BSc (Hons) in Information Technology

IT Project – IT2080

Activity 4

Sri Lanka Institute of Information Technology

Database Design & Development Activity

Information Technology Project (IT2080)

2025

Unified System for Insurance Claim Management

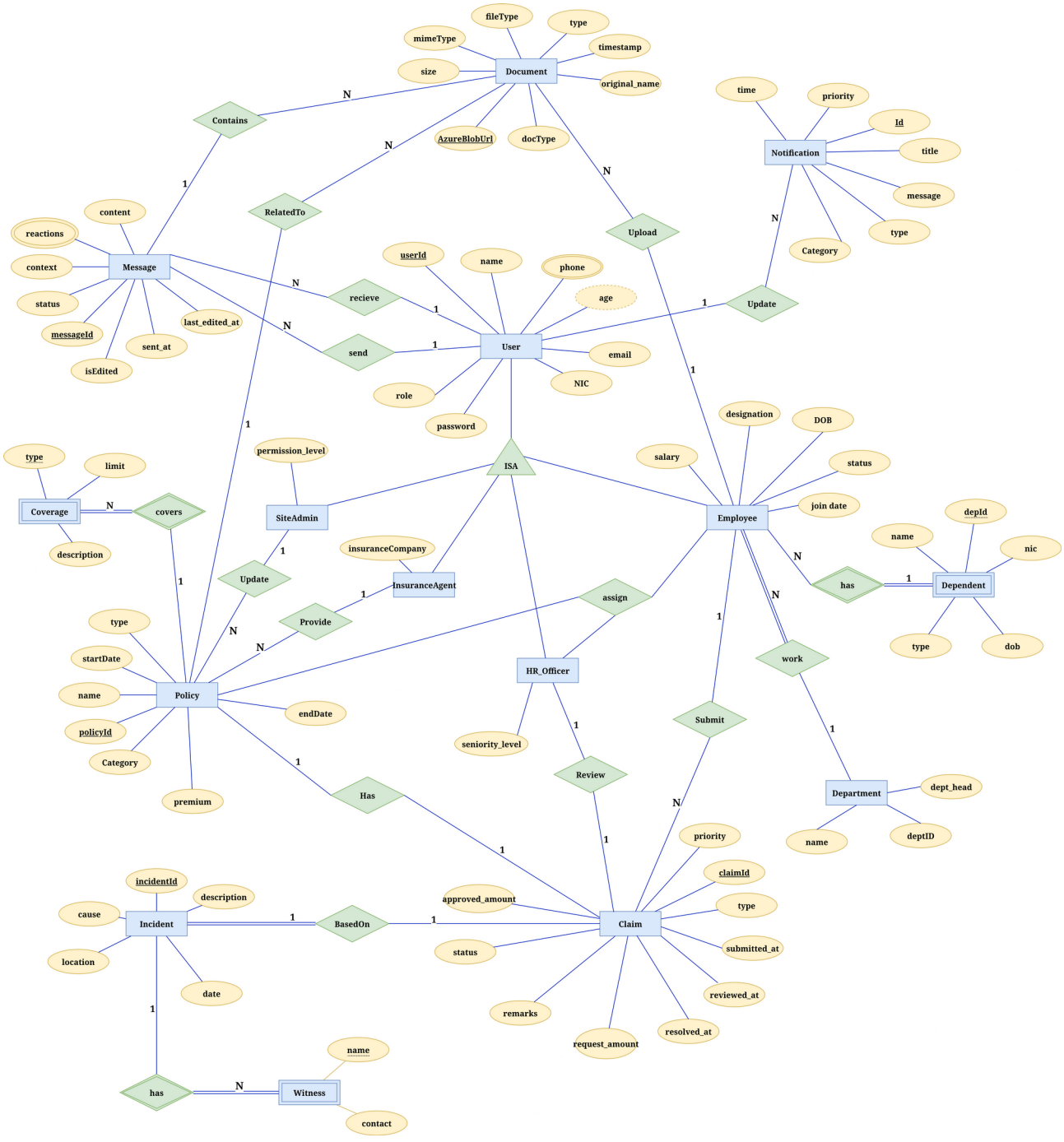
Group no : ITP25_B4_96

Campus : Malabe

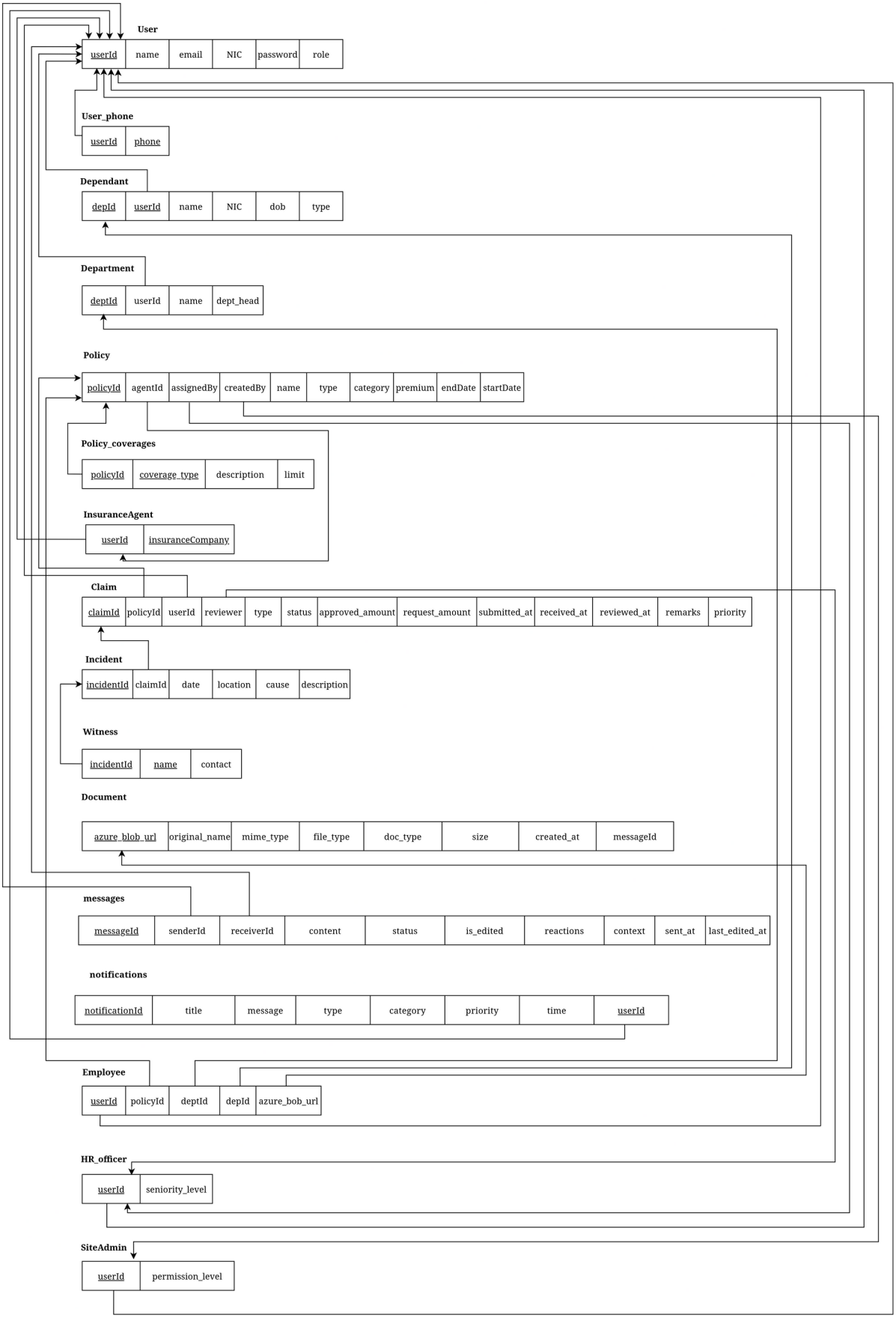
Submitted by:

	Reg Number	Name With Initials	Contact Number	Email
1.	IT23834774	NATH I M N	+94 77 429 0347	it23834774@my.sliit.lk
2.	IT23836440	FERNANDO PULLE N S	+94 70 311 4390	it23836440@my.sliit.lk
3.	IT23830332	PATHIRANA P U O R	+94 71 679 2331	it23830332@my.sliit.lk
4.	IT23725010	PERERA B I V	+94 70 134 6360	it23725010@my.sliit.lk
5.	IT23828766	SENARATHNA P G R M	+94 71 777 6610	it23828766@my.sliit.lk

