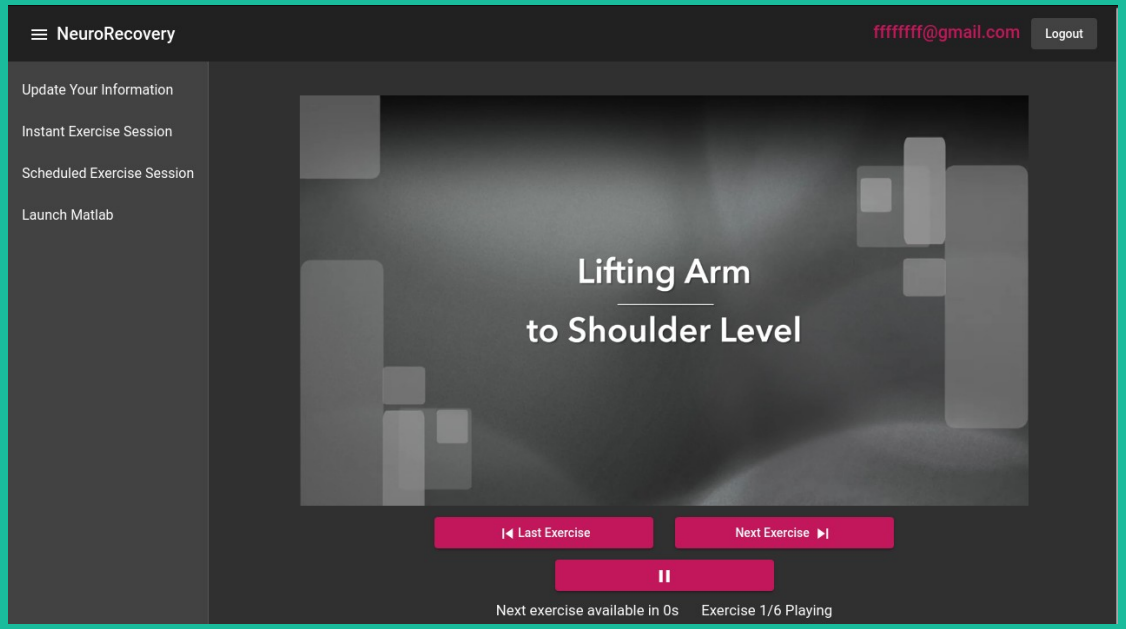


NeuroRecovery App

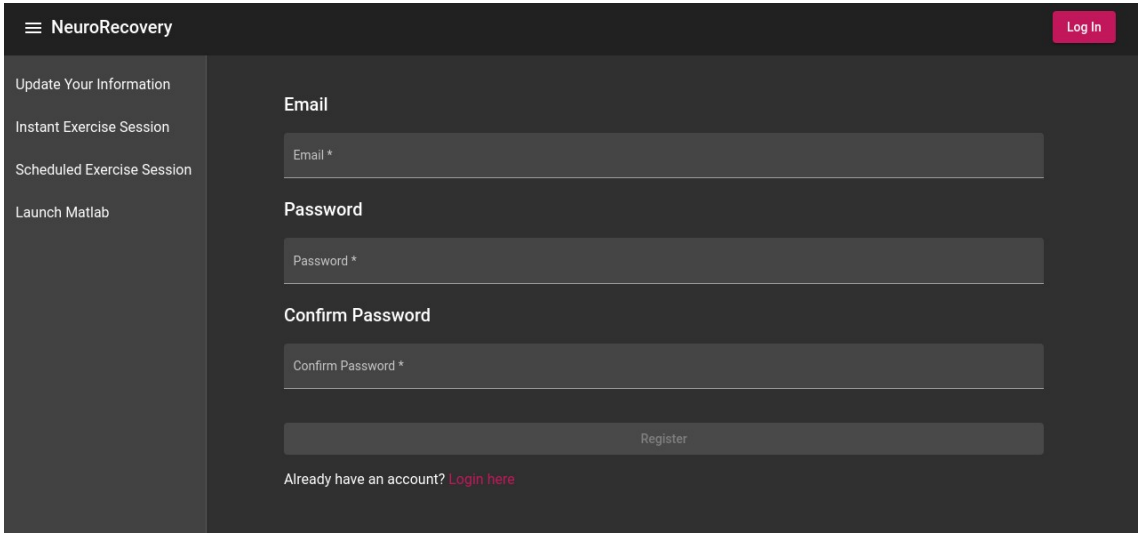
The NeuroRecovery App assists post-stroke patients with their recovery through a web application

Core Features

- Login/Register system with password hashing
- Instant exercise sessions
- Scheduled exercise sessions
- Patient and therapist user data forms
- User exercise statistics
- Exercise recommendations based on user data
- Therapist-Patient link
- Matlab Psychtoolbox simulation connection

Login/Register System

- Email/Password based authentication
- Passwords are stored as Argon2id hashes for security
- Secure Session ID is stored in browsers localStorage

