GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

SCHOOL OF POLYTECHNIC (GREATER NOIDA , UTTAR PRADESH)

PROJECT - II (DPCS9999) REPORT ON EncDoc

By

LAKSHY SHARMA
&
MOHD ABDULLAHA

Admission no.

19GPTC4060011(LAKSHY SHARMA)

19GPTC4060015(MOHD ABDULLAHA)

In partial fulfillment of requirements for the award of the degree

DIPLOMA IN COMPUTER SCIENCE & ENGINEERING

(Under the guidance of Er. Anand Dohare and Er. Nutan Gusain)

GALGOTIAS
UNIVERSITY
(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

CERTIFICATE

This is to certify that **Mohd Abdullaha(19GPTC4060015)/Lakshy sharma (19GPTC4060011)**, student of **Diploma in Computer Science & Engineering** ,Six Semester, Department of Computer Science of Galgotias University, has pursued the Major Project titled "**ENCDOC**" under the supervision of **Er. Anand Dohare**, Head Of Department (HOD) and Assistance Professor **Er. Nutan Gusain** and the report has been submitted in partial fulfillment of requirements for the award of the degree, Diploma in Computer Science & Engineering by Galgotias University in the Year 2021.

Er. Nutan Gusain

Assistance Professor

Er. Anand Dohare Head of

Department (HOD)

ACKNOWLEDGEMENT

I express my sincere regard and indebtedness to my project internal guide **Er. Anand Dohare** and **Er. Nutan Gusain** , for his valuable time, guidance, encouragement, support and cooperation throughout the duration of our project. I would sincerely like to thank ITDepartment for giving me the opportunity to work on enhancing my technical skills whileundergoing this project. This project was done under the guidance of **Er. Anand Dohare**,Head of Department and **Er. Nutan Gusain** . This project helped in understanding the various parameters which are involved in the development of a web application and the working and integration of front end along with the back end to create a fully functionalweb application.

I would like to thank Er. Anand Dohare (Head of Department), Er. Nutan Gusain and whole of department for their constant support.

Lakshy Sharma: (19GPTC4060011)

Enrollment no: (19014060010)

Mohd Abdullaha: (19GPTC4060015)

Enrollment no : (19014060013)

ABSTRACT

The main objective of the EncDoc is to provide Security. All user data is managed under EU guidelines. The project is totally built at administrative end and thus only the administrator is guaranteed the access. The purpose of the project is to build an application program to reduce the security and integrity of files. It tracks all the details about the of file.

INDEX

CHAPTER 1 : INTRODUCTION

1.1 INTRODUCTION

The project EncDoc is a backend and android application that allows user to handle their files and password.

1.2 AIM

Our proposed system is an EncDoc that enables ease for theuser. It overcomes the disadvantages of the security threat. Our proposed system is a medium to Encrypt file atfinger tip. This project can be used at production level to generate hash sum of their files to maintain integrity and will be used as password handler or provider. Our Aim is to make cryptography more usable in daily life.

1.3 FEASIBILITY STUDY

A feasibility study is a high-level capsule version of the entire System analysis and Design Process. The study begins by classifying the problem definition. Feasibility is to determine if it's worth doing. Once an acceptance problem definition has been generated, the analyst develops a logical model of the system. A search for alternatives is analyzed carefully. There are 3 parts in feasibility study.

1) Operational Feasibility

2) Technical Feasibility

3) Economical Feasibility

1.3.1 OPERATIONAL FEASIBILITY

Operational feasibility is the measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.The operational feasibility assessment focuses on the degree to which the proposed development projects fits in with the existing business environment and objectives with regard to development schedule, delivery date, corporate culture and existing business processes.To ensure success, desired operational outcomes must be imparted during design and development. These include such design-dependent parameters as reliability, maintainability, supportability, usability, producibility, disposability, sustainability, affordability and others. These parameters are required to be considered at the early stages of design if desired operational behaviours are to be realised. A system design and development requires appropriate and timely application of engineering and management efforts to meet the previously mentioned parameters. A system may serve its intended purpose most effectively when its technical and operating characteristics are engineered into the design. Therefore, operational feasibility is a critical aspect of systems engineering that needs to be an integral part of the early design phases.

1.3.2 TECHNICAL FEASIBILITY

This involves questions such as whether the technology needed for the system exists, how difficult it will be to build, and whether the firm has enough experience using that technology. The assessment is based on outline design of system requirements in terms of input, processes, output, fields, programs and procedures. This can be qualified in terms of volume of data, trends, frequency of updating inorder to give an introduction to the technical system. The application is the fact that it has been developed on windows XP platform and a high configuration of 1GB RAM on Intel Pentium Dual core processor. This is technically feasible .The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organization and their applicability to the expected needs of the proposed system. It is an evaluation of the hardware and software and how it meets the need of the proposed system.

1.3.3 ECONOMICAL FEASIBILITY

Establishing the cost-effectiveness of the proposed system i.e. if the benefits do not outweigh the costs then it is not worth going ahead. In the fast paced world today there is a great need of online social networking facilities. Thus the benefits of this project in the current scenario make it economically feasible. The purpose of the economic feasibility assessment is to determine the positive economic benefits to the organization that the proposed system will provide. It includes quantification and identification of all the benefits expected. This assessment typically involves a cost/benefits analysis.

1.4 ORGANISATION OF THE REPORT

1.4.1 INTRODUCTION

This section includes the overall view of the project i.e. the basic problem definition and the general overview of the problem which describes the problem in layman terms. It also specifies the software used and the proposed solution strategy.

1.4.2 SOFTWARE REQUIREMENTS SPECIFICATION

This section includes the Software and hardware requirements for the smooth running of the application.

1.4.3 DESIGN & PLANNING

This section consists of the Software Development Life Cycle model. It also contains technical diagrams like the Data Flow Diagram and the Entity Relationship diagram.

1.4.4 IMPLEMENTATION DETAILS

This section describes the different technologies used for the entire development processof the Front-end as well as the Back-end development of the application.

