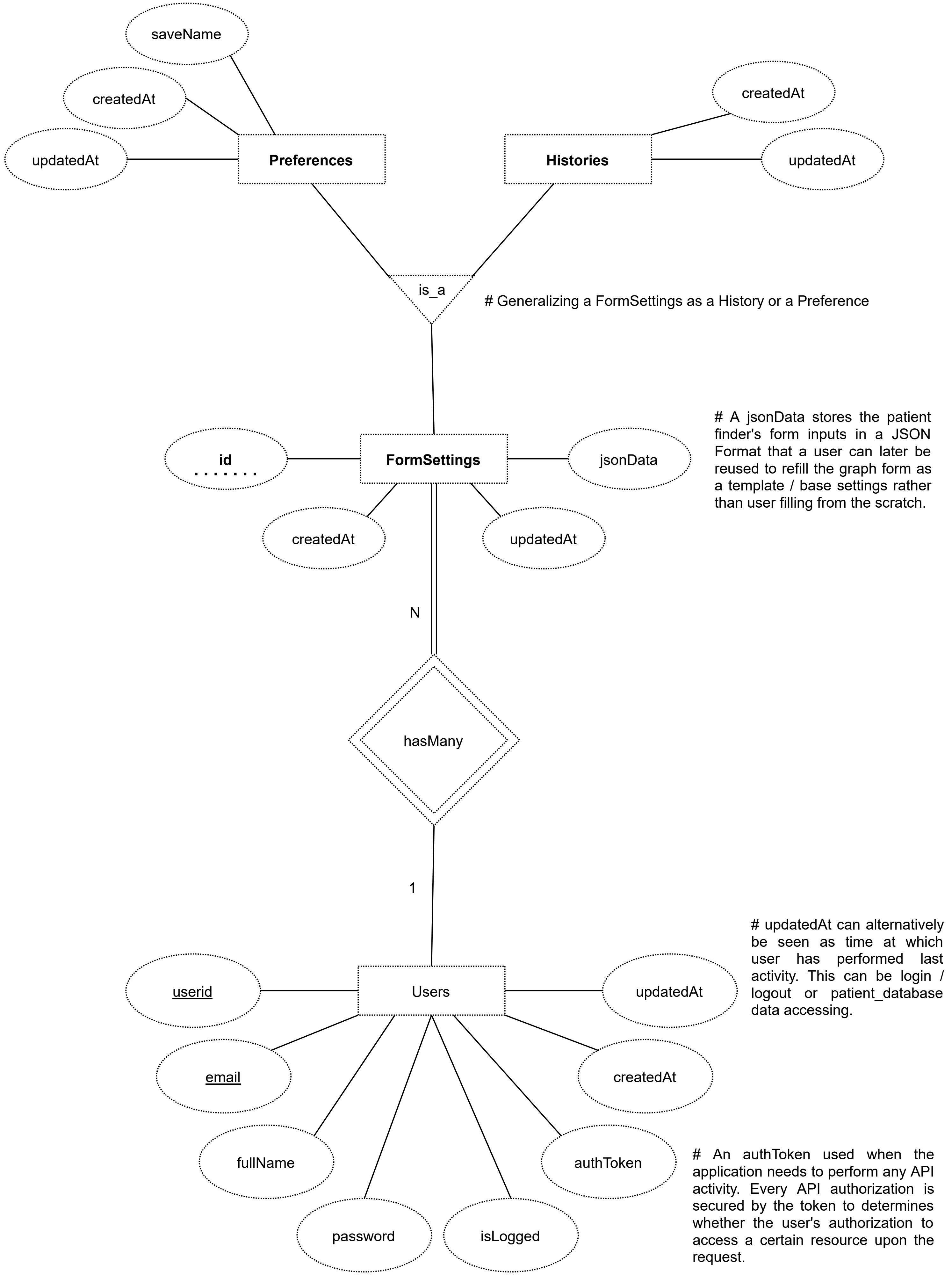
Short Description about the ER- diagram database for User preferences:
This product is targeted to build a database, whose design includes following properties:

1. Each user of Patient Finder application can save their Patient Finder form definition saved within the FormSettings entity.
2. Hence, A single user can have many Form Settings, that is stored in a way that is uniquely identifiable. The configuration related to inputs are stored into the database as a raw JSON format named jsonData.
3. A form settings can be generalized into two categories:
    1. Preferences: A FormSettings configurations that are manually saved by a user, so that users can default certain settings like a template rather than filling the long form from scratch. This is a means for users to efficiently work with the Patient Finder application, gaining a bonus interface experience.
    2. History: A FormSetting configuration that are stored automatically when a user clicks for graph generation (or data visualization) so that the potential product owner or respective users of the application can track what they have accessed across their product usage.
4. Note: Difference between a user's Preference and History lies at the part where Preference comes as a feature for assistance in working with the system and History as a feature for tracking their Patient Finder data accesses.

Also, definition of user must include a userid, email, their Full name and a secure password for securely accessing their account. Each time a user gets logged into their account they are generated with a unique accessToken. This user accessToken can be sent to as a authorization mechanism for every action they perform on their account and data accesses. If the access token is mismatch from what's in the database, then the Backend API must not carry out that request as they are marked as *Unauthorized Action* by default.

When a user logs in successfully, they are immediately marked as Logged in (i.e., isLogged becomes true). The time for any last changes made to their account are also saved (as updatedAt). This feature is also consistent for all other entities part of our database.



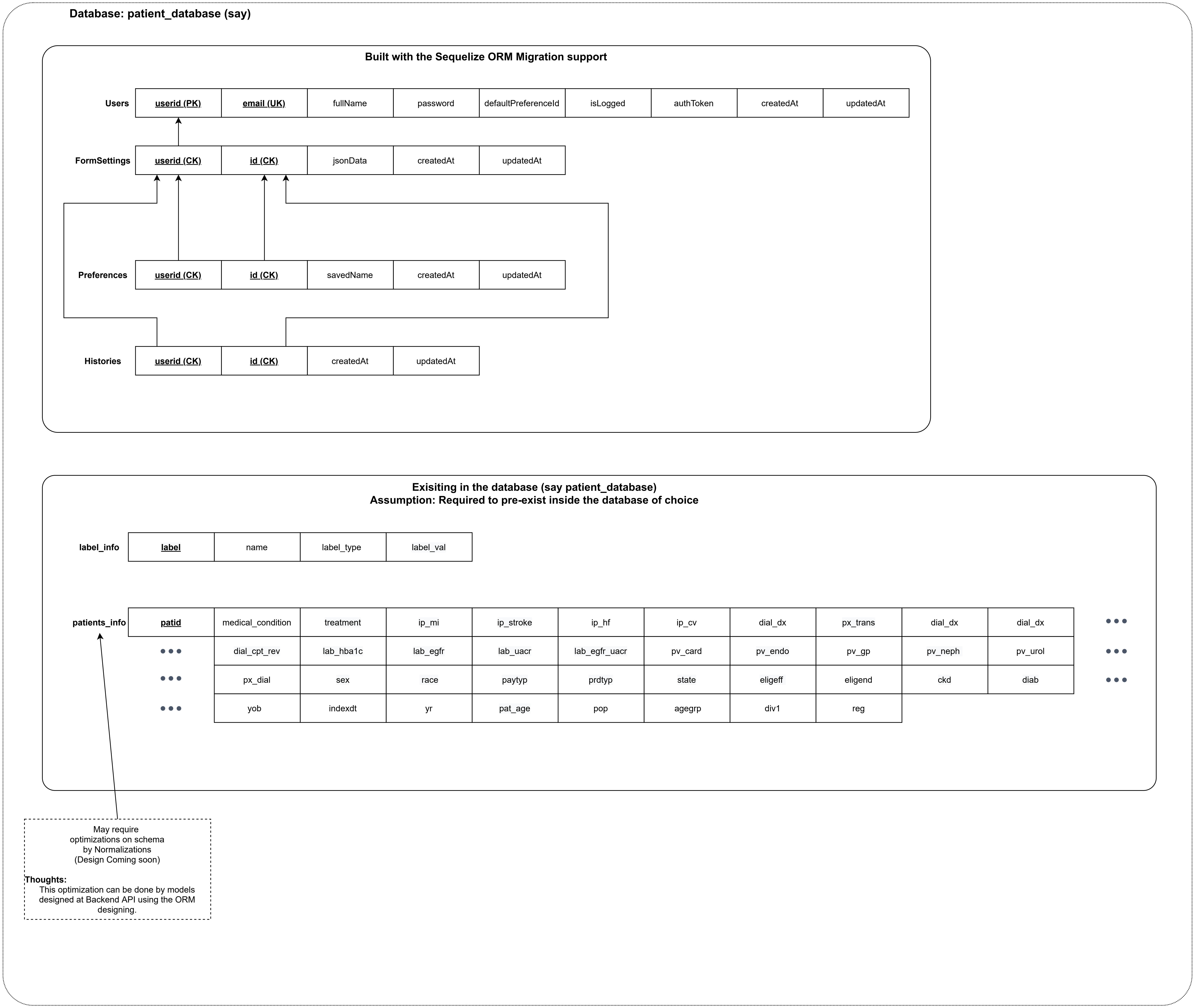
# Generalizing a FormSettings as a History or a Preference

