Chiri Freelancer Management System

▼ Software Requirement Specification (SRS)

Version: 1.0.0
Date: 09/September/2023

Table of Contents

1. Introduction

1.1 Purpose

The purpose of the Chiri Freelancer Management System is to empower freelancers with efficient tools and resources, streamline freelance operations, foster collaboration, and provide personalized financial insights. This project aims to simplify freelancers' lives and enhance their success in the freelance economy.

1.2 Scope

The scope of the Chiri Freelancer Management System encompasses the development of a comprehensive software platform that facilitates freelancers' workflow management, client interactions, project handling, financial tracking, and community engagement. This system is designed to serve freelancers, small businesses, agencies, and independent professionals seeking to optimize their freelance operations and thrive in the gig economy.

1.3 Definitions, Acronyms, and Abbreviations

- SRS: Software Requirement Specification
- API: Application Programming Interface

1.4 References

1.5 Overview

The Chiri Freelancer Management System is a feature-rich software platform tailored for freelancers and businesses. It offers tools for streamlined project management, client interactions, financial tracking, and collaboration. With AI-driven features and a supportive user community, Chiri aims to simplify freelancers' lives and empower them to succeed in the ever-evolving freelancing landscape.

2. System Description

2.1 System Overview

The Chiri Freelancer Management System is a multifaceted software platform designed to revolutionize freelance management. It serves as a central hub for freelancers to efficiently manage projects, clients, and financials. Key features include gig tracking, client relationship management, project bidding, financial insights, and collaborative tools. Chiri empowers freelancers to thrive in the gig economy by simplifying complex processes and enhancing their freelance experience.

2.2 System Architecture

The Chiri Freelancer Management System adopts a modern and scalable architecture to ensure robust performance and flexibility. It comprises three primary components:

1. **Client-Side Application:** The client-side application is built using modern web technologies, providing a responsive and user-friendly interface accessible from web browsers and mobile devices.
2. **Server-Side Application:** The server-side application is developed using Node.js and hosted on cloud infrastructure. It handles data processing, user authentication, and business logic.
3. **Third-Party Integrations:** Chiri seamlessly integrates with third-party services for payment processing, communication, and data analytics, ensuring a comprehensive and feature-rich ecosystem.

This architecture enables Chiri to provide a reliable, high-performance platform while allowing for future scalability and adaptability to changing requirements.

2.3 System Users

The Chiri Freelancer Management System caters to a diverse set of users, each with specific roles and permissions:

1. **Freelancers:** Independent professionals across various industries who utilize Chiri to streamline their freelance operations, manage projects, and interact with clients.

2. **Clients:** Businesses, organizations, and individuals seeking freelance services. They use Chiri to post projects, evaluate proposals, and communicate with freelancers.
3. **Administrators:** System administrators have elevated privileges to manage and oversee the Chiri platform. They can control user accounts, monitor system performance, and ensure platform integrity.

Chiri's user-centric design ensures that each user type enjoys a tailored experience, optimizing their interaction with the platform and fostering a collaborative freelance ecosystem.

3. Functional Requirements

3.1 User Management

Chiri's User Management module enables:

- User registration and profile creation.
- Secure authentication and login procedures.
- User role assignment (Freelancer, Client, Administrator).
- User profile updates, including personal and contact information.
- Password reset and recovery mechanisms for enhanced security.

This functionality ensures that users can easily join, access, and manage their profiles within the Chiri platform, fostering a seamless user experience.

3.2 Gig Tracking

Chiri's Gig Tracking feature allows freelancers to:

- Create and manage gigs or projects.
- Monitor project progress and milestones.
- Track project timelines and deadlines.
- Record gig-related expenses and income.
- Generate reports for individual gigs.

This functionality empowers freelancers to efficiently manage their projects, ensuring they stay organized and on top of their freelance work.

3.3 Client Management

Chiri's Client Management functionality offers the following features:

- Efficient client communication and interaction.
- Client profile creation and management.
- Client project history and interaction tracking.
- Integration with project management for seamless collaboration.

This functionality enables freelancers to build and maintain strong client relationships while keeping track of project-specific details and communication history.

3.4 Project Management

Chiri's Project Management module facilitates:

- Project creation, including descriptions and requirements.
- Task assignment and tracking within projects.
- Milestone setting and monitoring.
- Real-time project status updates.
- File and document sharing for project collaboration.

This functionality empowers freelancers to efficiently organize, collaborate on, and deliver projects to their clients.

3.5 Project Bidding

3.6 Seller Management

3.7 Support Ticket System

3.8 Project Reports

3.9 Buyer Reports

3.10 Target Reports

4. Non-Functional Requirements

Chiri is committed to meeting the following non-functional requirements:

- **Performance:** The system should maintain responsive performance even with a large user base and extensive data.
- **Security:** Robust security measures should protect user data, financial transactions, and system integrity.
- **Usability:** The user interface should be intuitive, ensuring a positive user experience.
- **Compatibility:** Chiri should be accessible across various devices and web browsers.
- **Scalability:** The system should easily scale to accommodate increasing user numbers and data loads.
- **Reliability:** Chiri should be highly available and reliable, minimizing downtime.