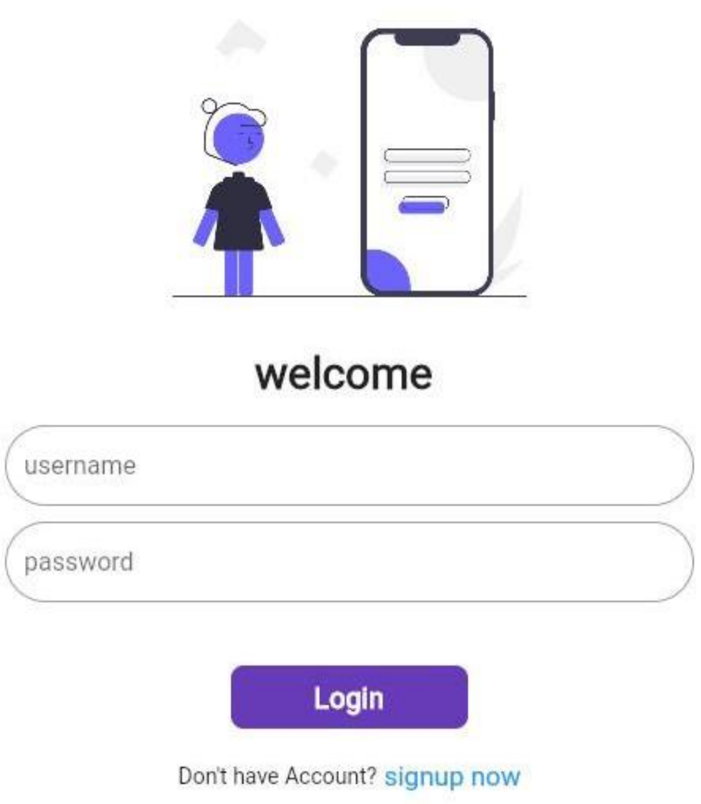
1.4.5 RESULTS AND DISCUSSION

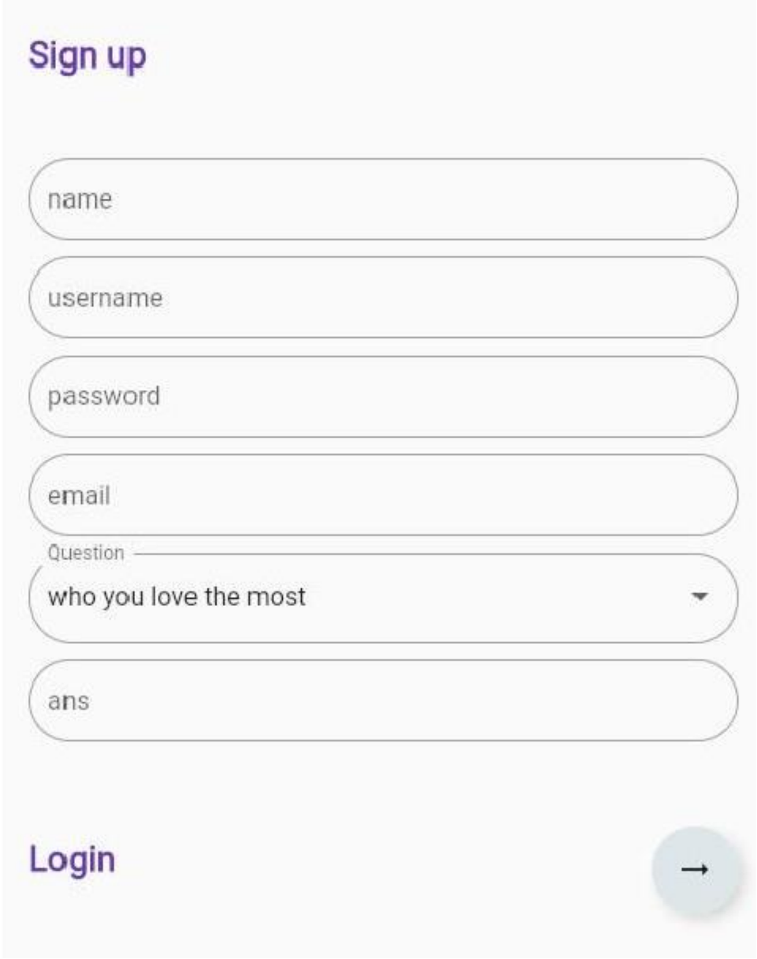
This section has screenshots of all the implementation i.e. user interface and their description.

1.4.6 SUMMARY AND CONCLUSION

This section has screenshots of all the implementation i.e. user interface and their description.

1.5 LOGIN & SIGNUP PAGE





Logging in tells the system who you are and what you have permission to do. Likewise, when you finish, you will log out so that no one else can access your files without permission.

CHAPTER 2 : SOFTWARE REQUIREMENTS SPECIFICATION

2.1 Hardware Requirements

Number	**Description**
1	PC with 250 GB or more Hard disk.
2	PC with 2 GB RAM.
3	PC with Pentium 1 and Above.

2.2 Software Requirements

Number	**Description**	**Type**
1	Operating System	Windows
2	Language	Node JS , Dart

3	Database	Mongo-db
4	IDE	Visual Code, Android Studio
5	Browser	Google Chrome

1

NODE JS

Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Node.js was developed
by Ryan Dahl in 2009 and its latest version is v0.10.36.

–

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X,
Microsoft Windows, and Linux.

Node.js also provides a rich library of various JavaScript modules which simplifies the development of web
applications using Node.js to a great extent.

DART

Dart is a programming language designed for client development,[8][9] such as for the web and mobile apps. It is developed by Google and can also be used to build server and desktop applications.

It is an object-oriented, class-based, garbage-collected language with C-style syntax.[10] It can compile to either native code or JavaScript, and supports interfaces, mixins, abstract classes, reified generics and type inference.[11]

Dart was unveiled at the GOTO conference in Aarhus, Denmark, October 10–12, 2011.[12] The project was founded by Lars Bak and Kasper Lund.[13] Dart 1.0 was released on November 14, 2013.[14]

Dart initially had a mixed reception and the Dart initiative has been criticized by some for fragmenting the web, due to the original plans to include a Dart VM in Chrome. Those plans were dropped in 2015 with the 1.9 release of Dart to focus instead on compiling Dart to JavaScript.[15]

Dart 2.0 was released in August 2018, with language changes including a sound type system.[16]

Dart 2.6 introduced a new extension, dart2native, which extends native compilation to the Linux, macOS, and Windows desktop platforms. Earlier developers could create new tools using only Android or iOS devices. With this extension it also becomes possible to compose a program into self-contained executables. According to company representatives, it’s no longer necessary to have the Dart SDK installed, as the self-contained executables can now start running in a few seconds. The new extension is also integrated with the Flutter toolkit, making it possible to use the compiler on small services (for example,
backend support).

MONGO DB

MongoDB is an open source NoSQL database management program. NoSQL is used as an alternative to traditional relational databases. NoSQL databases are quite useful for working with large sets of distributed data. MongoDB is a tool that can manage document-oriented information, store or retrieve information.

MongoDB supports various forms of data. It is one of the many nonrelational database technologies that arose in the mid-2000s under the NoSQL banner -- normally, for use in big data applications and other processing jobs involving data that doesn't fit well in a rigid relational model. Instead of using tables and rows as in relational databases, the MongoDB architecture is made up of collections and documents.

Organizations can use Mongo DB for its ad-hoc queries, indexing, load balancing, aggregation, server-side JavaScript execution and other features.

ANDROID STUDIO

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA . On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Apply Changes to push code and resource changes to your running app without restarting your app
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App
- Engine

This page provides an introduction to basic Android Studio features. For a summary of the latest changes, see Android Studio release notes.

