

Technical requirements

1. Use Spring MVC / Hibernate ORM / MySQL / PostgreSQL
2. Frontend Framework
 - React JS (optional)
 - Bootstrap
3. Cover all changes with JUnit tests
4. Add integration tests (*optional*)
5. Include code linters
6. For models, try to use:
 - Hibernate / JPA Single Table Inheritance
 - Polymorphic associations
 - Scopes
 - Validations and custom validator object, if necessary
 - Factory pattern
 - Encapsulate some logic in packages
7. For controllers, try to:
 - Keep them **thin**
 - Encapsulate business logic in service objects
8. Try to showcase background and cron jobs
9. Dockerize the Application (*optional*)
 - Create the application in the Docker environment
 - Use application and database containers
 - Use Docker compose - <https://docs.docker.com/compose>

Payment System Task

1. Relations:
 - 1.1. Ensure you have merchant and admin user roles (UI)

1.2. Merchants have many payment transactions of different types

1.3. Transactions are related (belongs_to)

- You can also have follow/referenced transactions that refer/depend to/on the initial transaction
 - Authorize Transaction -> Charge Transaction -> Refund Transaction
 - Authorize Transaction -> Reversal Transaction
 - Only **approved** or **refunded** transactions can be referenced, otherwise the status of the submitted transaction will be created with **error** status
- Ensure you prevent a merchant from being deleted unless there are no related payment transactions

2. Models:

2.1. Merchant: `name`, `description`, `email`, `status` (**active**, **inactive**), `total_transaction_sum`

2.2. Transaction: `uuid`, `amount`, `status` (**approved**, **reversed**, **refunded**, **error**), `customer_email`, `customer_phone`, `reference_id`

- Use validations for: `uuid`, `amount` > 0, `customer_email`, `status`
- Transaction Types
 - **Authorize transaction** - has `amount` and used to hold customer's amount
 - **Charge** transaction - has `amount` and used to confirm the amount is taken from the customer's account and transferred to the merchant
 - The merchant's total transactions amount has to be the sum of the **approved Charge** transactions
 - **Refund** transaction - has `amount` and used to reverse a specific amount (whole amount) of the Charge Transaction and return it to the customer
 - Transitions the **Charge** transaction to status **refunded**
 - The **approved Refund** transactions will decrease the merchant's total transaction amount
 - **Reversal** transaction - has no `amount`, used to invalidate the Authorize Transaction
 - Transitions the **Authorize** transaction to status **reversed**

3. Inputs and tasks:

- 3.1. Imports new merchants and admins from **CSV** (rake task)
- 3.2. A background Job for deleting transactions older than an hour (cron job)
- 3.3. Accepts payments using `XML / JSON` API (single point **POST** request)
 - Include API authentication layer (Basic authentication, OAuth or **JWT** authentication)
 - No transactions can be submitted unless the merchant is in **active** state
4. Presentation:
 - 4.1. Display, edit, destroy merchants
 - 4.2. Display transactions

Task submission

1. Add the task to a GitHub/Bitbucket/GitLab repo - either a public or a private one. Organize the Git commits the following way:
 - 1.1. Initial commit with all changes not directly related to the task - the newly installed rails app, `.gitignore` file, etc.
 - 1.2. All subsequent commits should be logically organized reflecting the steps you've taken developing the application
 - Neither one large commit with all changes nor a multitude of smaller commits for every little tiny change.
2. If for some reason you can't provide a GitHub/Bitbucket/GitLab repo, please, at least include the `.git` folder.
3. Document your code where needed and add a short `README` .