052400-1 VU Information Management and Systems Engineering (2025S)

Milestone 1

(Please note: submission deadline: Tue 11.04.2025 13:00)

Group 12

Student 1: Kadkhoda Masoumali, Samina, 52308861

Student 2: Matsas, Mario, 12446870

(*Please note: sort by last name*)

10/4/2025, Vienna

Contents

Requirements	Realization
Entities	
6- 9 Entities with key attribute & at least two other attributes	
	User
	Patient
	Doctor
	Prescription
	PrescriptionLine
	ActiveSubstance
	PharmaceuticalProduct
At least one weak entity	
	PrescriptionLine, Strong entity: Prescription, Prescriptions contain PrescriptionLines
Relationships	
At least one unary (recursive) relationship	
	Report
At least one binary relationship (1:n)	
	Download
At least one binary relationship (m:n)	
	Examine
At least one IS-A (inheritance) relationship with 2+ child entities	
	Doctor, Patient IS-A User

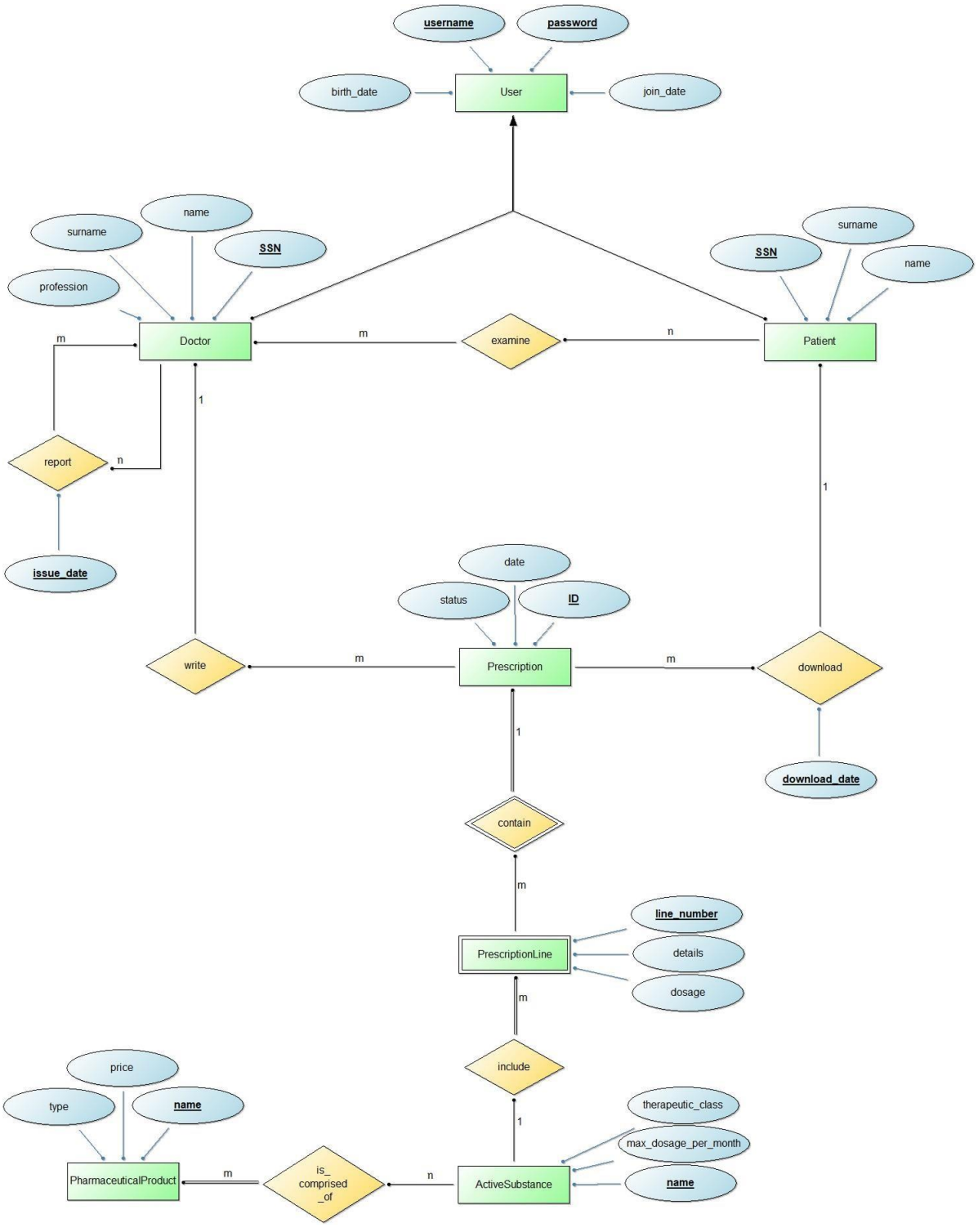
1 Milestone 1

1.1 Team - Conceptual Modeling

1.1.1 Describe the Application Domain

- Main idea: In this project we will create a e-Prescription website, that Doctors and Patients can use to write and view Prescriptions.
- Both the Patient and the Doctor are Users[username, password, birth_date, join_date] and are able to log into their accounts using their credential.
- Any Patient[name, surname, SSN] will be able to visit any Doctor[name, surname, SSN, profession] were he can be examined for any ailments he may have. The Doctor will be able to write many Prescription[ID, date, status]'s (if he visits him more than once) which will be assigned to the Patient, so he can view and download them at any given time.