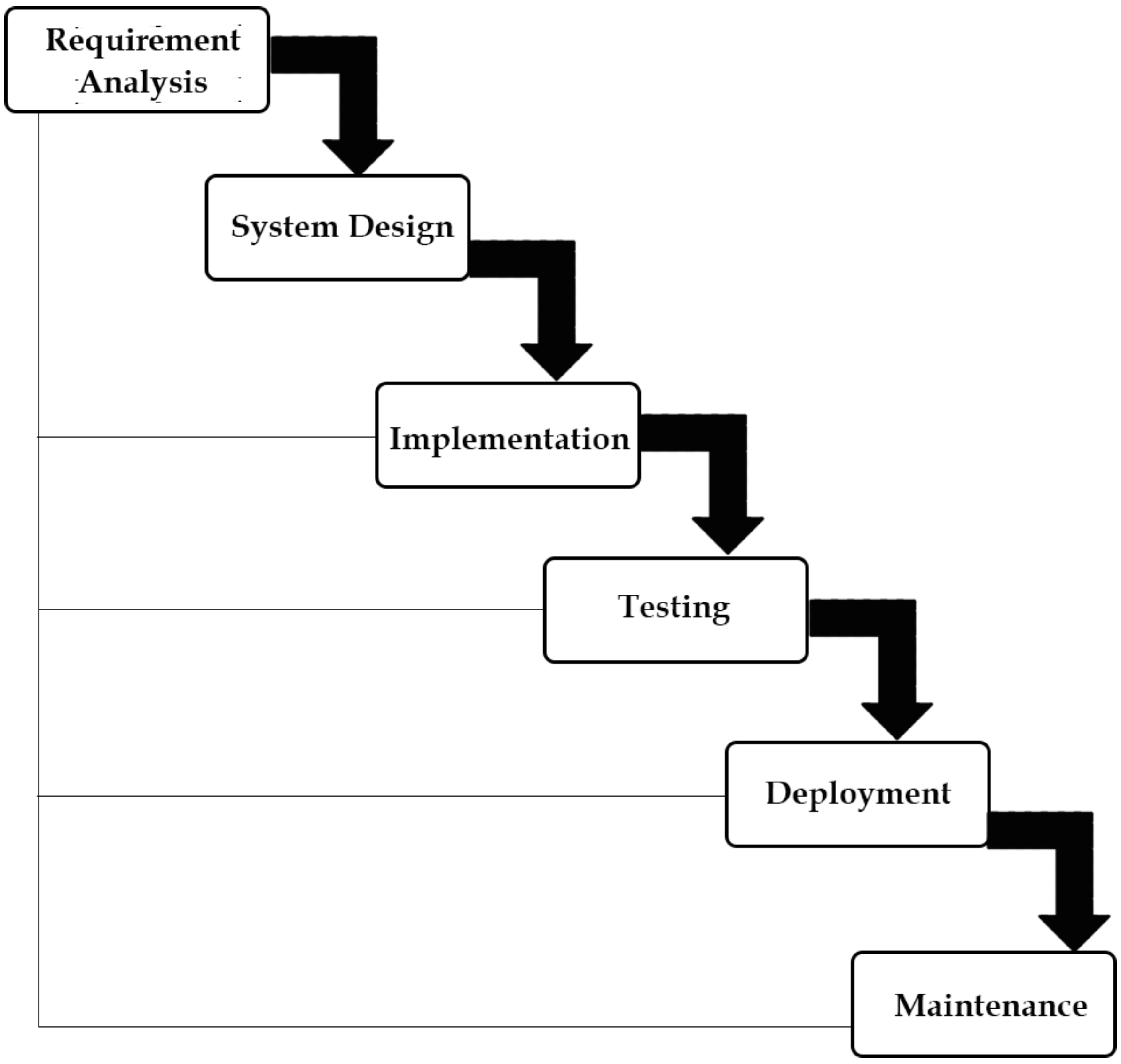
CHAPTER 3 : DESIGN & PLANNING

3.1 Software Development Life Cycle Model

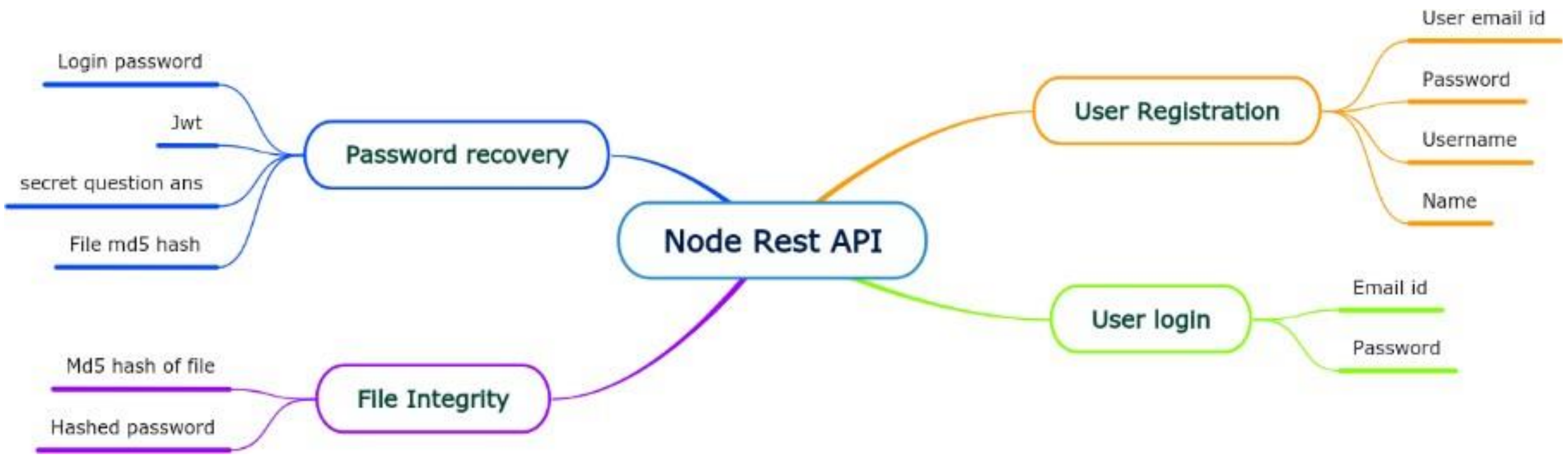
3.1.1WATERFALL MODEL

The waterfall model was selected as the SDLC model due to the following reasons:

- Requirements were very well documented, clear and fixed. Technology was
- adequately understood.
- Simple and easy to understand anduse.
- There were no ambiguous requirements.
- Easy to manage due to the rigidity of the model. Each phase has specific deliverablesand a review process.
- Clearly defined stages.
- Well understood milestones.Easy to arrange tasks.



3.2 ER Diagram



CHAPTER 4 : IMPLEMENTATION DETAILS

In this Section we will do Analysis of Technologies to use for implementing the project.

: *FRONT END 4.1.*

DART& FLUTTER

CHAPTER 5 : TESTING

5.1 : UNIT

TESTING

5.1.1Introduction

In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use. Intuitively, one can view a unit as the smallesttestable part of an application. In procedural programming, a unit could be an entire module, but it is more commonly an individual function or procedure. In object-orientedprogramming, a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are short code fragments created by programmers or occasionally bywhite box testers during the development process. It forms the basis for componenttesting. Ideally, each test case is independent from the others. Substitutes such asmethod stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation. Unit tests are typically written and run by software developers toensure that code meets its design and behaves as intended.

.

5.1.2

The goal of unit testing is to isolate each part of the program and show that the individualparts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits.

1) Find problems early : Unit testing finds problems early in the development cycle. In test- driven development (TDD), which is frequently used in both extreme programming and scrum, unit tests are created before the code itself is written. When the tests pass, that code is considered complete. The same unit tests are run against that function frequentlyas the larger code base is developed either as the code is changed or via an automated process with the build. If the unit tests fail, it is considered to be a bug either in the changed code or the tests themselves. The unit tests then allow the location of the fault or failure to be easily traced. Since the unit tests alert the development team of the problembefore handing the code off to testers or clients, it is still early in the development process.

2 **) Facilitates Change :** Unit testing allows the programmer to refactor code or upgrade system libraries at a later date, and make sure the module still works correctly (e.g., in regression testing). The procedure is to write test cases for all functions and methods sothat whenever a change causes a fault, it can be quickly identified. Unit tests detect changes which may break a design contract.

3 **) Simplifies Integration :** Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program firstand then testing the sum of its parts, integration testing becomes much easier.

4 **) Documentation :** Unit testing provides a sort of living documentation of the system. Developers looking to learn what functionality is provided by a unit, and how to use it, canlook at the unit tests to gain a basic understanding of the unit's interface (API).Unit testcases embody characteristics that are critical to the success of the unit. These characteristics can indicate appropriate/inappropriate use of a unit as well as negative behaviors that are to be trapped by the unit.

5.2 : INTEGRATION TESTING

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

5.2.1 Purpose

The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e., assemblages (or groups of units), are exercised through their interfaces using black-box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface. Test cases are constructed to test whether all the components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e., unit testing. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.Software integration testing is performed according to the software development life cycle (SDLC) after module and functional tests. The cross- dependencies for software integration testing are: schedule for integration testing, strategy and selection of the tools used for integration, define the cyclomatical complexity of the software and software architecture, reusability of modules and life-cycle and versioning management.Some different types of integration testing are big-bang, top- down, and bottom-up, mixed (sandwich) and risky-hardest. Other Integration Patterns[2] are: collaboration integration, backbone integration, layer integration, client-server integration, distributed services integration and high-frequency integration.

5.2.1.1 Big

In the big-bang approach, most of the developed modules are coupled together to form a complete software system or major part of the system and then used for integration testing. This method is very effective for saving time in the integration testing process. However, if the test cases and their results are not recorded properly, the entire integration process will be more complicated and may prevent the testing team from achieving the goal of integration testing.A type of big-bang integration testing is called "usage model testing" which can be used in both software and hardware integration testing. The basis behind this type of integration testing is to run user-like workloads in integrated user-like environments. In doing the testing in this manner, the environment is proofed, while the individual components are proofed indirectly through their use. Usage Model testing takes an optimistic approach to testing, because it expects to have few problems with the individual components. The strategy relies heavily on the component developers to do theisolated unit testing for their product. The goal of the strategy is to avoid redoing the testing done by the developers, and instead flesh-out problems caused by the interaction of the components in the environment.

5.2.1.2 Top-down And Bottom-up

Bottom-up testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.All the bottom or low- level modules, procedures or functions are integrated and then tested. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.Top-down testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.Sandwich testing is an approach to combine top down testing with bottomup testing.

5.3 : SOFTWARE VERIFICATION AND VALIDATION

5.3.1 Introduction

In software project management, software testing, and software engineering, verification and validation (V&V) is the process of checking that a software system meets specifications and that it fulfills its intended purpose. It may also be referred to as software quality control. It is normally the responsibility of software testers as part of the softwaredevelopment lifecycle. Validation checks that the product design satisfies or fits the intended use (high-level checking), i.e., the software meets the user requirements.This is done through dynamic testing and other forms of review.Verification and validation are not the same thing, although they are often confused. Boehm succinctly expressed the difference between

- Validation : Are we building the right product? Verification :
- Are we building the product right?