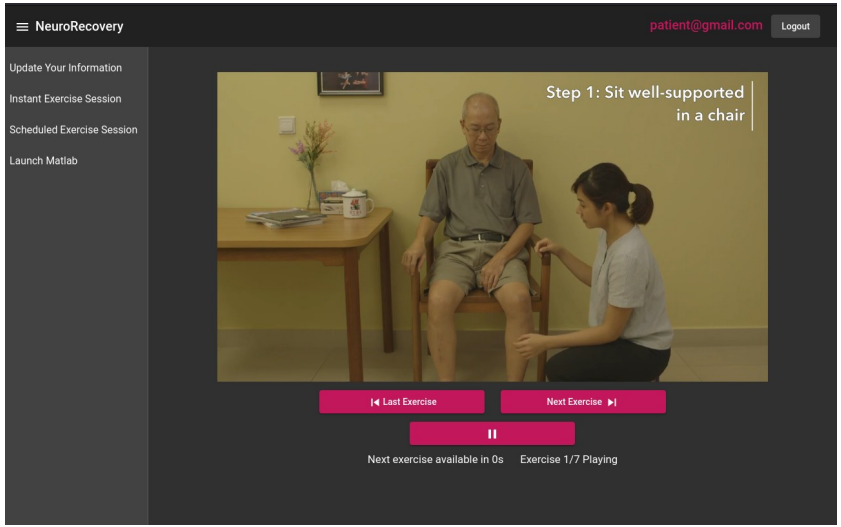



```rust
pub async fn login_user(
    Extension(state): Extension<Arc<State>>,
    Json(payload): Json<UserRequest>,
) -> Result<Json<LoginResponse>, StatusCode> {
    let user = match data::find_user_by_email(&state.db, &payload.email).await {
        Ok(user) => match user {
            Some(user) => user,
            None => {
                info!("User not found  {}", &payload.email);
                return Err(StatusCode::BAD_REQUEST);
            }
        },
        Err(e) => {
            error!("Failure finding user: {}", e);
            return Err(StatusCode::INTERNAL_SERVER_ERROR);
        }
    };

    match check_password_matches_hash(&payload.password, &user.hash, &user.salt) {
        Ok(check) => {
            if !check {
                info!("Password hash for user {} does not match", &payload.email);
                return Err(StatusCode::BAD_REQUEST);
            }
        }
        Err(e) => {
            error!("Failure checking hashed password: {}", e);
            return Err(StatusCode::INTERNAL_SERVER_ERROR);
        }
    };

    let session_id = gen_session_id();
    match data::update_user_session_id(&state.db, &payload.email, &session_id).await {
        Ok(_) => (),
        Err(e) => {
            error!("Failed to update user {} session ID: {}", &payload.email, e);
            return Err(StatusCode::INTERNAL_SERVER_ERROR);
        }
    }
    Ok(Json(LoginResponse { session_id }))
}
```

Instant Exercise Sessions

- Looped video clips of the exercise being demonstrated
- Countdown timer on each exercise before “Next Exercise” can be clicked
- Easy to add new videos with an mp4 file and an array of TimeSets

```
ngOnInit(): void {
  this.exercisesService.fill_external_values("Instant Lower Limb");
  this.exercisesService.exerciseTimes = [
    { StartTime: 46, EndTime: 80 },
    { StartTime: 88, EndTime: 111 },
    { StartTime: 116, EndTime: 157 },
    { StartTime: 159, EndTime: 178 },
    { StartTime: 187, EndTime: 231 },
    { StartTime: 240, EndTime: 284 },
    { StartTime: 330, EndTime: 357 },
  ];
}
```

