1. Annexes

1. Schemas:

```
var PatientSchema = new mongoose.Schema({
    identifier: [SubDocs.Identifier],
    active: Boolean, // Whether this patient's record is in active use
    name: SubDocs.HumanName,
    telecom: [SubDocs.ContactPoint],
    gender: String,
    birthDate: Date,

    // Indicates if the individual is deceased or not
    deceasedBoolean: Boolean,
    deceasedDateTime: Date,

    address: [SubDocs.Address],
    profession:[String],
    maritalStatus: SubDocs.CodeableConcept,// Marital (civil) status of a patient

    // Whether patient is part of a multiple birth
    multipleBirthBoolean: Boolean,
    multipleBirthInteger: Number,

    photo: [SubDocs.Attachment],// Attached images
    contact: [{//  A contact party (e.g. guardian, partner, friend) for the
patient
        relationship: [SubDocs.CodeableConcept], // The kind of relationship
        name: SubDocs.HumanName,
        telecom: [SubDocs.ContactPoint],
        address: [SubDocs.Address],
        gender: String,
        organization: SubDocs.Reference, // Organization that is associated with
the contact
        period: SubDocs.Period// The period during which this contact person or
organization is valid to be contacted relating to this patient

    }],
    communication: [{ //  A list of Languages which may be used to communicate
with the patient about his or her health
        language: SubDocs.CodeableConcept,   // The language which can be used to
communicate with the patient about his or her health
        preferred: Boolean // Language preference indicator
    }],
    careProvider: [SubDocs.Reference], // Patient's nominated primary care
provider
    managingOrganization: SubDocs.Reference,// Organization that is the custodian
of the patient record
    link: [{ // Link to another patient resource that concerns the same actual
person
        other: SubDocs.Reference, // The other patient resource that the link
refers to
        linkType: String, // replace | refer | seealso - type of link
    }]
});
```

```
var PractitionerSchema = new mongoose.Schema({
      identifier: [SubDocs.Identifier],
    active: Boolean,// Whether this practitioner's record is in active use
    name: SubDocs.HumanName,
    telecom: [SubDocs.ContactPoint],
    address: [SubDocs.Address],
    gender: String,
    birthDate: Date,
    photo: [SubDocs.Attachment],// Attached images
    practitionerRole: [{  // Roles/organizations the practitioner is associated
with
        managingOrganization: SubDocs.Reference,
        role: SubDocs.CodeableConcept, // Roles which this practitioner may
perform
        specialty: [SubDocs.CodeableConcept],// Specific specialty of the
practitioner
        period: SubDocs.Period, // The period during which the practitioner is
authorized to perform in these role(s)
        location: [SubDocs.Reference], //The location(s) at which this
practitioner provides care
        healthcareService: [SubDocs.Reference]
    }],
    qualification: [{//   Qualifications obtained by training and certification
        identifier: [SubDocs.Identifier],// An identifier for this qualification
for the practitioner
        code: SubDocs.CodeableConcept, // Coded representation of the
qualification
        period: SubDocs.Period, // Period during which the qualification is valid
        issuer: SubDocs.Reference,// Organization that regulates and issues the
qualification
    }],
    communication: [SubDocs.CodeableConcept],// A language the practitioner is
able to use in patient communication
});
```

```
var ConditionSchema = new mongoose.Schema({
      identifier : [ SubDocs.Identifier ],
      patient : SubDocs.Reference,     // Who has the condition?