EER Diagram


fileType
type
mimeType
timestamp
size
Document
original_name
Contains
AzureBlobUrl
docType
RelatedTo
Upload
time
priority
Id
title
Notification
message
type
Category
content
reactions
context
Message
status
messageId
sent_at
isEdited
last_edited_at
recieve
send
userId
name
phone
age
User
email
NIC
role
password
Update
ISA
permission_level
SiteAdmin
designation
DOB
salary
status
join date
Employee
depId
name
nic
Dependent
type
dob
has
type
limit
Coverage
covers
description
insuranceCompany
InsuranceAgent
assign
Provide
type
startDate
name
policyId
Category
premium
Policy
endDate
Has
HR_Officer
seniority_level
Review
work
Submit
Department
dept_head
name
deptID
incidentId
description
cause
location
Incident
date
BasedOn
approved_amount
status
Claim
priority
claimId
type
submitted_at
reviewed_at
resolved_at
request_amount
remarks
name
has
Witness
contact

Relational Schema



Mongoose Schemas

User Schema

```
import mongoose from "mongoose";


const UserSchema = new mongoose.Schema({
userId: { type: String, required: [true, "User ID is required"], unique: true, trim: true, uppercase:
true, match: [/^[A-Z0-9]{6,20}$/, "User ID must be 6-20 alphanumeric characters"] },
email: { type: String, required: [true, "Email is required"], unique: true, lowercase: true, trim: true,
match: [/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/, "Please enter a valid email"] },
passwordHash: { type: String, required: [true, "Password hash is required"], select: false },
role: { type: String, required: [true, "User role is required"], enum: { values: ["employee",
"hr_officer", "insurance_agent", "admin"], message: "Invalid user role" } },
name: { type: String, required: [true, "Name is required"], trim: true, maxlength: [100, "Name
cannot exceed 100 characters"] },
nic: { type: String, required: [true, "NIC number is required"], unique: true, trim: true, match:
[/^[0-9]{9}[vVxX]$|^[0-9]{12}$/, "Please enter a valid NIC number"] },
phoneNumber: { type: String, required: [true, "Phone number is required"], trim: true, match:
[/^(\+94|0)[0-9]{9}$/, "Please enter a valid Sri Lankan phone number"] },
address: { type: String, required: [true, "Address is required"], trim: true, maxlength: [200,
"Address cannot exceed 200 characters"] },
status: { type: String, default: "active", enum: ["active", "inactive", "suspended", "terminated"] },
designation: { type: String, trim: true, maxlength: [100, "Designation cannot exceed 100
characters"] },
dateOfBirth: { type: Date, validate: { validator: function (v) { return !v || v <= new Date(); },
message: "Date of birth cannot be in the future" } },
salary: { type: Number, min: [0, "Salary cannot be negative"] },
joinDate: { type: Date, validate: { validator: function (v) { return !v || v <= new Date(); }, message:
"Join date cannot be in the future" } },
permissionLevel: { type: String, enum: ["basic", "standard", "elevated", "admin"] },
seniorityLevel: { type: String, enum: ["junior", "mid", "senior", "lead", "executive"] },
insuranceCompany: { type: String, trim: true, maxlength: [100, "Insurance company name cannot
exceed 100 characters"] },
department: {
name: { type: String, required: [true, "Department name is required"], trim: true, maxlength:
[100, "Department name cannot exceed 100 characters"] },
code: { type: String, trim: true, uppercase: true, maxlength: [20, "Department code cannot
exceed 20 characters"] },
location: { type: String, trim: true, maxlength: [100, "Department location cannot exceed 100
characters"] }
},
dependents: [{
name: { type: String, required: [true, "Dependent name is required"], trim: true, maxlength: [100,
"Dependent name cannot exceed 100 characters"] },
relationship: { type: String, required: [true, "Relationship is required"], enum: ["spouse", "child",
```

```
"parent", "sibling", "other"] },
dateOfBirth: { type: Date, validate: { validator: function (v) { return !v || v <= new Date(); },
message: "Date of birth cannot be in the future" } },
nic: { type: String, trim: true, match: [/^[0-9]{9}[vVxX]$|^[0-9]{12}$/, "Please enter a valid NIC
number"] },
isInsured: { type: Boolean, default: false },
insurancePolicies: [{ type: String, trim: true }]
}]
}, { timestamps: true });

UserSchema.index({ userId: 1 });
UserSchema.index({ email: 1 });
UserSchema.index({ nic: 1 });
UserSchema.index({ role: 1 });
UserSchema.index({ status: 1 });

const User = mongoose.model("User", UserSchema);
export default User;
```