These non-functional requirements ensure that Chiri delivers a secure, user-friendly, and reliable platform for freelancers and clients.

5. User Interfaces

Chiri's user interfaces must adhere to the following non-functional requirements:

- **Responsiveness:** The interfaces should load quickly and adapt to different screen sizes and resolutions for a seamless user experience.
- **Accessibility:** The interfaces should comply with accessibility standards to ensure usability for all users, including those with disabilities.
- **Consistency:** The design elements, layout, and navigation should maintain consistency throughout the platform to enhance user familiarity and usability.
- **Intuitiveness:** User interfaces should be intuitive, requiring minimal user training and allowing users to perform tasks efficiently.
- **Performance:** Interfaces should be optimized for speed and responsiveness, ensuring smooth interaction even during peak usage periods.

These non-functional requirements contribute to user satisfaction and usability, making Chiri a user-friendly and accessible platform for freelancers and clients.

6. Data Management

Chiri's data management system must meet the following non-functional requirements:

- **Data Models:** Data should be structured efficiently to support quick retrieval and reporting.
- **Data Storage:** Reliable and scalable data storage solutions should be used to ensure data integrity and availability.
- **Data Backup and Recovery:** Regular data backups and a robust recovery mechanism should be in place to protect against data loss and ensure business continuity.

These non-functional requirements ensure that Chiri's data management system is secure, efficient, and resilient, guaranteeing the integrity and accessibility of user data.

7. Integration

Chiri's integrations with third-party services must meet the following non-functional requirements:

- **Seamless Integration:** Third-party services should integrate seamlessly with Chiri, providing a smooth user experience.
- **Reliability:** Integration points should be reliable, minimizing service disruptions and ensuring data consistency.
- **Security:** Integration mechanisms should adhere to strict security standards to protect sensitive data during data exchanges.
- **Scalability:** Integration solutions should be scalable to accommodate increasing data volumes and user traffic.

These non-functional requirements guarantee that Chiri's integrations are robust, secure, and capable of delivering a seamless experience for users.

8. Security and Compliance

Chiri's security and compliance measures must meet the following non-functional requirements:

- **Data Security:** Robust encryption, access controls, and secure authentication methods should safeguard user data and financial information.
- **Privacy:** Chiri should adhere to stringent privacy standards, including compliance with relevant data protection regulations like GDPR.
- **Regular Audits:** Periodic security audits and vulnerability assessments should be conducted to ensure ongoing system security.

These non-functional requirements ensure that Chiri provides a secure and compliant environment, protecting user data and privacy while meeting legal and regulatory standards.

9. Testing Requirements

Chiri's testing requirements must meet the following non-functional criteria:

- **Test Cases:** Comprehensive test cases should cover all system functionalities, including positive and negative scenarios, security testing, and performance testing.
- **Test Environment:** A dedicated test environment should replicate the production environment closely, ensuring accurate testing results and minimizing risks during deployment.

These non-functional requirements ensure rigorous testing, guaranteeing that Chiri is robust, secure, and reliable when used by freelancers, clients, and administrators.

10. Deployment Plan

Chiri's deployment plan should adhere to the following non-functional criteria:

- **Deployment Strategy:** A well-defined deployment strategy should minimize downtime, ensure data migration, and mitigate potential risks during the release.
- **Deployment Schedule:** A detailed deployment schedule should be created, considering optimal times for deployment to minimize user disruptions and

ensure a smooth transition to new system versions.

These non-functional requirements ensure a careful and organized deployment process, maximizing system availability and minimizing disruptions for users.

11. Maintenance and Support

Chiri's maintenance and support measures should meet the following non-functional criteria:

- **Ongoing Maintenance:** Regular updates, bug fixes, and system enhancements should be conducted to keep Chiri up-to-date and reliable.
- **Customer Support:** A dedicated customer support team should be available to address user inquiries, issues, and provide assistance promptly.

These non-functional requirements ensure that Chiri remains a well-maintained and user-friendly platform with reliable customer support services for its users.

12. Appendix

[Include any additional documentation, diagrams, or references.]

▼ Entities / Schema / Model

What we need to store?

- User (freelancer, buyer, sellers, admin)
  - id - int
  - name - string
  - email - string
  - password - string (hashed)
  - role - enum [FREELANCER, BUYER, SELLER, ADMIN] default FREELANCER

    - status - enum [PENDING, APPROVED, BLOCK, DECLINE] default PENDING
    - timestamp
- Profile (freelancer, buyer, sellers, admin)
    - id - int
    - userId - relation with user
    - name - string
    - email - string
    - dateOfBirth - datetime
    - gender - string
    - brief - text
    - profileImage - string
    - email - string
    - phone - string
    - fax - string
    - address - string
    - city - string
    - state - string
    - zip - string
    - socialMedia - string
    - role - enum [FREELANCER, BUYER, SELLER, ADMIN] default FREELANCER
    - status - enum [PENDING, APPROVED, BLOCK, DECLINE] default PENDING
    - timestamp
- Category