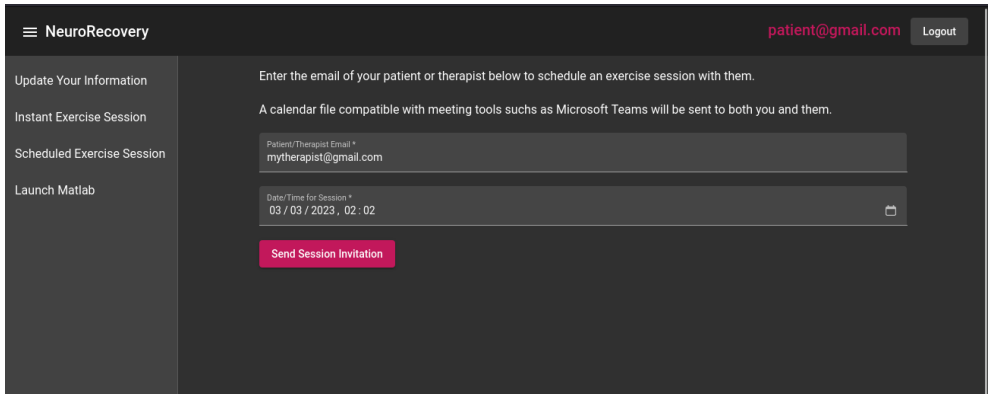


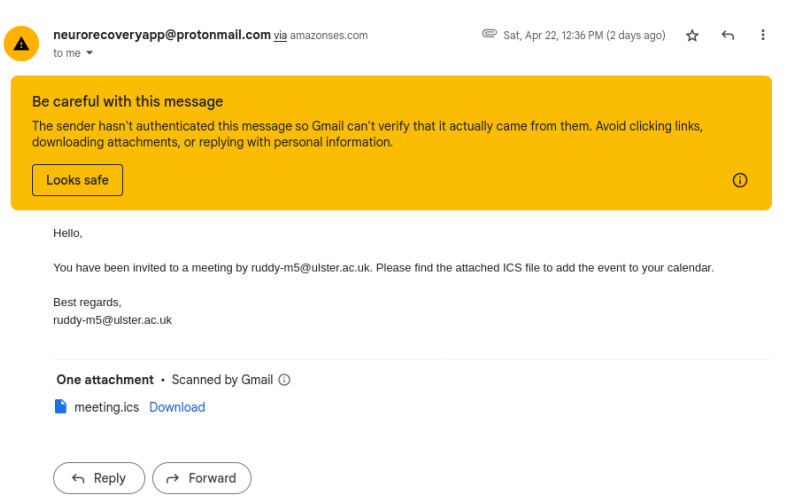

```
export class ExercisesService {

  constructor(private snackBar: MatSnackBar, private router: Router, private backendService: BackendService, private
  loginService: LoginService) { }

  preload: string = 'auto';
  api = new VgApiService;
  started = false;
  startEpoch = 0;

  timePerExercise = 3;
  timerCurrent = 0;
  timerFinished = true;
  interval: any = null;

  exerciseIndex = 0;
  exerciseTimes = new Array<TimeSet>;
  onLastExercise = false;
  highestCompletedExerciseIndex = 0;

  // external values
  kind = "";
  // exercise session values
  datetime = "";
  totalTimeTakenSecs = 0;
  numExercisesCompleted = 0;
```

Scheduled Exercise Sessions

- Form to enter email of meeting attendee and datetime
- Generates ICS calendar file and emails it to both parties

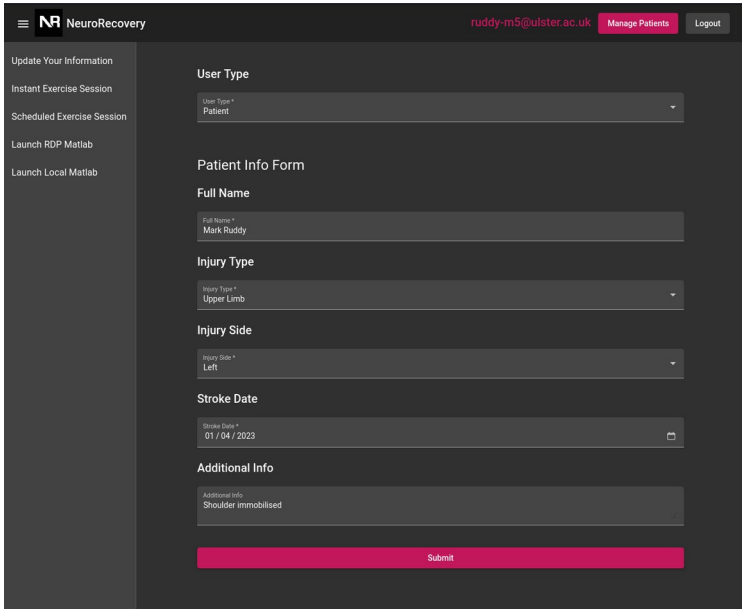
```
let { error, value } = ics.createEvent({
  start: [year, month, day, hour, minute],
  duration: { minutes: this.contactForm.value.estimatedTime! },
  title: 'NeuroRecovery Meeting',
  description: 'NeuroRecovery Meeting',
  location: 'Online Call',
})
if (error) {
  console.log(error);
}
let blob = new Blob([value!], { type: 'text/calendar;charset=utf-8' });
saveAs(blob, 'neurorecovery_meeting.ics');
```





