Okay, I need to help design an approval workflow API for a payment disbursement application using Spring Boot. The user mentioned that the approval process has three tiers based on the payment amount. Let me break down the requirements first.

The tiers are:

- Tier 1: Amount < $50,000

- Tier 2: $50,000 ≤ Amount < $1,000,000

- Tier 3: $1,000,000 ≤ Amount ≤ $3,000,000

So, when a payment request is created, the system needs to determine which approvers are required based on the amount. Then, each approver in the tier needs to approve it sequentially. Maybe Tier-1, then Tier-2, then Tier-3 if applicable. Wait, actually, the tiers are hierarchical? Like, if the amount is over $1M, does it need all three tiers or just Tier-3? The user said "based on amount," so probably each payment is assigned to a single tier. Wait, no—maybe if it's over $1M, it goes to Tier-3, but if between 50k and 1M, Tier-2, etc. So each payment request is assigned to one tier, and that tier's approvers need to approve it. But the user might have meant that each tier's approval is required up to the necessary level. Hmm, the original question says "needs to be approved by Tier-1, Tier-2 and Tier-3 approvers based on amount." Wait, maybe the tiers are cumulative? For example, a payment over 1M needs Tier-3 approval, but maybe also Tier-2 and Tier-1? That doesn't make sense. Probably, the payment is routed to the appropriate tier's approvers. For example, a $100k payment would go to Tier-2, and once approved by Tier-2, it's done. But the user might have intended that higher tiers require approvals from all lower tiers as well. Need to clarify that.

But the user's description says: Tier-1 for <50k, Tier-2 for 50k-1M, Tier-3 for 1M-3M. So each payment is assigned to one tier, and that tier's approvers must approve. However, the term "Tier" might imply that multiple tiers need to approve. For example, a payment in Tier-3 might require approvals from all three tiers. But that's not clear. The user might need to have a hierarchy where each tier's approval is required in sequence. For example, a Tier-3 payment first goes to Tier-1, then Tier-2, then Tier-3. But that might not be the case. The user's exact words: "needs to be approved by Tier-1, Tier-2 and Tier-3 approvers based on amount." Hmm, perhaps that the approval hierarchy depends on the amount. So, if the amount is in Tier-3, then all three tiers need to approve. But that seems odd. Alternatively, each tier represents a level of approval, so higher amounts require higher tiers. So a payment of $1.5M requires Tier-3 approval only. That seems more likely.

Assuming that each payment is assigned to a single tier based on the amount, and only the approvers of that tier need to approve. So the workflow would be:

1. Payment request created with amount X.

2. Determine the tier based on X.

3. Assign the request to the corresponding tier's approvers.

4. Once approved by the required approvers in that tier, the payment is processed.

But perhaps the tiers are sequential. For example, a Tier-3 payment must be approved by Tier-1, then Tier-2, then Tier-3. That seems more like a hierarchical approval process where each higher tier requires lower tiers' approvals. But the user's initial description may not clarify that. The original question says "based on amount," so maybe the required tiers are only the one corresponding to the amount. However, the user might have intended a step-up approval where higher amounts require more approvals. For example, Tier-1 approval for up to 50k, Tier-1 and Tier-2 for 50k-1M, and all three tiers for over 1M. But the user's tiers are defined as:

- Tier-1: <50k

- Tier-2: 50k-1M

- Tier-3: 1M-3M

So, perhaps each tier is responsible for a range, and only one approval from that tier is needed. For example, a payment of $75k would go to Tier-2, and once approved by a Tier-2 approver, it's approved.

But the user says "approval workflow," which often implies multiple steps. So maybe each tier represents a level of approval, and higher amounts require more approvals. For example:

- If amount <50k: only Tier-1 approval needed.

- If between 50k and 1M: Tier-1 and Tier-2 approvals needed.

- If between 1M and 3M: all three tiers need to approve.

But that would mean that for a $100k payment, both Tier-1 and Tier-2 must approve. But the user's original description might not have specified that. The user's initial statement says "based on amount. Tier-1 is for payment amount <50000 USD, Tier-2 is <1M USD and >50k USD and Tier-3 is for 3M USD and >1M USD." So, the tiers are non-overlapping ranges, and each payment falls into exactly one tier. Therefore, each payment is assigned to a single tier, and that tier's approvers must approve it. So the workflow would involve the approvers within that tier.