According to the Capability Maturity Model (CMMI-SW v1.1)

Software Verification: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of thatphase.

Software Validation: The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.

In other words, software verification is ensuring that the product has been built according to the requirements and design specifications, while software validation ensures that the product meets the user's needs, and that the specifications were correct in the first place. Software verification ensures that "you built it right". Software validation ensures that "you built the right thing". Software validation confirms that the product, as provided, will fulfill its intended use.

From Testing Perspective

- Fault – wrong or missing function in the code.
- Failure – the manifestation of a fault during execution.
- Malfunction – according to its specification the system does not meet its specified functionality

Both verification and validation are related to the concepts of quality and of software quality assurance. By themselves, verification and validation do not guarantee software quality; planning, traceability, configuration management and other aspects of software engineering are required.Within the modeling and simulation (M&S) community, the definitions of verification, validation and accreditation are similar:

- M&S Verification is the process of determining that a • computer model, simulation, or federation of models and simulations implementations and their associated data accurately represent the developer's conceptual description and specifications.
- M&S Validation is the process of determining the degree to which a model, simulation, or federation of models and simulations, and their associated data are accurate representations of the real world from the perspective of the intended use(s).

5.3.2 Classification

In mission-critical software systems, where flawless performance is absolutely necessary, formal methods may be used to ensure the correct operation of a system. However, often for non-mission-critical software systems, formal methods prove to be very costly and an alternative method of software V&V must be sought out. In such cases, syntactic methods are often used.

5.3.3 Test Cases

A test case is a tool used in the process. Test cases may be prepared for software verification and software validation to determine if the product was built according to the requirements of the user. Other methods, such as reviews, may be used early in the life cycle to provide for software validation.

5.4 : Black-Box

Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings. This method of test can be applied virtually to every level of software testing: unit, integration, system and acceptance. It typically comprises most if not all higher level testing, but can also dominate unit testing as well.

5.4.1 Test Procedures

Specific knowledge of the application's code/internal structure and programming knowledge in general is not required. The tester is aware of what the software is supposedto do but is not aware of how it does it. For instance, the tester is aware that a particular input returns a certain, invariable output but is not aware of how the software produces the output in the first place.

5.4.2 Test Cases

Test cases are built around specifications and requirements, i.e., what the application is supposed to do. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are primarily functional in nature, non-functional tests may also be used.The test designer selects both valid and invalid inputs and determines the correct output,often with the help of an oracle or a previous result that is known to be good, without any knowledge of the test object's internal structure.

5.5 : White-Box

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT). White-box testing can be applied at the unit, integration and system levels of the software testing process. Although traditional testers tended to think of white-box testing as being done at the unit level, it is used for integration and system testing more frequently today. It can test paths within a unit, paths between units during integration, and between subsystems during a system—level test. Though this method of test design can uncover many errors or problems, it has the potential to miss unimplemented parts of the specification or missing requirements.

5.5.1 Levels

1) Unit testing : White-box testing is done during unit testing to ensure that the code isworking as intended, before any integration happens with previously tested code. White-box testing during unit testing catches any defects early on and aids in any defects that happen later on after the code is integrated with the rest of the application and thereforeprevents any type of errors later on.

2) Integration testing : White-box testing at this level are written to test the interactions ofeach interface with each other. The Unit level testing made sure that each code was tested and working accordingly in an isolated environment and integration examines the correctness of the behaviour in an open environment through the use of white-box testingfor any interactions of interfaces that are known to the programmer.

3) Regression testing : White-box testing during regression testing is the use of recycledwhite-box test cases at the unit and integration testing levels.

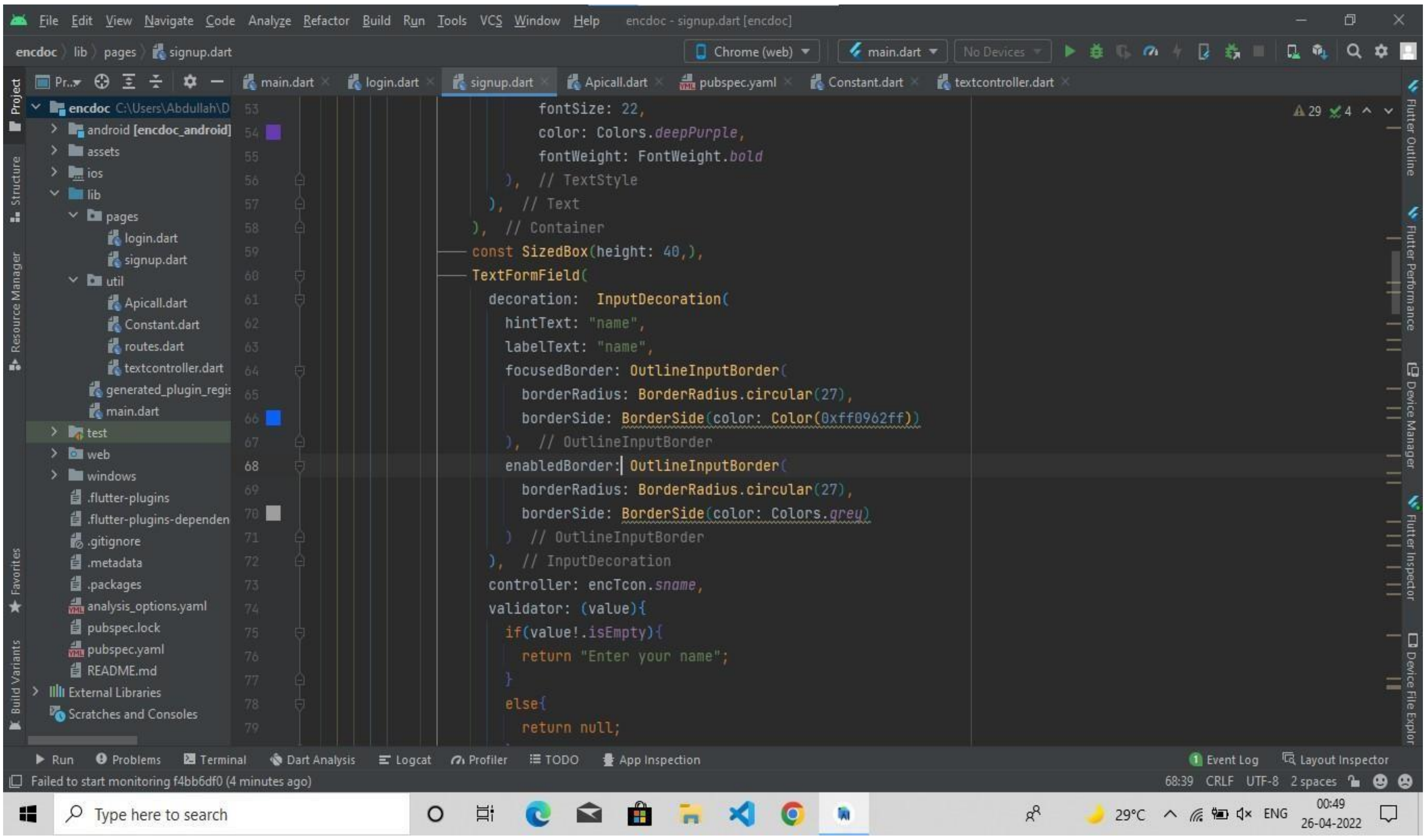
5.5.2

White-box testing's basic procedures involves the tester having a deep level of understanding of the source code being tested. The programmer must have a deep understanding of the application to know what kinds of test cases to create so that everyvisible path is exercised for testing. Once the source code is understood then the source code can be analyzed for test cases to be created. These are the three basic steps that white-box testing takes in order to create test cases:

- Input involves different types of requirements, functional specifications, detailed designing of documents, proper source code, security specifications. This is the preparation stage of white- box testing to layout all of the basic information.
- Processing involves performing risk analysis to guide whole testing process, proper test plan, execute test cases and communicate results. This is the phase of building test cases to make sure they thoroughly test the application the given results are recorded accordingly.
- Output involves preparing final report that encompasses all of the above preparationsand results.