# A jsonData stores the patient finder's form inputs in a JSON Format that a user can later be reused to refill the graph form as a template / base settings rather than user filling from the scratch.

# updatedAt can alternatively be seen as time at which user has performed last activity. This can be login / logout or patient_database data accessing.

# An authToken used when the application needs to perform any API activity. Every API authorization is secured by the token to determines whether the user's authorization to access a certain resource upon the request.

Database Design for a Patient Finder:



Database: patient_database (say)

Built with the Sequelize ORM Migration support

Users	**userid (PK)**	**email (UK)**	fullName	password	defaultPreferenceId	isLogged	authToken	createdAt	updatedAt

FormSettings	**userid (CK)**	**id (CK)**	jsonData	createdAt	updatedAt

Preferences	**userid (CK)**	**id (CK)**	savedName	createdAt	updatedAt

Histories	**userid (CK)**	**id (CK)**	createdAt	updatedAt

Exisiting in the database (say patient_database)
Assumption: Required to pre-exist inside the database of choice

label_info	**label**	name	label_type	label_val

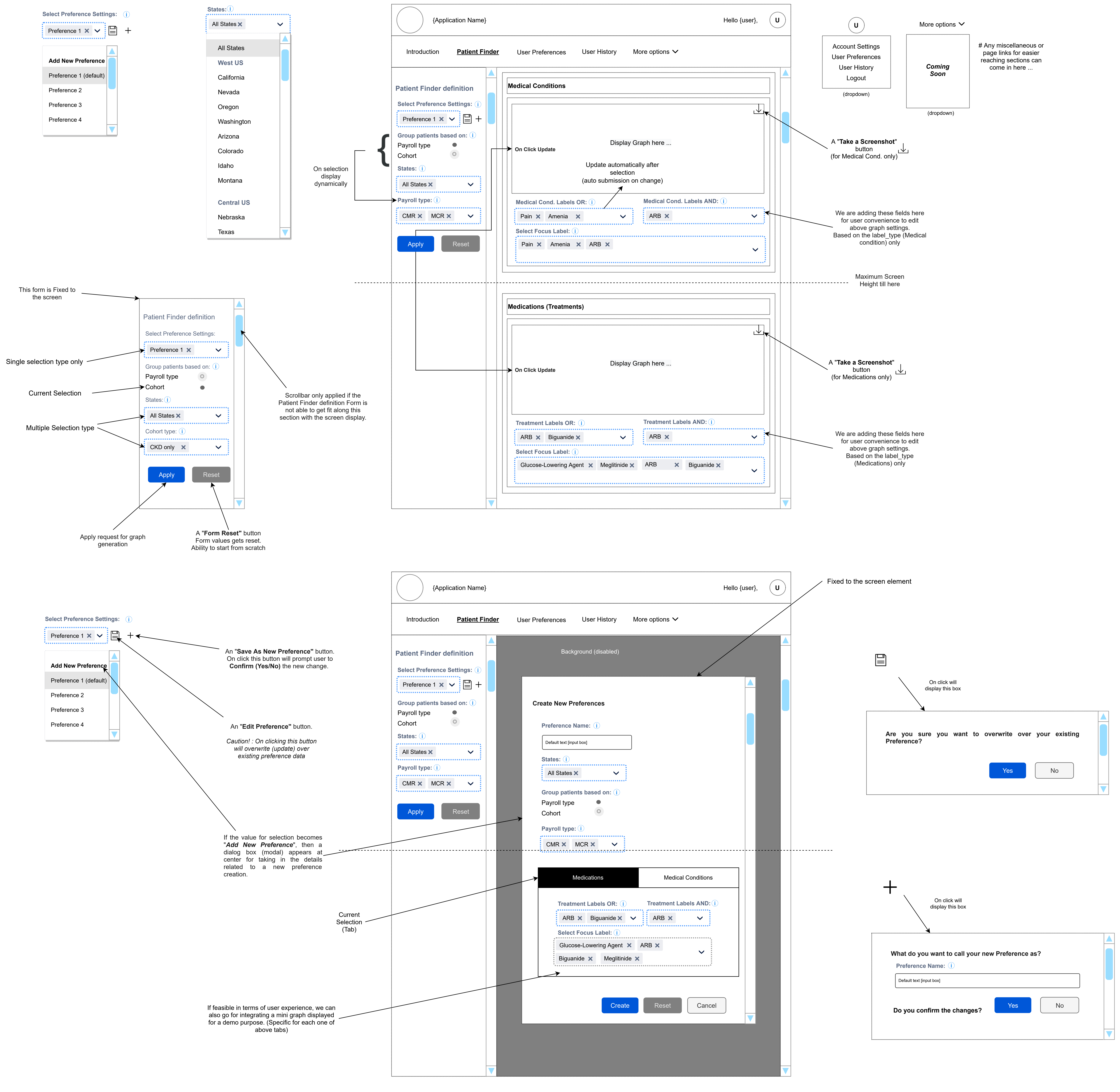
patients_info	**patid**	medical_condition	treatment	ip_mi	ip_stroke	ip_hf	ip_cv	dial_dx	px_trans	dial_dx	dial_dx	•••
	•••	dial_cpt_rev	lab_hba1c	lab_egfr	lab_uacr	lab_egfr_uacr	pv_card	pv_endo	pv_gp	pv_neph	pv_urol	•••
	•••	px_dial	sex	race	paytyp	prdtyp	state	eligeff	eligend	ckd	diab	•••
	•••	yob	indexdt	yr	pat_age	pop	agegrp	div1	reg			

May require
optimizations on schema
by Normalizations
(Design Coming soon)