      encounter : SubDocs.Reference,  // Encounter when condition first asserted
      asserter : SubDocs.Reference,    // Person who asserts this condition
      dateRecorded : Date,             // When first entered
      code : SubDocs.CodeableConcept, // Identification of the condition, problem
or diagnosis
      category : SubDocs.CodeableConcept, // complaint | symptom | finding |
diagnosis
      clinicalStatus : {
            type : String,
            enum : [ 'active', 'relapse', 'remission', 'resolved' ],
            default: 'active'
      },
      verificationStatus : {
```

```
            type : String,
            required : true,
            enum : [ 'provisional', 'differential', 'confirmed', 'refuted',
                        'entered-in-error', 'unknown' ],
            default: 'confirmed'
      },
      severity : SubDocs.CodeableConcept,// Subjective severity of condition
      // Estimated or actual date, date-time, or age
      onsetDateTime : Date,
      onsetQuantity : SubDocs.Quantity,
      onsetPeriod :  SubDocs.Period,
      onsetString : String,
      // If/when in resolution/remission
      abatementDateTime : Date,
      abatementQuantity :  SubDocs.Quantity,
      abatementBoolean : Boolean,
      abatementPeriod :  SubDocs.Period,
      abatementRange : {low : String, high : String},
      abatementString : String,
      stage : {                                    // Stage/grade, usually
assessed formally
            summary : SubDocs.CodeableConcept,     // Simple summary (disease
specific)
            assessment : [ SubDocs.Reference],     // Formal record of assessment
      },
      evidence : [ {                          // Supporting evidence
            code : SubDocs.CodeableConcept, // Manifestation/symptom
            detail : [ SubDocs.Reference],  // Supporting information found
elsewhere
      } ],
      bodySite : [ SubDocs.CodeableConcept ],// Anatomical location, if relevant
      notes : String                          // Additional information about the
Condition
});
```

```
var ObservationSchema = new mongoose.Schema({
    identifier: [SubDocs.Identifier],
    status: String,                    // registered | preliminary | final |
amended +
    category: SubDocs.CodeableConcept, // Classification of type of observation:
                                       // social-history, risk-factor, vital-
signs, imaging, laboratory, procedure, survey, exam, therapy
    code: SubDocs.CodeableConcept,     // Type of observation (code / type)
    subject: SubDocs.Reference,        // Who and/or what this is about
    encounter: SubDocs.Reference,      // Healthcare event during which this
observation is made
    effectiveDateTime: Date,           // Clinically relevant time/time-period for
observation
    effectivePeriod: SubDocs.Period,
    issued: Date,                      // Date/Time this was made available
    performer: [SubDocs.Reference],    // Who is responsible for the observation
    // Actual result
    valueQuantity: SubDocs.Quantity,
    valueCodeableConcept: SubDocs.CodeableConcept,
    valueString: String,
    valueRange: SubDocs.Range,
```

```
    valueRatio: SubDocs.Ratio,
    valueSampledData: SubDocs.SampledData,
    valueAttachment: SubDocs.Attachment,
    valueTime: String,    // A time during the day, with no date specified
                          // Regex: ([01][0-9]|2[0-3]):[0-5][0-9]:[0-5][0-9](\.[0-
9]+)?
    valueDateTime: Date,
    valuePeriod: SubDocs.Period,
    dataAbsentReason: SubDocs.CodeableConcept,// Why the result is missing
    interpretation: SubDocs.CodeableConcept, // High, low, normal, etc.
    comments: String,                        // Comments about result
    bodySite: SubDocs.CodeableConcept,
    method: SubDocs.CodeableConcept,         // How it was done
    specimen: SubDocs.Reference,
    device: SubDocs.Reference,
    referenceRange: [{          // Provides guide for interpretation
                                // Must have at least a low or a high or text
        low: String,
        high: String,
        meaning: SubDocs.CodeableConcept,// Indicates the meaning/use of this
range of this range
        age: SubDocs.Range,     // Applicable age range, if relevant
        text: String,           // Text based reference range in an observation
    }],
    related: [{           // Resource related to this observation
        fhirType: String, // has-member | derived-from | sequel-to | replaces |
qualified-by | interfered-by
        target: SubDocs.Reference,
    }],
    component: [{                     // Component results
        code: SubDocs.CodeableConcept,
        valueQuantity: SubDocs.Quantity,
        valueCodeableConcept: SubDocs.CodeableConcept,
        valueString: String,
        valueRange: SubDocs.Range,
        valueRatio: SubDocs.Ratio,
        valueSampledData: SubDocs.SampledData,
          valueAttachment: SubDocs.Attachment,
        valueTime: String,
        valueDateTime: Date,
        valuePeriod: SubDocs.Period,
        dataAbsentReason: SubDocs.CodeableConcept,  // Why the result is missing
        referenceRange: [{            // Provides guide for interpretation
                                      // Must have at least a low or a high or
text
            low: String,
            high: String,
            meaning: SubDocs.CodeableConcept,// Indicates the meaning/use of this
range of this range
            age: SubDocs.Range,       // Applicable age range, if relevant
            text: String,             // Text based reference range in an
observation
        }]
    }]
});



var EncounterSchema = new mongoose.Schema({
```

```
      identifier: [SubDocs.Identifier],