Policy Schema

```
import mongoose from "mongoose";

const PolicySchema = new mongoose.Schema({
policyId: { type: String, required: [true, "Policy ID is required"], unique: true, trim: true,
uppercase: true },
name: { type: String, required: [true, "Policy name is required"], trim: true, maxlength: [200,
"Policy name cannot exceed 200 characters"] },
type: { type: String, required: [true, "Policy type is required"], enum: { values: ["life", "medical",
"vehicle", "travel", "property", "disability"], message: "Invalid policy type" } },
category: { type: String, required: [true, "Policy category is required"], enum: ["individual",
"group", "family"] },
startDate: { type: Date, required: [true, "Policy start date is required"] },
endDate: { type: Date, required: [true, "Policy end date is required"], validate: { validator: function
(v) { return v > this.startDate; }, message: "End date must be after start date" } },
premium: {
amount: { type: Number, required: [true, "Premium amount is required"], min: [0, "Premium
amount cannot be negative"] },
frequency: { type: String, required: [true, "Premium frequency is required"], enum: ["monthly",
"quarterly", "semi-annual", "annual"] }
},
coverageDetails: [{
type: { type: String, required: true, enum: ["hospitalization", "surgery", "medication", "dental",
"optical", "maternity", "accident", "death", "disability", "vehicle_damage", "third_party_liability",
"theft", "other"] },
description: { type: String, required: true, trim: true, maxlength: [500, "Coverage description
cannot exceed 500 characters"] },
coverageAmount: { type: Number, required: true, min: [0, "Coverage amount cannot be
negative"] },
deductible: { type: Number, default: 0, min: [0, "Deductible cannot be negative"] },
usedAmount: { type: Number, default: 0, min: [0, "Used amount cannot be negative"] },
remainingAmount: { type: Number, min: [0, "Remaining amount cannot be negative"] }
}],
insuranceCompany: { type: String, required: [true, "Insurance company is required"], trim: true,
maxlength: [100, "Insurance company name cannot exceed 100 characters"] },
status: { type: String, default: "active", enum: ["active", "expired", "cancelled", "suspended",
"pending"] },
eligibleUsers: [{
userId: { type: String, required: true, ref: "User" },
relationship: { type: String, required: true, enum: ["self", "spouse", "child", "parent", "dependent"]
},
isActive: { type: Boolean, default: true }
}],
totalCoverageAmount: { type: Number, required: [true, "Total coverage amount is required"],
min: [0, "Total coverage amount cannot be negative"] },
```

```
totalUsedAmount: { type: Number, default: 0, min: [0, "Total used amount cannot be negative"] },
documents: [{ type: mongoose.Schema.Types.ObjectId, ref: "Document" }]
}, { timestamps: true });

PolicySchema.index({ policyId: 1 });
PolicySchema.index({ type: 1 });
PolicySchema.index({ category: 1 });
PolicySchema.index({ "eligibleUsers.userId": 1 });
PolicySchema.index({ status: 1 });
PolicySchema.index({ endDate: 1 });
PolicySchema.index({ insuranceCompany: 1 });

const Policy = mongoose.model("Policy", PolicySchema);
export default Policy;
```

