```typescript
import {
  Stitch,
  RemoteMongoClient,
  StitchAppClient,
  AnonymousCredential,
  RemoteMongoDatabase,
  StitchUser
} from 'mongodb-stitch-browser-sdk';
import { ObjectId } from 'bson';


/**
 * Type of log. Holds both the event and details fields when it comes to CSV file
 * generation.
 */
export enum LogItemType {
  CREATE_START,
  CREATE_PASSWORD,
  CREATE_RESET,
  CREATE_READY,
  CONFIRM_SHOW,
  CONFIRM_INCORRECT,
  CONFIRM_CORRECT,
  CONFIRM_COMPLETE,
  TEST_SHOW,
  TEST_INCORRECT,
  TEST_FAIL,
  TEST_PASS,
  FINISH
}

/**
 * Document holding a specific log
 */
export class LogItem {
  _id: ObjectId | undefined;
  readonly scheme: string = "colourCircles";
  readonly mode: string = "9:0-7";

  constructor(
    private time: Date,
    private userId: ObjectId,
    private site: string, // type
    private eventAndDetails: LogItemType, // event_details (separated by an
underscore)
    private data: string,
    private progressId: ObjectId
  ) { }

  // convert the object to Atlas-friendly JSON
  toJSON() {
    return {
      time: this.time.getTime(),
      userId: this.userId,
      site: this.site,
      scheme: this.scheme,
      mode: this.mode,
      eventAndDetails: LogItemType[this.eventAndDetails],
      data: this.data,
      progressId: this.progressId
    }
  }

  // build an object from Atlas
  static fromJSON(json: any): LogItem {
    const eventAndDetails: LogItemType = (LogItemType as any)[json.eventAndDetails];
    const item = new LogItem(
      new Date(json.time),
      json.userId,
      json.site,
      eventAndDetails,
      json.data,
      json.progressId
    );
    item._id = json._id;
    return item;
  }
}

/**
 * Document holding the progress of a participant through the entire process.
 */
export class ProgressItem {
  _id: ObjectId | undefined;

  constructor(
    private userId: ObjectId,
    private userName: string, // participant number === `Participant ${i}`
    private emailPassword: string,
    private bankingPassword: string,
    private shoppingPassword: string,
    private testedEmail: boolean,
    private testedBanking: boolean,
    private testedShopping: boolean
  ) { }

  // convert the object to Atlas-friendly JSON
  toJSON() {
    return {
      userId: this.userId,
      userName: this.userName,
      emailPassword: this.emailPassword,
      bankingPassword: this.bankingPassword,
      shoppingPassword: this.shoppingPassword,
      testedEmail: this.testedEmail,
      testedBanking: this.testedBanking,
      testedShopping: this.testedShopping
    }
  }

  // build an object from Atlas
  static fromJSON(json: any): ProgressItem {
    const progress = new ProgressItem(
      json.userId,
      json.userName,
      json.emailPassword,
      json.bankingPassword,
      json.shoppingPassword,
      json.testedEmail,
      json.testedBanking,
      json.testedShopping
    );
    progress._id = json._id;
    return progress;
  }
}

/**
 * Service class (provides the main functionality)
 */
export class StitchService {
  private static client: StitchAppClient;
  private static db: RemoteMongoDatabase;
  private static readonly PROGRESS_COL = "progress";
  private static readonly LOGS_COL = "logs";

  // connect to MongoDB Stitch and Atlas
  constructor() {
    if (!StitchService.client) {
      const client = StitchService.client =
        Stitch.initializeDefaultAppClient("comp-3008-project-jjpdf");
      const mongodb = client.getServiceClient(
        RemoteMongoClient.factory,
        "mongodb-atlas"
      );
      StitchService.db = mongodb.db("database");
    }
  }

  isLoggedIn() {
    return StitchService.client.auth.isLoggedIn;
  }

  /**
   * authenticate the user anonymously
   */
  login(): Promise<StitchUser> {
    if (!StitchService.client.auth.isLoggedIn) {
      return Stitch.defaultAppClient.auth
        .loginWithCredential(new AnonymousCredential());
    } else return Promise.resolve(this.getStitchUser());
  }

  logout() {
    StitchService.client.auth.logout();
  }

  getStitchUser(): StitchUser {
    const user = StitchService.client.auth.user;
    if (user === undefined) throw new Error("user is not logged in");
    return user;
  }

  /**
   * initiates a new progress document for a participant
   * @param stitchUser the participant
   */
  startProgress(stitchUser: StitchUser): Promise<ProgressItem> {
    return new Promise((resolve, reject) => {
      const participantNum = prompt("Please enter your participant number");
      if (participantNum === "") return reject("you must enter a participant
number");
      const progress = new ProgressItem(
        new ObjectId(stitchUser.id),
        `Participant ${participantNum}`,
        "", "", "", false, false, false
      );
      console.log(progress.toJSON());
      StitchService.db.collection(StitchService.PROGRESS_COL)
        .insertOne(progress.toJSON())
        .then(result => {
          progress._id = result.insertedId;
          resolve(progress);
        })
        .catch(reject);
    })
  }

  /**
   * Retrieves the progress document for a participant
   * @param progressId
   */
  getProgress(progressId: string): Promise<ProgressItem | null> {
    return new Promise((resolve, reject) => {
      console.log(progressId);
      StitchService.db.collection(StitchService.PROGRESS_COL)
        .findOne({ _id: new ObjectId(progressId) })
        .then(res => resolve(res ? ProgressItem.fromJSON(res) : null))
        .catch(reject);
    });
  }

  /**
   * Updates a participant's progress
   * @param progress
   */
  updateProgress(progress: ProgressItem): Promise<void> {
    return new Promise((resolve, reject) => {
      StitchService.db.collection(StitchService.PROGRESS_COL)
        .updateOne({ _id: new ObjectId(progress._id) }, progress.toJSON())
        .then(() => resolve())
        .catch(reject);
    });
  }

  /**
   * Posts a log to the database
   * @param log
   */
  postLog(log: LogItem): Promise<LogItem> {
    return new Promise((resolve, reject) => {
      StitchService.db.collection(StitchService.LOGS_COL)
        .insertOne(log.toJSON())
        .then(result => {
          log._id = result.insertedId;
          resolve(log);
        })
        .catch(reject);
    });
  }
}
```