Patient and Therapist User Data Form

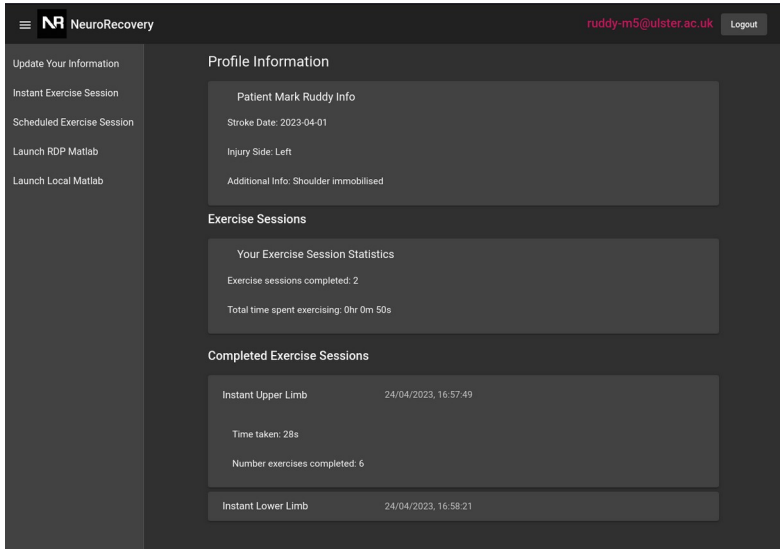
- Form to submit data on Patient/Therapist information




```rust
pub async fn post_patient_form(
    Extension(state): Extension<Arc<State>>,
    Json(payload): Json<data::PatientForm>,
) -> Result<(), StatusCode> {
    match utils::check_authenticated_request(&state.db, &payload.session_id, &payload.email).await {
        Ok(_) => (),
        Err(e) => return Err(e),
    };

    // Delete any existing user info as it will be overwritten
    match data::delete_user_info_if_existing(&state.db, &payload.email).await {
        Ok(()) => (),
        Err(e) => {
            error!("Failure deleting existing user data: {}", e);
            return Err(StatusCode::INTERNAL_SERVER_ERROR);
        }
    }

    match data::insert_patient_form(&state.db, payload).await {
        Ok(()) => (),
        Err(e) => {
            error!("Failure inserting patient form: {}", e);
            return Err(StatusCode::INTERNAL_SERVER_ERROR);
        }
    }
    Ok(())
}
```

User Statistics

- Submitted form information is displayed on user stats page
- Exercise session statistics, including total time spent exercising and sessions completed



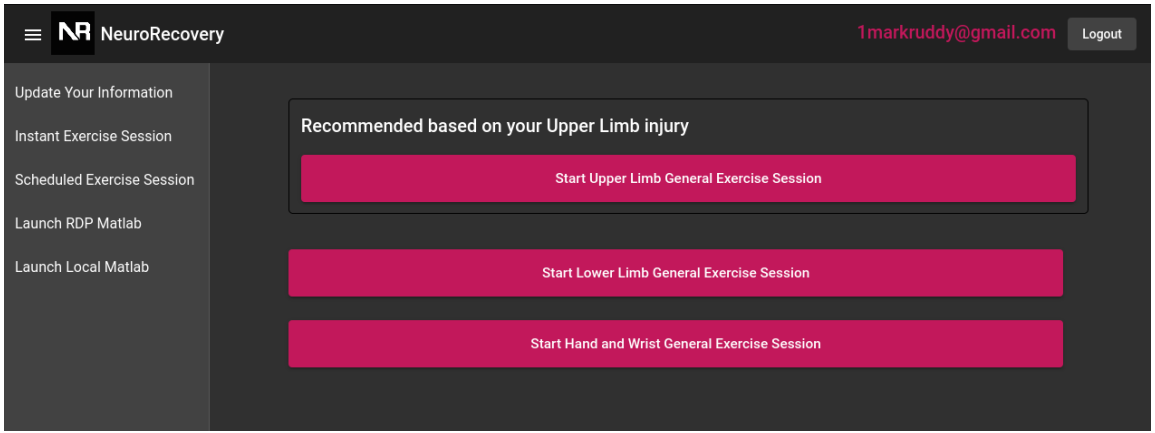

```
if (this.userType == "Patient") {
  this.patientForm = await this.backendService.getPatientForm(authenticatedRequest);
}

if (this.userType == "Therapist") {
  this.therapistForm = await this.backendService.getTherapistForm(authenticatedRequest);
}

this.exerciseSessions = await this.backendService.getExerciseSessions(authenticatedRequest);
if (this.exerciseSessions.length > 0) {
  this.totalExerciseSessionsCompleted = this.exerciseSessions.length;
  this.exerciseSessions.forEach(exerciseSession => this.totalTimeSpentExercisingSecs += parseInt(exerciseSession.
  total_time_taken_secs));
}
this.totalTimeSpentExercisingHumanReadable = this.secondsToHms(this.totalTimeSpentExercisingSecs)
this.userDataFetchInProgress = false;
}
```

```
pub async fn get_exercise_sessions(
    Extension(state): Extension<Arc<State>>,
    Json(payload): Json<AuthenticatedRequest>,
) -> Result<Json<Vec<data::ExerciseSession>>, StatusCode> {
    match utils::check_authenticated_request(&state.db, &payload.session_id, &payload.email).await {
        Ok(_) => (),
        Err(e) => return Err(e),
    };

    let exercise_essions = match data::get_exercise_sessions(&state.db, &payload.email).await {
        Ok(exercise_sessions) => match exercise_sessions {
            Some(exercise_sessions) => exercise_sessions,
            None => {
                info!("No exercise sessions available for request");
                return Err(StatusCode::BAD_REQUEST);
            }
        },
        Err(e) => {
            error!("Failure finding exercise sessions: {}", e);
            return Err(StatusCode::INTERNAL_SERVER_ERROR);
        }
    };
    Ok(Json(exercise_essions))
}
```