    - id - int
    - userId - relation with user
    - name - string
    - active - boolean default true
    - type - enum [TICKET, BID, INVOICE] default BID
    - timestamp
- Invoice
    - id - int
    - userId- relation with user
    - categoryId - string
    - invoiceNumber - string
    - dateOfCreation - datetime
    - dateSent - datetime
    - dateDue - datetime
    - tax - int
    - discount - int
    - amountDue - int
    - brief - text
    - status - enum [NEW, COMPLETE] default NEW
    - timestamp
- Bid
    - id - int
    - userId- relation with user
    - buyerId - relation with user
    - sellerId - relation with user
    - categoryId - relation with category

    - bidNumber - int
    - dateOfBid - datetime
    - bidNumber - int
    - bidStatus - enum [ACCEPTED, REJECTED, SELECTED, WAITING] default WAITING
    - paymentType - enum [CARD, CASH, CHEQUE, ESCROW] default CARD
    - price - int
    - tax - int
    - discount - int
    - amountDue - int
    - brief - text
    - timestamp
- Ticket
    - id - int
    - userId- relation with user
    - categoryId - relation with category
    - ticketNumber - int
    - name - string
    - email - string
    - phone - string
    - ticketSubject - string
    - dateOfCreation - datetime
    - type - enum [WEB_DESIGN, LOGO_DESIGN, BRANDING, SOFTWARE_CONSULTING] default WEB_DESIGN
    - timestamp

▼ ER Diagram

https://viewer.diagrams.net/?tags=%7B%7D&highlight=0000ff&edit=_blank&layers=1&nav=1&title=chiri.drawio#R7Z1bk9o4E4Z%2FzVwm5QMYuBxmMvlSmWxSSbZ291JgAd4Yi5VNZsiv%2FyTjAyAPkRgfhNVVSQXLxjjSq34sdat1496tn99TtFl9Ij4ObxzLf75x728cx7EmHvuHl%2Bz2JfbInexLljTws7Ky4FvwC2eFVla6DXwcH12YEBImwea4cE6iCM%2BTozJEKXk6vmxBwuNf3aAlFgq%2BzVEolv4V%2BMlqXzoeWmX5%2F3CwXOW%2FbFvZmTXKL84K4hXyydNBkfvuxr2jhCT7T%2BvnOxzy2svrZf%2B9hxfOFg9GcZTIfOF9PJ%2F4y8XD98Hq8Xa1%2BvmF3k3fOM4ge7pkl%2F%2BXsc9qIDskNFmRJYlQ%2BK4snVKyjXzM72uxo%2FKaR0I2rNBmhf%2FiJNllzYm2CWFFq2QdZmdx5N%2FyxmGH777%2BwpR8J59QtNufeQjCMLt1nCCalFeuUeR%2FjnB%2B4uDCEM1wOEXzH8v02e5ISCg7FZHsakp%2B4LzwxnFH4%2FHtrcvOLEiU5OU%2BXqBtyOpyKlZtVtsx2dI5PlOf2XW8Cg%2B%2BmDXIe0zWOKE7dgHFIUqCn8ciQ5lWl8V1xVe%2FkIA9imNlHcvORZV1K9ezjm%2FBqmeJk%2BxbpSjYh4PHKItSqSjIJnvgnyjcZv%2BFP2NMbwqpl2Jiqt%2FwjwmahUXDZbpwecuxjpugIMI0k8achCHaxEF6%2Bb5kFYT%2BI9qRbZLfKD%2BaLoJn7H%2Fd93I7VebTI7tZnMmCN28uZX4ahcEyYp%2FnrGH5L04pjtmzPKI4ya44EOmxys8rbMGkeKCvqXM3uZ1KK4%2BVD7zBZHh%2FTns%2FMU3w81lVZWfdrBmeSoNlj7Oy1YGxck5Vcyi8I8mo6sMR9HFeGF95B5%2BuCA1%2BcTkcdf9MLOnxU7AOUcQsLvJPiqYkRYx10hZZ8xyLil%2FkU7L5nnaSrGDD%2B0paD8Mp%2B8Nq5s56O7xhbeLcsWO7PE6bacOs3h2JWPuiIG0ozDT0hLmOpklqBVPR4EV%2Bf5rVO%2F88I0lC1hfIrA45ne3Qv9fYTk1jblMScwWJffn4osjY%2Fz8JUPiVvSGgaBnuJZG%2BMKBSEhW6qWzJovVOm%2FXU2hBWnYswZdcq8H3MLM%2F0aRUk%2BNsGpRR5Yi9Nl9ucxsTgSIvhoPVdxcbPblY2ifLdUMhseIQS1vdZ5cRNcG4gci4K%2FtviD%2FcdSy0H2f7aacwEFUTLx%2F03vRMtDo3Q4hn4DWvVptTtWhDnECArBVlLFy0rQ9brGrKevMTasXvYD%2FL7XTNhh9JK6DthR4LCIrTGqcQoI5rumO2%2FAI3D6hiw2nOsTrrG6gSw2ogSxtJK6DtWc%2FfJgcTwGgUhgFUbCRoHVlv0GgBZ%2B0XWwi%2FVGVptBdcDsFVFC%2FJi6D1cRd%2FDBsXxE6E%2B8FUfGZoHWNFdAYDtGWCHnQNWwe0AgFXRwkBaDL0HrOh3oIQ%2Fq4fWvOmiWcz%2FwdF2za7ilsPx0sCtGfuwTOOCKMbMPs15UJKXxXQ9FGe5bXGs2XbHT%2FOPMXuo%2FDPy10GUGiAgeMc6N4%2FgojsECN4zgo8lCT5ojOAKHg4guIoWRtJi%2BA1zz7W9OsFP79aGIRNdHMwmJVv2Y9Yb9vcMujc48tMB9EvcRpsNZRLxM4qHZP5j%2F9HH8zBgTQzs1kDhiuy%2BXPFSt2tB8vnUJ7C7t%2Bx2ZKe3G2O3o%2BBDAXarxJla0mLoO7tzSR1oLAnWrJfzwTejLPtxfgyM7VyJ5jF2IJq7VpYrssf%2BOzd4%

