

VIETNAM NATIONAL UNIVERSITY,  
HO CHI MINH CITY

UNIVERSITY OF SCIENCE  
FACULTY OF INFORMATION TECHNOLOGY

---

## Project 02

---

CSC10007 – OPERATING SYSTEM

Ngo Nguyen The Khoa 23127065

April 11, 2025

# Contents

<b>1</b>	<b>Group Information</b>	<b>2</b>
<b>2</b>	<b>Project Information</b>	<b>2</b>
<b>3</b>	<b>App Screenshots</b>	<b>2</b>
<b>4</b>	<b>Project in details</b>	<b>4</b>
4.1	Utility Function Implementations . . . . .	4
4.2	List partitions on a disk . . . . .	4
4.3	Read FAT32 file system information . . . . .	6
4.4	Use with GUI . . . . .	11
<b>5</b>	<b>References</b>	<b>13</b>

# 1 Group Information

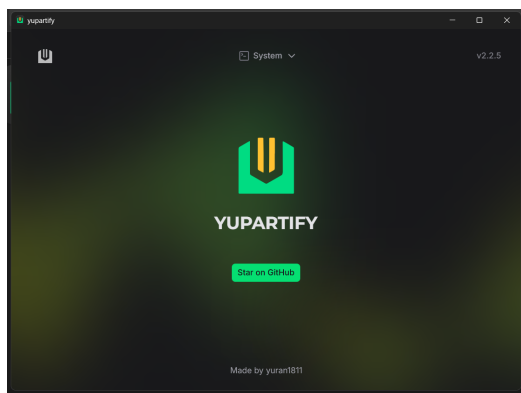
- **Subject:** Operating System.
- **Class:** 23CLC09.
- **Lecturer:** Cao Xuan Nam, Dang Hoai Thuong.
- **Team members:**

No.	Fullname	Student ID	Email
1	Ngo Nguyen The Khoa	23127065	mntkhoa23@clc.fitus.edu.vn

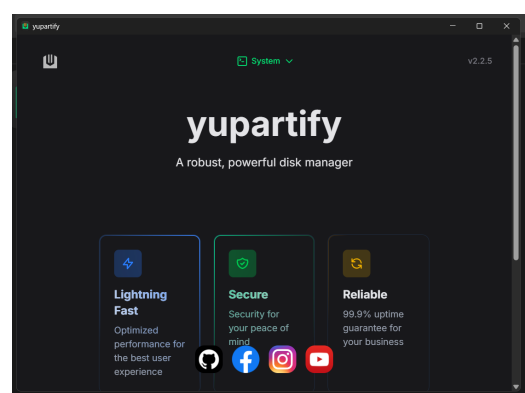
## 2 Project Information

- **Name:** FAT32.
- **Developing Environment:** Visual Studio Code (Windows).
- **Programming Language:** Rust, JavaScript.