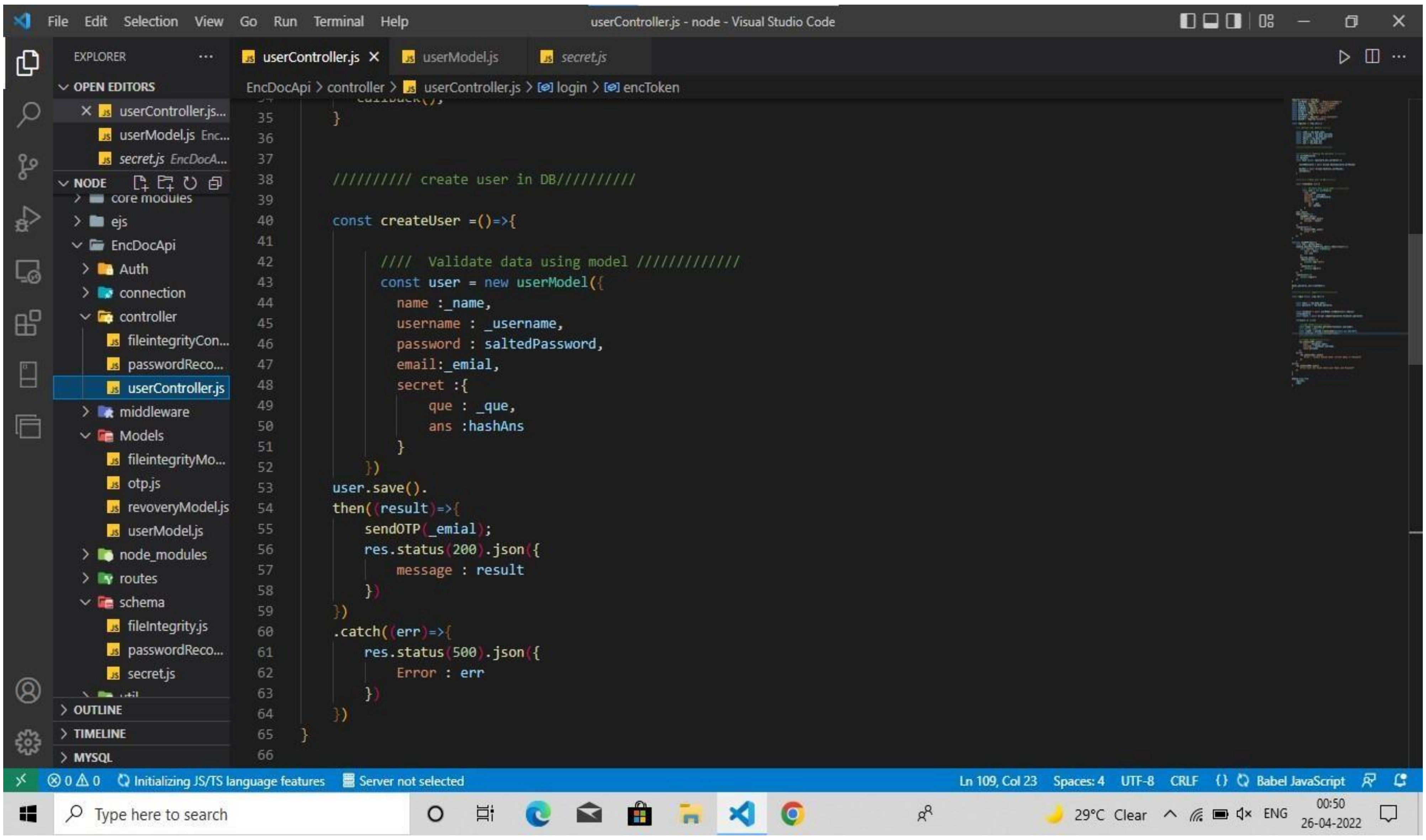
System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black-box testing, and as such, should require no knowledge of the inner design of the code or logic. As a rule, system testing takes, as its input, all of the "integrated" software components that have passed integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called assemblages) or between any of the assemblages and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

System testing is performed on the entire system in the context of a Functional Requirement Specification(s) (FRS) and/or a System Requirement Specification (SRS). System testing tests not only the design, but also the behavior and even the believed expectations of the customer.



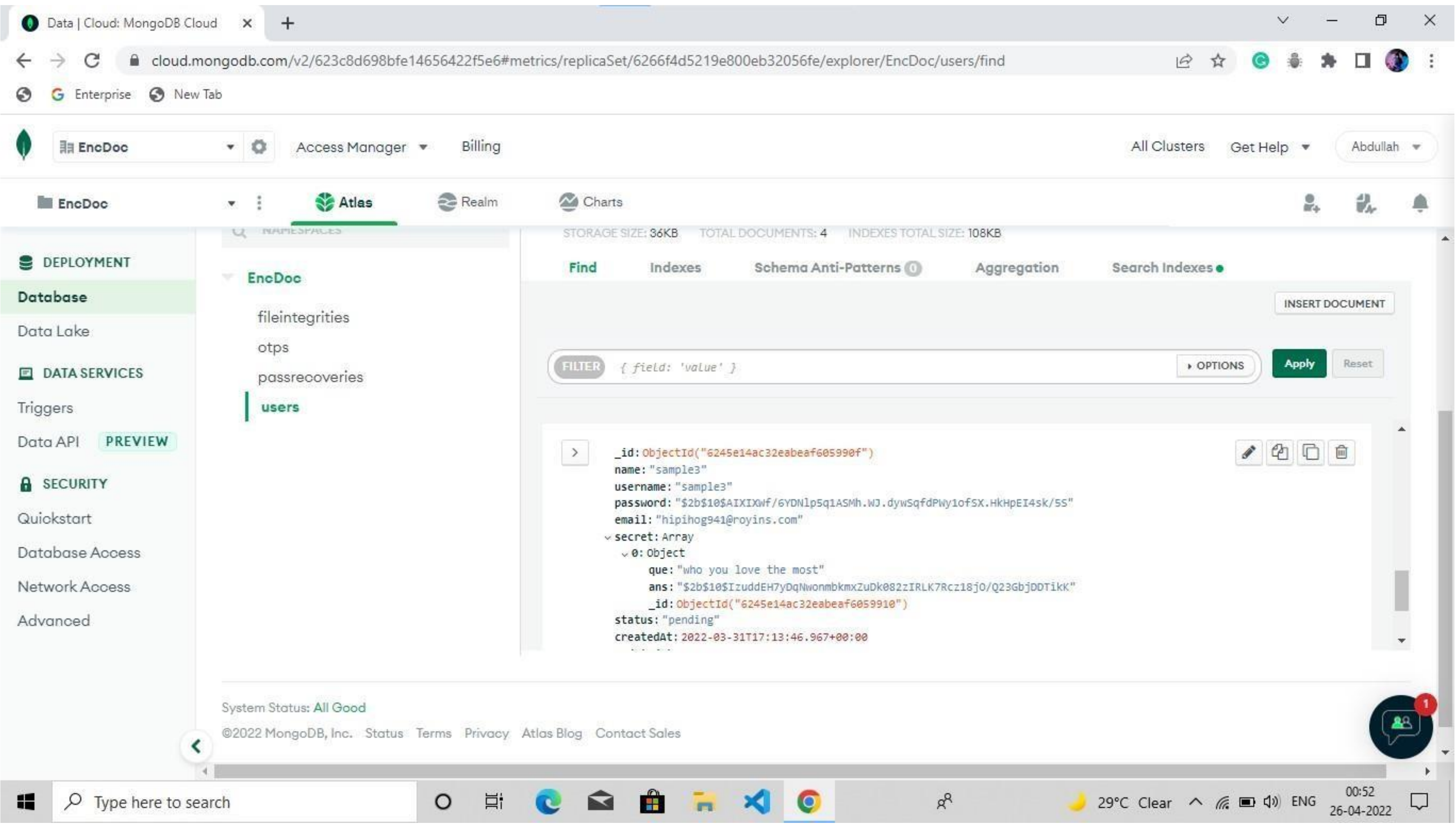
```
                fontSize: 22,
                color: Colors.deepPurple,
                fontWeight: FontWeight.bold
              ), // TextStyle
            ), // Text
          ), // Container
          const SizedBox(height: 40,),
          TextFormField(
            decoration:  InputDecoration(
              hintText: "name",
              labelText: "name",
              focusedBorder: OutlineInputBorder(
                borderRadius: BorderRadius.circular(27),
                borderSide: BorderSide(color: Color(0xff0962ff))
              ), // OutlineInputBorder
              enabledBorder: OutlineInputBorder(
                borderRadius: BorderRadius.circular(27),
                borderSide: BorderSide(color: Colors.grey)
              ) // OutlineInputBorder
            ), // InputDecoration
            controller: encTcon.sname,
            validator: (value){
              if(value!.isEmpty){
                return "Enter your name";
              }
              else{
                return null;
```

