Validate Invariants Between Data Models

Google Summer Of Code

Ankita Saxena

Name of the project

Validate Invariants between data model

Why I’m interested in working with Oppia?

Oppia serves the purpose to provide free and standard education to all, develop a world in which each student has access to education and can educate themselves in the most effective way. I’m driven by the idea Oppia is working on and so would love to be a part of such community.
I’m really impressed by the kind of mentorship I’ve received over past couple of months. The mentors are kind of awesome -- my PR’s get reviewed in no time, regular meetings, documented work and freedom to form a team and lead as I wish has also made my head clear that Oppia is the organization I would be working in, if given the opportunity.
The most important reason of why I am interested in working with Oppia is that I enjoy contributing code to Oppia and the past three months of contributing to Oppia has been a great learning experience for me.

What interests me about this project?

During my Outreachy intern, I implemented a style guide linter and I really liked testing the style guide stuff and making fixes. This project aims at validating Oppia datastore models and migrating them to a valid schema. It is challenging as well as an important improvement to the datastore models which makes this project really interesting. This project not only provides me an opportunity to test and improve the models but also a chance to improve my Python coding skills and learn about One off jobs and GAE Models.

Prior Experience

I have been coding in Python since last three years. I have done various course projects in python. Apart from the course projects, I participated in Outreachy winter round intern and implemented a style guide linter in python. The intern was a great learning experience for me and helped me to improve my coding skills in Python. The intern blog can be found here and my contributions for the intern can be found here. I also learnt a lot about contributing to open source and I learnt many skills from my mentor on how to write clean code and submit good PRs.
Apart from that, I have been contributing to Oppia since the past three months and have become quite familiar with the codebase. I have also added tests for exp_one_off_jobs in my initial PR (#5972) and have gained a fairly good knowledge of one off jobs and gae models by going through the documentation.
I have been investigating the project for the past several days and looking at my coding skills, testing skills and the information that I have currently, regarding this project, I feel I'm in a good position to take up this project.

Some of my PR's and Issues

Some of my PR's are:

1. PR #5972
2. PR #6015
3. PR #6530
4. PR #6170
5. PR #5997

I have also reported few issues since I've been contributing to Oppia:

1. Issue #6351
2. Issue #6251
3. Issue #6224

Apart from these issues and PR's I have been reviewing PR's of my fellow contributors too. I have also been leading the pre migration for Angular Migration project and it has been a great learning experience for me.

Project Prerequisites

List of models with their validation checks

Activity Models

Name	Purpose	Checks	Inherits checks from
ActivityReferencesModel	Model for list of activity references	• Model id -> can only be from the list `ALL_ACTIVITY_REFERENCE_LIST_TYPES` • activity_references list -> each item should follow the schema of ActivityReferences • Each item in activity_references -> dict • Keys in dict -> type and id • type -> string (can be either `constants.ACTIVITY_TYPE_EXPLORATION` or `constants.ACTIVITY_TYPE_COLLECTION`) • id -> string	BaseModel

Audit Models

Name	Purpose	Checks	Inherits Checks from
RoleQueryAuditModel	Model to store data of queries made through the admin panel	• user_id should be of an admin • Model id should be of the form `[user_id].[timestamp_in_sec].[intent].[random_number]` • The values used in model id should match with the corresponding user_id and intent	BaseModel

Base Models

Name	Purpose	Checks	Inherits checks from
BaseModel	Base model for all storage classes	• created_on <= last_updated <= current time	None
BaseCommitLogEntryModel	Base model for models that store commit logs	• commit_type should be either: 'create', 'revert', 'edit', 'delete' • post_commit_status should be either: 'public', 'private' (Add choices validation) • commit_cmds should have the parameters according to cmd name (Table) • post_commit_is_private is true if post_commit_status is private and vice versa	BaseModel
VersionedModel	Model to handle storage of version history	• `SNAPSHOT_METADATA_CLASS` is present and contains the fields: 'committer_id' (string), 'commit_type' (string -> 'create', 'revert', 'edit', 'delete'), 'commit_message'(string) and 'commit_cmds' (list of dicts) • Each dict in commit_cmds • `SNAPSHOT_CONTENT_CLASS` is present and contains the field 'content' (json) • `ALLOW_REVERT` -> boolean • If the version of the model is v then there should be v snapshot metadata models and v snapshot content models with id of the form `{instance_id}-{version_number}`	BaseModel
BaseSnapshotMetadataModel	Base class for Snapshot Metadata class	• commit_type should be either: 'create', 'revert', 'edit', 'delete' • committer_id should be a valid user id • commit_cmds should have the parameters according to cmd name (Table)	BaseModel
BaseSna	Base class for	None	BaseModel

pshotContentModel	Snapshot Content class		
BaseMapReduceBatchResultsModel	Base model for batch storage for MR jobs	• _use_cache -> boolean • _use_memcache -> boolean	BaseModel

Classifier Models

Name	Purpose	Checks	Inherits checks from
ClassifierTrainingJobModel	Model for storing classifier training jobs	• Model id should be of form: `{{exp_id}}.{{random_hash_of_16_chars}}` • next_scheduled_check_time >= current time • exp_id and exp_version correspond to a valid exploration version, for which state_name is a valid state	BaseModel
TrainingJobExplorationMappingModel	Model for mapping exploration attributes to a ClassifierTrainingJob	• Model id should be of type `{{exp_id}}.{{exp_version}}.{{utf8_encoded_state_name}}` • exp_id and exp_version correspond to a valid exploration version, for which state_name is a valid state	BaseModel

Collection Models

Name	Purpose	Checks	Inherits checks from
CollectionSnapshotMetadataModel	Model for collection snapshot metadata	None	BaseSnapshotMetadataModel

CollectionSnapshotContentModel	Model for collection snapshot content	None	BaseSnapshotContentModel
CollectionModel	Model for oppia collection	• language_code should be from `constants.ALL_LANGUAGE_CODES` • collection_contents should be a list of dicts with each dict having key 'exploration_id'. The value should correspond to a valid exploration • 1 <= schema_version <= Current schema version • CollectionModel should follow the schema for a Collection • For a model with version v, there should be v commit log models	VersionedModel
CollectionRightsSnapshotMetadataModel	Model for collection rights snapshot metadata	None	BaseSnapshotMetadataModel
CollectionRightsSnapshotContentModel	Model for collection rights snapshot content	None	BaseSnapshotContentModel
CollectionRightsModel	Model to store rights related to a collection	• Model id should be same as collection id • first_published_msec <= current time • For a model with version v, there should be v commit log models	VersionedModel
CollectionCommitLogEntryModel	Model to store commit logs to a collection	• Model id should be form `collection-{{COLLECTION_ID}}-{{COLLECTION_VERSION}}`	BaseCommitLogEntryModel

CollectionSummaryModel	Model to store collection summary	• Model id should be same as collection id • language_code should be from `constants.ALL_LANGUAGE_CODES` • collection_model_created_on <= collection_model_last_updated <= current time • Fields in this model should match the corresponding fields in CollectionModel and CollectionRightsModel • node_count should be equal to number of keys in collection_contents dict in CollectionModel	BaseModel

Config Models

Name	**Purpose**	**Checks**	**Inherits checks from**
ConfigPropertySnapshotMetadataModel	Model for config property snapshot metadata	None	BaseSnapshotMetadataModel
ConfigPropertySnapshotContentModel	Model for Config property snapshot content	None	BaseSnapshotContentModel
ConfigPropertyModel	Model to represent a config property	• Model id should be name of the config property	VersionedModel

Email Models

Name	Purpose	Checks	Inherits checks from
SentEmailModel	Records email sent from oppia to single user	• Model id should be of the form `[INTENT].[random hash]` • sent_datetime <= current time • recipient_id and sender_id should correspond to valid user ids	BaseModel
BulkEmailModel	Records email sent from oppia to multiple users	• Model id -> string of length 12	BaseModel
GeneralFeedbackEmailReplyToIdModel	Model to store unique_id for each <user, thread> combination	• Model id should be of form `[USER_ID].[THREAD_ID]` • USER_ID and THREAD_ID should correspond to valid user and feedback thread • length(reply_to_id) <= REPLY_TO_ID_LENGTH	BaseModel