Exercise Recommendations

- Patient's form information is used to provide exercise recommendations

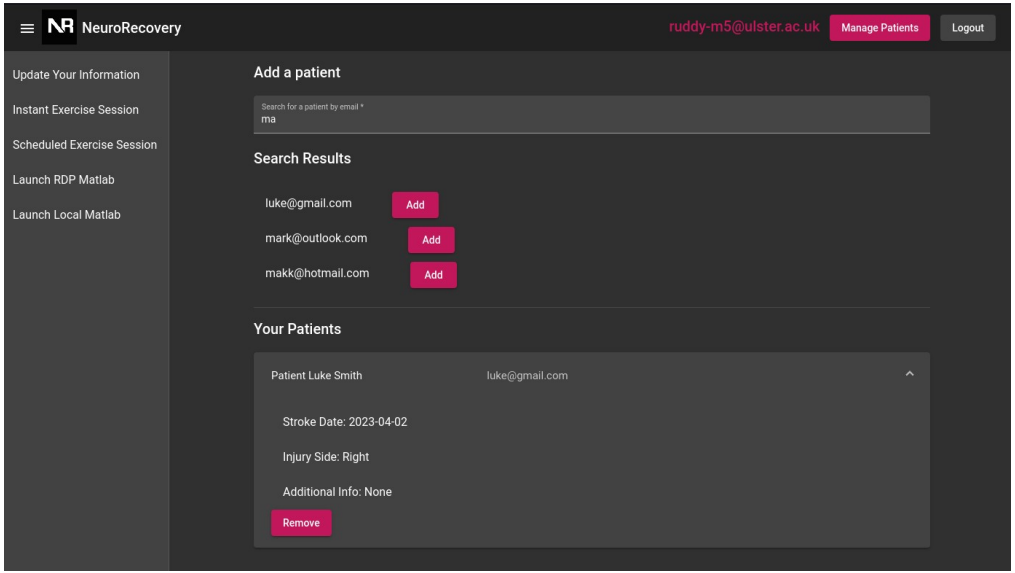
```
<div class='recommended-box' *ngIf="injuryType == 'Upper Limb' || injuryType == 'Lower Limb'">
    <ng-container *ngIf="injuryType == 'Upper Limb'">
        <h2>Recommended based on your {{ injuryType }} injury</h2>
        <button class='recommended-button' routerLink='/instant-ul' color='primary' mat-raised-button>Start Upper
        Limb General Exercise Session</button>
    </ng-container>
    <ng-container *ngIf="injuryType == 'Lower Limb'">
        <h2>Recommended based on your {{ injuryType }} injury</h2>
        <button class='recommended-button' routerLink='/instant-ll' color='primary' mat-raised-button>Start Lower
        Limb General Exercise Session</button>
    </ng-container>
</div>
```




```
async ngOnInit() {
  if (this.loginService.isLoggedIn()) {
    this.loggedIn = true;

    let authenticatedRequest = {
      email: localStorage.getItem('email'),
      session_id: localStorage.getItem('session_id'),
    } as AuthenticatedRequest;

    this.userDataFetchInProgress = true;

    this.userType = await this.backendService.getUserType(authenticatedRequest);

    if (this.userType == "Patient") {
      this.patientForm = await this.backendService.getPatientForm(authenticatedRequest);
      this.injuryType = this.patientForm.injury_type;
    }
  }
```

Therapist-Patient Link

- Therapist can search patients by email
- Therapist can add patients and view their statistics