    status: {
      type: String,
      enum : ["planned" , "arrived" , "in-progress" , "onleave" , "finished" ,
"cancelled"],
      required : true,
      default: 'finished'
      },
    statusHistory: [{// List of past encounter statuses
        status: {
             type: String,
             enum : ["planned" , "arrived" , "in-progress" , "onleave" ,
"finished" , "cancelled"],
             required : true
      },
        period: SubDocs.Period
    }],
    classification: {
      type: String,
      enum : ["inpatient" , "outpatient" , "ambulatory" , "emergency", "home",
"field", "daytime", "virtual", "other"],
      required: true,
            default: 'ambulatory'
    },
    encounterType: [SubDocs.CodeableConcept],// Specific type of encounter
    priority: SubDocs.CodeableConcept, //Indicates the urgency of the encounter
    patient: SubDocs.Reference, // The patient present at the encounter
    episodeOfCare: [SubDocs.Reference],// Episode(s) of care that this encounter
should be recorded against
    incomingReferral: [SubDocs.Reference],// The ReferralRequest that initiated
this encounter
    participant: [{// List of participants involved in the encounter
        role: [SubDocs.CodeableConcept],// Role of participant in encounter
        period: SubDocs.Period, // Period of time during the encounter participant
was present
        individual: SubDocs.Reference,// Persons involved in the encounter other
than the patient
    }],
    appointment: SubDocs.Reference,// The appointment that scheduled this
encounter
    period: SubDocs.Period,// The start and end time of the encounter
    length: SubDocs.Quantity,// Quantity of time the encounter lasted (less time
absent)
    reason: [SubDocs.CodeableConcept], // Reason the encounter takes place (code)
    indication: [SubDocs.Reference],// Reason the encounter takes place (resource:
condition, procedure)
    hospitalization: {    // Details about the admission to a healthcare service
        preAdmissionIdentifier: SubDocs.Identifier,
        origin: SubDocs.Reference,// The location from which the patient came
before admission
        admitSource: SubDocs.CodeableConcept,// From where patient was admitted
(physician referral, transfer)
        admittingDiagnosis: [SubDocs.Reference], // The admitting diagnosis as
reported by admitting practitioner (condition)
        reAdmission: SubDocs.CodeableConcept,// The type of hospital re-admission
that has occurred (if any). If the value is absent, then this is not identified as
a readmission
```

```
        dietPreference: [SubDocs.CodeableConcept],  // Diet preferences reported
by the patient
        specialCourtesy: [SubDocs.CodeableConcept], // Special courtesies (VIP,
board member)
        specialArrangement: [SubDocs.CodeableConcept],// Wheelchair, translator,
stretcher, etc.
        destination: SubDocs.Reference,      // Location to which the patient is
discharged
        dischargeDisposition: SubDocs.CodeableConcept,// Category or kind of
location after discharge
        dischargeDiagnosis: [SubDocs.Reference]// The final diagnosis given a
patient before release from the hospital after all testing, surgery, and workup
are complete

    },
    location: [{    // List of locations where the patient has been
        location: SubDocs.Reference,// Location the encounter takes place
        status: {               //     planned | active | reserved | completed

             type: String,      // EncounterLocationStatus (Required)
             enum : ["planned" , "active" , "reserved" , "completed"],
             required: true,
             default: 'completed'
        },
        period: SubDocs.Period  // Time period during which the patient was
present at the location
    }],
    serviceProvider: SubDocs.Reference,// The custodian organization of this
Encounter record
    partOf: SubDocs.Reference,         // Another Encounter this encounter is part
of
    conclusion : String                // (Not in original model)
});




var DiagnosticReportSchema = new mongoose.Schema({
      identifier: [SubDocs.Identifier],
    status: {                                          // planned | arrived |
in-progress | onleave | finished | cancelled
      type: String,
      enum : ["register" , "partial" , "final" , "corrected" , "appended" ,
"cancelled", "entered-in-error"],
      required : true,
      default: 'final'
    },
    category: SubDocs.CodeableConcept, // Service category ex: BLB Blood Bank, CG
Cytogenetics, CH Chemistry, CP CAT Scan,...
    code: SubDocs.CodeableConcept,// Name/Code for this diagnostic report
    subject: SubDocs.Reference, // The subject of the report, usually, but not
always, the patient
    encounter: SubDocs.Reference,      // Health care event when test ordered
    effectiveDateTime: Date,           // Clinically Relevant time/time-period for
report
    effectivePeriod: SubDocs.Period,   // Clinically Relevant time/time-period for
report
    issued: Date,                // DateTime this version was released
    performer: SubDocs.Reference,// Responsible Diagnostic Service
```

```
    request: [SubDocs.Reference],// What was requested (DiagnosticOrder |
ProcedureRequest | ReferralRequest)
    specimen: [SubDocs.Reference],     // Specimens this report is based on
    result: [SubDocs.Reference],       // Observations - simple, or complex nested
groups
    imagingStudy: [SubDocs.Reference], // Reference to full details of imaging
associated with the diagnostic report
    image: [{                                // Key images associated with this
report
        comment: String,                     // Comment about the image (e.g.