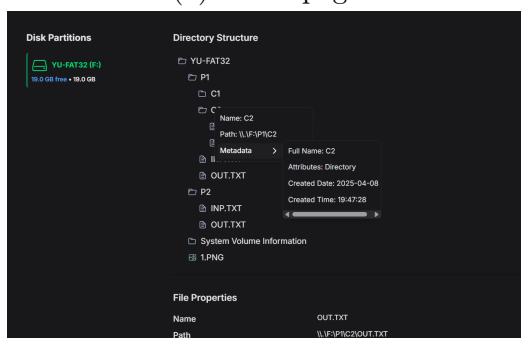
## 3 App Screenshots



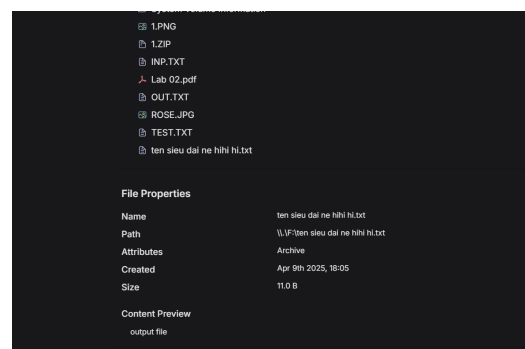
(a) Home page



(b) About page

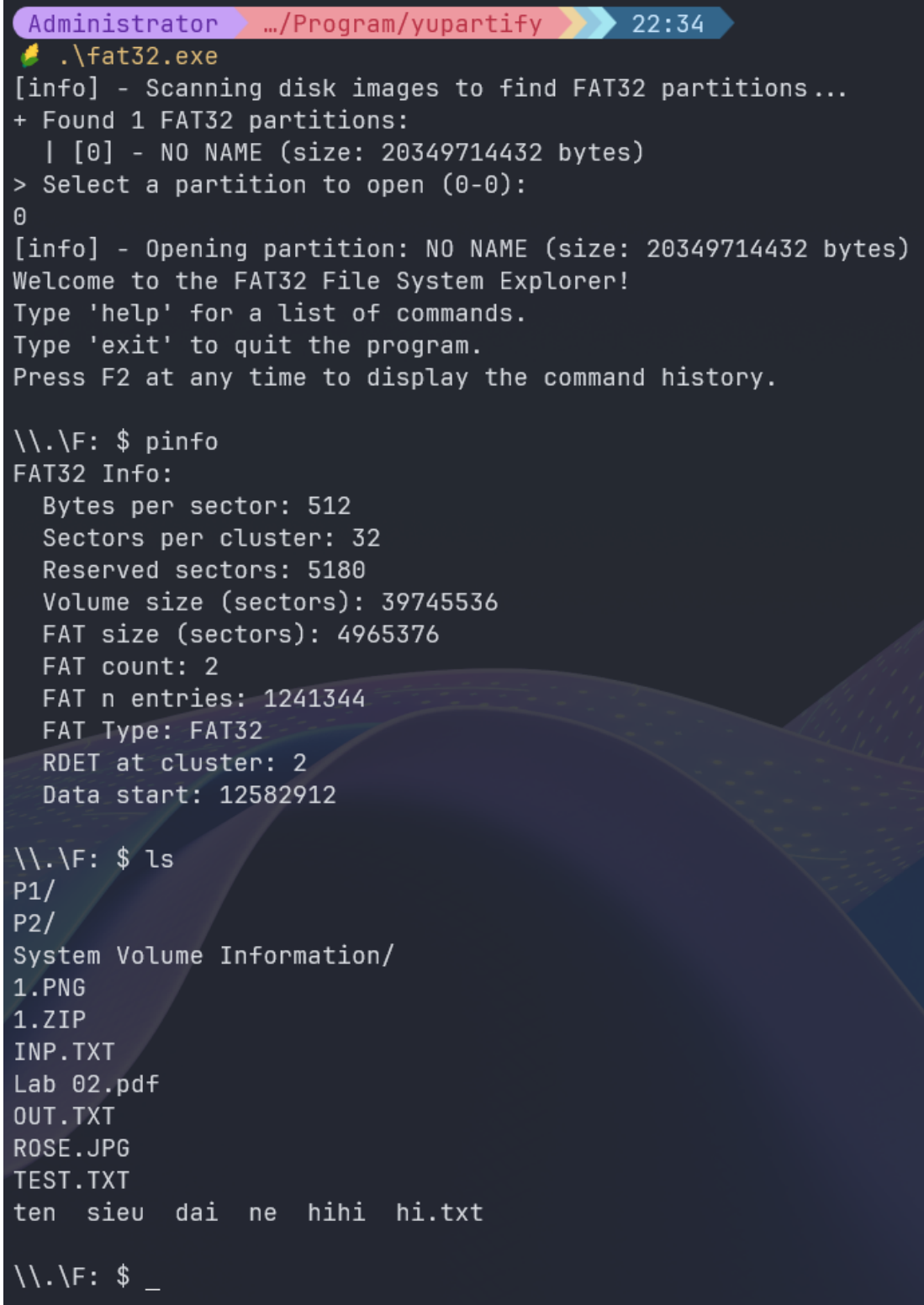


(c) Display directory tree with metadata on right-click



(d) Display text file

Figure 1: GUI App Screenshots



```
Administrator .../Program/yupartify 22:34
.\fat32.exe
[info] - Scanning disk images to find FAT32 partitions...
+ Found 1 FAT32 partitions:
  | [0] - NO NAME (size: 20349714432 bytes)
> Select a partition to open (0-0):
0
[info] - Opening partition: NO NAME (size: 20349714432 bytes)
Welcome to the FAT32 File System Explorer!
Type 'help' for a list of commands.
Type 'exit' to quit the program.
Press F2 at any time to display the command history.

\\.\F: $ pinfo
FAT32 Info:
  Bytes per sector: 512
  Sectors per cluster: 32
  Reserved sectors: 5180
  Volume size (sectors): 39745536
  FAT size (sectors): 4965376
  FAT count: 2
  FAT n entries: 1241344
  FAT Type: FAT32
  RDET at cluster: 2
  Data start: 12582912

\\.\F: $ ls
P1/
P2/
System Volume Information/
1.PNG
1.ZIP
INP.TXT
Lab 02.pdf
OUT.TXT
ROSE.JPG
TEST.TXT
ten sieu dai ne hihi hi.txt

\\.\F: $ _
```

Figure 2: CLI Version

## 4 Project in details

### 4.1 Utility Function Implementations

#### Convert numbers from little-endian bytes

---

```

1 // src-tauri/src/utils/data/reader.rs
2 pub fn read_u16_le(bytes: &[u8]) -> u16;
3 pub fn read_u24_le(bytes: &[u8]) -> u32;
4 pub fn read_u32_le(bytes: &[u8]) -> u32;
5 pub fn read_u48_le(bytes: &[u8]) -> u64;
6 pub fn read_u64_le(bytes: &[u8]) -> u64;