But the problem is that the user mentions "Tier-1, Tier-2 and Tier-3 approvers based on amount," which might imply that higher tiers require multiple approvals. Alternatively, maybe each tier has different approvers, and the payment is routed to the appropriate tier's approvers.

So, the first step is to clarify the workflow logic. Let me assume that each payment is assigned to a single tier based on the amount, and that tier's approvers must approve it. For example, if a payment is $1.5M, it's Tier-3, and needs approval from Tier-3 approvers. But how many approvers are needed per tier? Maybe each tier has a group of approvers, and any one of them can approve. Or perhaps it requires multiple approvals within the tier. The user hasn't specified that, so I'll assume that each tier has one or more approvers, and the payment needs approval from one user in the respective tier.

Alternatively, maybe each tier requires a single approval from a user in that tier. So the workflow steps would be:

- Create payment request, determine tier, assign to that tier's approvers, once approved, process payment.

But perhaps the tiers are hierarchical, so higher tiers require lower tiers' approvals first. For example, a Tier-3 payment must first be approved by Tier-1, then Tier-2, then Tier-3. But the user's initial description doesn't mention that, so this might be an incorrect assumption.

Given the ambiguity, perhaps the best approach is to design a flexible approval workflow where each payment is assigned a sequence of approval steps based on the tier. For example, each tier corresponds to an approval step. So:

- For Tier-1 (amount <50k): only one approval step (Tier-1) is needed.

- For Tier-2 (50k ≤ amount <1M): two approval steps (Tier-1 and Tier-2).

- For Tier-3 (1M ≤ amount ≤3M): three approval steps (Tier-1, Tier-2, Tier-3).

But this may not be what the user intended. Alternatively, each tier is a separate group, and higher tiers require only their own approval. For example, Tier-3 payments only need approval from Tier-3 approvers, not the lower tiers.

The user needs to clarify this, but since they are asking for an API design, perhaps we should proceed with the assumption that each payment is assigned to a single tier, and that tier's approvers must approve it. So the workflow is:

1. User submits payment request with amount.

2. Determine the tier based on amount.

3. Assign the request to the corresponding tier's approvers.

4. Once approved by the required approvers (maybe one or more in the tier), the payment is processed.

But another possibility is that each tier is a sequential step. For example:

- For Tier-1: only Tier-1 approval needed.

- For Tier-2: first Tier-1 approval, then Tier-2.

- For Tier-3: Tier-1, Tier-2, then Tier-3.

This would mean that higher tiers require more approvals. This is a common approach in approval workflows where larger amounts require more levels of approval. So even if a payment is in Tier-3, it must go through all three tiers. This seems more likely to be the intended behavior.

So, based on that assumption, the workflow steps would be:

- For a payment of $30k (Tier-1): only Tier-1 approval needed.

- For $750k (Tier-2): Tier-1 approves, then Tier-2 approves.

- For $2M (Tier-3): Tier-1 approves, then Tier-2 approves, then Tier-3 approves.

If that's the case, then the API needs to manage a multi-step approval process where each step corresponds to a tier, and each step must be approved in sequence.

Now, moving to the API design. The Spring Boot application needs to handle creating payment requests, determining the required approval steps, managing the approval process (approvers, status transitions), and triggering the payment once fully approved.

Possible API endpoints:

1. POST /api/payments - Create a new payment request. The request body includes the amount, details, etc. The response includes the payment ID and initial status.

2. GET /api/payments/{id} - Retrieve payment request details, including current approval status.

3. POST /api/payments/{id}/approvals - Submit an approval for a payment. The approver's tier must match the current required tier. The request body includes the approver's user ID and approval decision (approve/reject).

4. GET /api/payments/{id}/approvals - Get the list of approvals for the payment request.

5. Possibly a PATCH or PUT endpoint to update payment details (if allowed before approval).

Database Design:

Tables needed:

- PaymentRequest: id, amount, status (PENDING, APPROVED, REJECTED, PROCESSED), created\_by, created\_date, etc.

- ApprovalStep: id, payment\_id, tier (1, 2, 3), status (PENDING, APPROVED, REJECTED), required\_approvers (maybe), approver\_user\_id (who approved), comments, date.

But since each step is a tier, and each tier may require one or more approvals, the ApprovalStep table can track each tier's approval. For example, a payment in Tier-3 would have three ApprovalStep records, one for each tier. Each step must be approved before moving to the next.