Claim Schema

```
import mongoose from "mongoose";

const ClaimSchema = new mongoose.Schema({
claimId: { type: String, required: [true, "Claim ID is required"], unique: true, trim: true, uppercase: true },
userId: { type: String, required: [true, "User ID is required"], ref: "User" },
policyId: { type: mongoose.Schema.Types.ObjectId, required: [true, "Policy ID is required"], ref: "Policy" },
type: { type: String, required: [true, "Claim type is required"], enum: ["medical", "life", "vehicle", "travel", "property", "disability"] },
status: { type: String, default: "submitted", enum: { values: ["draft", "submitted", "under_review", "approved", "rejected", "partially_approved", "paid", "closed"], message: "Invalid claim status" } },
priority: { type: String, default: "normal", enum: ["low", "normal", "high", "urgent"] },
requestAmount: { type: Number, required: [true, "Request amount is required"], min: [0, "Request amount cannot be negative"] },
approvedAmount: { type: Number, default: 0, min: [0, "Approved amount cannot be negative"] },
submittedAt: { type: Date, default: Date.now, required: [true, "Submitted date is required"] },
reviewedAt: { type: Date },
resolvedAt: { type: Date },
remarks: { type: String, trim: true, maxlength: [1000, "Remarks cannot exceed 1000 characters"] },
incidentDate: { type: Date, required: [true, "Incident date is required"], validate: { validator: function (v) { return v <= new Date(); }, message: "Incident date cannot be in the future" } },
description: { type: String, required: [true, "Claim description is required"], trim: true, maxlength: [1000, "Description cannot exceed 1000 characters"] },
documents: [{ type: mongoose.Schema.Types.ObjectId, ref: "Document" }],
reviewedBy: { type: String, ref: "User" },
dependentId: { type: mongoose.Schema.Types.ObjectId },
insuranceCompany: { type: String, required: [true, "Insurance company is required"], trim: true }
}, { timestamps: true });

ClaimSchema.index({ claimId: 1 });
ClaimSchema.index({ userId: 1 });
ClaimSchema.index({ policyId: 1 });
ClaimSchema.index({ status: 1 });
ClaimSchema.index({ type: 1 });
ClaimSchema.index({ incidentDate: -1 });
ClaimSchema.index({ submittedAt: -1 });
ClaimSchema.index({ insuranceCompany: 1 });

const Claim = mongoose.model("Claim", ClaimSchema);
export default Claim;
```

Document Schema

```
import mongoose from "mongoose";

const DocumentSchema = new mongoose.Schema({
filename: { type: String, required: [true, "Filename is required"], trim: true, maxlength: [255, "Filename cannot exceed 255 characters"] },
originalName: { type: String, required: [true, "Original filename is required"], trim: true, maxlength: [255, "Original filename cannot exceed 255 characters"] },
type: { type: String, required: [true, "Document type is required"], enum: { values: ["policy", "claim", "user", "general"], message: "Type must be either policy, claim, user, or general" } },
docType: { type: String, required: [true, "Document category is required"], enum: { values: ["nic", "passport", "invoice", "medical_bill", "police_report", "photo", "receipt", "policy_document", "claim_form", "supporting_document", "other"], message: "Invalid document category" } },
mimeType: { type: String, required: [true, "MIME type is required"], validate: { validator: function (v) { const allowedTypes = ["image/jpeg", "image/jpg", "image/png", "image/gif", "image/webp", "application/pdf", "application/msword", "application/vnd.openxmlformats-officedocument.wordprocessingml.document", "application/vnd.ms-excel", "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet", "text/plain", "text/csv"]; return allowedTypes.includes(v); }, message: "File type not supported" } },
size: { type: Number, required: [true, "File size is required"], min: [1, "File size must be greater than 0"], max: [10 * 1024 * 1024, "File size cannot exceed 10MB"] },
azureBlobUrl: { type: String, required: [true, "Azure blob URL is required"], trim: true },
azureContainerName: { type: String, required: [true, "Azure container name is required"], trim: true },
userId: { type: String, trim: true, default: null }, // Will be populated when user authentication is implemented
refId: { type: mongoose.Schema.Types.ObjectId, default: null }, // Reference to related claim, policy, or user document
uploadedBy: { type: String, required: [true, "Uploader information is required"], trim: true },
uploadedByRole: { type: String, required: [true, "Uploader role is required"], enum: { values: ["employee", "hr_officer", "insurance_agent", "admin"], message: "Invalid uploader role" } },
status: { type: String, default: "active", enum: { values: ["active", "archived", "deleted"], message: "Status must be active, archived, or deleted" } },
isVerified: { type: Boolean, default: false },
verifiedBy: { type: String, trim: true, default: null },
verifiedAt: { type: Date, default: null },
metadata: {
description: { type: String, maxlength: [500, "Description cannot exceed 500 characters"], trim: true },
tags: [{ type: String, trim: true, maxlength: [50, "Tag cannot exceed 50 characters"] }],
version: { type: Number, default: 1, min: [1, "Version must be at least 1"] },
isConfidential: { type: Boolean, default: false },
expiryDate: { type: Date, default: null },
documentNumber: { type: String, trim: true, default: null }
},
```

```
accessLog: [{
accessedBy: { type: String, required: true },
accessedByRole: { type: String, required: true, enum: ["employee", "hr_officer",
"insurance_agent", "admin"] },
accessedAt: { type: Date, default: Date.now },
action: { type: String, required: true, enum: ["view", "download", "update", "delete"] }
}]
}, { timestamps: true });

DocumentSchema.index({ type: 1, docType: 1 });
DocumentSchema.index({ userId: 1 });
DocumentSchema.index({ refId: 1 });
DocumentSchema.index({ uploadedBy: 1 });
DocumentSchema.index({ createdAt: -1 });
DocumentSchema.index({ status: 1 });

const Document = mongoose.model("Document", DocumentSchema);
export default Document;
```