Exploration Models

Name	Purpose	Checks	Inherits checks from
ExplorationSnapshotMetadataModel	Model for exploration snapshot metadata	None	BaseSnapshotMetadataModel
ExplorationSnapshotContentModel	Model for exploration snapshot content	None	BaseSnapshotContentModel
ExplorationM	Model for	• language_code should be from	VersionedMod

odel	oppia exploration	`constants.ALL_LANGUAGE_CODES` • 0 <= states_schema_version <= latest version • states dict should be non empty • param_specs dict -> keys = param_names and values = dict with single key obj_type => allowed object types • param_changes = list of dict => each dict should follow the schema for ParamChange • For a model with version v, there should be v commit log models and v state id mapping models • There should be one summary model • It should follow the schema of a exploration (Note: Due to this separate check is not required for param changes and state - they are validated in exploration validate function itself)	el
ExplorationRightsSnapshotMetadataModel	Model for exploration rights snapshot metadata	None	BaseSnapshotMetadataModel
ExplorationRightsSnapshotContentModel	Model for exploration rights snapshot content	None	BaseSnapshotContentModel
ExplorationRightsModel	Model to store rights related to an exploration	• Model id should be same as exploration id • first_published_msec <= current time • cloned_from should be a valid exploration id of another ExplorationModel • Each id in owner_ids, editor_ids, translator_ids, viewer_ids should correspond to a non-deleted UserSettingsModel id	VersionedModel
ExplorationCommitLogEntryModel	Model to store commit	• Model id should be of form `exploration-{{EXP_ID}}-{{EXP_VERSION}}`	BaseCommitLogEntryModel

	logs to an exploration		
ExpSummaryModel	Model to store exploration summary	• Model id should be same as exploration id • language_code should be from `constants.ALL_LANGUAGE_CODES` • exploration_model_created_on <= exploration_model_last_updated <= current time • first_published_msec <= current time • ratings should be a dict with keys as 1, 2, 3, 4, 5 and values should be non-negative ints representing counts for the keys • Model fields should match the corresponding fields in ExplorationModel and ExplorationRightsModel • Each id in owner_ids, editor_ids, translator_ids, viewer_ids should correspond to a non-deleted UserSettingsModel id	BaseModel
StateIdMappingModel	Model to map each exploration version's state to unique id	• The state_name_to_ids dict must contain the same state names as those in the ExplorationModel • largest_state_id_used should be largest id in state_name_to_ids dict • Model id should be of the form `{exp_id}.{exp_version}`	BaseModel

Feedback Models

Name	Purpose	Checks	Inherits checks from
GeneralFeedbackThreadModel	Model for feedback thread for an entity	• Model id should be of the form `[ENTITY_TYPE].[ENTITY_ID].[GENERATED_STRING]` • entity_type can be exploration or topic • entity_id should be a valid exploration_id or topic_id respectively • message_count should match the number of GeneralFeedbackMessageModel for the thread • last_updated <= current time	BaseModel
GeneralFeedbackMessageModel	Model for feedback messages	• Model id should be of the form `[THREAD_ID].[MESSAGE_ID]`	BaseModel
GeneralFeedbackThreadUserModel	Model for storing ids of messages in a thread read by the user	• Model id should be of form `[user_id].[thread_id]` • message_ids_read_by_user is <= total number of message for that thread_id	BaseModel
FeedbackAnalyticsModel	Model for storing feedback thread analytics for an exploration	• Model id should be same as exploration id which should correspond to a valid exploration	BaseMapReduceBatchResultsModel
UnsentFeedbackEmailModel	Model for storing feedback messages that need to be sent to creators	• Model id should be same as the user id to whom the message is to be sent. The user_id should correspond to a valid user • feedback_message_references is a list of dict which should follow the schema of FeedbackMessageReference	BaseModel

File Models

Name	Purpose	Checks	Inherits checks from
FileMetadataSnapshotMetadataModel	Model for file metadata snapshot metadata	None	BaseSnapshotMetadataModel
FileMetadataSnapshotContentModel	Model for file metadata snapshot content	None	BaseSnapshotContentModel
FileMetadataModel	Model for file metadata	• Model id should be of the form `{exploration_id}/{filepath}` • exploration_id should correspond to a valid exploration	VersionedModel
FileSnapshotMetadataModel	Model for file snapshot metadata	None	BaseSnapshotMetadataModel
FileSnapshotContentModel	Model for file snapshot content	None	BaseSnapshotContentModel
FileModel	Model for file data	• Model id should be of the form `{exploration_id}/{filepath}` • exploration_id should correspond to a valid exploration	VersionedModel

Job Models

Name	Purpose	Checks	Inherits checks from
JobModel	Model for long running job	• Model id should be of the form `{job_type}-{current_time_str}-{random_int}` • time_queued_msec <= time_started_msec <= time_finished_msec <= current time • error should be None if status is not canceled or failed • output should be a list of strings if job has status completed else None • has_been_cleaned_up = deleted	BaseModel
ContinuousComputationModel	Model for continuous computation	• Model id should be same as continuous computation manager class • last_started_msec <= (last_finished_msec or last_stopped_msec) <= current time	BaseModel

Question Models

Name	Purpose	Checks	Inherits checks from
QuestionSnapshotMetadataModel	Model for question snapshot metadata	None	BaseSnapshotMetadataModel
QuestionSnapshotContentModel	Model for question snapshot content	None	BaseSnapshotContentModel
QuestionModel	Model for storing	• Model id should be a random hash of 16 chars	VersionedModel

	questions	• language_code should be from `constants.ALL_LANGUAGE_CODES` • question_state_data_schema_version <= latest version • It should follow the schema for Question • question_state_data should follow the schema of State • For a model with version v, there should be v commit log models	
QuestionSkillLinkModel	Model for storing question skill links	• Model id should be a random hash of 12 chars • question_id and skill_id should correspond to valid QuestionModel and SkillModel	BaseModel
QuestionCommitLogEntryModel	Model for storing commit logs to questions	• Model id should be of the form `question-{{QUESTION_ID}}-{{QUESTION_VERSION}}`	BaseCommitLogEntryModel
QuestionSummaryModel	Model for storing summary of a question	• Model id should be same as the question id • creator_id should correspond to a valid user • question_model_created_on <= question_model_last_updated <= current time • The fields in the model should match the corresponding fields in QuestionModel and QuestionRightsModel	BaseModel
QuestionRightsSnapshotMetadataModel	Model for question rights snapshot metadata	None	BaseSnapshotMetadataModel

QuestionRightsSnapshotContentModel	Model for question rights snapshot content	None	BaseSnapshotContentModel
QuestionRightsModel	Model for storing question rights	• Model id should be same as the question id • creator_id should correspond to a valid user	VersionedModel

Recommendation Models

Name	**Purpose**	**Checks**	**Inherits checks from**
ExplorationRecommendationsModel	Model to store list of recommended explorations similar to an exploration	• Model id should be same as exploration id for which recommendations are stored • Each recommended_exploration_id is also a valid exploration. • No recommended_exploration_id should be equal to the model instance ID	BaseMapReduceBatchResultsModel
TopicSimilaritiesModel	Model to store similarity between two topics	• Model key should be 'topics' • The content should be a 2d dict with topic being a key and the value should be similarity • keys can be only from `recommendation_services.RECOMMENDATION_CATEGORIES` • The values should be real number between 0.0 and 1.0 • The content dict should be symmetrical	BaseModel

Skill Models

Name	Purpose	Checks	Inherits checks from
SkillSnapshotMetadataModel	Model for skill snapshot metadata	None	BaseSnapshotMetadataModel
SkillSnapshotContentModel	Model for skill snapshot content	None	BaseSnapshotContentModel
SkillModel	Model for storing a skill	• misconceptions_schema_version <= latest version • Each item in misconceptions should follow the schema of Misconception • language_code should be from `constants.ALL_LANGUAGE_CODES` • skill_contents_schema_version <= latest version • skill_contents should follow the schema for SkillContents • superseding_skill_id should correspond to a valid skill or should be None • all_questions_merged should be true only if superseding_skill_id is not None and there are no questions present for the given skill id • For a model with version v, there should be v commit log models	VersionedModel
SkillCommitLogEntryModel	Model for storing commit logs to a skill	• Model id should be of the form `skill-{{SKILL_ID}}-{{SKILL_VERSION}}`	BaseModel
SkillSummaryModel	Model for storing summary of a skill	• The model id should be same as skill id • language_code should be from `constants.ALL_LANGUAGE_CODES` • skill_model_created_on <= skill_model_last_updated <= current time	BaseModel

		• Fields in the model should match the corresponding fields in SkillModel and SkillRightsModel	
SkillRights Snapshot Metadata Model	Model for storing skill righs snapshot metadata	None	BaseSnapshotMetadataModel
SkillRights SnapshotContentModel	Model for storing skill rights snapshot content	None	BaseSnapshotContentModel
SkillRights Model	Model for storing rights related to a skill	• Model id should be same as the skill id • creator_id should correspond to a valid user	Versioned Model