- Every Prescription will contain at least one PrescriptionLine[line_number, dosage, details] in which the Doctor will have written extra info such as dosage information, or details in regards to the time the patient should use the medication. Every PrescriptionLine will also contain exactly one ActiveSubstance[name, max_dosage_per_month, therapeutic_class] so that when the patient goes to buy the medicine he can show the Prescription to the pharmacist and he will know what PharmaceuticalProduct[name, price, type]'s to give him.
- Every PharmaceuticalProduct must have at least one ActiveSubstance and an ActiveSubstance can appear in more than one PharmaceuticalProduct.
- Finally, when a Doctor is logged in, he will be able to type in a chat box available to all Doctors, where he can report other Doctors on suspicions of malpractice as well as view this week's reports.

1.1.2 Logical Design – ER Diagram in Chen Notation



1.1.3 Relational Modeling – SQL CREATE Statements

User: <u>username</u>, <u>password</u>, birth_date, join_date

Doctor: <u>SSN</u>, <u>username</u>, <u>password</u>, name, surname, profession

Patient: SSN, username, password, name, surname

Prescription: ID, date, status, doctor_id, patient_id

PrescriptionLine: line_number, ID, active_substance_name, dosage, details

ActiveSubstance: name, max_dosage_per_month, therapeutic_class

PharmaceuticalProduct: name, price, type, active_substance_name

Examine: doctor_id, patient_id

HaveActiveSubstance: pharmaceutical_product_name, active_substance_name

Report: doctor_id_reporter, doctor_id_reportee, issue_date

Download: pratient_id, prescription_id, download_date

1.2 Individual - Student 1

Student 1: Kadkhoda Masoumali, Samina, 52308861

1.2.1 Use Case Definition and Design

Version 1 and 2

Textual Description

Downloading Prescription (Weak Entity and IS-A relationship)

Trigger: A patient, who is a type of User, logs into the system to view and download their latest prescription.

Preconditions:

1. The patient must have a valid SSN.
2. The patient must be registered into the system as a verified User.

Main Flow:

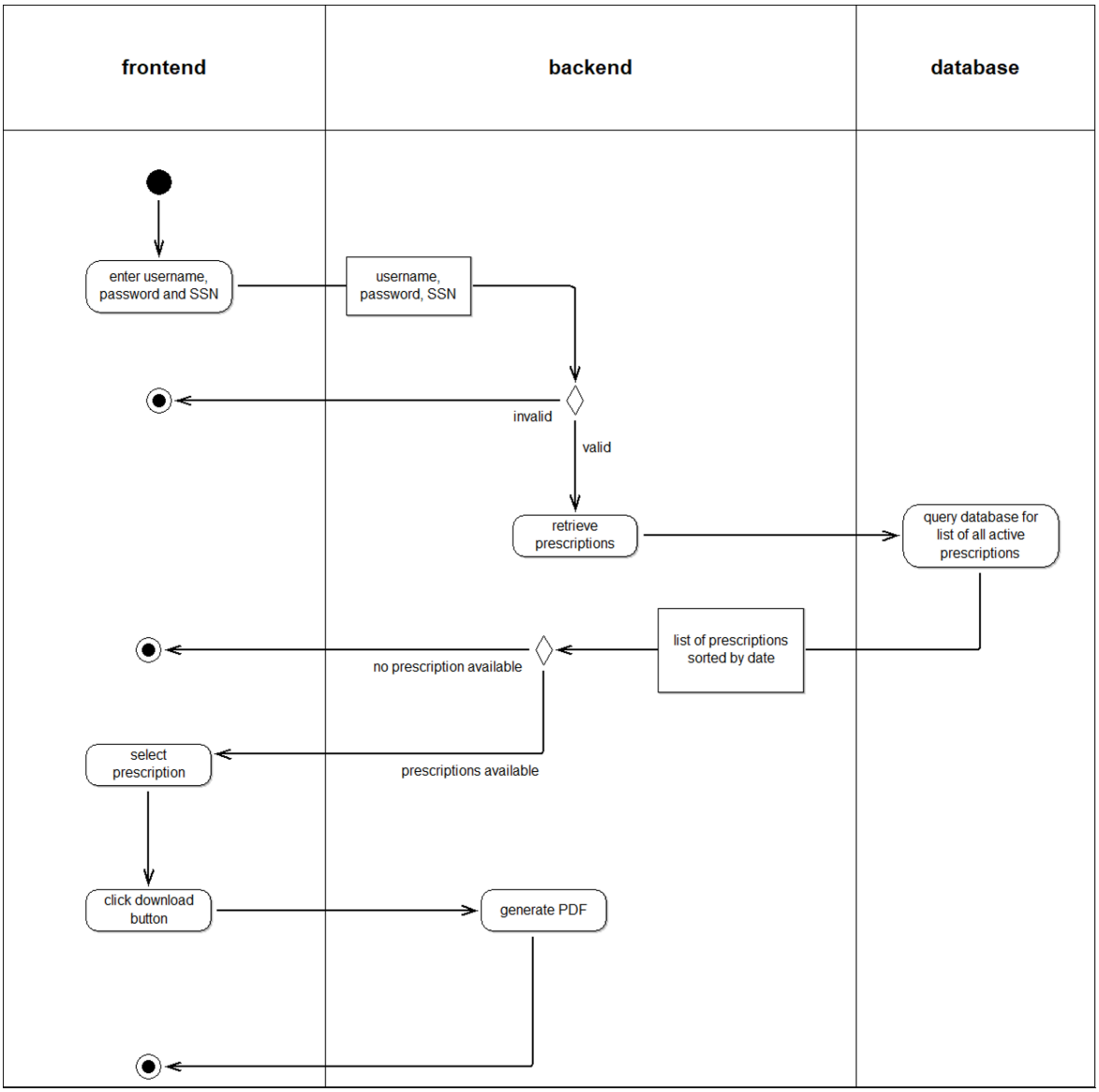
1. The patient, who is a User, enters their username, password, and SSN to log in.
2. The system authenticates the patient and if the entered data is invalid the session will be terminated.
3. If the entered data is valid, the system retrieves their associated prescriptions from the database.
4. The system displays a list of the patient's prescriptions, sorted by date, if they are available.
5. The patient selects a prescription clicks the "Download" button.
6. The system generates a PDF of the prescription details, including the ID, date, status, doctor's info and prescription lines (including Active Substances).

Postconditions:

1. The patient views and accesses the prescription details in PDF format.
2. The system records the download action.

Entities: User, Patient, Prescription, Doctor, PrescriptionLine

Graphical Representation



1.2.2 Analytics Report

Concept

The use case described in the previous section represents the scenario where a patient downloads a prescription. Based on this, I created a report that analyzes this process and is updated whenever the use case is executed.

The report displays the number of prescriptions downloaded by patients, including details such as Patient Information (Name, Surname, SSN), Prescription Details (ID, Date, Status, prescriptionLines), Doctor Information (Name, Surname, Profession, SSN), and the Download Date. The download date is filtered within a specific date range to show only prescriptions downloaded during that period. Also for better readability, details of prescription lines are represented as a single line string in the following format:

“Line) active substance, dosage, details; ”

Ultimately, the results are sorted by download date, with the most recent downloads appearing first. Each time a patient downloads a prescription, a new entry is recorded in the Download table. Running the query before and after executing the use case will show an increase in the number of downloads, reflecting the changes made to the database.

```sql
-- Query to generate analytics report
SELECT
    U.username AS User_Username,
    U.password AS User_Pass,
    P.SSN AS Patient_SSN,
    P.name AS Patient_Name,
    P.surname AS Patient_Surname,
    PR.ID AS Prescription_ID,
    PR.date AS Prescription_Date,
    PR.status AS Prescription_Status,
    D.SSN AS Doctor_SSN,
    D.name AS Doctor_Name,
    D.surname AS Doctor_Surname,
    D.profession AS Doctor_Profession,
    STRING_AGG(
        CONCAT(PL.line_number, ')', PL.active_substance_name, ',',PL.dosage, ',', PL.details),'; '
    ) AS Prescription_Lines,
    DL.download_date AS Download_Date
FROM Download DL
JOIN Prescription PR ON DL.prescription_id = PR.ID
JOIN Patient P ON DL.patient_id = P.SSN
JOIN Doctor D ON PR.doctor_id = D.SSN
JOIN PrescriptionLine PL ON PL.ID = PR.ID
JOIN User U ON P.username = U.username
WHERE DL.download_date BETWEEN '2024-01-01' AND '2024-12-31'
GROUP BY DL.download_date, PR.ID
ORDER BY DL.download_date DESC;
```