2FOAffvB2mB%2FePx%2BevN%2FVqKByVaL0Ysij1ZXlZcdLK0U1yq5xdDL3V5K%2FSbx0YTZTXN9iyNehs2LFECXMDnGt38DSxUyE3S5dfOPmr1P5dInkG5fX3FpG8KrJvdfrQ1BXWna6rGZ0RK8aLGesSw7gWCuqQnSswYJGbdVonPvLAffXlU2hqaO28zWNTpX768QH8fCx2v0AM201SKY2L9n141j0km1jTD%2F454ajmvDYAPkZx9%2F8J4G%2F%2FeVv54sfXXBhNaOFovcCWF3RhQVpBfSSoHlwFf0PANeewbX79Y%2BugrcC6Kpi2uTF0Hu6is4KSC6gmQbNw6tCrjLAqw5qvgCvna9%2BdBU8FIBXFdPmSYuh93gVHRQ85vLzYhpQ9qiHEZjH6yEBu51r0zzsKuQyA%2BzqoOYLsNv5ksX8xoDduk3bRFoMfV%2F2MBD9EkvMWojCsFYfERq34iFf8AB87S9fu19WOFDwTQBfFcRQdF%2Fg60D0TMxogBesiFVNAnDtXIHGwXUI8cb9h6tswHFjcB1Cvp2GNrGAfDtFVYjTcJv94tYPa77DLQxhtZGicZTNDTBQtr%2BUda2uKetBWHEzYii6L1DWEyfjIPBJMw2ah1eIK%2B4%2FXged4xXiihsybbXFFV8%2FXsW44s2KVz%2FgVRsNGodXW8U3BnzVQc4X8LXzCCe7wj0GgK1nvwkgbFkXImIX6BkAq5EGjSOsByt3eg%2FYQechTh6s3Glo8FDbyp2rx6snevqR71Mcx0BYfVRoHmBhjU7%2FAdt5mNMI1ug0ZNpgjU6hMdHJPw%2BSHdBVHwkaR9eRGBQAdO0ZXYedhzeNYIVOM2Ioui%2FQdSS6IPimneB%2F1UiDxuF1ojCsALzqoOYL8Np5eNMEooebEUPRfQGvE3Gg8CvYAFz1UaB5cIXYpv7DtfPYpgmENjVk2iCyqagKMbApJnOmpE%2BsrRFAVh8lmgdZcL%2F2HrJe5%2FFNdj5FDZSt27aB%2F7UUmThPwjMRM9O05iPZMjMxYLZjKRqHWdsSB7Mfop8kmPMHPk%2FcE7q61l6XCaMZppkAjqmZlqyC0H9EO7JN8hvlR0zDz9hnII8L%2BTw9spvFucRP9tbOFTtn7c9%2FcUpxzJ7lEXGEpldcKMXDvsQEN3XuJrfTmiRqK4sxf0vPEwtXbUY3qZJTYwn9bUthcGr025k2RrDs6ArpSeR01qDMxPHpl48dY%2FLUCF0rFcs%2BDJtH2Ja4mubPKPhviz%2Fc6%2F5WdqjHoSF6PPOa5tWqT6nbtSJQSLp5ZdMhFwC3Khq9ZeBWZd30Qt4EM%2FZhyT88fMyL2G8UpR2byZ7MmpT9HKBsW%2BIU8DbG9IOvO5JNUJ95CHbEwQgguG8Ilp1baQ7BTtXCfkBwe6Ip%2Bjkg2HbEYcec%2FeKS0B1gWAcFmodhG0Lb%2B49huyoyoF0O2xDc3pAcih4MgLVtR1BZsHe%2F%2FrFdz9JtFLkJANJ2LkUDSQtx7gaQdtg9aSHSvSnzVluoew9IK07f8cC7z4s7ilESkOgGIvF0EqSBvIWUngbwtmphWcu8haSeTZm32rJ69oC3YigBx%2Bs3XoWOxyPf3Wk0izdp9QJ5NZKmeeR1Ibqq%2F%2BR17M7J68Kmxg3JwYW4qbIuxLgpjtd7dgDg1VmZ5oHXhlXeBoC3KlFZy0FVsMy7qXFFbeu8rx%2B8jhgykPB9ogTogktXC0Gax9uBgh0E3mqh5wt4O%2BqctwMInmpqTzwInirrwhEHukE8Zz%2BYAGR1UaF5kHUUZvkAslro%2BZLkGJ1D1lGYPAHIKi3AGEvLofeQdcUhA1pzxFbPJwN1tZCledR1FYYcQF0t9HwBdbvPSeWKgw6gbj2eMvnUeP2nrrj2YkYDvKggLqsrQK4GmjQQuZAF0gDkymalai5Jt6uQeQWQq2TeassBef1Zul1x%2BQXfyHHLfs16w%2F7iaLtm%2F3C7cZqPJcJP1QlZ2KNYc7LehDjBqY0BTHetY0VMvyLFt8ztWpmPVhixAKa10LM