Statistics Models

Name	**Purpose**	**Checks**	**Inherits checks from**
StateCounterModel	Model to store following counts related to a state: • Number of times state was first entered in a reader session • Number of times state was entered second time or	• It should exist for each exploration and state pair (if an exploration has s states then there should be s such models for that exploration) • Model id should be of the form `[EXPLORATION_ID].[STATE_NAME]` • Exploration id should correspond to a valid exploration	BaseModel

	later in a reader session • Number of resolved answers • Number of active answers		
AnswerSubmittedEventLogEntryModel	Model to statistics of an answer submitted by a student	• Model id should be of the form `{{timestamp}:{exp_id}:{session_id}}` • event_schema_version <= latest version • exp_id and exp_version should correspond to a valid exploration • state_name should be present in StateIdMappingModel for the exp_id and exp_version	BaseModel
ExplorationActualStartEventLogEntryModel	Model for storing statistics when a student completes one state of exploration and moves to next	• Model id should be of the form `{{timestamp}:{exp_id}:{session_id}}` • event_schema_version <= latest version • exp_id and exp_version should correspond to a valid exploration • state_name should be present in StateIdMappingModel for the exp_id and exp_version	BaseModel
SolutionHitEventLogEntryModel	Model for storing statistics when a student triggers a solution	• Model id should be of the form `{{timestamp}:{exp_id}:{session_id}}` • event_schema_version <= latest version • exp_id and exp_version should correspond to a valid exploration • state_name should be present in StateIdMappingModel for the exp_id and exp_version	BaseModel
StartExplorationEventLogEntryModel	Model for storing statistics when a student starts an exploration	• Model id should be of the form `{{timestamp}:{exp_id}:{session_id}}` • event_schema_version <= latest version • event_type should be 'start'	BaseModel

		• exp_id and exp_version should correspond to a valid exploration • state_name should be present in StateIdMappingModel for the exp_id and exp_version	
MaybeLeaveExplorationEventLogEntryModel	Model for storing statistics when a reader attempting to leave the exploration without completing	• Model id should be of the form `{{timestamp}:{exp_id}:{session_id}}` • event_schema_version <= latest version • event_type should be 'leave' • exp_id and exp_version should correspond to a valid exploration • state_name should be present in StateIdMappingModel for the exp_id and exp_version	BaseModel
CompleteExplorationEventLogEntryModel	Model for storing statistics when the user reaches the terminal state of the exploration	• Model id should be of the form `{{timestamp}:{exp_id}:{session_id}}` • event_schema_version <= latest version • event_type should be 'complete' • exp_id and exp_version should correspond to a valid exploration • state_name should be present in StateIdMappingModel for the exp_id and exp_version	BaseModel
RateExplorationEventLogEntryModel	Model for storing statistics when a learner rates an exploration	• Model id should be of the form `{{timestamp}:{exp_id}:{session_id}}` • event_schema_version <= latest version • event_type should be 'rate_exploration' • exp_id and exp_version should correspond to a valid exploration	BaseModel
StateHitEventLogEntryModel	Model for storing statistics when a student reaches a particular state	• Model id should be of the form `{{timestamp}:{exp_id}:{session_id}}` • event_schema_version <= latest version • event_type should be 'state_hit' • exp_id and exp_version should correspond to a valid exploration • state_name should be present in	BaseModel

		StateIdMappingModel for the exp_id and exp_version	
StateCompleteEventLogEntryModel	Model for storing statistics when a student completes a particular state	• Model id should be of the form `{{timestamp}:{exp_id}:{session_id}}` • event_schema_version <= latest version • exp_id and exp_version should correspond to a valid exploration • state_name should be present in StateIdMappingModel for the exp_id and exp_version	BaseModel
LeaveForRefresherExplorationEventLogEntryModel	Model for storing statistics when a student leaves for a refresher exploration	• Model id should be of the form `{{timestamp}:{exp_id}:{session_id}}` • event_schema_version <= latest version • exp_id and exp_version should correspond to a valid exploration • state_name should be present in StateIdMappingModel for the exp_id and exp_version	BaseModel
ExplorationStatsModel	Model for storing analytics data for an exploration. It contains statistics data aggregated from version 1 to the version given in the key	• Model id should be of the form `{{exp_id}}.{{exp_version}}` • state_stats_mapping should contain should be a dict mapping each state name in the exploration to state stats dict. The keys for state stats dict should be: { `total_answers_count_v1`, `total_answers_count_v2`, `useful_feedback_count_v1`, `useful_feedback_count_v2`, `total_hit_count_v1`, `total_hit_count_v2`, `first_hit_count_v1`, `first_hit_count_v2`, `num_times_solution_viewed_v2`, `num_completions_v1`, `num_completions_v2`} • There should be v such models for a published exploration with version v • This model should not be present for an unpublished exploration • exp_id and exp_version should	BaseModel

		correspond to a valid exploration	
ExplorationIssuesModel	Model for storing the list of playthroughs for an exploration grouped by issues.	• Model id should be of the form `{{exp_id}}.{{exp_version}}` • unresolved_issues should be a list of dict where each dict represents an issue - it should follow the schema of issues from extensions/issues • There should be v such models for a published exploration with version v • This model should not be present for an unpublished exploration • exp_id and exp_version should correspond to a valid exploration	BaseModel
PlaythroughModel	Model for storing playthrough data	• Model id should be of the form `{{exp_id}}.{{random_hash_of_16_chars}}` • issue_customization_args should the match the format for the corresponding issue_type • actions should be a list of dict with each dict should follow the LearnerAction schema • exp_id and exp_version should correspond to a valid exploration	BaseModel
ExplorationAnnotationsModel	Batch model for storing MapReduce calculation output for exploration-level statistics	• Model id should be of the format `{[EXPLORATION_ID]:[EXPLORATION_VERSION]}` • EXPLORATION_ID and EXPLORATION_VERSION should correspond to a valid exploration	BaseMapReduceBatchResultsModel
StateAnswersModel	Model to store answers of a state	• Model id should be of the format `{[EXPLORATION_ID]:[EXPLORATION_VERSION]:[STATE_NAME]:[SHARD_ID]}` • EXPLORATION_ID and EXPLORATION_VERSION should correspond to a valid exploration • submitted_answer_list should be a list of answer dicts where each dict should follow the submitted answer schema • shard_count should be set only for	BaseModel

		shard_id = 0 • Other than shard_count, metadata fields should be same across shards • Number of shards should match the shard count (excluding the shard_id 0) • accumulated_answer_json_size_bytes should match the sum of dict sizes in submitted_answer_list • schema_version <= latest version • exploration_id, exploration_version and state_name should correspond to a valid exploration and state • state_name should be present in StateIdMappingModel for the exploration's version • interaction_id should match the interaction for the exploration version's state	
StateAnswersCalcOutputModel	Model to store output of calculations done on state answers	• Model id should be of the form `{[EXPLORATION_ID]:[EXPLORATION_VERSION]:[STATE_NAME]:[CALCULATION_ID]}` • EXPLORATION_ID and EXPLORATION_VERSION should correspond to a valid exploration • calculation_output_type should be either `stats_domain.CALC_OUTPUT_TYPE_ANSWER_FREQUENCY_LIST` or `stats_domain.CALC_OUTPUT_TYPE_CATEGORIZED_ANSWER_FREQUENCY_LISTS` • exploration_id, exploration_version and state_name should correspond to a valid exploration and state • state_name should be present in StateIdMappingModel for the exploration's version • The model should follow the schema of stats_domain.StateAnswersCalcOutput	BaseMapReduceBatchResultsModel

Additional common checks:

For each exploration, timestamp for:

- StartExplorationEventLogEntryModel >= ExplorationActualStartEventLogEntryModel >= CompleteExplorationEventLogEntryModel.
- StartExplorationEventLogEntryModel >= MaybeLeaveExplorationEventLogEntryModel.

For each state, timestamp for:

- StateHitEventLogEntryModel > StateCompleteEventLogEntryModel

Story Models

Name	**Purpose**	**Checks**	**Inherits checks from**
StorySnapshotMetadataModel	Model for story snapshot metadata	None	BaseSnapshotMetadataModel
StorySnapshotContentModel	Model for story snapshot content	None	BaseSnapshotContentModel
StoryModel	Model for storing story	• language_code should be from `constants.ALL_LANGUAGE_CODES` • story_contents dict should match the schema for StoryContents • The value for exploration_id key in `story_contents['node']` should correspond to a valid exploration • story_contents_schema_version <= latest version • For a model with version v, there should be v commit log models	VersionedModel
StoryCommitLogEntryModel	Model to store commit logs to story	• Model id should be of the form story-`{{STORY_ID}}-{{STORY_VERSION}}`	BaseCommitLogEntryModel
StorySummaryModel	Model to story summary	• language_code should be from `constants.ALL_LANGUAGE_CODES` • story_model_created_on <=	BaseModel

		story_model_last_updated <= current time • node_count should be equal to number of nodes in `story_contents['node']` in StoryModel • Fields in the model should match the corresponding fields in StoryModel and StoryRightsModel	
StoryRightsSnapshotMetadataModel	Model to store story rights snapshot metadata	None	BaseSnapshotMetadataModel
StoryRightsSnapshotContentModel	Model to store story rights snapshot content	None	BaseSnapshotContentModel
StoryRightsModel	Model to store rights related to a story	• Model id should be same as story id • manager_ids should correspond to valid users	VersionedModel