Proof of Concept

1. First, the related tables were created according to part 1.1.3.
2. Then, I added some random data to the tables related to the use case.
3. Then I executed the query. The results shown here are for before executing the use case, so we can refer to these results and analyze the changes. The screenshot, which is shown below, is sorted by download date, just like how it was explained in the concept part.


```
[
  {
    "User_Username": "pat_sara",
    "User_Pass": "@2",
    "Patient_SSN": "222-22-2222",
    "Patient_Name": "Sara",
    "Patient_Surname": "Taylor",
    "Prescription_ID": "PER-001",
    "Prescription_Date": "2024-03-10",
    "Prescription_Status": "Active",
    "Doctor_SSN": "111-11-1111",
    "Doctor_Name": "Tom",
    "Doctor_Surname": "Smith",
    "Doctor_Profession": "Ophthalmologist",
    "Prescription_Lines": "1)Atorvastatin,10mg,Once daily; 2)Aspirin,75mg,Morning after breakfast",
    "Download_Date": "2024-10-11"
  },
  {
    "User_Username": "pat_ryan",
    "User_Pass": "@6",
    "Patient_SSN": "666-66-6666",
    "Patient_Name": "Ryan",
    "Patient_Surname": "Philips",
    "Prescription_ID": "PER-005",
    "Prescription_Date": "2024-12-09",
    "Prescription_Status": "Active",
    "Doctor_SSN": "333-33-3333",
    "Doctor_Name": "Maria",
    "Doctor_Surname": "Müller",
    "Doctor_Profession": "Neurologist",
    "Prescription_Lines": "1)Simvastatin,20mg,Once daily at bedtime",
    "Download_Date": "2024-06-07"
  },
  {
    "User_Username": "pat_ben",
    "User_Pass": "@4",
    "Patient_SSN": "444-44-4444",
    "Patient_Name": "Ben",
    "Patient_Surname": "Wilson",
    "Prescription_ID": "PER-004",
    "Prescription_Date": "2025-01-03",
    "Prescription_Status": "Active",
    "Doctor_SSN": "555-55-5555",
    "Doctor_Name": "John",
    "Doctor_Surname": "Nova",
    "Doctor_Profession": "Dentist",
    "Prescription_Lines": "1)Omeprazole,40mg,Before meals",
    "Download_Date": "2024-03-23"
  }
]
```

4. Now I assume that the use case is executed once, which means there has been an entry for the table download. Here is the entry:

   INSERT INTO Download (patient_id, prescription_id, download_date) VALUES ('222-22-2222', 'PER-003', '2024-07-14');

5. I execute the query again to see the changes our use case has caused. So, there will be one new data inserted into the results, and they are sorted by download date. This means the use case works correctly.


```
[
  {
    "User_Username": "pat_sara",
    "User_Pass": "@2",
    "Patient_SSN": "222-22-2222",
    "Patient_Name": "Sara",
    "Patient_Surname": "Taylor",
    "Prescription_ID": "PER-001",
    "Prescription_Date": "2024-03-10",
    "Prescription_Status": "Active",
    "Doctor_SSN": "111-11-1111",
    "Doctor_Name": "Tom",
    "Doctor_Surname": "Smith",
    "Doctor_Profession": "Ophthalmologist",
    "Prescription_Lines": "1)Atorvastatin,10mg,Once daily; 2)Aspirin,75mg,Morning after breakfast",
    "Download_Date": "2024-10-11"
  },
  {
    "User_Username": "pat_sara",
    "User_Pass": "@2",
    "Patient_SSN": "222-22-2222",
    "Patient_Name": "Sara",
    "Patient_Surname": "Taylor",
    "Prescription_ID": "PER-003",
    "Prescription_Date": "2024-10-20",
    "Prescription_Status": "Active",
    "Doctor_SSN": "333-33-3333",
    "Doctor_Name": "Maria",
    "Doctor_Surname": "Müller",
    "Doctor_Profession": "Neurologist",
    "Prescription_Lines": "1)Lisinopril,20mg,Once daily with lunch; 2)Hydrochlorothiazide,25mg,Once daily with water",
    "Download_Date": "2024-07-14"
  },
  {
    "User_Username": "pat_ryan",
    "User_Pass": "@6",
    "Patient_SSN": "666-66-6666",
    "Patient_Name": "Ryan",
    "Patient_Surname": "Philips",
    "Prescription_ID": "PER-005",
    "Prescription_Date": "2024-12-09",
    "Prescription_Status": "Active",
    "Doctor_SSN": "333-33-3333",
    "Doctor_Name": "Maria",
    "Doctor_Surname": "Müller",
    "Doctor_Profession": "Neurologist",
    "Prescription_Lines": "1)Simvastatin,20mg,Once daily at bedtime",
    "Download_Date": "2024-06-07"
  },
  {
    "User_Username": "pat_ben",
    "User_Pass": "@4",
    "Patient_SSN": "444-44-4444",
    "Patient_Name": "Ben",
    "Patient_Surname": "Wilson",
    "Prescription_ID": "PER-004",
    "Prescription_Date": "2025-01-03",
    "Prescription_Status": "Active",
    "Doctor_SSN": "555-55-5555",
    "Doctor_Name": "John",
    "Doctor_Surname": "Nova",
    "Doctor_Profession": "Dentist",
    "Prescription_Lines": "1)Omeprazole,40mg,Before meals",
    "Download_Date": "2024-03-23"
  }
]
```