Message Schema

```
import mongoose from "mongoose";

const MessageSchema = new mongoose.Schema({
conversationId: { type: String, required: [true, "Conversation ID is required"], trim: true, index: true },
sender: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: [true, "Sender is required"] },
recipients: [{
user: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },
readAt: { type: Date },
deliveredAt: { type: Date, default: Date.now }
}],
content: {
text: { type: String, required: [true, "Message content is required"], trim: true, maxlength: [2000, "Message cannot exceed 2000 characters"] },
attachments: [{ type: mongoose.Schema.Types.ObjectId, ref: "Document" }]
},
messageType: { type: String, default: "text", enum: ["text", "file", "system", "notification"] },
context: {
relatedTo: { type: String, enum: ["claim", "policy", "general", "support"] },
referenceId: { type: mongoose.Schema.Types.ObjectId }
},
priority: { type: String, default: "normal", enum: ["low", "normal", "high", "urgent"] },
status: { type: String, default: "sent", enum: ["draft", "sent", "delivered", "read", "archived"] },
isEdited: { type: Boolean, default: false },
editedAt: { type: Date },
reactions: [{
user: { type: mongoose.Schema.Types.ObjectId, ref: "User" },
reaction: { type: String, enum: ["like", "dislike", "helpful", "resolved"] },
timestamp: { type: Date, default: Date.now }
}]
}, { timestamps: true });

MessageSchema.index({ conversationId: 1, createdAt: -1 });
MessageSchema.index({ sender: 1 });
MessageSchema.index({ "recipients.user": 1 });
MessageSchema.index({ status: 1 });

const Message = mongoose.model("Message", MessageSchema);
export default Message;
```

Notification Schema

```
import mongoose from "mongoose";

const NotificationSchema = new mongoose.Schema({
recipient: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: [true, "Recipient is
required"] },
title: { type: String, required: [true, "Notification title is required"], trim: true, maxlength: [100,
"Title cannot exceed 100 characters"] },
message: { type: String, required: [true, "Notification message is required"], trim: true,
maxlength: [500, "Message cannot exceed 500 characters"] },
type: { type: String, required: [true, "Notification type is required"], enum: { values:
["claim_status", "policy_update", "document_request", "payment_processed", "system",
"reminder", "alert"], message: "Invalid notification type" } },
category: { type: String, default: "info", enum: ["info", "success", "warning", "error"] },
relatedTo: {
model: { type: String, enum: ["Claim", "Policy", "User", "Message"] },
id: { type: mongoose.Schema.Types.ObjectId }
},
actionRequired: { type: Boolean, default: false },
actionUrl: { type: String, trim: true },
expiresAt: { type: Date },
priority: { type: String, default: "normal", enum: ["low", "normal", "high", "urgent"] },
status: { type: String, default: "active", enum: ["active", "read", "archived", "expired"] }
}, { timestamps: true });

NotificationSchema.index({ recipient: 1, createdAt: -1 });
NotificationSchema.index({ type: 1 });
NotificationSchema.index({ status: 1 });
NotificationSchema.index({ expiresAt: 1 });

const Notification = mongoose.model("Notification", NotificationSchema);
export default Notification;
```

DB and Storage Hosting