Suggestion Models

Name	**Purpose**	**Checks**	**Inherits checks from**
GeneralSuggestionModel	Model to store suggestions given by users	• The model should follow the schema of BaseSuggestion • target_id should correspond to a valid model for the target_type and target_version_at_submission • change_cmd should follow the schema for the corresponding suggestion_type • author_id should correspond to a valid user • score_category should be of the form `{SCORE_TYPE_CHOICES}.{suggestion subcategory}`	BaseModel

		• For suggestion_type 'Edit exploration state content' author_id should correspond to owners/editors of exp or users who can edit any exp	
ReviewerRotationTrackingModel	Model to keep track of the position in reviewer rotation	• Model id should be same as score category	BaseModel

Topic Models

Name	**Purpose**	**Checks**	**Inherits checks from**
TopicSnapshotMetadataModel	Model for topic snapshot metadata	None	BaseSnapshotMetadataModel
TopicSnapshotContentModel	Model for topic snapshot content	None	BaseSnapshotContentModel
TopicModel	Model for storing topics	• language_code should be from `constants.ALL_LANGUAGE_CODES` • canonical_name should be same as name in lowercase • No two models should have same canonical_name • canonical_story_ids and additional_story_ids should correspond to valid story ids and should be distinct • uncategorized_skill_ids should correspond to valid skill ids that should not be part of any subtopic • next_subtopic_id > all existing subtopic ids • subtopic_schema_version <= latest version • Each item in subtopics should	VersionedModel

		follow the schema of a Subtopic • For a model with version v, there should be v commit log models	
TopicCommitLogEntryModel	Model to store commit logs to a topic	• Model id should be of the form `topic-{{TOPIC_ID}}-{{TOPIC_VERSION}}`	BaseCommitLogEntryModel
TopicSummaryModel	Model to store topic summary	• Model id should be same as topic id • language_code should be from `constants.ALL_LANGUAGE_CODES` • canonical_name should be same as name in lowercase • topic_model_created_on <= topic_model_last_updated <= current time • Fields in the model should match the corresponding fields in TopicModel and TopicRightsModel	BaseModel
SubtopicPageSnapshotMetadataModel	Model for subtopic page snapshot metadata	None	BaseSnapshotMetadataModel
SubtopicPageSnapshotContentModel	Model for subtopic page snapshot content	None	BaseSnapshotContentModel
SubtopicPageModel	Model for storing subtopic pages	• language_code should be from `constants.ALL_LANGUAGE_CODES` • page_contents should consist of subtitled_html,content_ids_to_audio_translations and written_translations fields • subtitled_html, and written_translations should follow the corresponding schema of SubtitledHtml, WrittenTranslation • content_ids_to_audio_translations should contain the keys content and default_outcome • page_contents_schema_version	VersionedModel

		<= latest version	
SubtopicPageCommitLogEntryModel	Model to store commit logs to subtopic page	• Model id should be of the form `subtopicpage-{{SUBTOPIC_PAGE_ID}}-{{SUBTOPIC_PAGE_VERSION}}`	BaseCommitLogEntryModel
TopicRightsSnapshotMetadataModel	Model for topic rights snapshot metadata	None	BaseSnapshotMetadataModel
TopicRightsSnapshotContentModel	Model for topic rights snapshot content	None	BaseSnapshotContentModel
TopicRightsModel	Model for storing rights related to a topic	• Model id should be same as topic id • manager_ids should correspond to valid users	VersionedModel

User Models

Name	**Purpose**	**Checks**	**Inherits checks from**
UserSettingsModel	Model to store settings for a user	• Model id should be same as user_id • All time fields should be less than current time • role should correspond to one of the ROLE_ID in feconf • No two models should have the same normalized_usernames • Each model corresponds to a CompletedActivities, IncompleteActivities, LearnerPlaylist, UserEmailPreferences, UserSubscriptions, UserContributions, UserEmailPreferences, UserSubscriptions, UserRecentChangesBatch, UserStats, UserBulkEmails model	BaseModel
CompletedActivitiesModel	Model to keep track of	• Model id should be same as user id • exploration_ids should correspond to valid explorations	BaseModel

	explorations and collections completed by a user	• collection_ids should correspond to valid collections • Model should not have common explorations and collections with IncompleteActivitiesModel	
IncompleteActivitiesModel	Model to keep track of explorations and collections currently being completed by a user	• Model id should be same as user id • exploration_ids should correspond to valid explorations • collection_ids should correspond to valid collections • Model should not have common explorations and collections with CompleteActivitiesModel • There should be N ExpUserLastPlaythroughModel for N items in exploration_ids	BaseModel
ExpUserLastPlaythroughModel	Model to store the last playthrought info about partially completed exploration	• Model id should be of the form `[user_id].[exploration_id]` • user_id should correspond to valid user • exploration_id, last_played_exploration_version should correspond to valid exploration and last_played_state_name should correspond to valid state in the exploration • exploration_id should be present in IncompleteActivitiesModel for the user_id	BaseModel
LearnerPlaylistModel	Model to keep track of all explorations and collections in user's playlist	• Model id should be same as user id • exploration_ids should correspond to valid explorations • collection_ids should correspond to valid collections	BaseModel
UserContributionsModel	Models to keep track of exploration created/edited by user	• Model id should be same as user id • created_exploration_ids should correspond to valid explorations • edited_exploration_ids should correspond to valid explorations and should have a positive commit by the user	BaseModel
UserEmail	Model to	• Model id should be same as user id	BaseMo

PreferencesModel	store email preferences of a user	• editor_role_notifications should be True if user has never set a preference	del
UserSubscriptionsModel	Model to store subscriptions of a user	• Model id should be same as user id • activity_ids should correspond to valid explorations • collection_ids should correspond to valid collections • feedback_thread_ids, general_feedback_thread_ids should correspond to valid feedback threads • creator_ids should correspond to valid users • last_checked <= current time	BaseModel
UserSubscribersModel	Model to store subscribers of a user	• Model id should be same as user id • subscriber_ids should correspond to valid users and should not include the user themselves • user_id should be present in the creator_ids of UserSubscriptionsModel of subscribers	BaseModel
UserRecentChangesBatchModel	Model to store list of recent changes in user subscriptions	• Model id should be same as user id • job_queued_msec <= current time	BaseMapReduceBatchResultsModel
UserStatsModel	Model to store user specific statistics	• Model id should be same as user id • schema_version <= latest version • weekly_creator_stats_list should be a list with each item dict keyed by a datetime string and value as another Json object containing key-value pairs to be stored	BaseMapReduceBatchResultsModel
ExplorationUserDataModel	Model to store user specific data related to specific exploration	• Model id should be of the form `[USER_ID].[EXPLORATION_ID]` • The creator fields present in the model should be set only if the user is present as an editor in the ExplorationRightsModel for the exploration • rating should be between 1 and 5 inclusive • user_id and exploration_id should	BaseModel

		correspond to valid user and exploration • rated_on <= current time • draft_change_list_last_updated <= current time • draft_change_list_exp_version <= current version • draft_change_list should follow the commands and parameters as for commit_cmds	
CollectionProgressModel	Model to store progress a user has made within a collection	• Model id should be of the form `{user_id}. {collection_id}` • user_id and collection_id should correspond to valid user and collection • Explorations in completed_explorations should be present in CompletedActivitesModel and should correspond to valid exploration	BaseModel
StoryProgressModel	Model to store progress a user has made within a story	• Model id should be of the form `{user_id}. {story_id}` • user_id and story_id should correspond to valid user and story • Nodes in completed_node_ids should correspond to valid nodes	BaseModel
UserQueryModel	Model for storing result of queries	• Model id is alphanumeric id of length 12 • user_ids and submitter_id should correspond to valid users • sent_email_model_id should correspond to a valid BulkEmailModel	BaseModel
UserBulkEmailsModel	Model to store ids of bulk emails sent to a user	• Model id should be same as user id • sent_email_model_ids should correspond to valid BulkEmailModels	BaseModel
UserSkillMasteryModel	Model for storing a user's degree of mastery of a skill in Oppia.	• Model id should be of the form `{{USER_ID}}.{{SKILL_ID}}` • user_id and skill_id should correspond to valid user and skill • degree_of_mastery should be between 0.0 and 1.0 inclusive	BaseModel

UserContributionScoringModel	Model for storing the scores of a user for various suggestions created by the user.	• Model id should be of the form `{{score_category}}.{{user_id}}` • user_id should correspond to valid user • score_category should correspond to a valid score category • score should be non-negative.	BaseModel