1.2.3 NoSQL Design

Entities: User, Patient, Prescription, Doctor, PrescriptionLine

Design Overview

Patients:


```
{
  "_id": "p123",
  "username": "sara_smith",
  "password": "s123",
  "name": "Sara",
  "surname": "Smith",
  "prescriptions": [
    {
      "prescription_id": "pers1",
      "doctor_id": "d123",
      "date": { "$date": "2024-10-25T00:00:00Z" },
      "status": "Active",
      "prescription_lines": [
        {
          "line_number": 1,
          "dosage": "10mg",
          "details": "Once in the morning",
          "active_substance": {
            "name": "Lisinopril",
            "max_dosage_per_month": 30,
            "therapeutic_class": "ACE Inhibitor"
          }
        },
        {
          "line_number": 2,
          "dosage": "200mg",
```

```
      "details": "Twice daily",
      "active_substance": {
      "name": "Metformin",
        "max_dosage_per_month": 60,
        "therapeutic_class": "Biguanide"
      }
    }
  ],
  "download_history_ids": [
    "download1",
    "download2"
  ]
},
{
  "prescription_id": "pers2",
  "doctor_id": "d456",
  "date": { "$date": "2023-05-17T00:00:00Z" },
  "status": "Completed",
  "prescription_lines": [
    {
      "line_number": 1,
      "dosage": "100mg",
      "details": "Once before lunch",
      "active_substance": {
        "name": "Amoxicillin",
        "max_dosage_per_month": 500,
        "therapeutic_class": "Antibiotic"
      }
    }
```

```
      ],
      "download_history_ids": []
    }
  ]
}
```


Doctors:


```
[
    {
        "_id": "d123",
        "name": "John",
        "surname": "Williams",
        "profession": "General Practitioner"
    },
    {
        "_id": "d456",
        "name": "Lisa",
        "surname": "Jones",
        "profession": "Dentist"
    }
]
```


Downloads:


```
  [
    {
        "_id": "download1",
        "prescription_id": "pers1",
        "patient_id": "p123",
        "download_date": { "$date": "2024-11-28T04:00:00Z" }
    },
    {
```

```
    "_id": "download2",
    "prescription_id": "pers1",
    "patient_id": "p123",
    "download_date": { "$date": "2024-12-01T04:00:00Z" }
  }
]
```


Expected Execution and Possible Changes

Expected Execution:

1. Patient logs in and System authenticates: When a patient attempts to log in, the system performs a read on the user's collection to verify the username. Then, the input password and SSN are compared with the stored values. This is handled in a single read operation. If authentication succeeds, the system continues to retrieve the patient's prescriptions.
2. The system retrieves all prescriptions and displays results: Once logged in, the system fetches all prescriptions embedded within the patient document. Since the data is already part of the patient record, only one read is needed. Each prescription includes doctor_id, prescription lines, and other relevant details, and the system can display them immediately without extra database calls.
3. Patient selects a prescription and System generates a PDF: When the patient selects a prescription, the data is already available from the earlier read. However, since doctor's details are referenced (not embedded), an additional query is made to the doctor's collection using doctor_id to retrieve the doctor's full information. This combined data is used to generate the PDF.
4. System Records Download Action: After the PDF is generated, the system records the download event. A new document is inserted into the downloads collection, which is separate, referencing the patient_id and prescription_id. The download_history array in the patient document stores a reference to this entry, making it easy to track past downloads.

Possible Changes:

1. Frequent Reads for Patient Data and Prescriptions: With embedding, all prescription details are loaded with the patient document, which is fast but can become inefficient as the number of prescriptions increases. It loads more data than needed for most operations. By referencing prescriptions instead, we fetch only basic IDs with the patient and retrieve full prescription details only when needed (e.g., when a patient selects one). However, since the use case is about patients downloading prescriptions, I decided to go with the embedded version to better clarity.