6pgcazEfDbstNTfw50nLo%2FcjYEQcdsJmGnlo0b0Q8EN8h77J8aeeRe4JX2EtDSqGv3kujzADzuy2oCok1ECwFq8%2Fk3s%2F0MYID%2BdWUZzbTqNJZgzF5YlgKbKZRmyBg%2FVlZF2JcCmymoa8eFfc8e8Vbmszt2hBo%2FhwA3KuZEFEHbpXbol3gDqsiUiCTd3uiKfo5QNkeipPAsJmGLuozD8EexMj3H8FVm2m0i2APQuQbUoMHIfJFXYzE8USEuAuCyYwymgFiuxehgYiF%2Far6j9jKjTKqGNtcdJ6nkLYWGKtk3mrbier6o%2FM80UkGbn89taiI2ldE2MncrpUXQNHiYX%2BJc08Co9SKLEmEwndl6Um7ltc8klRyvOn%2FxUmyy9CLtgk5FgZrMbr7m3%2F%2F7WAwyAv%2BYQVvrLeWPcxL7p9zu5ge7Q6PvmDKek3q8S9hf0tpqtB3X39hSr6TTyja5ecegjAsbuCXV65R5H9ONciKDy6qQ68%2BXqBtmFSqNZtjjcmWzvFZtmYGJMnfN168cn8db8CzMqc4REnwEx89xxkRfyHp2ti8exSRB0Uowolw9w%2BafasJ0Q67Fe2hYPlL3e%2FU6r8gy2O9nQi4lOWperURpi2ry3x33vqU%2BboXv9GoUwHZrxBQO9ZKWoutmLV8V0lt5CNOAU%2FZN%2FIBDUTKdRspd0wnVzan2GDS2EBzBMliry1ObiQ%2FQXcmTq7dZQxjcTYX4uRqEwSkiy2FJsaHQJycvno0bzXD2AHgXpkHQR24ssv7GwRu1bpBiJNrTzRFPwco22NxeeFsu4NAOT3kZyCDwYvffwZXBcq1zOAqJz4wuEUjWJuvvwcMFn39MfseQFgP%2FRkHYadiL1uAcN8gXBlK1yqFncrt8IDCramm7OhAYadiz7x5lsuEc9g656kFGremQ%2BNobI%2BBxgbQeNg1je0xLB5rarABmC3qYiJidhb4f2zXM0xvYD8VTWRoHmUnsJ%2BKAZQdd07ZiSOvM6Csihwm8kGf%2FaesGGPAl4x9XuxDjmH5mD5aNBC1sI9K%2F1HrVGWabRm1sI9KU%2Battn1UeoBaMZQABrTaydBAykKabQMoO%2BiesgruCaCsknmDJNtlXYjOCUbZb%2FmGZTCg1UeLxqHWyfeiAtT2GbWyS8Kbi5eyFHwUgFqVAJSiBwNqHUv0UGzQbs1q8PtuA1k%2BddKigahV2HQRUKuDni9KitE9ahV8FIBaJfMmv9dr%2F1Ereig2NJhjmDfWRYIGElYhvzEQVgc9X0LYzvNOOZaCfwIlq2Te5NOq95%2BwonciQc%2FAV10EaCBfIa2jAXztPM1UEX8FfK3dvEFSx1JlokvCD%2BI5%2B8EEIKuLCs2DrKPgKQPI6qDnSyDbeR4pJ5cVQLZu86aQwr33kHVEpxdac8Teb2GqWBsZmkdZW8H6AWV10PMFlB3I5olqbMtFx1Zw%2BgNllRLvONJy6PuWi45dkQqZBnjBiljdMMR6fO9FdxrN4s1N5ynI%2BotcBU0qIvcVWy%2FK3K4VkUJ%2BZAOQq8HsMexy3JR5g8zHZV3ALsfXosUejW7ZISUkObyctc%2FqE%2FExv%2BL%2F

▼ API Endpoints

Namespaces

- Public - Anyone with the API access

- Admin / Private - Internal use only (restricted, different CORS policy)

We will follow no namespace at this moment

▼ Auth

▼ Create a new account

Method: POST

Access: Public

Path: /auth/signup

Request Body:

- name
- email
- password

Response

- 201
  - statusCode
  - message
  - data
    - access token
    - refresh token
  - links
    - self
    - signin
- 400
  - statusCode
  - message
  - data (Array of error messages)

        - field
        - message
    - ▼ Signin to existing account

      Method: POST

      Access: Public

      Path: /auth/signin

      Request Body:

      - email
      - password

      Response

      - 200
        - statusCode
        - message
        - data
          - access token
          - refresh token
        - links
          - self
        - 400
          - statusCode
          - message
          - data (Array of error messages)
            - field
            - message
- ▼ User
  - ▼ Get all users

Method: GET

Access: Private

Role: Admin

Path: /users?query=params

Query:

- page (default 1) - current page number
- limit (default 10) - the number of objects should be returned
- sortType (default desc) - the type of sort, it could be either asc or desc
- sortBy (default updatedAt) - the property that will used to sort. It could be either updatedAt or title.
- expand (default none) - possible values (profile)

Response:

- 200
    - statusCode
    - data array
        - id
        - name
        - email
        - password
        - role
        - status
        - profile
            - id
            - userId
            - name

                - email
                - dateOfBirth
                - gender
                - brief
                - profileImage
                - email
                - phone
                - fax
                - address
                - city
                - state
                - zip
                - socialMedia
                - timestamp
            - timestamp
        - pagination
            - page
            - limit
            - nextPage
            - prevPage
            - totalPage
            - totalBook
        - links
            - self
            - nextPage
            - prevPage

- 400
  - message

▼ Get a single users

Method: GET

Access: Private

Role: Admin

Path: /users/:id

Response:

- 200
  - statusCode
  - data
    - id
    - name
    - email
    - password
    - role
    - status
    - profile
      - id
      - userId
      - name
      - email
      - dateOfBirth
      - gender
      - brief

- profileImage
- email
- phone
- fax
- address
- city
- state
- zip
- socialMedia
- timestamp

- timestamp

- 404
  - message

▼ Create a user

Method: POST

Access: Private

Role: Admin

Path: /users

Request Body:

- name
- email
- password
- role
- status

Response

- 201
  - statusCode
  - message
  - data
    - id
    - name
    - email
    - role
    - status
    - timestamp
  - links
    - self
    - edit
    - delete
    - view
- 400
  - message
  - data (Array of error messages)
    - field
    - message
- 401
  - message

▼ Update a user

Method: PATCH

Access: Private

Role: Admin

Path: /users/:id

Request Body:

- name
- email
- role
- status (only admin can update)

Response

- 200
  - statusCode
  - message
  - data
    - name
    - email
    - role
    - status
  - links
    - self
- 404
  - message
- 400
  - message
  - data (Array of error messages)
    - field
    - message
- 401
  - message

▼ Delete a user

Method: Delete

Access: Private

Role: Admin

Path: /users/:id

Response:

- 204
- 404
  - statusCode
  - message
- 401
  - statusCode
  - message

▼ Change password

Method: PATCH

Access: Private

Role: Admin

Path: /users/:id/password

Request Body:

Response:

- 200
- 400
  - message
  - data (Array of error messages)
    - field
    - message

- 404
  - statusCode
  - message
- 401
  - statusCode
  - message

▼ Category

▼ Get all Categories

Method: GET

Access: Private

Role: Admin

Path: /categories?query=params

Query:

- page (default 1) - current page number
- limit (default 10) - the number of objects should be returned
- sortType (default desc) - the type of sort, it could be either asc or desc
- sortBy (default updatedAt) - the property that will used to sort. It could be either updatedAt or title.
- type - the type of the category
- search - the search keyword of the category

Response:

- 200
  - statusCode
  - data
    - id

        - userId
        - name
        - active
        - type
        - timestamp
    - pagination
        - page
        - limit
        - nextPage
        - prevPage
        - totalPage
        - totalBook
    - links
        - self
        - nextPage
        - prevPage
- 400
    - message

▼ Get a single Category

Method: GET

Access: Private

Role: Admin

Path: /categories/:id

Response:

- 200

        - statusCode
        - data
            - id
            - userId
            - name
            - active
            - type
            - timestamp
    - 404
        - message

▼ Create a Category

Method: POST

Access: Private

Role: Admin

Path: /categories

Request Body:

- id
- userId
- name
- active
- type

Response

- 201
    - statusCode
    - message

        - user data
            - id
            - userId
            - name
            - active
            - timestamp
        - links
            - self
            - edit
            - delete
            - view
    - 400
        - message
        - data (Array of error messages)
            - field
            - message
    - 401
        - message

▼ Update a Category

Method: PATCH

Access: Private

Role: Admin

Path: /categories/:id

Request Body:

- id
- userId

- name
- active
- type

Response

- 200
  - statusCode
  - message
  - invoice data
    - id
    - userId
    - name
    - active
    - timestamp
  - links
    - self
- 404
  - message
- 400
  - message
  - data (Array of error messages)
    - field
    - message
- 401
  - message

▼ Delete a Category

Method: Delete

Access: Private

Role: Admin

Path: /categories/:id

Response:

- 204
- 404
  - statusCode
  - message
- 401
  - statusCode
  - message

▼ Invoice

▼ Get all invoices

Method: GET

Access: Private

Role: Admin

Path: /invoices?query=params

Query:

- page (default 1) - current page number
- limit (default 10) - the number of objects should be returned
- sortType (default desc) - the type of sort, it could be either asc or desc
- sortBy (default updatedAt) - the property that will used to sort. It could be either updatedAt or title.
- search - the search keyword of the invoice
- expand (default none) - possible values (category)

Response:

- 200
  - statusCode
  - data
    - id
    - userId
    - categoryId
    - invoiceNumber
    - dateOfCreation
    - dateSent
    - dateDue
    - tax
    - discount
    - amountDue
    - brief
    - status
    - timestamp
  - pagination
    - page
    - limit
    - nextPage
    - prevPage
    - totalPage
    - totalBook
  - links
    - self

        - nextPage
        - prevPage
    - 400
        - statusCode
        - message

▼ Get a single invoice

Method: GET

Access: Private

Role: Admin

Path: /invoices/:id

Response:

- 200
    - statusCode
    - data
        - id
        - userId
        - categoryId
        - invoiceNumber
        - dateOfCreation
        - dateSent
        - dateDue
        - tax
        - discount
        - amountDue
        - brief

        - status
        - timestamp
    - 404
        - statusCode
        - message

▼ Create a invoice

Method: POST

Access: Private

Role: Admin

Path: /invoices

Request Body:

- categoryId
- invoiceNumber
- dateOfCreation
- dateSent
- dateDue
- tax
- discount
- amountDue
- brief
- status

Response

- 201
    - statusCode
    - message

    - data
        - id
        - userId
        - categoryId
        - invoiceNumber
        - dateOfCreation
        - dateSent
        - dateDue
        - tax
        - discount
        - amountDue
        - brief
        - status
        - timestamp
    - links
        - self
        - edit
        - delete
        - view
- 400
    - statusCode
    - message
    - data (Array of error messages)
        - field
        - message
- 401

        - statusCode
        - message

▼ Update a invoice

Method: PATCH

Access: Private

Role: Admin

Path: /invoices/:id

Request Body:

- categoryId
- invoiceNumber
- dateOfCreation
- dateSent
- dateDue
- tax
- discount
- amountDue
- brief
- status

Response

- 200
    - statusCode
    - message
    - data
        - id
        - userId
        - categoryId

        - invoiceNumber
        - dateOfCreation
        - dateSent
        - dateDue
        - tax
        - discount
        - amountDue
        - brief
        - status
        - timestamp
      - links
        - self
  - 404
    - statusCode
    - message
  - 400
    - statusCode
    - message
    - data (Array of error messages)
      - field
      - message
  - 401
    - statusCode
    - message

▼ Delete a invoice

Method: Delete

Access: Private

Role: Admin

Path: /invoices/:id

Response:

- 204
- 404
  - message
- 401
  - message

▼ Bid

▼ Get all bids

Method: GET

Access: Private

Role: Admin

Path: /bids?query=params

Query:

- page (default 1) - current page number
- limit (default 10) - the number of objects should be returned
- sortType (default desc) - the type of sort, it could be either asc or desc
- sortBy (default updatedAt) - the property that will used to sort. It could be either updatedAt or title.
- search - the search keyword of the bid
- expand (default none) - possible values (category, user)

Response:

- 200

- statusCode
- data
  - id
  - userId
  - buyerId
  - sellerId
  - categoryId
  - bidNumber
  - dateOfBid
  - bidNumber
  - bidStatus
  - paymentType
  - price
  - tax
  - discount
  - amountDue
  - brief
  - timestamp
- pagination
  - page
  - limit
  - nextPage
  - prevPage
  - totalPage
  - totalBook
- links

        - self
        - nextPage
        - prevPage
    - 400
        - statusCode
        - message

▼ Get a single bid

Method: GET

Access: Private

Role: Admin

Path: /bids/:id

Response:

- 200
    - statusCode
    - data
        - id
        - userId
        - buyerId
        - sellerId
        - categoryId
        - bidNumber
        - dateOfBid
        - bidNumber
        - bidStatus
        - paymentType

        - price
        - tax
        - discount
        - amountDue
        - brief
        - timestamp
    - 404
        - statusCode
        - message

▼ Create a bid

Method: POST

Access: Private

Role: Admin

Path: /bids

Request Body:

- id
- userId
- buyerId
- sellerId
- categoryId
- bidNumber
- dateOfBid
- bidNumber
- bidStatus
- paymentType

- price
- tax
- discount
- amountDue
- brief
- timestamp

Response

- 201
  - statusCode
  - message
  - data
    - id
    - userId
    - categoryId
    - invoiceNumber
    - dateOfCreation
    - dateSent
    - dateDue
    - tax
    - discount
    - amountDue
    - brief
    - status
    - timestamp
  - links
    - self

        - edit
        - delete
        - view
    - 400
        - statusCode
        - message
        - data (Array of error messages)
            - field
            - message
    - 401
        - statusCode
        - message

▼ Update a bid

Method: PATCH

Access: Private

Role: Admin

Path: /bids/:id

Request Body:

- id
- userId
- buyerId
- sellerId
- categoryId
- bidNumber
- dateOfBid
- bidNumber

- bidStatus
- paymentType
- price
- tax
- discount
- amountDue
- brief
- timestamp

Response

- 200
  - statusCode
  - message
  - invoice data
    - id
    - userId
    - categoryId
    - invoiceNumber
    - dateOfCreation
    - dateSent
    - dateDue
    - tax
    - discount
    - amountDue
    - brief
    - status
    - timestamp