explanation)
        link: SubDocs.Reference              // Reference to the image source
    }],
    conclusion: String,// Clinical Interpretation of test results
    codedDiagnosis: [SubDocs.CodeableConcept],// Codes for the conclusion
    presentedForm: [SubDocs.Attachment]      // Entire report as issued
});



var DiagnosticOrderSchema = new mongoose.Schema({
    subject: SubDocs.Reference,
    orderer: SubDocs.Reference,
    identifier: [SubDocs.Identifier],
    encounter: SubDocs.Reference,
    reason: [SubDocs.CodeableConcept],
    supportingInformation: [SubDocs.Reference],
    specimen: [SubDocs.Reference],
    status: {
             type : String,
             enum : [ 'proposed', 'draft', 'planned', 'requested', 'received',
'accepted', 'in-progress', 'review', 'completed', 'cancelled', 'suspended',
'rejected', 'failed' ],
             required: true,
      default: 'requested'
      },
    priority: {
      type: String,
             enum : [ 'routine', 'urgent', 'stat', 'asap'],
             required : true,
             default: 'routine'
      },
    event: [{
        status: {
             type : String,
             enum : [ 'proposed', 'draft', 'planned', 'requested', 'received',
'accepted', 'in-progress', 'review', 'completed', 'cancelled', 'suspended',
'rejected', 'failed' ],
             required: true,
             default: 'requested'
      },
        description: SubDocs.CodeableConcept,
        dateTime: Date,
        actor: SubDocs.Reference
    }],
    item: [{
        code: SubDocs.CodeableConcept,
        specimen: [SubDocs.Reference],
```

```
        bodySite: SubDocs.CodeableConcept,
        status: {
             type : String,
             enum : [ 'proposed', 'draft', 'planned', 'requested', 'received',
'accepted', 'in-progress', 'review', 'completed', 'cancelled', 'suspended',
'rejected', 'failed' ],
             required: true,
             default: 'requested'
      },
        event: [{
        }]
    }],
    note: SubDocs.Annotation
});



var MedicationOrderSchema = new mongoose.Schema({
      identifier: [SubDocs.Identifier],
    dateWritten: Date,    // When prescription was authorized
    status: {
             type : String,
             enum : [ 'active', 'on-hold', 'completed', 'entered-in-error',
'stopped', 'draft' ],
             required : true
      },
    dateEnded: Date,                    // When prescription was stopped
    reasonEnded: SubDocs.CodeableConcept,// Why the prescription was stopped
    patient: SubDocs.Reference,
    prescriber: SubDocs.Reference,
    encounter: SubDocs.Reference,      // Created during encounter/admission/stay
    reasonCodeableConcept: SubDocs.CodeableConcept, // Reason or indication for
writing the prescription
    reasonReference: SubDocs.Reference,
    note: String,  // Information about the prescription
    medicationReference: SubDocs.Reference,  // Medication to be taken

    // How medication should be taken
    dosageInstruction: [{
        text: String,     // Dosage instructions expressed as text
        additionalInstructions: SubDocs.CodeableConcept,// Supplemental
instructions - e.g. "with meals"
        timing: SubDocs.Timing, // When medication should be administered
        asNeededBoolean: Boolean,
        asNeededCodeableConcept: SubDocs.CodeableConcept,
        siteCodeableConcept: SubDocs.CodeableConcept,// Body site to administer to
        siteReference : SubDocs.Reference,
        route: SubDocs.CodeableConcept,// How drug should enter body
        method: SubDocs.CodeableConcept,// Technique for administering medication
        // Amount of medication per dose
        doseRange: SubDocs.Range,// Ex: between x ml and x ml
        doseQuantity: String,
        // Amount of medication per unit of time
        rateRatio: SubDocs.Ratio,
        rateRange: SubDocs.Range,
        // Upper limit on medication per unit of time
        maxDosePerPeriod: SubDocs.Ratio
    }],
```

```
    // Medication supply authorization
    dispenseRequest : {
      //Product to be supplied
      medicationCodeableConcept: SubDocs.CodeableConcept,
      medicationReference: SubDocs.Reference,
      validityPeriod: SubDocs.Period, // Time period supply is authorized for
      numberOfRepeatsAllowed: {type: Number, min: 0}, // Number of refills
authorized
      quantity: String,               // Amount of medication to supply per
dispense
      expectedSupplyDuration: String  // Number of days supply per dispense
    },
    // Any restrictions on medication substitution
    substitution : {
      substitutionType: SubDocs.CodeableConcept,// generic | formulary +