Failed to start monitoring f4bb6df0 (4 minutes ago)



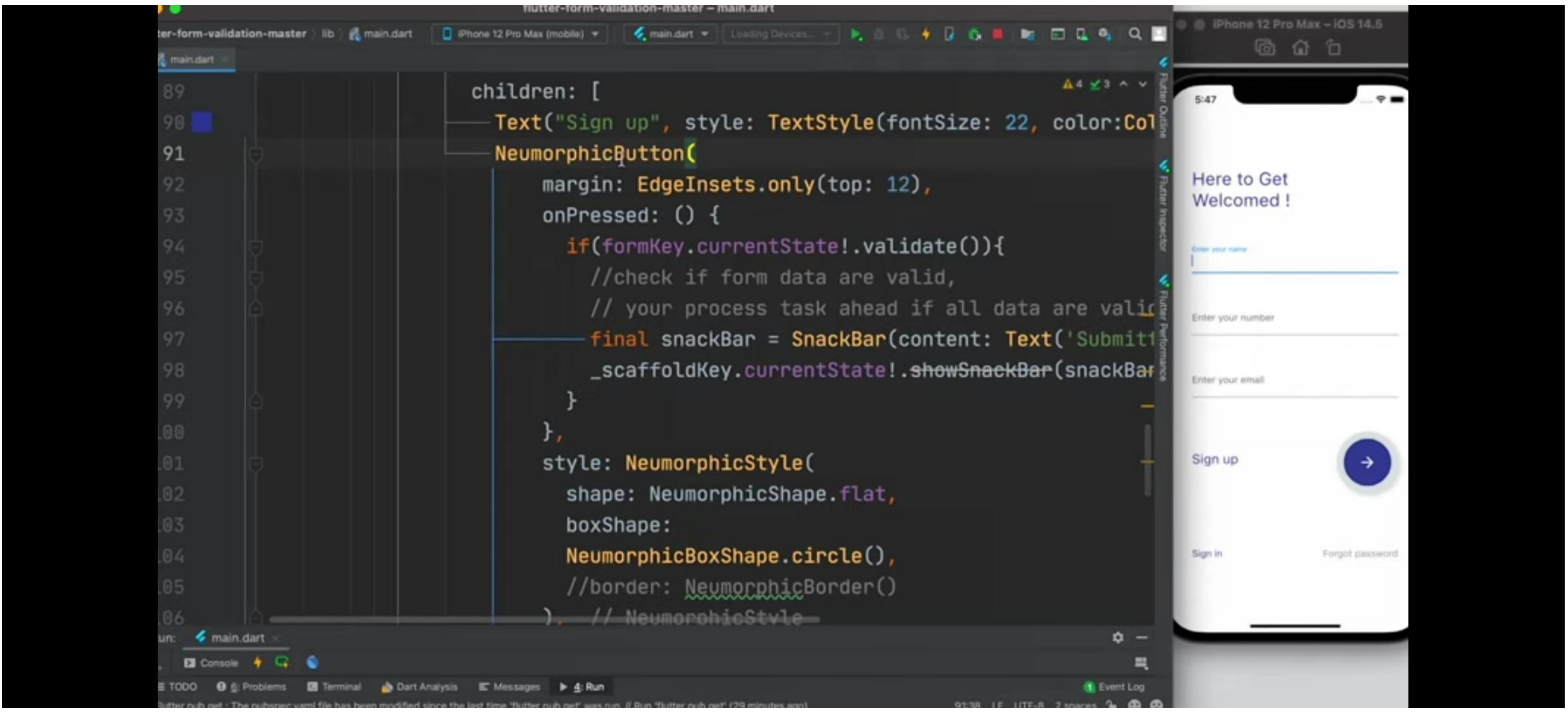
```
    }


    ////////// create user in DB//////////

    const createUser =()=>{

          ////  Validate data using model /////////////
          const user = new userModel({
            name :_name,
            username : _username,
            password : saltedPassword,
            email:_emial,
            secret :{
                que : _que,
                ans :hashAns
            }
        })
    user.save().
    then((result)=>{
        sendOTP(_emial);
        res.status(200).json({
            message : result
        })
    })
    .catch((err)=>{
        res.status(500).json({
            Error : err
        })
    })
}
```

Database

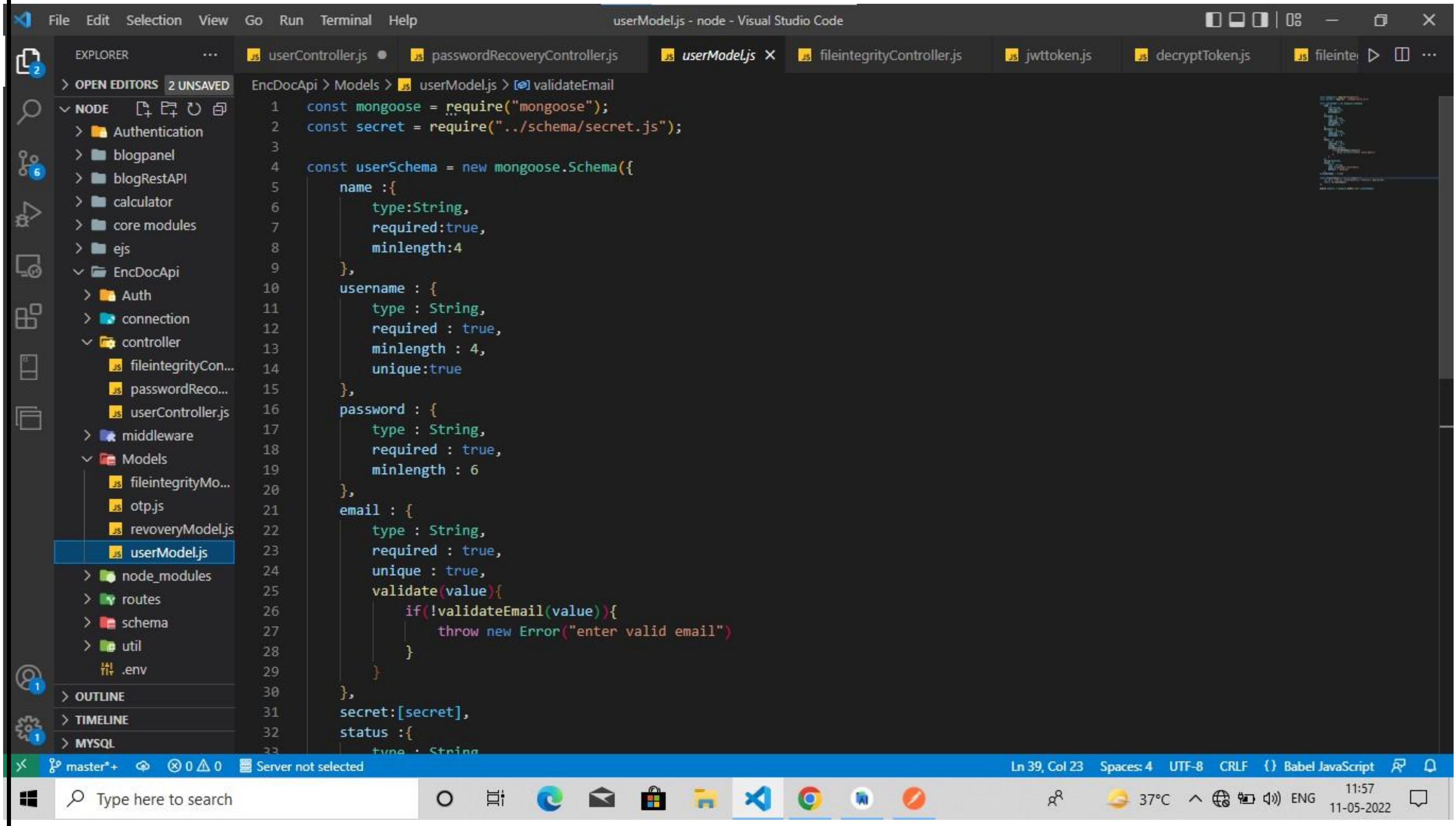


```
_id: ObjectId("6245e14ac32eabeaf605990f")
name: "sample3"
username: "sample3"
password: "$2b$10$AIXIXWf/6YDNlp5q1ASMh.WJ.dywSqfdPWy1ofSX.HkHpEI4sk/5S"
email: "hipihog941@royins.com"
secret: Array
  0: Object
    que: "who you love the most"
    ans: "$2b$10$IzuddEH7yDqNwonmbkmxZuDk082zIRLK7Rcz18jO/Q23GbjDDTikK"
    _id: ObjectId("6245e14ac32eabeaf6059910")
status: "pending"
createdAt: 2022-03-31T17:13:46.967+00:00
```