Wait, but if the steps are sequential, then the approval process would be:

- When a payment is created, the first ApprovalStep (Tier-1) is created as PENDING.

- Once Tier-1 is approved, create Tier-2 step.

- Once Tier-2 is approved, create Tier-3 step (if needed).

So the ApprovalStep table would have a payment\_id, tier, status, approver\_user\_id, date, etc.

But how to model the required steps based on the tier. For example, a Tier-3 payment requires steps for Tier-1, Tier-2, and Tier-3.

So the workflow engine needs to determine how many steps are required based on the amount.

Alternative approach: When a payment is created, determine the required tiers (steps) and create all necessary ApprovalStep entries upfront. For example:

- If the payment is Tier-1, create one step (Tier-1).

- If Tier-2, create two steps (Tier-1, Tier-2).

- If Tier-3, create three steps (Tier-1, Tier-2, Tier-3).

Each step must be approved in order. Each step has a status (PENDING, APPROVED, REJECTED). Once all steps are approved, the payment can be processed.

But how to track the current step. Maybe the payment has a current\_step field that points to the current tier. For example:

PaymentRequest: current\_step (1, 2, 3), status (if any step is rejected, status is REJECTED; if all steps approved, status is APPROVED).

But this may complicate things. Alternatively, each step is processed sequentially. After the first step is approved, the next step is triggered.

So when a payment is created:

- Determine the max tier needed (based on amount).

- Create ApprovalStep records for tiers 1 up to max tier, each with status PENDING.

But they must be approved in order. So the first step is Tier-1. Once approved, Tier-2 is next, etc.

Alternatively, when creating the payment, only create the first step (Tier-1), and once that's approved, create the next step (if needed). This approach may be better for dynamic step creation.

But let's outline the database schema:

PaymentRequest:

- id (PK)

- amount

- currency (if needed)

- status (PENDING, APPROVED, REJECTED, PROCESSED)

- created\_by (user ID)

- created\_date

- updated\_date

ApprovalStep:

- id (PK)

- payment\_id (FK to PaymentRequest)

- tier (1, 2, 3)

- status (PENDING, APPROVED, REJECTED)

- approver\_user\_id (nullable, FK to users)

- comments

- created\_date

- updated\_date

So for a Tier-3 payment, the ApprovalSteps would be:

1. Tier 1: PENDING

2. Tier 2: PENDING (only after Tier 1 is approved)

3. Tier 3: PENDING (only after Tier 2 is approved)

But how to model this. Maybe when a payment is created, we determine the max tier and create the first step (Tier-1). Once Tier-1 is approved, check if the max tier is higher than current tier. If yes, create the next step (Tier-2). Repeat until all tiers are approved.

Alternatively, create all steps upfront but mark them as not started until the previous tier is approved. But the initial approach of dynamically creating steps as previous tiers are approved might be more efficient.

So the process would be:

1. Payment is created with amount X. Determine the max tier (1, 2, or 3) based on X.

2. Create the first ApprovalStep for Tier-1, status PENDING.

3. When an approver from Tier-1 approves, check if max tier > 1. If yes, create the next ApprovalStep for Tier-2, status PENDING.

4. Repeat for Tier-3 if needed.

This way, steps are created dynamically as needed.

Now, the API endpoints need to handle submitting approvals and progressing through the steps.

For example, when a POST to /api/payments/{id}/approvals is made, the system checks:

- The user is an approver for the current tier (Tier-1, 2, or 3).

- The current step for the payment is in PENDING status.

If the approval is successful, the step is marked as approved, and the next step (if any) is created. If it's the last step, the payment is marked as approved overall.

If any step is rejected, the payment is rejected.

Now, to manage approvers, we need a way to determine which users are in each tier. So perhaps a User table with a tier attribute, or a separate ApproverTier table that maps users to tiers.

User:

- id (PK)

- username

- email

- role

- etc.

ApproverTier:

- user\_id (FK to User)

- tier (1, 2, 3)

So when checking if a user can approve a step, we verify that the user is in the tier corresponding to the current step.

Now, the API flow:

1. Create payment:

POST /api/payments

Request body: { "amount": 750000, "details": "..." }

Response: { "id": 123, "status": "PENDING", "currentTier": 1 }

2. The system creates an ApprovalStep for Tier-1, status PENDING.

3. A Tier-1 approver logs in, sees pending requests, and approves.

POST /api/payments/123/approvals

Request body: { "approverUserId": 456, "decision": "APPROVE", "comments": "Looks good" }

The system checks if user 456 is a Tier-1 approver. If yes, approves the Tier-1 step. Then checks the max tier for the payment. Since amount is 750k, max tier is 2. So create Tier-2 ApprovalStep, status PENDING.