      reason: SubDocs.CodeableConcept // Why should (not) substitution be made
    },
    priorPrescription : SubDocs.Reference// An order/prescription that this
supersedes
});



/**
 * Concept - reference to a terminology or just text
 */
var CodeableConceptSchema = new mongoose.Schema({
      coding : [ {                    // Code defined by a terminology system
            system : String,          // Identity of the terminology system
            version : String,         // Version of the system - if relevant
            code : String,            // Symbol in syntax defined by the system
(this is where we put the SNOMED CT code)
            display : String,         // Representation defined by the system
            userSelected : Boolean    // If this coding was chosen directly by
the user
      } ],
      text : String                   // Plain text representation of the concept
   });


/**
 * A reference from one resource to another
 */
var ReferenceSchema = new mongoose.Schema({
      reference : String,             // Relative, internal or absolute URL
reference
      display : String                // Text alternative for the resource
   });


/**
 * Time range defined by start and end date/time
 */
var PeriodSchema = new mongoose.Schema({
      start : Date,
      end : Date
   });


/**
```

```
 * An identifier intended for computation
 */
var IdentifierSchema = new mongoose.Schema({
      use : {
            type : String,
            enum : [ 'usual', 'official', 'temp', 'secondary' ],
            required : true,
            default: 'usual'
      },
      idType : CodeableConceptSchema, // Description of identifier
      system : String,                // The namespace for the identifier (uri)
      value : String,                 // The value that is unique
      period : PeriodSchema,          // Time period when id is/was valid for use
      assigner : ReferenceSchema       // Organization that issued id (may be
just text)
   });

/**
 * A postal address
 */
var AddressSchema = new mongoose.Schema({
    use : {                           // purpose of this address
      type: String,
            enum : [ 'home', 'word', 'temp', 'old' ],
            required : true,
            default: 'home'
    },
    addressType : {
      type: String,
            enum : [ 'postal', 'physical', 'both'],
            required : true,
            default: 'postal'
    },
    text : String,                // Text representation of the address
    line : [String],              // Street name, number, direction & P.O. Box etc.
    city : String,                // Name of city, town etc.
    district : String,            // District name (aka county)
    state : String,               // Sub-unit of country (abbreviations ok)
    postalCode : String,          // Postal code for area
    country : String,             // Country (can be ISO 3166 3 letter code)
    period : PeriodSchema         // Time period when address was/is in use
   });

/**
 *    Details of a Technology mediated contact point (phone, fax, email, etc.)
 */
var ContactPointSchema = new mongoose.Schema({
        system : {                    // C? phone | fax | email | pager | other
            type : String,
            enum : [ 'phone', 'fax', 'email', 'pager', 'other' ],
            required : true,
            default: 'phone'
            },
        value : String,               // The actual contact point details
        use : {                             // purpose of this contact point
            type : String,
            enum : [ 'home', 'work', 'temp', 'old', 'mobile' ],
            required : true,
```

```
            default: 'home'
            },
        rank : Number,          // Specify preferred order of use (1 = highest)
        period : PeriodSchema   // Time period when the contact point was/is in
use
   });

/**
 * Name of a human - parts and usage
 */
var HumanNameSchema = new mongoose.Schema({
    use: {
            type : String,
            enum : [ 'usual', 'official', 'temp', 'nickname', 'anonymous', 'old',
'maiden' ],
            required : true,
            default: 'usual'
      },
    text: String,         // Text representation of the full name
    family: [String],     // Family name (often called 'Surname')
    given: [String],      // Given names (not always 'first'). Includes middle
names
    prefix: [String],     // Parts that come before the name
    suffix: [String],     // Parts that come after the name
    period: PeriodSchema  // Time period when name was/is in use
   });

var AttachmentSchema = new mongoose.Schema({
      contentType : {           // Mime type of the content, with charset etc.