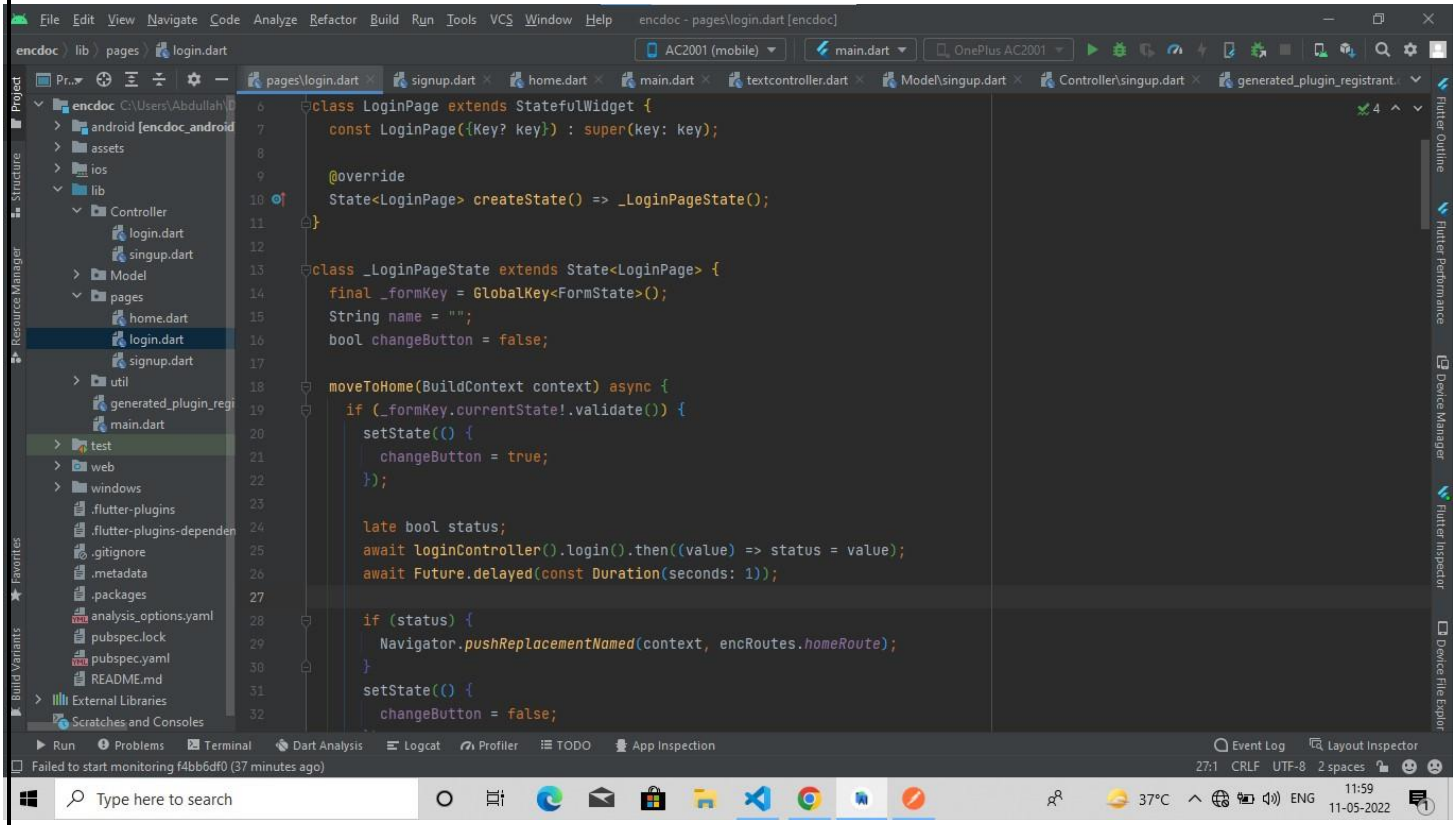
Some Code Screen Shot



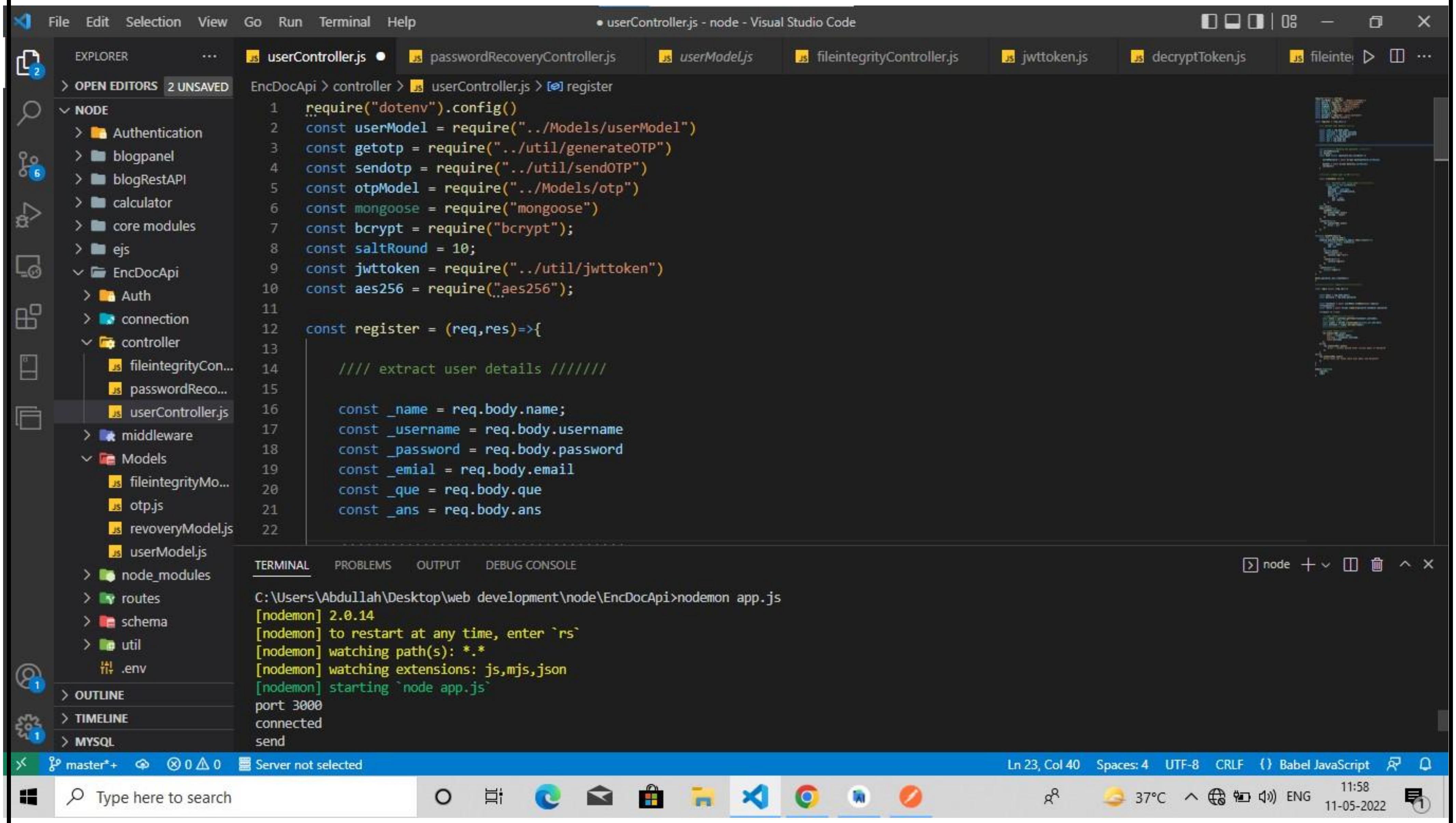
```
                    children: [
                      Text("Sign up", style: TextStyle(fontSize: 22, color:Col
                      NeumorphicButton(
                          margin: EdgeInsets.only(top: 12),
                          onPressed: () {
                            if(formKey.currentState!.validate()){
                              //check if form data are valid,
                              // your process task ahead if all data are valic
                              final snackBar = SnackBar(content: Text('Submitt
                              _scaffoldKey.currentState!.~~showSnackBar~~(snackBar
                            }
                          },
                          style: NeumorphicStyle(
                            shape: NeumorphicShape.flat,
                            boxShape:
                            NeumorphicBoxShape.circle(),
                            //border: NeumorphicBorder()
                          ), // NeumorphicStyle
```



```
const mongoose = require("mongoose");
const secret = require("../schema/secret.js");

const userSchema = new mongoose.Schema({
    name :{
        type:String,
        required:true,
        minlength:4
    },
    username : {
        type : String,
        required : true,
        minlength : 4,
        unique:true
    },
    password : {
        type : String,
        required : true,
        minlength : 6
    },
    email : {
        type : String,
        required : true,
        unique : true,
        validate(value){
            if(!validateEmail(value)){
                throw new Error("enter valid email")
            }
        }
    },
    secret:[secret],
    status :{
        type : String
```