2. Write Frequency for Download History: If downloads happen rarely, embedding download history under each prescription could work. However, since downloads can grow over time, embedding may increase document size and slow down updates. To keep performance stable, the design stores download history in a separate collection with references to patient and prescription IDs. This makes writing and querying more efficient as data grows.

3. Search Queries for Specific Prescriptions: Referencing prescriptions in their own collection and indexing key fields like doctor_id improves search speed. For example, the system can quickly find all prescriptions by a specific doctor without scanning through entire patient documents. This design could support efficient, flexible queries that would be slower with embedded data.

Five Rules of Thumb

1. I embedded prescription lines inside prescriptions since they are always accessed together. Prescriptions are also embedded within the patient document because they are directly tied to the patient and commonly retrieved at login.

2. Doctor data is referenced using doctor_id because it's not needed unless the user views or downloads a specific prescription. Similarly, downloads are stored in a separate collection and referenced, as each download record may need to be accessed independently.

3. Download history can grow significantly over time, so embedding it would lead to oversized prescription documents. Instead, it is stored in its own collection and referenced, which keeps documents smaller and improves performance.

4. Prescriptions are mostly read and rarely updated, so embedding them improves read performance. On the other hand, download history involves frequent writes, so referencing it helps reduce write overhead and avoids bloated parent documents.

5. The structure aligns with how the system queries data. The prescriptions are read with patient data, while doctors and downloads are queried separately. This ensures both fast access and flexibility in searches and updates.

1.3 Individual - Student 2

Student 2: Matsas, Mario, 12446870

1.3.1 Use Case Definition and Design

Version 1

Textual Description

Write Prescription (weak entity)
Trigger: A doctor initiates a prescription after examining a patient.

Preconditions:

1. The doctor is logged in.

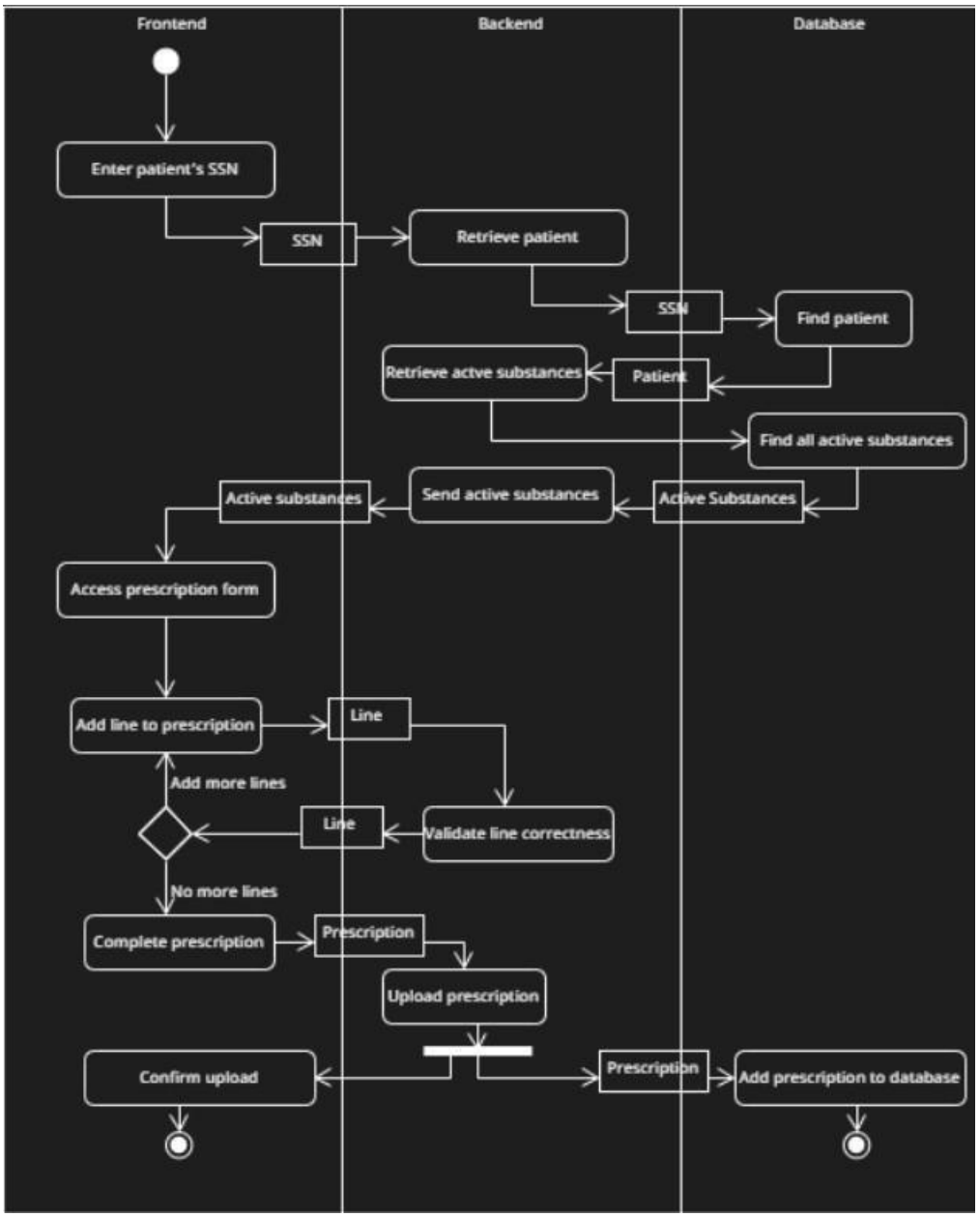
Main Flow:

1. The doctor enters the patient's SSN.
2. The system loads the patient with that SSN.
3. The system loads all the available active substances from the database.
4. The system presents the doctor with a prescription form he can fill.
5. Loop:
    5.1. The doctor adds an active substance, the dosage and the details to the prescription line.
    5.2. The doctor adds the line to the prescription.
    5.3. The system confirms the validity of the line.
6. The doctor completes the prescription and clicks "Upload".
7. The system confirms the upload.
8. The system saves the prescription to the database.

Postconditions: The prescription is uploaded to the site, saved in the database and is visible to the patient that it addresses.

Entities: Doctor, Patient, Prescription, PrescriptionLine, ActiveSubstance

Graphical Representation



1.3.2 Analytics Report

Concept

My analytics report will analyze the number of prescriptions written by a specific doctor as well as the total prescription lines that were written by him. The filter field will be the doctor's SSN and the entities involved are: Doctor, Prescription and PrescriptionLine. I use the tables we have already defined previously in part 1.1.3 (I create all the tables, but I only use the ones that are relevant to my use case) and add the dummy data that can be found in Student2_InsertData_Initial.sql. Afterwards I run this query

```
SELECT
    D.SSN AS Doctor_SSN,
    COUNT(DISTINCT P.ID) AS Total_Prescriptions,
    COUNT(PL.line_number) AS Total_PrescriptionLines
FROM Doctor D
JOIN Prescription P ON D.SSN = P.doctor_id JOIN PrescriptionLine PL ON P.ID = PL.ID
WHERE D.SSN = '1111'
GROUP BY D.SSN;
```

that can be found in Student2_Querystatement.sql, so I can view the initial results and be able to make comparisons later. Afterwards I add some extra data to simulate one execution of my use case (the data can be found in Student2_InsertData_UseCase.sql) and run the query again so I can make sure that the results have now changed and that I get the expected results.

Proof of Concept

After creating the tables and inserting the first batch of data, which can be found in Student2_InsertData_Initial.sql, I ran the query inside of Student2_Querystatement.sql and got these results:

```
+------------+---------------------+-------------------------+
| Doctor_SSN | Total_Prescriptions | Total_PrescriptionLines |
+------------+---------------------+-------------------------+
| 1111       |                   1 |                       1 |
+------------+---------------------+-------------------------+
1 row in set (0.01 sec)
```

Then I inserted new data (simulating what would happen if we performed the "Write Prescription" usecase) and got these results:

```
+------------+---------------------+-------------------------+
| Doctor_SSN | Total_Prescriptions | Total_PrescriptionLines |
+------------+---------------------+-------------------------+
| 1111       |                   2 |                       3 |
+------------+---------------------+-------------------------+
1 row in set (0.00 sec)
```

Which matches the expected result.

1.3.3 NoSQL Design

The involved entities are: Doctor, Patient, ActiveSubstance, Prescription and PrescriptionLine

Design Overview

My NoSQL design (not for the entire project, but for my use case) is the following:

Doctor.json:


```
[
    {
        "_id": "1111",
        "username": "doc1",
        "password": "pass1",
        "name": "A",
        "surname": "M",
        "profession": "Cardiologist",
        "birth_date": "1980-01-01",
        "join_date":  "2020-01-01"
    },
    {
        "_id": "2222",
        "username": "doc2",
        "password": "pass2",
        "name": "X",
        "surname": "M",
        "profession": "Dermatologist",
        "birth_date": "1985-02-02",
        "join_date":  "2021-02-02"
    },
    {
        "_id": "3333",
        "username": "doc3",
        "password": "pass3",
        "name": "E",
```

```
    "surname": "M",
    "profession": "Neurologist",
    "birth_date": "1990-03-03",
    "join_date": "2022-03-03"
  }
]
```