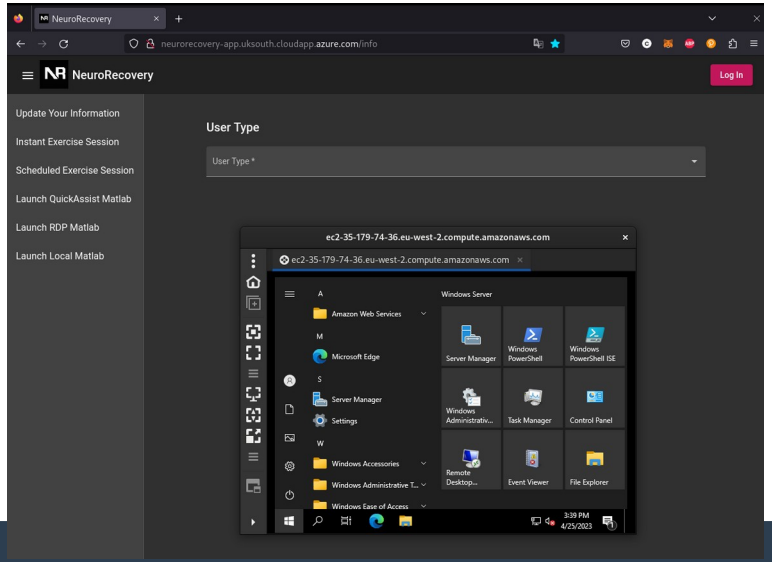

```rust
pub async fn post_therapist_patient(
    Extension(state): Extension<Arc<State>>,
    Json(payload): Json<TherapistPatientRequest>,
) -> Result<(), StatusCode> {
    match utils::check_authenticated_request(&state.db, &payload.session_id, &payload.email).await {
        Ok(_) => (),
        Err(e) => return Err(e),
    };

    // verify that this patient(user) exists
    match data::get_user_type(&state.db, &payload.patient_email).await {
        Ok(user_type) => {
            if user_type != "Patient" {
                info!("Insert patient therapist requested from a non-patient");
                return Err(StatusCode::BAD_REQUEST);
            }
        }
        Err(e) => {
            error!("Failure identifying user type: {}", e);
            return Err(StatusCode::INTERNAL_SERVER_ERROR);
        }
    }

    match data::add_therapist_patient(
        &state.db,
        &payload.email,
        &payload.patient_email,
        &payload.session_id,
    )
    .await
    {
        Ok(_) => (),
        Err(e) => {
            error!("Failure inserting therapist patient: {}", e);
            return Err(StatusCode::INTERNAL_SERVER_ERROR);
        }
    }
    Ok(())
}
```

```rust
pub async fn find_patients_by_email_substring(
    db: &Database,
    email: &str,
) -> Result<Option<Vec<Patient>>, Box<dyn Error>> {
    let coll = db.collection::<User>("users");
    let filter = doc! { "email": { "$regex": format!(".*{}.*", email) } };
    let mut cursor = coll.find(filter, None).await?;

    let mut patients = vec![];
    while let Some(user) = cursor.try_next().await? {
        if get_user_type(db, &user.email).await? == "Patient" {
            patients.push(Patient { email: user.email });
        }
    }
    if patients.len() == 0 {
        return Ok(None);
    }
    Ok(Some(patients))
}
```

MatLab Psychtoolbox Simulation Integration

- NeuroRecovery app provides a frontend button to reach MatLab Psychtoolbox Simulation
- Local Python server on client triggers QuickAssist, RDP or Local Matlab program for Psychtoolbox Simulation

```
launchRdpMatlab() {
  fetch('http://localhost:9090/launch_matlab_rdp', {
    method: 'GET'
  })
    .then(response => {
      console.log(response);
      if (!response.ok) {
        this.snackBar.open(errorMessages['matlabFailedLaunch'], '', {
          duration: 3000,
          panelClass: ['mat-toolbar', 'mat-warn'],
          verticalPosition: 'top',
          horizontalPosition: 'center',
        });
        throw new Error('Non-200 code from launch_matlab/');
      }
      this.snackBar.open(successMessages['matlabSuccessfulLaunch'], '', {
        duration: 3000,
        panelClass: ['mat-toolbar'],
        verticalPosition: 'top',
        horizontalPosition: 'center',
      });
      console.log('Successfully launched MATLAB!');
    })
```




```
@app.route('/launch_matlab_rdp')
def launch_matlab_rdp():
    if platform.system().lower() == "windows":
        subprocess.call(f'cmdkey /generic:"ec2-35-179-74-36.eu-west-2.compute.amazonaws.com" /user:"Administrator" /pass:
        "{RDP_PASS}"', shell=True);
        subprocess.call(windows_rdp_cmd, shell=True)
    else:
        subprocess.call(linux_rdp_cmd, shell=True)

    response = jsonify("Matlab RDP should be launched")
    response.headers.add("Access-Control-Allow-Origin", "*")
    response.headers.add("Access-Control-Request-Headers", "X-Requested-With, accept, content-type")
    response.headers.add("Access-Control-Allow-Methods", "GET")
    return response

@app.route('/launch_matlab_quickassist')
def launch_matlab_quickassist():
    subprocess.call("explorer.exe shell:AppsFolder\\MicrosoftCorporationII.QuickAssist_8wekyb3d8bbwe!App", shell=True)

    response = jsonify("Matlab QuickAssist should be launched")
    response.headers.add("Access-Control-Allow-Origin", "*")
    response.headers.add("Access-Control-Request-Headers", "X-Requested-With, accept, content-type")
    response.headers.add("Access-Control-Allow-Methods", "GET")
    return response
```

Technical Implementation

- TypeScript Angular SPA Frontend
- Rust Axum Backend
- MongoDB NoSQL database
- Hosted in a Kubernetes Cluster(K3s) running in an Azure RHEL Virtual Machine
- Tilt CI/CD is ran in the VM to provision the full app