            type : String,
            required : true,
            default: 'image/png'
      },
      language : {                    // Human language of the content (BCP-47)

            type : String,
            required : true,
            default: 'en'
      },
      data : Buffer,                  // Data inline, base64ed
      url : String,                   // Uri where the data can be found
      size : {                        // Number of bytes of content (if url
provided)
            type : Number,
            min : 0
      },
      hash : Buffer,                  // Hash of the data (sha-1, base64ed)
      title : String,                 // Label to display in place of the data
      creation : Date                 // Date attachment was first created
   });

/**
 * A measured amount (or an amount that can potentially be measured).
 */
var QuantitySchema = new mongoose.Schema({
    value : String,             // Numerical value (with implicit precision)
```

```
    comparator : {              // How to understand the value
            type : String,
            enum : [ '=', '<', '<=', '>=', '>' ],
            required : true,
            default: '='
      },
    unit : String,              // Unit representation
    system : String,            // System that defines coded unit form
    code : String               // Coded form of the unit
   });

/**
 * A set of ordered Quantity values defined by a low and high limit.
 */
var RangeSchema = new mongoose.Schema({
      low : String,
      high : String
});



/**
 * A relationship between two Quantity values expressed as a numerator and a
denominator.
 */
var RatioSchema = new mongoose.Schema({
      numerator : QuantitySchema,
      denominator: QuantitySchema
   });

/**
 * Data that comes from a series of measurements taken by
 * a device, with upper and lower limits.
 * There may be more than one dimension in the data.
 */
var SampledDataSchema = new mongoose.Schema({
        origin : String,              // Zero value and units
        period : Number,              // Number of milliseconds between samples
        factor : Number,              // Multiply data by this before adding to
origin
        lowerLimit : Number,          // Lower limit of detection
        upperLimit : Number,          // Upper limit of detection
        dimensions : {                // Number of sample points at each time
point
                    type : Number,
                    min : 0
            },
        data : String                 // Decimal values with spaces, or "E" | "U"
| "L"
   });

/**
 * A text note which also contains information about who made the statement and
when.
 */
var AnnotationSchema = new mongoose.Schema({
      //     Individual responsible for the annotation
```

```
        authorReference : ReferenceSchema,
        authorString: String,
        // When the annotation was made
        time: Date,
        //     The annotation - text content
        text: String
    });

/**
 * A timing schedule that specifies an event that may occur multiple times.
 */
var TimingSchema = new mongoose.Schema({
        event: [Date],                  // When the event occurs
        /*     When the event is to occur
               Either frequency or when can exist, not both
               if there's a duration, there needs to be duration units
               if there's a period, there needs to be period units
               If there's a periodMax, there must be a period
               If there's a durationMax, there must be a duration
         */
        repeat :{
               // Length/Range of lengths, or (Start and/or end) limits
               boundsQuantity: String,
               boundsRange: RangeSchema,
               boundsPeriod: PeriodSchema,
               duration : Number,        // How long when it happens
            durationMax : Number,        // How long when it happens (Max)
            durationUnits : {
                     type : String,
                     enum : [ 's', 'min', 'h', 'd', 'wk', 'mo', 'a' ]
               },
            frequency : Number,          // Event occurs frequency times per period
            frequencyMax : Number,       // Event occurs up to frequencyMax times
per period
            period : Number,             // Event occurs frequency times per period
            periodMax : Number,          // Upper limit of period (3-4 hours)
            periodUnits : {
                     type : String,
                     enum : [ 's', 'min', 'h', 'd', 'wk', 'mo', 'a' ]
               },
            when : String                // Regular life events the event is tied to
        },
        code : CodeableConceptSchema     // QD | QOD | Q4H | Q6H | BID | TID | QID |
AM | PM + (TimingAbbreviation)
});

var UserSchema = new mongoose.Schema({
        created : {
               type : Date,
               required: true
        },
        reference : {       // refer to a patient and/or practitioner resource
               familyName: {
                     type : String,
                     required : false
               },
               patientId : {
                     type : String,
```

```
                required : false
        },
        practitionerId : {
                type : String,
                required : false
        }
    },
    email : {
        type : String,
        required : true,
        unique : true
    },
    password : {
        type : String,
        required : true
    },
    language: {
        type: String,
        required: false
    },
    isPatient: {
        type: Boolean,
        required: true
    },
    isPractitioner: {
        type: Boolean,
        required: true
    }
});

var ResourceHistorySchema = new mongoose.Schema({
  resourceType: String,
  createdBy: String,
  history: [{resourceId: mongoose.Schema.Types.ObjectId, updatedBy: String}]
});
```