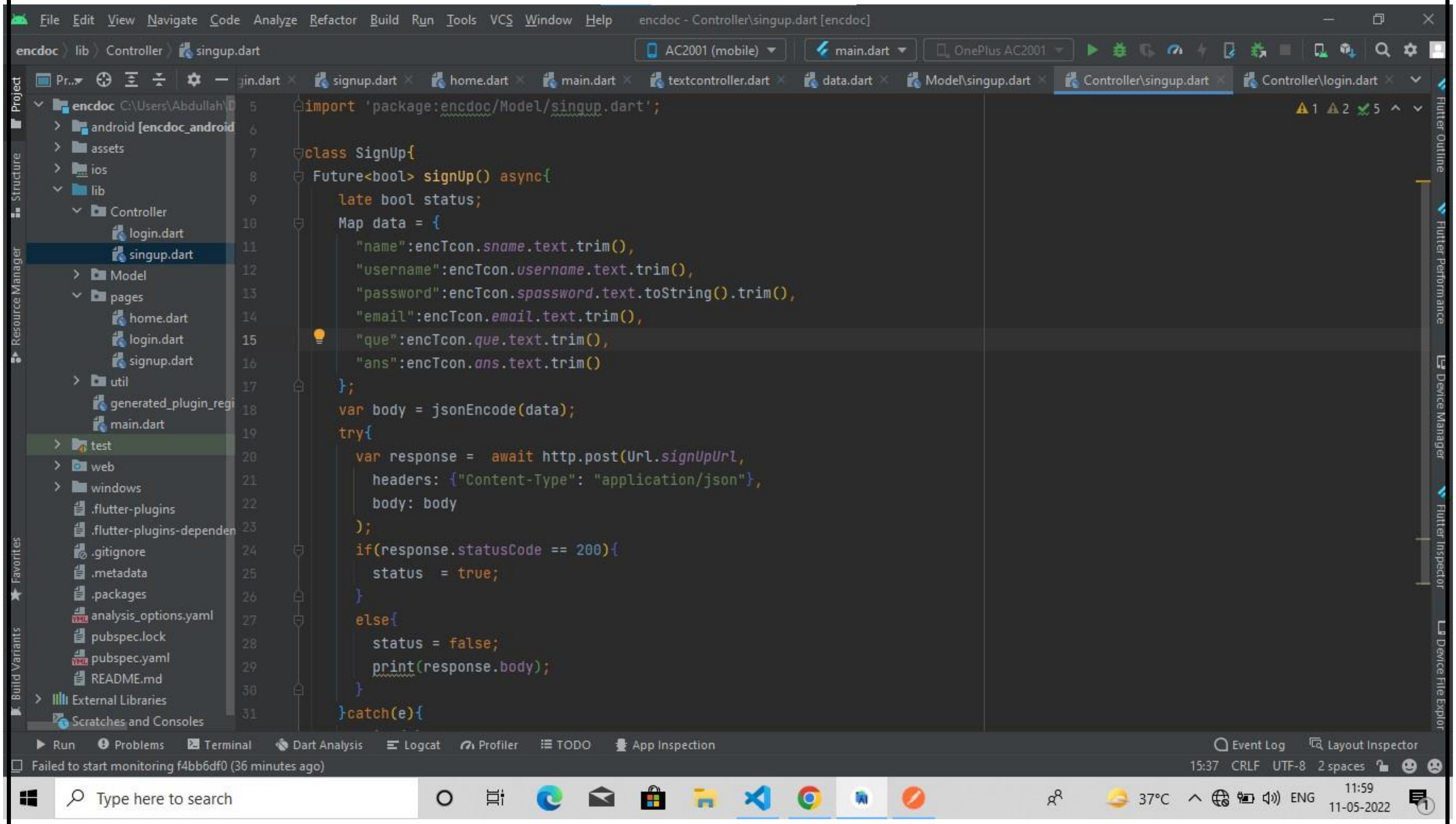


```
class LoginPage extends StatefulWidget {
  const LoginPage({Key? key}) : super(key: key);

  @override
  State<LoginPage> createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  final _formKey = GlobalKey<FormState>();
  String name = "";
  bool changeButton = false;

  moveToHome(BuildContext context) async {
    if (_formKey.currentState!.validate()) {
      setState(() {
        changeButton = true;
      });

      late bool status;
      await loginController().login().then((value) => status = value);
      await Future.delayed(const Duration(seconds: 1));

      if (status) {
        Navigator.pushReplacementNamed(context, encRoutes.homeRoute);
      }
      setState(() {
        changeButton = false;
```



```
require("dotenv").config()
const userModel = require("../Models/userModel")
const getotp = require("../util/generateOTP")
const sendotp = require("../util/sendOTP")
const otpModel = require("../Models/otp")
const mongoose = require("mongoose")
const bcrypt = require("bcrypt");
const saltRound = 10;
const jwttoken = require("../util/jwttoken")
const aes256 = require("aes256");

const register = (req,res)=>{

    //// extract user details ///////

    const _name = req.body.name;
    const _username = req.body.username
    const _password = req.body.password
    const _emial = req.body.email
    const _que = req.body.que
    const _ans = req.body.ans
```

```
C:\Users\Abdullah\Desktop\web development\node\EncDocApi>nodemon app.js
[nodemon] 2.0.14
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
port 3000
connected
send
```



```
import 'package:encdoc/Model/singup.dart';

class SignUp{
 Future<bool> signUp() async{
   late bool status;
   Map data = {
     "name":encTcon.sname.text.trim(),
     "username":encTcon.username.text.trim(),
     "password":encTcon.spassword.text.toString().trim(),
     "email":encTcon.email.text.trim(),
     "que":encTcon.que.text.trim(),
     "ans":encTcon.ans.text.trim()
   };
   var body = jsonEncode(data);
   try{
     var response =  await http.post(Url.signUpUrl,
       headers: {"Content-Type": "application/json"},
       body: body
     );
     if(response.statusCode == 200){
       status  = true;
     }
     else{
       status = false;
       print(response.body);
     }
   }catch(e){
```

CHAPTER 7 : ADVANTAGES

- It's fast, easy and
- comfortable. Less hasslefor
  you.
- An online menu is simpler to manage.It's
- just one click away.

CHAPTER 8 : CONCLUSION

We develop test project which help to manage the file and provide cryptographypower to everyone so they can secure their documents and may used

In product for cryptography purpose.