TypeScript Angular SPA Frontend

- Single Page Application(SPA) returns a HTML, CSS and JavaScript bundle to user's browser
- Instantaneous page switches when no backend interaction is required
- Backend interaction requires requests to be made from the user's browser using the JavaScript fetch API
- Component-based frontend development, each page in the app is a separate component

```
@NgModule({
  declarations: [
    AppComponent,
    InfoComponent,
    InstantComponent,
    ScheduledComponent,
    LoginComponent,
    RegisterComponent,
    UserComponent,
    InstantUlComponent,
    InstantLlComponent,
    InstantHandComponent,
    ScheduledResultComponent,
    TherapistPatientsComponent,
  ],
  imports: modulesImports,
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule { }
```

```
export class BackendService {
  // NOTE: In production this URL should be the host server address
  public backendBaseUrl = 'http://localhost:8080';

  public loginUserEndpoint = 'login_user';
  public registerUserEndpoint = 'register_user';
  public postPatientFormEndpoint = 'post_patient_form';
  public postTherapistFormEndpoint = 'post_therapist_form';
  public postExerciseSessionEndpoint = 'post_exercise_session';
  public getPatientFormEndpoint = 'get_patient_form';
  public getTherapistFormEndpoint = 'get_therapist_form';
  public getTherapistPatientsEndpoint = 'get_therapist_patients';
  public postTherapistPatientEndpoint = 'post_therapist_patient';
  public removeTherapistPatientEndpoint = 'remove_therapist_patient';
  public getExerciseSessionsEndpoint = 'get_exercise_sessions';
  public getUserTypeEndpoint = 'get_user_type';
  public sendEmailEndpoint = 'send_email';
  public searchPatientsEndpoint = 'search_patients';
```

Rust Axum Backend

- Rust HTTP API developed with Axum framework
- Structs dictate request and response data, which is serialised and deserialised as JSON
- Uses mongodb Rust crate for database interaction
- Frontend sends requests to the backend defined HTTP endpoints

```rust
fn create_router(state: Arc<routes::State>) -> Router {
    Router::new()
        .route("/", get(routes::index))
        .route("/register_user", post(routes::register_user))
        .route("/login_user", post(routes::login_user))
        .route("/post_patient_form", post(routes::post_patient_form))
        .route("/post_therapist_form", post(routes::post_therapist_form))
        .route(
            "/post_exercise_session",
            post(routes::post_exercise_session),
        )
        .route("/get_patient_form", post(routes::get_patient_form))
        .route("/get_therapist_form", post(routes::get_therapist_form))
        .route(
            "/get_therapist_patients",
            post(routes::get_therapist_patients),
        )
        .route(
            "/post_therapist_patient",
            post(routes::post_therapist_patient),
        )
        .route(
            "/remove_therapist_patient",
            post(routes::remove_therapist_patient),
        )
        .route(
            "/get_exercise_sessions",
            post(routes::get_exercise_sessions),
        )
        .route("/get_user_type", post(routes::get_user_type))
        .route("/send_email", post(routes::send_email))
        .route("/search_patients", post(routes::search_patients))
        .layer(CorsLayer::permissive())
        .layer(Extension(state))
}
```