Thoughts:
This optimization can be done by models designed at Backend API using the ORM designing.

Bayer's Patient Finder wireframe design

Please note that the scrollbars on fixed to the screen elements are only present to improve screen responsiveness in this design.



On selection display dynamically

Update automatically after selection (auto submission on change)

A "**Take a Screenshot**" button (for Medical Cond. only)

We are adding these fields here for user convenience to edit above graph settings. Based on the label_type (Medical condition) only

Maximum Screen Height till here

A "**Take a Screenshot**" button (for Medications only)

We are adding these fields here for user convenience to edit above graph settings. Based on the label_type (Medications) only

# Any miscellaneous or page links for easier reaching sections can come in here ...

This form is Fixed to the screen

Single selection type only

Current Selection

Multiple Selection type

Scrollbar only applied if the Patient Finder definition Form is not able to get fit along this section with the screen display.

Apply request for graph generation

A "**Form Reset"** button Form values gets reset. Ability to start from scratch

An "**Save As New Preference"** button. On click this button will prompt user to **Confirm (Yes/No)** the new change.

An "**Edit Preference"** button.

Caution! : On clicking this button will overwrite (update) over existing preference data

If the value for selection becomes "***Add New Preference***", then a dialog box (modal) appears at center for taking in the details related to a new preference creation.