List of commit_cmds with required parameters

Command Name	**Related to Model**	**Parameters**
CMD_ADD_COLLECTION_NODE	Collection	- exploration_id (string) (should correspond to valid exploration)
CMD_DELETE_COLLECTION_NODE	Collection	- exploration_id (string) (should correspond to valid exploration)
CMD_SWAP_COLLECTION_NODES		- first node index (int) - second node index (int)
CMD_EDIT_COLLECTION_PROPERTY	Collection	- property_name (collection_domain.CollectionChange.COLLECTION_PROPERTIES) - new_value (value according to the schema for property_name) - old_value (value according to the schema for property_name) (optional)
CMD_EDIT_COLLECTION_NODE_PROPERTY	Collection	- property_name (from collection_domain.COLLECTION_NODE_PROPERTY_*) - new_value (value according to the schema for property_name) - olde_value (value according to the schema for property_name) (optional)
CMD_MIGRATE_SCHEMA_TO_LATEST_VERSION	Collection	- from_version (int) (<= latest version) - to_version (int) (<= latest version) - from_version <= to_version

CMD_ADD_COLLECTION_SKILL	Collection	- name (string)
CMD_DELETE_COLLECTION_SKILL	Collection	- skill_id (string) (should correspond to valid skill)
CMD_ADD_QUESTION_ID_TO_SKILL	Collection	- question_id (string) (should correspond to valid question) - skill_id (string) (should correspond to valid skill)
CMD_REMOVE_QUESTION_ID_FROM_SKILL	Collection	- question_id (string) (should correspond to valid question) - skill_id (string) (should correspond to valid skill)
CMD_UPDATE_SKILL_PROPERTY	Skill	- property_name (from skill_domain.SKILL_PROPERTY_*) - new_value (according to the schema for property_name) - old_value (according to the schema for property_name) (optional)
CMD_UPDATE_SKILL_CONTENTS_PROPERTY	Skill	- property_name (from skill_domain.SKILL_CONTENTS_PROPERTY_*) - new_value (according to the schema for property_name) - old_value (according to the schema for property_name) (optional)
CMD_UPDATE_SKILL_MISCONCEPTIONS_PROPERTY	Skill	- property_name (from skill_domain.SKILL_MISCONCEPTIONS_PROPERTY_*) - new_value (according to the schema for property_name) - old_value (according to the schema for property_name) (optional)
CMD_ADD_SKILL_MISCONCEPTION	Skill	- new_misconception_dict (should match misconception schema)
CMD_DELETE_SKILL_MISCONCEPTION	Skill	- id (int) (should correspond to valid misconception)
CMD_CREATE_NEW	Skill, SkillRights	None
CMD_MIGRATE_CONTENTS_SCHEMA_TO_LATEST_VERSION	Skill	- from_version (int) (<= latest version) - to_version (int) (<= latest version) - from_version <= to_version

CMD_MIGRATE_MISCONCEPTIONS_SCHEMA_TO_LATEST_VERSION	Skill	- from_version (int) (<= latest version) - to_version (int) (<= latest version) - from_version <= to_version
CMD_PUBLISH_SKILL	SkillRights	None
CMD_CREATE_NEW	Exploration	- title (string) - category (from constants.ALL_CATEGORIES)
CMD_ADD_STATE	Exploration	- state_name (string)
CMD_RENAME_STATE	Exploration	- old_state_name (string) (should correspond to valid state) - new_state_name (string)
CMD_DELETE_STATE	Exploration	- state_name (string) (should correspond to valid state)
CMD_EDIT_STATE_PROPERTY	Exploration	- property_name (from exp_domain.STATE_PROPERTY_*) - new_value (should be according to schema of property) - old_value (should be according to schema of property) (optional)
CMD_EDIT_EXPLORATION_PROPERTY	Exploration	- property_name (from exp_domain.ExplorationChange.EXPLORATION_PROPERTIES) - new_value (should be according to schema of property) - old_value (should be according to schema of property) (optional)
CMD_MIGRATE_STATES_SCHEMA_TO_LATEST_VERSION	Exploration	- from_version (int) (<= latest version) - to_version (int) (<= latest version) - from_version <= to_version
CMD_CREATE_NEW	Topic, TopicRights	- name (string) (For TopicModel) - None (For TopicRightsModel)
CMD_CHANGE_ROLE	TopicRights	- assignee_id (string) (should correspond to valid user) - new_role (topic_domain.ROLE_MANAGER, topic_domain.ROLE_NONE) - old_role (topic_domain.ROLE_MANAGER, topic_domain.ROLE_NONE)
CMD_REMOVE_MANAGER_	Topicrights	- removed_user_id (string) (should

ROLE		correspond to valid user)
CMD_PUBLISH_TOPIC	TopicRights	None
CMD_UNPUBLISH_TOPIC	TopicRights	None
CMD_ADD_SUBTOPIC	Topic	- title (string) - subtopic_id (int)
CMD_DELETE_SUBTOPIC	Topic	- subtopic_id (int) (should correspond to valide subtopic)
CMD_ADD_UNCATEGORIZED_SKILL_ID	Topic	- new_uncategorized_skill_id (int)
CMD_REMOVE_UNCATEGORIZED_SKILL_ID	Topic	- uncategorized_skill_id (int) (should correspond to valid skill id)
CMD_MOVE_SKILL_ID_TO_SUBTOPIC	Topic	- old_subtopic_id (int) (should correspond to valid subtopic) - new_subtopic_id (int) - skill_id (string) (should correspond to valid skill)
CMD_REMOVE_SKILL_ID_FROM_SUBTOPIC	Topic	- subtopic_id (int) (should correspond to valid subtopic) - skill_id (string) (should correspond to valid skill)
CMD_UPDATE_TOPIC_PROPERTY	Topic	- property_name (from topic_domain.TOPIC_PROPERTY_*) - new_value (should be according to schema of property) - old_value (should be according to schema of property) (optional)
CMD_UPDATE_SUBTOPIC_PROPERTY	Topic	- subtopic_id (int) (should correspond to valid subtopic) - property_name (from topic_domain.SUBTOPIC_PROPERTY_*) - new_value (should be according to schema of property) - old_value (should be according to schema of property) (optional)
CMD_MIGRATE_SUBTOPIC_SCHEMA_TO_LATEST_VERSION	Topic	- from_version (int) (<= latest version) - to_version (int) (<= latest version) - from_version <= to_version
CMD_UPDATE_QUESTION_PROPERTY	Question	- property_name (from question_domain.QUESTION_PROPE

		RTY_*) - new_value (should be according to schema of property) - old_value (should be according to schema of property) (optional)
CMD_CREATE_NEW_FULLY_SPECIFIED_QUESTION	Question	- question_dict (should match question schema) - skill_id (string) (should correspond to valid skill)
CMD_MIGRATE_STATE_SCHEMA_TO_LATEST_VERSION	Question	- from_version (int) (<= latest version) - to_version (int) (<= latest version) - from_version <= to_version
CMD_ADD_QUESTION_SKILL	QuestionSkillLink	deprecated
CMD_REMOVE_QUESTION_SKILL	QuestionSkillLink	deprecated
CMD_CREATE_NEW	(present in question_domain - but not used anywhere)	None
CMD_UPDATE_STORY_PROPERTY	Story	- property_name (from story_domain.StoryChange.STORY_PROPERTIES) - new_value (should be according to schema of property) - old_value (should be according to schema of property) (optional)
CMD_UPDATE_STORY_NODE_PROPERTY	Story	- property_name (from story_domains.StoryChange.STORY_NODE_PROPERTIES) - new_value (should be according to schema of property) - old_value (should be according to schema of property) (optional)
CMD_UPDATE_STORY_CONTENTS_PROPERTY	Story	- property_name (story_domain.StoryChange.STORY_CONTENT_PROPERTIES) - new_value (should be according to schema of property) - old_value (should be according to schema of property) (optional)

CMD_ADD_STORY_NODE	Story	- node_id (string)
CMD_DELETE_STORY_NODE	Story	- node_id (string) ((should correspond to valid story node)
CMD_UPDATE_STORY_NODE_OUTLINE_STATUS	Story	- node_id (string) (should correspond to valid story node)
CMD_CREATE_NEW	Story, StoryRights	- title (string) (For Story) - None (For StoryRights)
CMD_MIGRATE_SCHEMA_TO_LATEST_VERSION	Story	- from_version (int) (<= latest version) - to_version (int) (<= latest version) - from_version <= to_version
CMD_CHANGE_ROLE	StoryRights	- assignee_id (string) (should correspond to valid user) - new_role (story_domain.ROLE_MANAGER, story_domain.ROLE_NONE) - old_role (story_domain.ROLE_MANAGER, story_domain.ROLE_NONE)
CMD_PUBLISH_STORY	StoryRights	None
CMD_UNPUBLISH_STORY	StoryRights	None

List of additional checks and clean ups

(These tasks will be performed during GSoC itself. This will be included in migration cleanup jobs for the models as mentioned in the timeline. This part will be done in milestone 2.1, 2.2, 2.3 and 3.1, 3.2, 3.3 where the migration jobs will be added to fix validation issues. Along with those jobs, cleanup jobs will be added for the models for which clean up is listed in this list)