Patient.json


```
[
  {
    "_id": "4444",
    "username": "patient4",
    "password": "pass4",
    "name": "M",
    "surname": "M",
    "birth_date": "2000-05-05",
    "join_date": "2023-05-05"
  },
  {
    "_id": "5555",
    "username": "patient5",
    "password": "pass5",
    "name": "P",
    "surname": "X",
    "birth_date": "2000-05-05",
    "join_date": "2023-05-05"
  },
  {
    "_id": "6666",
    "username": "patient6",
    "password": "pass6",
    "name": "D",
```

```
    "surname": "X",
    "birth_date": "2000-05-05",
    "join_date": "2023-05-05"
  }
]
```


ActiveSubstance.json:


```
[
  {
    "_id": "Paracetamol",
    "max_dosage_per_month": 3000,
    "therapeutic_class": "Painkiller"
  },
  {
    "_id": "Ibuprofen",
    "max_dosage_per_month": 2400,
    "therapeutic_class": "Anti-inflammatory"
  },
  {
    "_id": "Amoxicillin",
    "max_dosage_per_month": 1000,
    "therapeutic_class": "Antibiotic"
  }
]
```


Prescription.json:


```
[
  {
    "_id": "1",
    "date": "2025-03-30",
    "status": "Active",
    "doctor_id": "1111",
    "patient_id": "4444",
```

```
  "lines": [
    {
      "line_number": 1,
      "active_substance_name": "Paracetamol",
      "dosage": "500mg",
      "details": "Take twice daily"
    }
  ]
},
{
  "_id": "2",
  "date": "2025-03-30",
  "status": "Active",
  "doctor_id": "2222",
  "patient_id": "5555",
  "lines": [
    {
      "line_number": 1,
      "active_substance_name": "Ibuprofen",
      "dosage": "400mg",
      "details": "Take three times daily"
    }
  ]
},
{
  "_id": "3",
  "date": "2025-03-30",
  "status": "Active",
  "doctor_id": "3333",
  "patient_id": "6666",
  "lines": [
```

```
      {
        "line_number": 1,
        "active_substance_name": "Amoxicillin",
        "dosage": "250mg",
        "details": "Take once daily"
      }
    ]
  }
]
```


Expected Execution and Possible Changes

Following the activity diagram of my use case, we can see that there will be 1 read for the patient, 1 read for the active substances and finally one write of the prescription to the database. My NoSQL design works well for this specific use case, because due to the way they are being read, patients and active substances need to be separate collections and not embedded anywhere, while the most important advantage is the embedding of the prescription lines into the prescription, something that makes total sense considering that prescription lines will never need to be read outside the prescription they are part of.

If there were to be more read and write operations (for example if the program was used in a hospital setting where there are many patients) the changes would be as follows:

Read: For the read there wouldn’t be much for us to change or improve since we already use indexing efficiently (we use the SSN as the index of the patient and the name for the active substance) but using a cache for the active substances (maybe even the patients if the cache is large enough) would surely help.

Write: In contrast to the read, if there were too many prescriptions written with many lines each there would be a very real chance that we might face memory issues. To combat this, we could try referencing instead of embedding the prescription lines and saving them into a different JSON document. Another problem however would be not just the lines but the prescriptions themselves. If we received too many prescription writes, using sharding would probably provide a very good solution.

Five Rules of Thumb

1. My NoSQL adheres to this rule, because we take advantage of the fact that prescription lines don't appear outside of prescriptions so that we can embed them into the prescriptions.

2. We also follow this rule, because doctors, patients and active substances are all objects that we want to access on their own (in the use case we want to get the patient and the active substances on their own, as for the doctor, my query on 1.3.2 shows that we want to access this object on its own) and thus we don't embed them anywhere.

3. While in my design I assume that we will not have too many lines in our prescriptions, I pointed out in the "Expected Execution and Possible Changes" part, that if that was to be the case (aka we had high-cardinality arrays) it would be a good idea to use referencing rather than embedding.

4. In my design it makes sense not to denormalize the doctors, patients and active substances, because while they are read far more often than written, we usually only care about their indices (the SSNs of the doctors and patients or the name of the active substances), so there is no need to embed or duplicate data. The prescription is written once and then can only be read but yet again denormalizing doesn't make sense because they are the main entities and contain info about all others. Finally, the prescription lines are written once and read often but unlike with the prescriptions, here it makes total sense to embed the prescription lines to the prescriptions, because when we read a prescription we obviously want to read the lines and get all the info, so now we can do this much faster and efficiently.

5. Yet again this is also something we take into account, since our data is structured in such a way that our use case can be executed as effectively as possible, using object referencing and embedding whenever needed (it is important to note that while the design is based on the use case, this design can also be used throughout the entire program, because it is overall very effective).