4. A Tier-2 approver approves.

POST /api/payments/123/approvals

Request body: { "approverUserId": 789, "decision": "APPROVE" }

Now, since max tier is 2 and all steps are approved, the payment status becomes APPROVED, and can be processed.

If the payment amount was 1.5M, max tier is 3. After Tier-2 approval, create Tier-3 step, and so on.

Now, error handling:

- If a user tries to approve a step that's not current (e.g., Tier-2 when current step is Tier-1), reject.

- If a user is not in the correct tier for the current step, reject.

- If a payment is already approved or rejected, reject further approvals.

Possible states for the payment:

- PENDING: waiting for approvals.

- APPROVED: all required tiers approved.

- REJECTED: any tier rejected.

- PROCESSED: payment has been disbursed (after approval).

Now, regarding tools to avoid reinventing the wheel:

- Camunda: A BPMN (Business Process Model and Notation) workflow engine that can integrate with Spring Boot. It allows modeling the approval workflow as a BPMN diagram, handling state transitions, user tasks, etc. This could manage the approval steps, user task assignments, and transitions.

- Activiti: Another BPMN workflow engine, now forked as Flowable. Similar to Camunda.

- Spring State Machine: A Spring project that allows defining state machines for complex state transitions. This is lighter than Camunda but requires more manual coding.

Using Camunda or Flowable would provide a visual workflow designer and handle persistence, retries, etc. But it adds complexity. For a simple three-tier approval, maybe Spring State Machine is sufficient.

Alternatively, for a simple case, custom code can manage the approval steps without a full BPM engine. But for extensibility, using a workflow engine might be better.

So, the recommendation would be:

- If the workflow is simple and unlikely to change, implement it with custom code and a database model as outlined.

- If the workflow might become more complex (e.g., adding more tiers, conditional steps, parallel approvals), use Camunda or Flowable.

But given the user's scenario, a custom implementation might suffice. However, using a workflow engine could save development time and provide flexibility.

Now, designing the Spring Boot services:

1. PaymentService: Handles creating payments, determining the required tiers, and managing the approval process.

2. ApprovalService: Handles approval submissions, validation, and progressing through tiers.

3. UserService: Manages user roles and approver tiers.

The key components:

- When a payment is created, determine the max tier.

- Create the initial ApprovalStep (Tier-1).

- When an approval is submitted, validate the user's tier, update the step, and progress to the next tier if needed.

- Once all tiers are approved, mark the payment as approved.

Code outline:

PaymentService:

public class PaymentService {

@Autowired

private PaymentRepository paymentRepository;

@Autowired

private ApprovalStepRepository approvalStepRepository;

@Autowired

private UserService userService;

public PaymentRequest createPayment(BigDecimal amount, User creator) {

PaymentRequest payment = new PaymentRequest();

payment.setAmount(amount);

payment.setStatus(PaymentStatus.PENDING);

payment.setCreatedBy(creator);

payment.setCreatedDate(LocalDateTime.now());

payment = paymentRepository.save(payment);

int maxTier = determineMaxTier(amount);

ApprovalStep firstStep = new ApprovalStep();

firstStep.setPayment(payment);

firstStep.setTier(1);

firstStep.setStatus(ApprovalStatus.PENDING);

firstStep.setCreatedDate(LocalDateTime.now());

approvalStepRepository.save(firstStep);

return payment;

}

private int determineMaxTier(BigDecimal amount) {

if (amount.compareTo(new BigDecimal("50000")) < 0) {

return 1;

} else if (amount.compareTo(new BigDecimal("1000000")) < 0) {

return 2;

} else {

return 3;

}

}

}

ApprovalService:

public class ApprovalService {

@Autowired

private ApprovalStepRepository approvalStepRepository;

@Autowired

private PaymentRepository paymentRepository;

@Autowired

private UserService userService;

public void processApproval(Long paymentId, Long approverUserId, ApprovalDecision decision, String comments) {

PaymentRequest payment = paymentRepository.findById(paymentId)

.orElseThrow(() -> new PaymentNotFoundException(paymentId));

if (payment.getStatus() != PaymentStatus.PENDING) {

throw new PaymentNotPendingException(paymentId);

