request body should be a JSON object with the following structure: "'json { "user": { "username": "string", "dateOfBirth": "string", "preferredLanguage": "string" }, "landId": 5, "message": "string", "type": "text" // or "image" to generate an image response }

- **Response**:
  - `success`: Boolean - Indicates if the operation was successful.
  - `message`: Depending on the `type` parameter in the request, this can be either:
    - String - The response from Cleo the Chameleon for text-based chats.
    - String - A URL to the generated image for image requests.

- **Error Handling**:
  - Returns `500 Internal Server Error` for server-side issues.

Implementation Details: This endpoint corresponds to the `getFreeformMessage` function in the backend. It processes the user's message, and depending on the requested type (text or image), invokes the appropriate Python script. For text responses, the `executePython` function uses OpenAI's language model to generate Cleo's response based on the `templateText` input template, considering the user's age, language, and chat history. For image responses, it executes a different script to generate an image URL based on the message. The endpoint also handles detection of inappropriate or offensive content to ensure a respectful chat environment.

5. Get User Chat

- **Endpoint**: `/api/chat`
- **Method**: `GET`
- **Description**: Retrieves chat of user for location.
- **URL Parameters**:
  - `username`: String (required)
  - `location_id`: Integer (required)
- **Response**:
  - `success`: Boolean - Indicates if the operation was successful.
  - `messages`: List of chat messages in which each message contains sender (User or AI) and content.
- **Error Handling**:
  - Returns `500 Internal Server Error` for any server-side errors.

6. **Get Lessons and Mini-Lessons Names**

- **Endpoint**: `/api/langchain/lessonNames`
- **Method**: `GET`
- **Description**: Retrieves lessons and mini lessons names for location.
- **URL Parameters**:
  - `locationName`: String (required)
- **Response**:
  - `success`: Boolean - Indicates if the operation was successful.
  - `message`: List of lessons in which each lesson containts its name and array of mini-lessons names.
- **Error Handling**:
  - Returns `500 Internal Server Error` for any server-side errors.

7. **Evaluate User Answer to a Question**

- **Endpoint**: `/api/langchain/evaluateQuestion`
- **Method**: `POST`
- **Description**: This endpoint evaluates a user's answer to a given question. It determines the relevance and correctness of the answer compared to a provided example of a correct answer. The evaluation and explanation are generated using the OpenAI language model, tailored to the specified language.
- **Request Body**:

  The request body should be a JSON object with the following structure: "'json { "user": { "username": "string", "preferredLanguage": "string" }, "question": "string", "userAnswer": "string", "correctAnswerExample": "string" }
- **Response**:
  - `success`: Boolean - Indicates if the operation was successful.
  - `message`: JSON Object - Contains two fields:
    - `evaluation`: String - Indicates the correctness of the user's answer ('correct' or 'incorrect').
    - `explanation`: String - Provides a rationale for the evaluation of the user's answer.
- **Error Handling**:
  - Returns `500 Internal Server Error` for server-side issues.

Implementation Details: The `getQuestionEvaluation` function in the backend handles this endpoint. It receives the user's answer, the question, the preferred language, and an example of a correct answer. These inputs are then passed to a Python script that utilizes the OpenAI language model to

evaluate the answer's correctness and relevance. The script employs structured output parsing to ensure the response is formatted as a JSON object with the evaluation and explanation.

Examples and Tutorials

Dive into this section for a series of step-by-step guides and real-world use-case examples designed to help you seamlessly interact with and harness the full potential of the Finance Friends platform. Whether you're a new user or looking to explore more advanced features, these tutorials and scenarios will provide the practical insights you need.

Step-by-Step Guides

1. Getting Started with User Registration Navigate the initial steps of joining the Finance Friends community by creating a new user account. - **Prepare Registration Details**: Gather essential details such as username, email, and password. - **Send Registration Request**: Utilize the `/api/register` endpoint to submit your registration details. - **Handle Response**: Confirm successful registration and note any instructions or welcome messages.

2. Interacting with the Chatbot Engage with our intelligent chatbot for a personalized learning journey and instant support. - **Set Up User Profile**: Ensure your profile reflects your learning preferences and style. - **Initiate Conversation**: Use the designated endpoint to send your queries or messages to the chatbot. - **Process Chatbot Response**: Understand and act on the guidance provided by the chatbot, tailoring your learning path accordingly.

Use-Case Driven Examples

Scenario 1: Tracking Learning Progress Monitor your journey through the diverse financial modules and visualize your achievements over time. - **Access Progress Dashboard**: Utilize specific features or endpoints to view your learning milestones. - **Analyze Performance**: Gain insights into your quiz scores, module completion rates, and areas for improvement.

Scenario 2: Participating in Quizzes Test your knowledge and reinforce your learning through interactive quizzes, receiving instant feedback. - **Engage with Quiz Modules**: Navigate to the quiz section corresponding to your current learning module. - **Submit Responses**: Answer the quiz questions and submit your responses through the platform. - **Review Feedback**: Receive real-time evaluation of your answers and understand the rationale behind each solution.

By following these guides and exploring the use-case scenarios, you'll be able to make the most of the Finance Friends platform, turning financial learning into a fun, engaging, and rewarding experience. Each step and scenario is designed to

enhance your interaction with the platform, ensuring a smooth and productive journey into the world of financial literacy.

Project Management and Design Tools

The development of Finance Friends is meticulously organized and creatively designed using industry-leading tools:

Notion for Project Management

- **Overview**: Our team uses Notion as the central hub for project management. It helps us track progress, organize tasks, and document every phase of the project.

Figma for Design

- **Creativity at Work**: The user interface of Finance Friends is crafted using Figma, ensuring an engaging and child-friendly experience. Figma aids our designers in creating intuitive and attractive designs that resonate with our young audience.

Repositories

Our codebase is split into backend and frontend repositories, each containing all necessary components and documentation for setting up and running the respective parts of the platform.

Backend Repository

- **Technology Stack**: The backend is built on Node.js and MongoDB, supplemented by Python scripts utilizing LangChain for advanced functionalities.
- **Repository Link**: Backend Repository

Frontend Repository

- **User Interface**: The frontend is developed using React, ensuring a dynamic and responsive user experience.
- **Repository Link**: Frontend Repository

Team and Contributions

Finance Friends is brought to life by a dedicated team of developers and guided by experienced mentors. Our project thrives on collaboration and academic contributions.

Authors

- Karlo Vrančić
- Filip Buljan
- Pavle Ergović
- Karla Šmuk
- Martina Rodić

Mentor

- **Guidance and Insight**: Our project is mentored by Prof. Ivica Botički, whose expertise and guidance have been invaluable in the development of Finance Friends.

Academic Contributions

- **Collaboration**: As a university project, we welcome feedback, suggestions, and contributions from fellow students and faculty members. Please refer to our contribution guidelines in the respective repositories for more information.

License and Acknowledgements

License

- **Academic Integrity**: This project is part of an academic program and adheres to the university's guidelines and policies.

Acknowledgements

- **Special Thanks**: Our heartfelt gratitude goes to Prof. Ivica Botički for his continuous support and mentorship. His insights have significantly shaped the trajectory and success of Finance Friends.