- Ensure each image and audio file in datastore is linked with an exploration - Remove files not linked with any exploration
- Update validation of html fields before they are saved using the predefined methods in html_validation_service as follows:

```
html_list = self.get_all_html_content_strings()
err_dict = (
    html_validation_service.validate_customization_args(
        html_list)
```

- Remove all models with deleted field as True
- Add schema_version to commit_cmds (The schema in the table will be the current version)
- Add schema to any json property which has a predefined set of rules to be associated with it.
- Add choices type validation to fields which are allowed to have only selected values:
    - post_commit_status -> BaseCommitLogEntryModel
    - commit_type -> BaseSnapshotMetadataModel
    - language_code -> (All models where it is specified)
    - ratings -> ExpSummaryModel
    - event_type -> LogEntryModels (Statistics Model)
    - calculation_output_type -> StateAnswersCalcOutputModel
- Remove all the deprecated properties from models
    - nodes -> CollectionModel
    - skill_tags, default_skin, skin_customizations -> ExplorationModel
- Complete all TODO tasks:
    - VersionedModel - In reconstituting a model from snapshot dict, the created_on and last_updated properties slightly differ from the entity model since they correspond to snapshot dict instead of the model. Check if it is an issue and and if it is required to store the contents of the actual model in these fields
    - CollectionModel - Check if put_async in CollectionCommitLogEntryModel leads to any update issues with the model
    - SentEmailModel - Implement functionality to get all emails sent to a particular user with a given intent within a given time period
    - ExplorationModel - Remove deprecated properties, Check if put_async in ExplorationCommitLogEntryModel leads to any update issues with the model
    - StartExplorationEventLogEntryModel, MaybeLeaveExplorationEventLogEntryModel, StateHitEventLogEntryModel - Migrate the events that do not have entity id to ensure that each event has an entity id

- UserSettingsModel - Write a one off job to give role to all users and remove the default value from role field, Add a new field to store the language that the user wants the site to display in (This todo just needs to be removed, the field is already added)
- UserSubscriptionsModel - Perform a migration to rename activity_ids to exploration_ids
- CollectionProgressModel - Write a job to remove stale models by checking that both user id and collection for the model still exist in the datastore

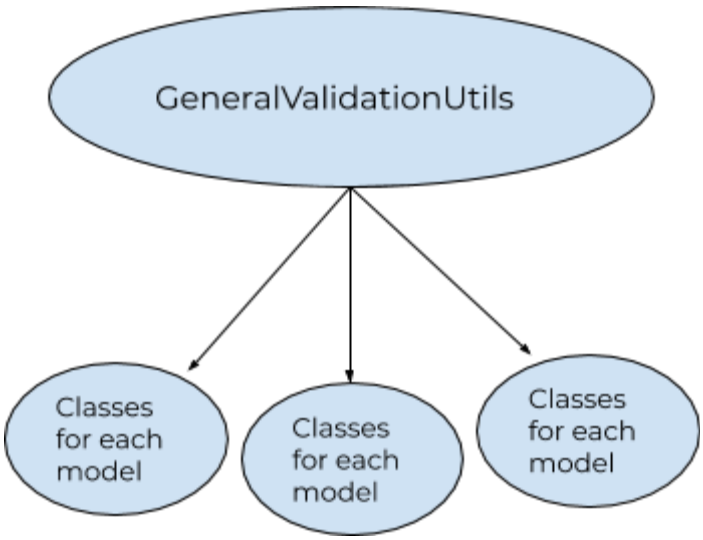
Technical Design

Files to be added (core/domain)

- model_validation_jobs_one_off_utils.py
- model_validation_jobs_one_off_utils_test.py
- commit_cmds_domain.py
- commit_cmds_domain_test.py
- model_validation_jobs_one_off.py
- model_validation_jobs_one_off_test.py
- model_migration_jobs_one_off.py
- model_migration_jobs_one_off_test.py

File Design

- model_validation_jobs_one_off_utils.py - This file will contain classes for each model to run test specific to that model and a superclass for all these classes to have the test general to all models.



Stuff that goes in the general class

- **Model id validation check** - The validation class for each model will have a id regex against which the id will be validated
- **Model schema validation check** - The validation class for each model will specify a instance against which we need to validate the complete the model schema. If no such validation check is needed no such instance will be specified. Model schema validation check is to ensure that model is a valid instance of the class for which it is created. For example, a ExplorationModel should form a valid instance of Exploration class.
- **Argument schema validation check** - The validation class for each model will specify a class for the arguments which are required to follow a particular schema. The arguments will be validated using the validate function of this class. (argname, argname_schema_class). If no such validation check is needed no schema class will be specified. Argument schema validation check is to ensure that model properties are a valid instance of the class to which they belong. For example, param_changes property in ExplorationModel should form a valid instance of ParamChanges class.
- **Schema version validation check** - Any argument specifying a version will be validated to ensure that the version falls between the valid range. The validation class for each model should specify the range for the version arguments being used by the model
- **Model relation check** - If any model requires a certain number of other model instances to be present according to the model version, this check will query if the condition holds valid. The validation class for model will specify a dict of models which it requires where key will be the model class and values will be the required ids.

- **Creation and Last update check** - This check will ensure that the created_on field is less than equal to last_updated field for each model instance
- **Language Code validation check** - This check will ensure that the language code is in the specified language codes in constants.js

- model_validation_jobs_one_off_utils_test.py - This file will contain the test for utility functions for the audit jobs.
- commit_cmds_domain.py - This file will contain a new CommitCmd class which will specify the schema for commit commands used in various models according to the table.
- commit_cmds_domain_test.py - This file will contain test for the commit cmds schema classes.
- model_validation_jobs_one_off.py - This file will contain one off jobs for each model. The jobs will be named as **ModelNameAuditOneOffJob**. Each job will call the utility class instance for validation of the model.
- model_validation_jobs_one_off_test.py - This file will contain the test for the audit one off jobs.
- model_migration_jobs_one_off.py - This file will contain the migration jobs to clean up and migrate the models to a valid schema.
- model_migration_jobs_one_off_test.py - This file will contain the test for migration one off jobs.

Implementation Strategy

Audit one off jobs for models

Audit one off jobs will be written for each model to find out the violations of the validation checks. Audit jobs will also be written for the list of additional checks.

Example code for audit one off job for ExplorationModel:

In model_validation_jobs_one_off_utils.py:

```
from constants import constants
import feconf
```

```
import re

from core.domain import exp_services
from core.platform import models

(exp_models,) = models.Registry.import_models([models.NAMES.exploration])
datastore_services = models.Registry.import_datastore_services()

class GeneralValidationUtils(object):
    """Class for general validation of models."""

    def __init__(
        self, model, id_regex, model_instance_to_validate,
        arg_schema_class_dict, required_models_list, version_range_dict):
        self.model = model
        self.id_regex = id_regex
        self.model_instance_to_validate = model_instance_to_validate
        self.arg_schema_class_dict = arg_schema_class_dict
        self.required_models_list = required_models_list
        self.version_range_dict = version_range_dict
        self.test_output = []

    def test_id_is_valid(self):
        """Checks if model id matches the id regex."""
        if not re.match(self.id_regex, self.model.id):
            self.test_output.append('Failed: Model id %s is not valid' % (
                self.model.id))
        else:
            self.test_output.append('Success: Model id is valid.')

    def test_model_schema_is_valid(self):
        """Checks if model follows a required schema."""
        model_schema_name = type(self.model_instance_to_validate).__name__
        try:
            self.model_instance_to_validate.validate()
            self.test_output.append('Success: Valid %s' %
model_schema_name)
        except Exception as e:
            self.test_output.append(
                'Failed: Model not valid %s due to %s' %
(model_schema_name, e))

    def test_language_code_is_valid(self):
        """Checks if the model language code is valid."""
        test_output_msg = 'Success: Language code is valid'
```

```
        if self.model.language_code not in (
            [item['code'] for item in constants.ALL_LANGUAGE_CODES]):
            test_output_msg = 'Failed: Language code %s is not a valid
language code.' % (
                    self.model.language_code)

        self.test_output.append(test_output_msg)

    def test_schema_version_falls_between_valid_range(self):
        """Checks if version is between lower and upper bound."""
        for version_key, range_list in self.version_range_dict.items():
            version = getattr(self.model, version_key)
            lower_bound = range_list[0]
            upper_bound = range_list[1]
            if version < lower_bound:
                self.test_output.append(
                    'Failed: %s %d is lesser than lower bound %d' % (
                        version_key, version, lower_bound))
            else:
                self.test_output.append(
                    'Success: %s is >= lower bound' % version_key)

            if version > upper_bound:
                self.test_output.append(
                    'Failed: %s %d is greater than upper bound %d' % (
                        version_key, version, upper_bound))
            else:
                self.test_output.append(
                    'Success: %s is <= upper bound' % version_key)

    def test_required_models_exist(self):
        """Checks if all the required models exist."""
        all_model_list = (
            datastore_services.fetch_multiple_entities_by_ids_and_models(
                self.required_models_list))
        for type_index, model_list in enumerate(all_model_list):
            missing_model_ids = []
            model_class_name = self.required_models_list[type_index][0]
            for model_index, model in enumerate(model_list):
                if not model:
                    missing_model_ids.append(

self.required_models_list[type_index][1][model_index])
            if len(missing_model_ids):
                self.test_output.append(
```

```
                    'Failed: Following %s missing: %s' % (
                        model_class_name, missing_model_ids))
            else:
                self.test_output.append(
                    'Success: All %s found.' % (model_class_name))


class ExplorationModelValidationUtils(GeneralValidationUtils):
    """Class for validation of exploration model."""

    def __init__(self, model):
        required_models_list = [
            ('ExpSummaryModel', [model.id]),
            (
                'ExplorationCommitLogEntryModel', [
                'exploration-%s-%s' % (model.id, i + 1) for i in range(
                    model.version)]
            ), (
                'StateIdMappingModel', [
                '%s.%s' % (model.id, i + 1) for i in range(model.version)]
            )
        ]
        version_range_dict = {
            'states_schema_version': [
            0, feconf.CURRENT_STATES_SCHEMA_VERSION]
        }
        model_instance_to_validate =
exp_services.get_exploration_from_model(
            model)

        super(ExplorationModelValidationUtils, self).__init__(
            model, None, model_instance_to_validate, None,
            required_models_list, version_range_dict)

    def validate(self):
        self.test_language_code_is_valid()
        self.test_required_models_exist()
        self.test_schema_version_falls_between_valid_range()
        self.test_model_schema_is_valid()
```