        - links
            - self
    - 404
        - statusCode
        - message
    - 400
        - statusCode
        - message
        - data (Array of error messages)
            - field
            - message
    - 401
        - statusCode
        - message

▼ Delete a bid

Method: Delete

Access: Private

Role: Admin

Path: /bids/:id

Response:

- 204
- 404
    - statusCode
    - message
- 401
    - statusCode

        - message

▼ Ticket

  ▼ Get all tickets

    Method: GET

    Access: Private

    Role: Admin

    Path: /tickets?query=params

    Query:

    - page (default 1) - current page number
    - limit (default 10) - the number of objects should be returned
    - sortType (default desc) - the type of sort, it could be either asc or desc
    - sortBy (default updatedAt) - the property that will used to sort. It could be either updatedAt or title.
    - search - the search keyword of the ticket
    - expand (default none) - possible values (category, user)

    Response:

    - 200
      - statusCode
      - data
        - id
        - userId
        - ticketNumber
        - name
        - email
        - phone

        - ticketSubject
        - type
        - timestamp
    - pagination
        - page
        - limit
        - nextPage
        - prevPage
        - totalPage
        - totalBook
    - links
        - self
        - nextPage
        - prevPage
- 400
    - statusCode
    - message

▼ Get a single ticket

Method: GET

Access: Private

Role: Admin

Path: /tickets/:id

Response:

- 200
    - statusCode

        - data
            - id
            - userId
            - ticketNumber
            - name
            - email
            - phone
            - ticketSubject
            - type
            - timestamp
    - 404
        - statusCode
        - message

▼ Create a ticket

Method: POST

Access: Private

Role: Admin

Path: /tickets

Request Body:

- id
- userId
- ticketNumber
- name
- email
- phone

- ticketSubject
- type
- timestamp

Response

- 201
  - statusCode
  - message
  - data
    - id
    - userId
    - categoryId
    - invoiceNumber
    - dateOfCreation
    - dateSent
    - dateDue
    - tax
    - discount
    - amountDue
    - brief
    - status
    - timestamp
  - links
    - self
    - edit
    - delete
    - view

- 400
  - statusCode
  - message
  - data (Array of error messages)
    - field
    - message
- 401
  - statusCode
  - message

▼ Update a ticket

Method: PATCH

Access: Private

Role: Admin

Path: /tickets/:id

Request Body:

- id
- userId
- ticketNumber
- name
- email
- phone
- ticketSubject
- type
- timestamp

Response

- 200

    - statusCode
    - message
    - invoice data
        - id
        - userId
        - categoryId
        - invoiceNumber
        - dateOfCreation
        - dateSent
        - dateDue
        - tax
        - discount
        - amountDue
        - brief
        - status
        - timestamp
    - links
        - self
- 404
    - statusCode
    - message
- 400
    - statusCode
    - message
    - data (Array of error messages)
        - field

        - message
    - 401
        - statusCode
        - message

▼ Delete a ticket

Method: Delete

Access: Private

Role: Admin

Path: /tickets/:id

Response:

- 204
- 404
    - statusCode
    - message
- 401
    - statusCode
    - message

▼ Report

▼ Get Bid Reports

Method: GET

Access: Private

Role: Admin

Path: /reports/bids

Query:

- timeFrame (default DAILY) - the type of Time Frame, it could be either DAILY, WEEKLY, MONTHLY, or YEARLY

Response:

- 200
  - statusCode
  - data
    - bidStatus
    - date
    - total
- 400
  - statusCode
  - message

▼ Get Bid Category Reports

Method: GET

Access: Private

Role: Admin

Path: /reports/bids/category

Response:

- 200
  - statusCode
  - data
    - category
    - total
- 400
  - statusCode
  - message

▼ Get Buyers & Sellers Reports

Method: GET

Access: Private

Role: Admin

Path: /reports/bids/buyer-seller

Query:

- role (default BUYER) - the type of Role , it could be either BUYER or SELLER

Response:

- 200
  - statusCode
  - data
    - stats
    - total
- 400
  - statusCode
  - message

▼ Get Buyer Seller Summary Reports

Method: GET

Access: Private

Role: Admin

Path: /reports/bids/buyer-seller-summary

Query:

- timeFrame (default DAILY) - the type of Time Frame, it could be either DAILY, WEEKLY, MONTHLY, or YEARLY
- role (default BUYER) - the type of Role , it could be either BUYER or SELLER

Response:

- 200

        - statusCode
        - data
            - stats
            - date
            - total
    - 400
        - statusCode
        - message

▼ Get Earnings Reports

Method: GET

Access: Private

Role: Admin

Path: /reports/earnings

Query:

- timeFrame (default DAILY) - the type of Time Frame, it could be either DAILY, WEEKLY, MONTHLY, or YEARLY

Response:

- 200
    - statusCode
    - data
        - total
- 400
    - statusCode
    - message

▼ Get Invoice category Reports

Method: GET

Access: Private

Role: Admin

Path: /reports/earnings/category

Query:

- timeFrame (default DAILY) - the type of Time Frame, it could be either DAILY, WEEKLY, MONTHLY, or YEARLY

Response:

- 200
  - statusCode
  - data
    - category
    - total
- 400
  - statusCode
  - message