Current Selection (Tab)

If feasible in terms of user experience, we can also go for integrating a mini graph displayed for a demo purpose. (Specific for each one of above tabs)

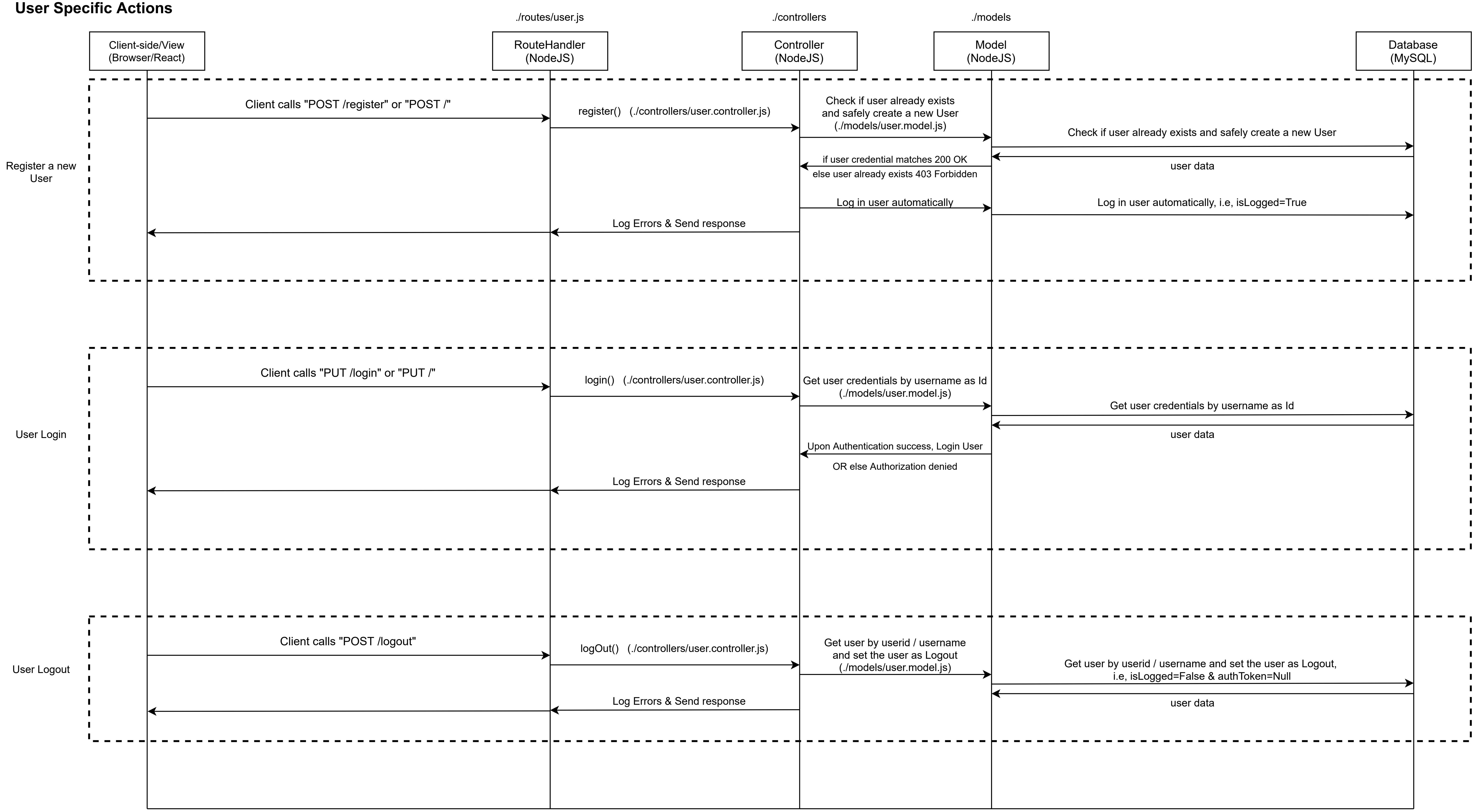
Fixed to the screen element

On click will display this box

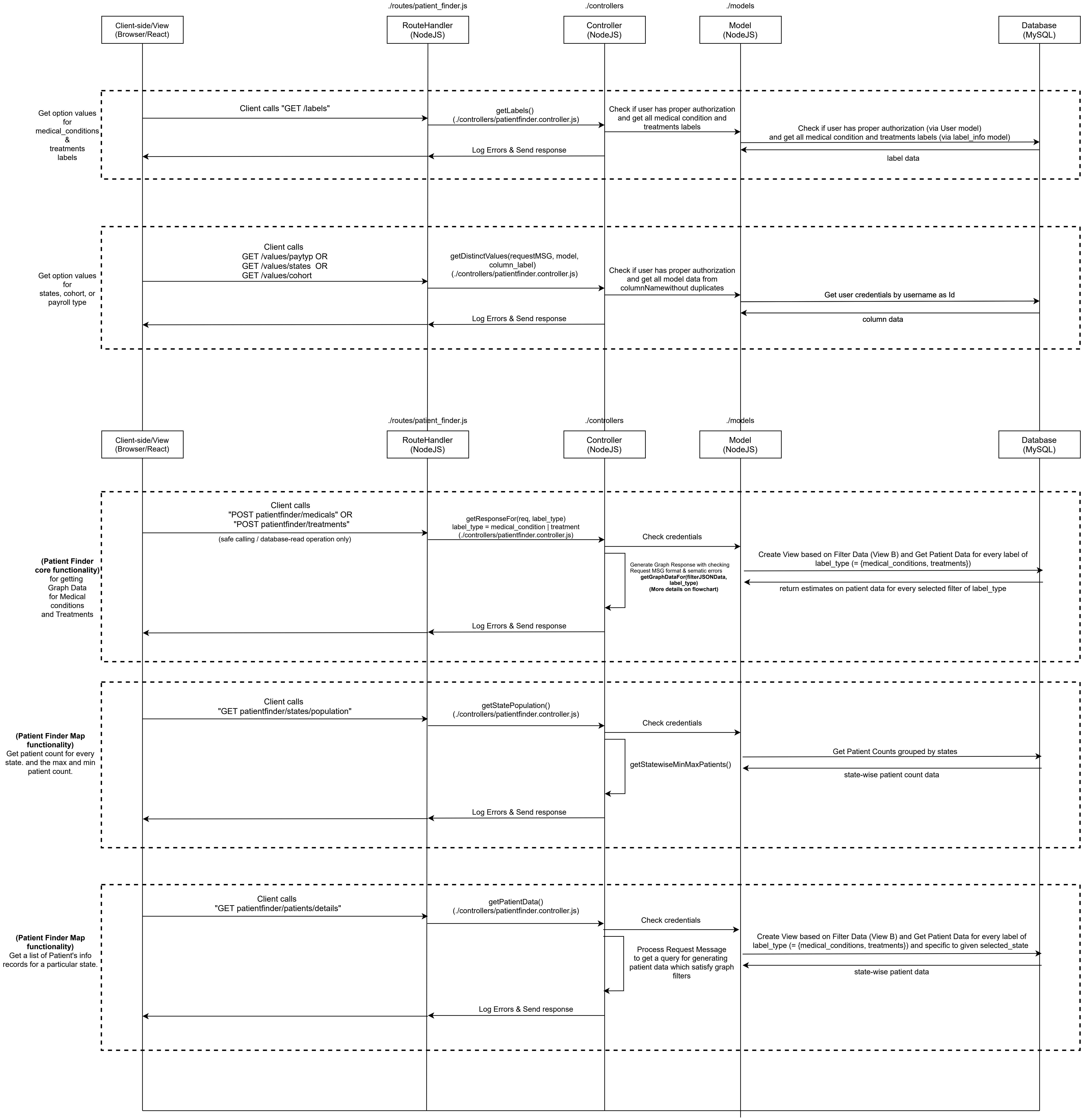
On click will display this box

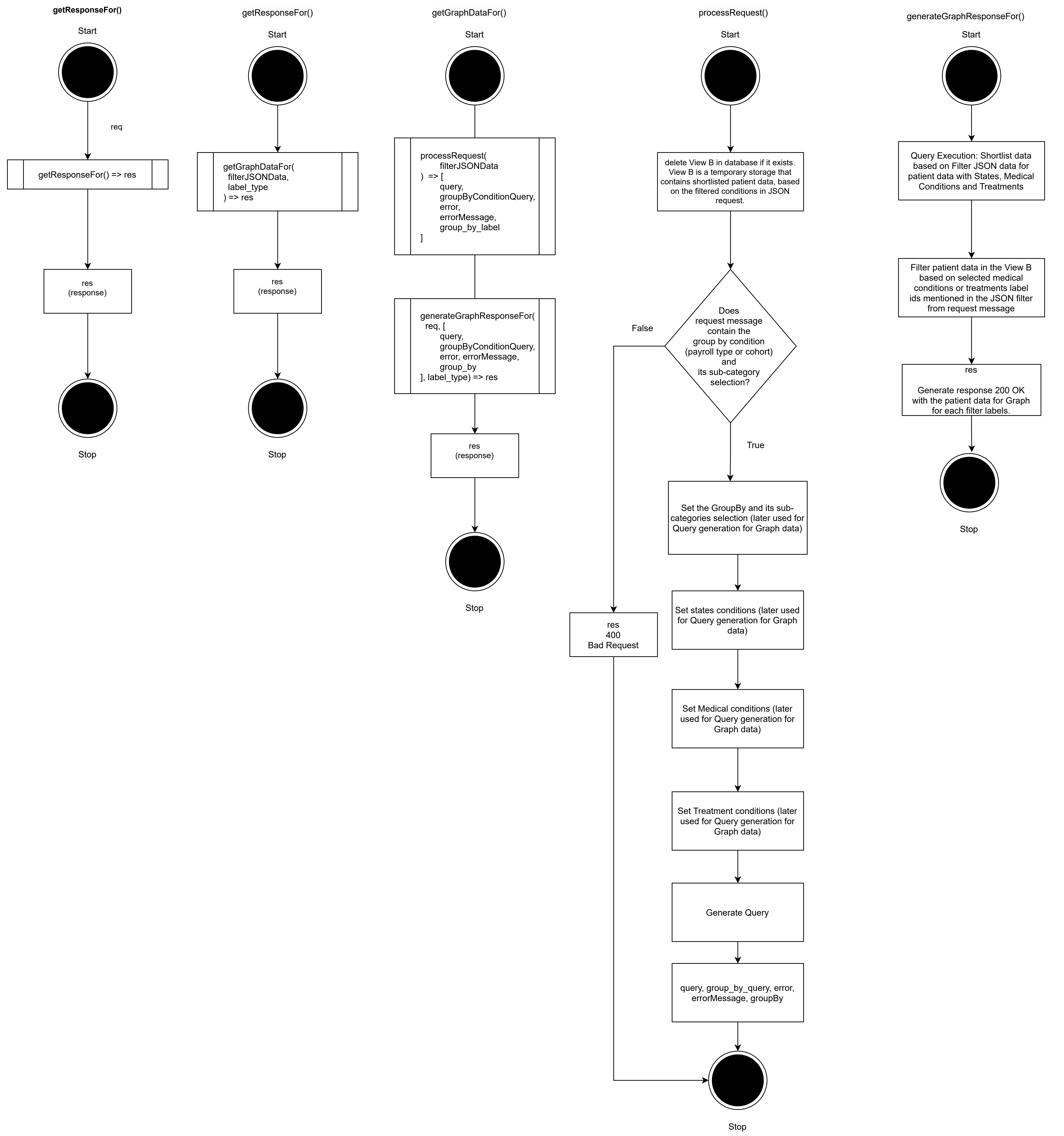
Backend Architecture & Working

User Specific Actions


./routes/user.js
./controllers
./models
Client-side/View (Browser/React)
RouteHandler (NodeJS)
Controller (NodeJS)
Model (NodeJS)
Database (MySQL)
Register a new User
Client calls "POST /register" or "POST /"
register() (./controllers/user.controller.js)
Check if user already exists and safely create a new User (./models/user.model.js)
Check if user already exists and safely create a new User
user data
if user credential matches 200 OK
else user already exists 403 Forbidden
Log in user automatically
Log in user automatically, i.e, isLogged=True
Log Errors & Send response
User Login
Client calls "PUT /login" or "PUT /"
login() (./controllers/user.controller.js)
Get user credentials by username as Id (./models/user.model.js)
Get user credentials by username as Id
user data
Upon Authentication success, Login User
OR else Authorization denied
Log Errors & Send response
User Logout
Client calls "POST /logout"
logOut() (./controllers/user.controller.js)
Get user by userid / username and set the user as Logout (./models/user.model.js)
Get user by userid / username and set the user as Logout, i.e, isLogged=False & authToken=Null
user data
Log Errors & Send response