}

ApprovalStep currentStep = approvalStepRepository.findByPaymentIdAndStatus(paymentId, ApprovalStatus.PENDING)

.orElseThrow(() -> new NoPendingStepsException(paymentId));

if (!userService.isUserApproverForTier(approverUserId, currentStep.getTier())) {

throw new UnauthorizedApprovalException(approverUserId, currentStep.getTier());

}

currentStep.setStatus(decision.getStatus());

currentStep.setApproverUserId(approverUserId);

currentStep.setComments(comments);

currentStep.setUpdatedDate(LocalDateTime.now());

approvalStepRepository.save(currentStep);

if (decision == ApprovalDecision.REJECT) {

payment.setStatus(PaymentStatus.REJECTED);

paymentRepository.save(payment);

return;

}

int maxTier = determineMaxTier(payment.getAmount());

if (currentStep.getTier() < maxTier) {

ApprovalStep nextStep = new ApprovalStep();

nextStep.setPayment(payment);

nextStep.setTier(currentStep.getTier() + 1);

nextStep.setStatus(ApprovalStatus.PENDING);

nextStep.setCreatedDate(LocalDateTime.now());

approvalStepRepository.save(nextStep);

} else {

payment.setStatus(PaymentStatus.APPROVED);

paymentRepository.save(payment);

}

}

}

UserService:

public class UserService {

@Autowired

private ApproverTierRepository approverTierRepository;

public boolean isUserApproverForTier(Long userId, int tier) {

return approverTierRepository.existsByUserIdAndTier(userId, tier);

}

}

This is a basic outline. The ApprovalStepRepository would need a method to find the current pending step for a payment.

Potential issues:

- Concurrency: If two approvers try to approve the same step at the same time. Using @Transactional and optimistic locking can help.

- Notifications: After a step is approved, the next tier's approvers need to be notified. This could be done via email, in-app notifications, etc. A messaging system or Spring Events could be used.

- Security: Ensure that only approvers from the correct tier can approve steps. Use Spring Security with roles or custom permissions.

Tools recommendation:

- Camunda/Flowable: For complex workflows, but might be overkill for this scenario.

- Spring State Machine: To model the approval process as a state machine. This can help manage the state transitions more cleanly.

- Lombok: To reduce boilerplate code in entities.

- MapStruct: For DTO mapping.

- Spring Security: For authentication and authorization.

- Hibernate Envers: For auditing changes to payment requests and approval steps.

- Quartz Scheduler or Spring Scheduler: For any scheduled tasks, like escalating pending approvals.

Example with Spring State Machine:

Define states as the tiers, and transitions on approval.

But for simplicity, the custom code approach might be sufficient.

Testing:

- Unit tests for tier determination.

- Integration tests for the approval flow, checking that each step progresses correctly.

- Security tests to ensure only authorized approvers can approve steps.

Additional considerations:

- Timeouts: If an approval step is pending for too long, escalate or send reminders.

- History: Tracking all approval steps and decisions.

- API documentation with Swagger/OpenAPI.

- Logging and monitoring for the approval process.

In conclusion, the API design involves creating endpoints to manage payment requests and their approval steps, with services that handle the business logic of progressing through tiers. Using a workflow engine like Camunda could provide more flexibility but adds complexity. For the given requirements, a custom implementation with careful state management is feasible.

To design an approval workflow API for your payment disbursement system using Spring Boot, follow this structured approach:

**1. API Endpoints**

Design RESTful endpoints to handle payment requests and approvals:

* **POST /api/payments**  
  Create a payment request. Validate the amount and determine the required approval tiers.

json

Copy

Request Body: { "amount": 750000, "details": "Payment for vendor", "currency": "USD" }

Response: { "id": 1, "status": "PENDING", "currentTier": 1 }

* **GET /api/payments/{id}**  
  Retrieve payment details and approval status.

json

Copy

Response: { "id": 1, "amount": 750000, "status": "PENDING", "currentTier": 1 }

* **POST /api/payments/{id}/approve**  
  Submit an approval/rejection for the current tier. Validate approver authority.

json

Copy

Request Body: { "approverId": 456, "decision": "APPROVE", "comments": "Approved by Tier-1" }

* **GET /api/payments/{id}/approvals**  
  List all approval steps (history).

json

Copy

Response: [

{ "tier": 1, "status": "APPROVED", "approverId": 456, "timestamp": "2023-10-01T10:00:00" },