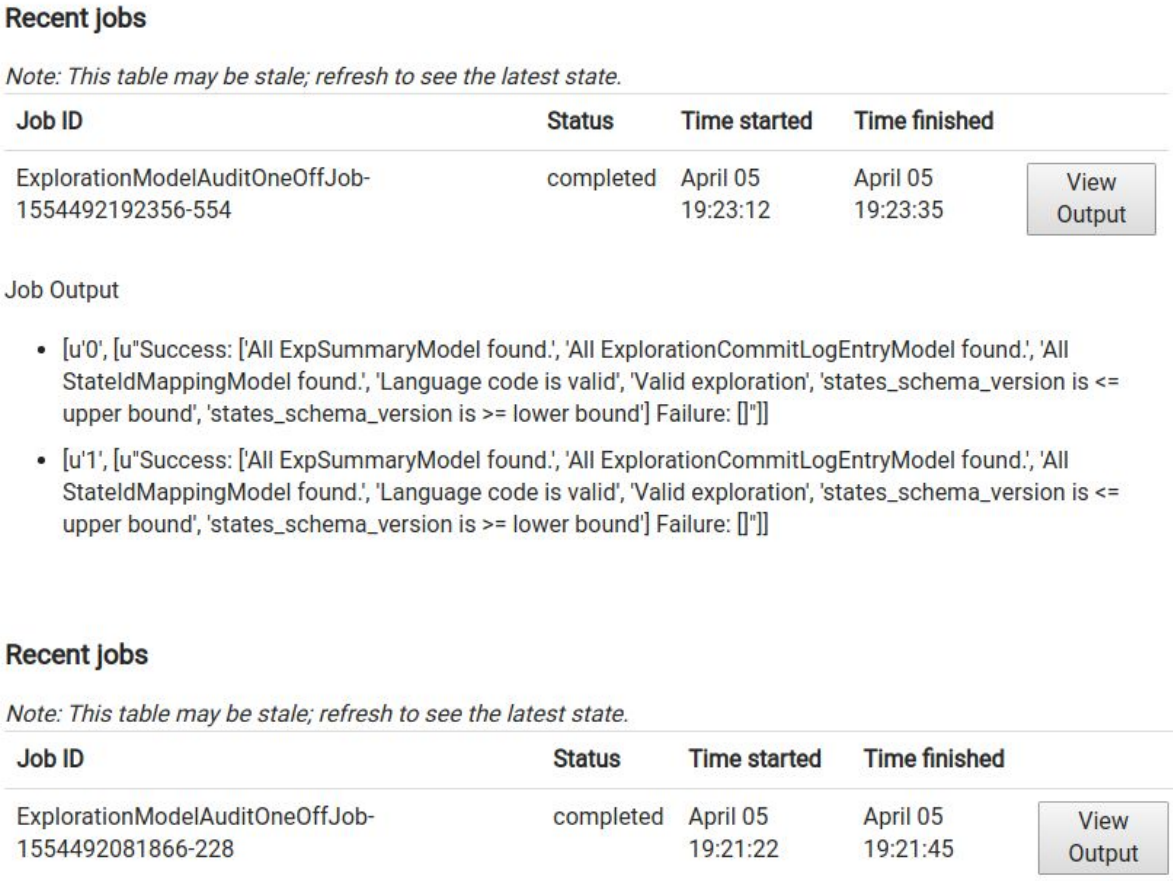
In model_validation_jobs_one_off.py:

```
from core import jobs
from core.domain import model_validation_jobs_one_off_utils
```

```
from core.platform import models

(exp_models,) = models.Registry.import_models([models.NAMES.exploration])

class ExplorationModelAuditOneOffJob(jobs.BaseMapReduceOneOffJobManager):
    """Job that audits and validates ExplorationModel."""

    @classmethod
    def entity_classes_to_map_over(cls):
        return [exp_models.ExplorationModel]

    @staticmethod
    def map(item):
        if not item.deleted:
            exploration_model_checks = (
                model_validation_jobs_one_off_utils.
                ExplorationModelValidationUtils(item))
            exploration_model_checks.validate()

            failed_test = []

            success_cnt = 0
            for output in exploration_model_checks.test_output:
                if output.startswith('Success'):
                    success_cnt += 1
                else:
                    failed_test.append(output[output.find(': ') + 2:])

           failed_test.sort()

            yield (item.id, 'Success: %d Failure: %s' % (
                success_cnt, failed_test))

    @staticmethod
    def reduce(key, values):
        yield(key, values)
```

Sample Output:

Note: Instead of success messages as shown below, the output will only contain a count of successful test and messages for failed test.



Recent jobs

Note: This table may be stale; refresh to see the latest state.

Job ID	Status	Time started	Time finished	
ExplorationModelAuditOneOffJob-1554492192356-554	completed	April 05 19:23:12	April 05 19:23:35	View Output

Job Output

- [u'0', [u"Success: ['All ExpSummaryModel found.', 'All ExplorationCommitLogEntryModel found.', 'All StateIdMappingModel found.', 'Language code is valid', 'Valid exploration', 'states_schema_version is <= upper bound', 'states_schema_version is >= lower bound'] Failure: []"]]
- [u'1', [u"Success: ['All ExpSummaryModel found.', 'All ExplorationCommitLogEntryModel found.', 'All StateIdMappingModel found.', 'Language code is valid', 'Valid exploration', 'states_schema_version is <= upper bound', 'states_schema_version is >= lower bound'] Failure: []"]]

Recent jobs

Note: This table may be stale; refresh to see the latest state.

Job ID	Status	Time started	Time finished	
ExplorationModelAuditOneOffJob-1554492081866-228	completed	April 05 19:21:22	April 05 19:21:45	View Output

Job Output

- [u'16', [u'Success: [\'All ExplorationCommitLogEntryModel found.\', \'All StateIdMappingModel found.\', \'Language code is valid\', \'Valid exploration\', \'states_schema_version is <= upper bound\', \'states_schema_version is >= lower bound\'] Failure: ["Following ExpSummaryModel missing: [\'16\']"]']]

Tests for Audit one off jobs

Before running the jobs on production data, test will be added to ensure that jobs cover all the cases for valid and invalid data.