Patient Finder Actions


./routes/patient_finder.js
./controllers
./models
Client-side/View (Browser/React)
RouteHandler (NodeJS)
Controller (NodeJS)
Model (NodeJS)
Database (MySQL)
Get option values for medical_conditions & treatments labels
Client calls "GET /labels"
getLabels() (./controllers/patientfinder.controller.js)
Check if user has proper authorization and get all medical condition and treatments labels
Check if user has proper authorization (via User model) and get all medical condition and treatments labels (via label_info model)
label data
Log Errors & Send response
Get option values for states, cohort, or payroll type
Client calls GET /values/paytyp OR GET /values/states OR GET /values/cohort
getDistinctValues(requestMSG, model, column_label) (./controllers/patientfinder.controller.js)
Check if user has proper authorization and get all model data from columnNamewithout duplicates
Get user credentials by username as Id
column data
Log Errors & Send response
./routes/patient_finder.js
./controllers
./models
Client-side/View (Browser/React)
RouteHandler (NodeJS)
Controller (NodeJS)
Model (NodeJS)
Database (MySQL)
(Patient Finder core functionality) for getting Graph Data for Medical conditions and Treatments
Client calls "POST patientfinder/medicals" OR "POST patientfinder/treatments"
(safe calling / database-read operation only)
getResponseFor(req, label_type) label_type = medical_condition | treatment (./controllers/patientfinder.controller.js)
Check credentials
Generate Graph Response with checking Request MSG format & sematic errors getGraphDataFor(filterJSONData, label_type) (More details on flowchart)
Create View based on Filter Data (View B) and Get Patient Data for every label of label_type (= {medical_conditions, treatments})
return estimates on patient data for every selected filter of label_type
Log Errors & Send response
(Patient Finder Map functionality) Get patient count for every state. and the max and min patient count.
Client calls "GET patientfinder/states/population"
getStatePopulation() (./controllers/patientfinder.controller.js)
Check credentials
getStatewiseMinMaxPatients()
Get Patient Counts grouped by states
state-wise patient count data
Log Errors & Send response
(Patient Finder Map functionality) Get a list of Patient's info records for a particular state.
Client calls "GET patientfinder/patients/details"
getPatientData() (./controllers/patientfinder.controller.js)
Check credentials
Process Request Message to get a query for generating patient data which satisfy graph filters
Create View based on Filter Data (View B) and Get Patient Data for every label of label_type (= {medical_conditions, treatments}) and specific to given selected_state
state-wise patient data
Log Errors & Send response


getResponseFor()
Start
req
getResponseFor() => res
res
(response)
Stop
getResponseFor()
Start
getGraphDataFor(
filterJSONData,
label_type
) => res
res
(response)
Stop
getGraphDataFor()
Start
processRequest(
filterJSONData
) => [
query,
groupByConditionQuery,
error,
errorMessage,
group_by_label
]
generateGraphResponseFor(
req, [
query,
groupByConditionQuery,
error, errorMessage,
group_by
], label_type) => res
res
(response)
Stop
processRequest()
Start
delete View B in database if it exists. View B is a temporary storage that contains shortlisted patient data, based on the filtered conditions in JSON request.
Does request message contain the group by condition (payroll type or cohort) and its sub-category selection?
False
True
res
400
Bad Request
Set the GroupBy and its sub-categories selection (later used for Query generation for Graph data)
Set states conditions (later used for Query generation for Graph data)
Set Medical conditions (later used for Query generation for Graph data)
Set Treatment conditions (later used for Query generation for Graph data)
Generate Query
query, group_by_query, error, errorMessage, groupBy
Stop
generateGraphResponseFor()
Start
Query Execution: Shortlist data based on Filter JSON data for patient data with States, Medical Conditions and Treatments
Filter patient data in the View B based on selected medical conditions or treatments label ids mentioned in the JSON filter from request message
res
Generate response 200 OK with the patient data for Graph for each filter labels.
Stop

For References,

Bayer Patient Finder Backend API Documentation:
https://github.com/RPG-coder/bayer-njit-backend/blob/master/documentation/API%20Documentation%20for%20Bayer%20Patient%20Finder.pdf

Bayer Patient Finder Repository for Backend:
https://github.com/RPG-coder/bayer-njit-backend

Bayer Patient Finder Repository for Frontend:
https://github.com/sp2728/bayer-njit-frontend

More Updates are coming soon.