# 4. System Specification

## 4.1 Technical Requirements

* Compatibility:
  + Network Protocols: Ensure the system supports communication protocols used by the campus network, such as HTTP/HTTPS, TCP/IP, etc.
  + Network Security: Align with the campus network security policies, including firewall rules, VPN access, and intrusion detection systems.
  + API Integration: The system must be able to interact with the campus's existing payment gateways via APIs, handling Alipay, WeChat Pay, and bank transfers.
  + Data Formats: Ensure the system can process and exchange data formats used within the campus, such as CSV, XML, or JSON.
  + Operating Systems: Compatibility issues within Android and iOS systems.
* Security:
* Data Encryption: Use strong encryption standards (such as AES) to protect stored and transmitted data. Encrypt sensitive information, such as Personally Identifiable Information (PII) and payment details.
* User Authentication: Implement Multi-Factor Authentication (MFA) to add security layers, ensuring that only authorized users can access the system.
* Access Permissions: Set different system permissions for users and administrators to ensure the security of the system's operation and user data.
* Payment Security: Comply with the Payment Card Industry Data Security Standard (PCI DSS). Use secure payment gateways and third-party payment services to ensure the security of the payment process.
* Network Security: Use firewalls and Intrusion Detection Systems (IDS) to protect network boundaries. Implement network isolation and segmentation to reduce potential security threats.
* Data Backup and Recovery: Regularly back up data and ensure the security and integrity of the backups. Develop a disaster recovery plan to quickly recover in case of data loss or system failure.
  + Reliability:
  + Stability: Ensure the system does not crash under high load and high data transfer operations, which could impair user experience or even cause user financial losses.
  + Fault Handling:
  + System Failure: Implement fault transfer mechanisms to avoid service interruptions due to single point of failure.
  + User-end Failure: For issues such as incorrect data input and user operation errors, the system should provide timely feedback and instruct users on how to make changes.
* User Interface:
  + Intuitive Design: Design an intuitive user interface that allows users to easily find and use functions.
  + Consistency: Maintain consistency in interface elements and operations to reduce the learning curve for users.
  + Feedback: Provide timely user feedback to let users know if their operations are successful.
  + Aesthetics: The interface should have a professional appearance and feel, making users enjoy the process of using the system.
  + User Testing: Conduct user testing, collect feedback, and adjust according to user needs and preferences.

## 4.2 Performance Metrics

* + Response Time:
  + Optimize response time and user experience by taking various measures such as code optimization, database indexing, caching, load balancing, hardware upgrades, network optimization, and asynchronous processing.
  + Maximum Concurrent Users:
  + Improve concurrent processing capacity by expanding server resources, using load balancing, and optimizing database connections to ensure the system can stably respond to user requests during peak times.
  + Transaction Processing Capacity:
  + High transaction processing capacity is key to ensuring user experience and business continuity for payment systems. By optimizing transaction processing processes and using efficient database designs and indexing, we can enhance processing speed.
  + Mean Time to Recovery (MTTR):
  + A low MTTR can minimize system downtime and reduce the impact on business. We plan to establish a disaster recovery plan, using redundant systems and quick switching mechanisms to minimize the time to recover from faults.
  + Load Testing:
  + We need to test the system's performance under the expected maximum load. Load testing can verify whether the system can meet performance requirements and identify potential performance bottlenecks. Finally, adjust the system according to the test results, increase resources, or optimize code.
  + Performance Monitoring:
  + Use professional monitoring tools, set alert thresholds, and quickly locate and resolve issues. When performance degradation or system abnormalities are detected, ensure the system can respond quickly, resolve issues in a timely manner, and ensure a good user experience.

|  |
| --- |
| Figure x Data flow graph |