MongoDB NoSQL Database

- No explicit database schema is required as MongoDB is a NoSQL database
- Rust structs define the data structures that are stored in the database
- Only the backend server interacts with the MongoDB database. It is not exposed to the internet and is only accessible from within the Kubernetes cluster

```
test> use neurorecovery
switched to db neurorecovery
neurorecovery> show collections
exercise_sessions
patient_forms
therapist_forms
therapist_patients
users
neurorecovery> db.users.find({})
[
  {
    _id: ObjectId("6446727d603866c6b4b134ff"),
    email: 'ruddy-m5@ulster.ac.uk',
    hash: '$argon2id$v=19$m=4096,t=3,p=1$2er4YWLgRtaH3gYAgagrow$vCc8ahAiZkczFDJpK+ybetcl88+oyTP0dZ4myMGW7m8',
    salt: '2er4YWLgRtaH3gYAgagrow',
    session_id: 'vfmBTYT86Zkejd70cOfoiYaQwt75jIfAd6ZgZUZz9GWnZ8W274pICXtPqUuJyFuuotfTIuTsy7rtlmjZ4xVX0Zpypb4v9fPYfyS4h4s42yjf2DH2yYYscb0VNGk6419O'
  },
  {
    _id: ObjectId("64467fbb603866c6b4b13505"),
    email: 'luke@gmail.com',
    hash: '$argon2id$v=19$m=4096,t=3,p=1$aKF1Ddq+twgy+CWAOiQGOA$ViILq4PCFd9AssR4R2s0X+UR9lqgheprYF+neG8WHv4',
    salt: 'aKF1Ddq+twgy+CWAOiQGOA',
    session_id: 'V26ulU5CCbO64tLElEsOAebtd9WZI7aQdXNH92Qhfk6Dz7JtO3Uq4KK6sAOHrnr78WBI8YWWL6xZDIakhrBaRNv4t8YYSHykYdbdDhNKafImAixfBM9A5Ck6ejeenKon'
  },
  {
    _id: ObjectId("64467fdc603866c6b4b13507"),
    email: 'mark@outlook.com',
    hash: '$argon2id$v=19$m=4096,t=3,p=1$ZoDz65BzNdpwSARFJ6ntbQ$p2qZI77Sob0zhu/m7t0Zv7nvOQwapsdyMv88D3JCpNs',
    salt: 'ZoDz65BzNdpwSARFJ6ntbQ',
    session_id: 'UNaqeWosrWVnYQ7nPwu1qRwh3Pw51DOnRuZz0pHhLJS4y7apFDaLYocew987zSqklCWG4igEeudNeNI4YI9GkMfstwhAhMgM4T8A2CcFeqvuHqAPH9ImN8Gx9uGenOX8'
  },
  {
    _id: ObjectId("644680336038866c6b4b13509"),
    email: 'makk@hotmail.com',
    hash: '$argon2id$v=19$m=4096,t=3,p=1$ZWknwR3KABCohGhZ7ct7fw$4te8FWK2M504PGJx0892MJc1eIyOBDrX8NSErCpuvtM',
    salt: 'ZWknwR3KABCohGhZ7ct7fw',
    session_id: '7s7BbY5vU3J1ZdjlF8qbspcR4Sbgb0jOuaUxy9z745QUNomrOlsbK4TgfxFt91cCZNA3DMq59AYpAxnPgfQ1L68nWDm4mXcfcGixl4EdY1y9wWvGKccnu3gGqRe65KQO'
  }
]
neurorecovery>
```

```rust
#[derive(Debug, Default, Clone, Serialize, Deserialize)]
pub struct User { 5 implementations
    pub email: String,
    pub hash: String,
    pub salt: String,
    pub session_id: String,
}

#[derive(Debug, Default, Clone, Serialize, Deserialize)]
pub struct PatientForm { 5 implementations
    pub full_name: String,
    pub stroke_date: String,
    pub injury_type: String,
    pub injury_side: String,
    pub additional_info: String,
    pub email: String,
    pub session_id: String,
}
```

Kubernetes Cluster in Azure VM

- Runs in K3s, a lightweight Kubernetes cluster distribution. Hosted in an Azure RHEL VM.
- Angular Frontend, Axum Backend, MongoDB all run in separate Docker containers
- Docker containers allow for consistent and reproducible deployments of software that won't be affected by changes on the host machine
- Kubernetes is a Container Orchestrator, which provides solutions such as automatic container restarts on failure to create resilient deployments

```
FROM docker.io/library/rust:1.60

WORKDIR app
COPY .env .
COPY . .

#RUN rustup toolchain install nightly-x86_64-unknown-linux-gnu && \
RUN cargo build --release

# Expose Axum server
EXPOSE 8080

ENTRYPOINT ["bash", "-c", "source /root/.bashrc && ./target/release/backend --addr 0.0.0.0:8080 --mongodb-url mongodb://
10.43.252.173:27017 --mongodb-username root --mongodb-name neurorecovery"]
```

```
[azureuser@neuro-vm neurorecovery]$ kubectl get all -n neurorecovery
NAME                                                  READY   STATUS    RESTARTS   AGE
pod/mongodb-84ff9df899-dblbb                          1/1     Running   0          26h
pod/neurorecovery-backend-chart-f97d7675d-9m6nw       1/1     Running   0          26h
pod/neurorecovery-frontend-chart-68f7d788bd-skms6     1/1     Running   0          62m

NAME                                          TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/mongodb                               ClusterIP      10.43.252.173   <none>        27017/TCP        26h
service/neurorecovery-backend-chart-balancer  LoadBalancer   10.43.232.105   10.0.0.4      8080:30161/TCP   26h
service/neurorecovery-frontend-chart-balancer LoadBalancer   10.43.220.110   10.0.0.4      80:30460/TCP     62m

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mongodb                       1/1     1            1           26h
deployment.apps/neurorecovery-backend-chart   1/1     1            1           26h
deployment.apps/neurorecovery-frontend-chart  1/1     1            1           62m

NAME                                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/mongodb-84ff9df899                      1         1         1       26h
replicaset.apps/neurorecovery-backend-chart-f97d7675d   1         1         1       26h
replicaset.apps/neurorecovery-frontend-chart-68f7d788bd 1         1         1       62m
[azureuser@neuro-vm neurorecovery]$
```

Tilt CI/CD

- While Kubernetes and containers provide a solution for running a resilient app deployment, they do not provide a way to boot all of it up automatically
- Tilt CI/CD can deploy the entire NeuroRecovery webapp with a single command: “tilt up”
- Once running, an admin accessing the Tilt CI/CD dashboard can rebuild each microservice with a click. This allows for fast developer feedback after making code changes.
- Automatically runs the Rust Axum backend tests, providing confidence that the code works