{ "tier": 2, "status": "PENDING", "approverId": null, "timestamp": null }

]

**2. Database Design**

Use these tables to model payments and approvals:

**payment\_request Table**

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| id | UUID/PK | Unique payment ID. |
| amount | DECIMAL(18,2) | Payment amount. |
| currency | VARCHAR(3) | Currency code (e.g., USD). |
| status | ENUM | PENDING, APPROVED, REJECTED. |
| created\_by | VARCHAR(255) | User who created the request. |
| created\_at | TIMESTAMP | Request creation time. |

**approval\_step Table**

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| id | UUID/PK | Unique approval step ID. |
| payment\_id | UUID/FK | Links to payment\_request. |
| tier | INT | Approval tier (1, 2, or 3). |
| status | ENUM | PENDING, APPROVED, REJECTED. |
| approver\_id | VARCHAR(255) | User who approved/rejected. |
| comments | TEXT | Approval remarks. |
| created\_at | TIMESTAMP | Step creation time. |
| updated\_at | TIMESTAMP | Last update time. |

**approver\_tier Table**

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| user\_id | VARCHAR(255) | Approver’s user ID. |
| tier | INT | Tier the user can approve (1, 2, 3). |

**3. Workflow Logic**

Implement the approval flow using this logic:

1. **Determine Required Tiers**:

java

Copy

if (amount < 50\_000) → Tier 1

else if (amount < 1\_000\_000) → Tiers 1 + 2

else → Tiers 1 + 2 + 3

1. **Sequential Approval**:  
   Each tier must approve in order. Tier 1 must approve before Tier 2, etc.
2. **State Management**:  
   Use payment\_request.status to track overall progress. Reject immediately if any tier rejects.

**4. Spring Boot Implementation**

**Key Components**

* **PaymentService**: Creates payments, manages approval steps, and updates statuses.
* **ApprovalService**: Validates approvers, processes approvals, and triggers transitions.
* **UserService**: Checks if a user is authorized to approve a tier.

**Example Code Snippets**

**Determine the Required Tiers**:

java

Copy

public int getMaxTier(BigDecimal amount) {

if (amount.compareTo(BigDecimal.valueOf(50\_000)) < 0) return 1;

else if (amount.compareTo(BigDecimal.valueOf(1\_000\_000)) < 0) return 2;

else return 3;

}

**Process Approval**:

java

Copy

@Transactional

public void approvePayment(UUID paymentId, String approverId) {

PaymentRequest payment = paymentRepository.findById(paymentId)

.orElseThrow(() -> new PaymentNotFoundException(paymentId));

ApprovalStep currentStep = approvalStepRepository.findByPaymentIdAndStatus(

paymentId, ApprovalStatus.PENDING

).orElseThrow(() -> new NoPendingStepsException(paymentId));

if (!userService.isApproverForTier(approverId, currentStep.getTier())) {

throw new UnauthorizedApprovalException("User cannot approve this tier");

}

currentStep.setStatus(ApprovalStatus.APPROVED);

currentStep.setApproverId(approverId);

approvalStepRepository.save(currentStep);

if (currentStep.getTier() < payment.getMaxTier()) {

createNextApprovalStep(payment, currentStep.getTier() + 1);

} else {

payment.setStatus(PaymentStatus.APPROVED);

paymentRepository.save(payment);

}

}

**5. Tools & Libraries**

* **Camunda/Flowable**: For complex workflows (BPMN support).
* **Spring State Machine**: For lightweight state transitions.
* **Lombok**: Simplify entity/DTO code.
* **Spring Security**: Secure endpoints and authorize approvers.
* **Swagger/OpenAPI**: Document the API.

**6. Recommendations**

* **Use Camunda** if you anticipate complex future requirements (e.g., dynamic tiers, parallel approvals).
* **Implement Notifications**: Use Spring Events or Azure Service Bus to alert approvers.
* **Add Auditing**: Track changes with Hibernate Envers or Spring Data JPA Auditing.
* **Security**: Integrate Azure AD for authentication and role-based access control (RBAC).

This design ensures a scalable and maintainable approval workflow while leveraging Spring Boot’s ecosystem.

Are there any github projects with similar spring boot examples

Thought for 0 seconds

The server is busy. Please try again later.

Can u generate above solution using Spring statemachine

Thought for 0 seconds

The server is busy. Please try again later.

New chat