Sample test for ExplorationModelAuditOneOffJob (test for valid case and commit log model missing case only):

In model_validation_jobs_one_off_test.py:

```
from core.domain import exp_domain
from core.domain import model_validation_jobs_one_off
from core.domain import exp_services
from core.platform import models
from core.tests import test_utils

(exp_models,) = (models.Registry.import_models([models.NAMES.exploration]))

class ExplorationModelAuditOneOffJobTests(
        test_utils.GenericTestBase):

    ALBERT_EMAIL = 'albert@example.com'
    ALBERT_NAME = 'albert'

    FIRST_EXP_ID = 'exp_id1'
    SECOND_EXP_ID = 'exp_id2'

    def setUp(self):
        super(ExplorationModelAuditOneOffJobTests, self).setUp()

        # Setup user who will own the test explorations.
        self.albert_id = self.get_user_id_from_email(self.ALBERT_EMAIL)
        self.signup(self.ALBERT_EMAIL, self.ALBERT_NAME)
        self.process_and_flush_pending_tasks()

        self.exploration1 =
exp_domain.Exploration.create_default_exploration(
            self.FIRST_EXP_ID, title='title1', category='category')
        exp_services.save_new_exploration(self.albert_id,
self.exploration1)

        self.exploration2 =
exp_domain.Exploration.create_default_exploration(
            self.SECOND_EXP_ID, title='title2', category='category')
        exp_services.save_new_exploration(self.albert_id,
self.exploration2)
        change_list = [exp_domain.ExplorationChange({
            'cmd': 'add_state',
```

```
            'state_name': 'State1'
        })]
        exp_services.update_exploration(
            self.albert_id, self.SECOND_EXP_ID, change_list, 'Add new
state')

    def test_no_errors_are_produced_for_a_valid_exploration(self):
        """Tests no errors are produced for valid exploration."""
        # Start validation job on sample exploration.
        job_id = (
            model_validation_jobs_one_off
            .ExplorationModelAuditOneOffJob.create_new())


        model_validation_jobs_one_off.
        ExplorationModelAuditOneOffJob.enqueue(job_id)
        self.process_and_flush_pending_tasks()

        actual_output = (
            model_validation_jobs_one_off
            .ExplorationModelAuditOneOffJob.get_output(job_id))
        expected_output = [(
            u'[u\'exp_id1\', '
            '[u"Success: 7'
            'Failure: []"]]'
        ), (
            u'[u\'exp_id2\', '
            '[u"Success: 7'
            'Failure: []"]]'
        )]
        self.assertEqual(actual_output, expected_output)

    def test_error_is_raised_when_all_commit_log_models_are_not_found(
            self):
        """Test error is raised if some of the commit log models are
        missing.
        """

        exp_models.ExplorationCommitLogEntryModel.get_by_id(
            'exploration-%s-%s' % (self.SECOND_EXP_ID, 1)).delete()

        # Start validation job on sample exploration.
        job_id = (
            model_validation_jobs_one_off
            .ExplorationModelAuditOneOffJob.create_new())
```

```
        model_validation_jobs_one_off.
        ExplorationModelAuditOneOffJob.enqueue(
            job_id)
        self.process_and_flush_pending_tasks()

        actual_output = (
            model_validation_jobs_one_off
            .ExplorationModelAuditOneOffJob.get_output(job_id))
        expected_output = [(
            u'[u\'exp_id1\', '
            '[u"Success: 7'
            'Failure: []"]]'
        ), (
            u'[u\'exp_id2\', '
            '[u\'Success: 6'
            'Failure: ["Following ExplorationCommitLogEntryModel missing: '
            '[\'exploration-exp_id2-1\']"]\']]'
        )]
        self.assertEqual(actual_output, expected_output)
```

Migration jobs for Models to fix the discrepancies

Based on the discrepancies found out after running the audit jobs, migration jobs will be written to migrate the models to a valid schema.

Test for Migration jobs fixing discrepancies

Test will be added for migration jobs before running them on production data. The test will be written by using the production data discrepancies obtained from audit jobs such that each specific discrepancy has a unique test. Test will also be added to ensure that the models after migration pass through the audit pipeline.

Migration jobs for clean up

These jobs will be written to remove the properties from models which are deprecated or are no longer required. These jobs will also be used to clean up the datastore by removing the deleted models.

Sample cleanup process to rename activity_ids to exploration_ids:

- Add a new property exploration_ids to UserSubscriptionsModel definition and remove the activity_ids property.
- The regular models update will not work and a better way to handle this is to override the put as done in feedback models.

```
last_updated = ndb.DateTimeProperty(indexed=True, required=True)

def put(self, update_last_updated_time=True):
    """Writes the given model instance to the datastore.

    Args:
        update_last_updated_time: bool. Whether to update the
        last_updated_field of the thread.

    Returns:
        ModelClass. The model instance.
     """
     if update_last_updated_time:
         self.last_updated = datetime.datetime.utcnow()

     return super(ModelClass, self).put()
```

Now we can turn off the auto update by passing update_last_updated_time=False whenever doing an item.put() in migration.

- Run a MigrationJob to set exploration_ids to activity_ids and activity_ids to None:

```
if 'activity_ids' in item._properties:
    item.exploration_ids = item.activity_ids
    del item._properties['activity_ids']
    item.put()
```

Test for Migration jobs performing clean up

Test will be added for clean up migration jobs which ensure that the model after clean up passes the audit pipeline and only the deprecated properties and models are removed. Mocks of old models will be created and then the test will run the clean up jobs on there mocks to verify the cleanup process.

Timeline

[Link to Project Tracker]

Note:

1. Migration job means the job written to remove the errors caught by the audit jobs. These jobs will be written and run after finding out the root cause of the errors and updating the model code to avoid that error in future.
2. Migration jobs for cleanup job means the job written to finish the todo tasks like removing old properties from a model

Task to be done in PR	PR Issued on	PR to be merged by
Tasks before GSoC starts (Current period - 27 May): • Contributing to pre migration project of Angular2 • Contributing to bug fixes and other issues • Reviewing PRs • Improving my proposal • Helping new contributors to get onboard		
Milestone 1.1: Add Audit Jobs and test for the following models as per the validation table: • Activity • Email • Exploration	27 May	5 June
Milestone 1.2: Add Audit Jobs and test for the following models as per the validation table: • Audit • Topic • Collection	27 May	5 June
Milestone 1.3: Add Audit Jobs and test for the following models as per the validation table: • Statistics • Config • File	27 May	5 June
Milestone 1.4: Add Audit	29 May	6 June

Jobs and tests for remaining models • Classifier • Question • Story		
Milestone 1.5: Add Audit Jobs and tests for remaining models • Feedback • Recommendation • User	29 May	6 June
Milestone 1.6: Add Audit Jobs and tests for remaining models • Job • Skill • Suggestion	29 May	6 June
The audit jobs need to be run on test server data and production server data to obtain a list of errors in the models which do not follow the validation rule. The output will be used to the migration one off jobs and test for them. Obtain the output from test server and copy of production data by **8th June**. Update validation jobs if required if there are unforeseen errors during running the jobs on test and production data. Get the changes merged and obtain the output latest by **12th June**. In the meantime start coding for cleanup migration jobs for the models in milestone 2.1, 2.2 and 2.3.		
Milestone 2.1: Add migration jobs and MigrationValidation jobs to clear all issues that arise when its run on production and test server. Add migration jobs for cleanup along with tests. For models: • Activity • Audit • Exploration	14 June	24 June
Milestone 2.2: Add migration jobs and MigrationValidation jobs to clear all issues that arise when its run on production and test server. Add migration jobs	19 June	29 June

for cleanup along with tests. For models: • Collection • Recommendation • Story		
Milestone 2.3: Add migration jobs and MigrationValidation jobs to clear all issues that arise when its run on production and test server. Add migration jobs for cleanup along with tests. For models: • Config • Topic • File	24 June	4 July
These 3 PRs should go into the July release and should be run on test and production data. First the MIgrationValidation jobs will be run which will ensure that after a migration, the models pass the validation. The output from these jobs will be used to update the Migration Jobs. Once the output has no errors, the migration jobs will be run on the test and production server data. This workflow should be done latest by **13 july**. Once the Migration is done, validation jobs will be run to ensure that no issues remain. If any issue is left out it should be fixed by updating the migration jobs. This workflow should be done latest by **17 july.**		
Milestone 3.1: Add migration jobs and MigrationValidation jobs to clear all issues that arise when its run on production and test server. Add migration jobs for cleanup along with tests. For models: • Feedback • Job • Statistics	8 July	18 July
Milestone 3.2: Add migration jobs and MigrationValidation jobs to clear all issues that arise when its run on production and test server. Add migration jobs for cleanup along with tests. For models:	13 July	23 July

• Classifier • Question • Skill		
Milestone 3.3: Run the audit jobs clear off all the issues that arise when its run on production and test server. Add migration jobs for cleanup along with tests. For models: • User • Email • Suggestion	18 July	28 July
These 3 PRs should go into the August release and should be run on test and production data. First the MigrationValidation jobs will be run which will ensure that after a migration, the models pass the validation. The output from these jobs will be used to update the Migration Jobs. Once the output has no errors, the migration jobs will be run on the test and production server data. This workflow should be done latest by **13 August**. Once the Migration is done, validation jobs will be run to ensure that no issues remain. If any issue is left out it should be fixed by updating the migration jobs. This workflow should be done latest by **17 August.**		

Summer Plan

I have no commitments in the summer. I'll be staying back home for the complete GSoC coding period. I plan to spend 40 - 45 hours per week on this project during the coding period.

Communication

Name: Ankita Saxena

Email: ankitasonu24@gmail.com

Phone: +91 7248186787

Timezone: Indian Standard Time (IST)

I'm always available at my mailing address. I would also like to have meetings on google hangouts. If the mentors of my project prefer some other means of communication, I'm fine with that as well.

Devlogs

I will maintain daily devlogs to document my work. This will help me as well as my mentor to monitor my progress.