```

---

#### Other helper functions

---

```

1 // src-tauri/src/utils/base.rs
2
3 // check if a file is a text file
4 pub fn is_text_file(path: &str) -> io::Result<bool>;
5
6 // calculate the size of a FAT32 partition based on the number of sectors used
7 pub fn calc_fat32_used_space(
8     fats: &Vec<FATTable>,
9     sectors_per_cluster: u64,
10    bytes_per_sector: u64,
11 ) -> u64;
12
13 // get attributes of a FAT32 file as a displayable string
14 pub fn format_attributes(attrs: u32) -> String;
15
16 // convert timestamp to a human-readable string
17 pub fn system_time_to_local_strings(time: SystemTime) -> (String, String);
18
19 // get the file metadata in a human-readable format
20 pub fn get_fat32_file_metadata(file: FATEntry) -> Result<FileMetadata, String>;

```

---

### 4.2 List partitions on a disk

#### Read MBR

---

```

1 // src-tauri/src/utils/data/reader.rs
2 pub fn read_mbr_disk(path: &str, partition_idx: Option<usize>) ->
3     ⇨ Result<Vec<PartitionRawInfo>> {
4     if path.is_empty() || !path.starts_with(r"\\.\") {
5         return Ok(vec![]);
6     }
7 }

```

---

```

6
7     if File::open(path).is_err() {
8         return Ok(vec![]);
9     }
10
11     let mut file = File::open(path)?;
12     let mut buffer = [0u8; 512];
13     file.read_exact(&mut buffer)?;
14
15     // Check boot signature
16     if &buffer[0x1FE..] != [0x55, 0xAA] {
17         return Err(std::io::Error::new(
18             std::io::ErrorKind::InvalidData,
19             "Invalid MBR boot sector",
20         ));
21     }
22
23     let mut partitions = Vec::new();
24     for i in 0..4 {
25         // MBR disk has four partitions (16-byte entries)
26         let offset = 0x1BE + i * 16;
27         let entry = &buffer[offset..offset + 16];
28
29         // ... handle partition entry
30     }
31
32     Ok(partitions)
33 }

```

---

## List FAT32 partitions

```

1 // src-tauri/src/utils/list_partitions.rs
2
3 // list all fat32 partitions on all disks
4 pub fn list_fat32_partitions() -> Vec<(PartitionRawInfo, String)> {
5     let mut part_list = Vec::new();
6
7     for i in 0..32 {
8         let path = format!(r"\\.\PhysicalDrive{}", i);
9         match read_mbr_disk(&path, None) {
10             Ok(partitions) => {
11                 if partitions.is_empty() && i > 0 {
12                     break;
13                 }
14
15                 for partition in partitions.iter() {
16                     if partition.raw_type.starts_with("FAT32") {
17                         part_list.push((partition.clone(), path.clone()));

```

```

18         }
19     }
20 }
21 Err(_) => {
22     break;
23 }
24 }
25 }
26
27 part_list
28 }
29
30 // list all fat32 partitions on all disks with drive letter
31 pub fn list_fat32_partitions_by_letter() -> Vec<PartitionInfo> {
32     (b'A'..=b'Z')
33     .into_iter()
34     .filter_map(|drive| {
35         let drive_letter = format!("{}", drive as char);
36         match FAT32::open(&format!("\\\\\\.\{drive_letter}", drive_letter), None) {
37             Ok(fat) => Some(PartitionInfo {
38                 drive_letter,
39                 label: fat.get_label(),
40                 fs_type: fat.raw_part_info.clone().unwrap_or_default().raw_type,
41                 total_size: fat.volume_size * fat.bytes_per_sector as u64,
42                 free_space: fat.volume_size * fat.bytes_per_sector as u64 -
43                     ↪ fat.used_space,
44                 },
45             _ => None,
46         })
47     })
48     .collect()
49 }

```

---

## 4.3 Read FAT32 file system information

### Read boot sector

---

```

1 // src-tauri/src/utils/data/parser.rs
2 pub fn parse_boot_sector(data: &[u8]) -> FAT32BootSector {
3     let bytes_per_sector = read_u16_le(&data[0xB..0xD]);
4     let sectors_per_cluster = data[0xD];
5     let reserved_sectors = read_u16_le(&data[0x0E..0x10]);
6     let volume_size = read_u32_le(&data[0x20..0x24]);
7     let fat_count = data[0x10];
8     let sectors_per_fat = read_u32_le(&data[0x24..0x28]);
9     let root_dir_cluster = read_u32_le(&data[0x2C..0x30]);
10    let fat_type = String::from_utf8_lossy(&data[0x52..0x52 + 8])
11        .trim()

```

```

12     .to_string();
13
14     let fat_size = sectors_per_fat as u64 * bytes_per_sector as u64;
15     let data_start = (reserved_sectors as u64 + fat_count as u64 * sectors_per_fat as
16 ↪      u64)
17         * bytes_per_sector as u64;
18
19     FAT32BootSector {
20         bytes_per_sector,
21         sectors_per_cluster,
22         reserved_sectors,
23         volume_size,
24         fat_count,
25         sectors_per_fat,
26         root_dir_cluster,
27         fat_size,
28         fat_type,
29         data_start,
30     }
31 }

```

---

## Read FAT table

```

1 // src-tauri/src/models/fat_fs.rs
2 impl FATTable {
3     pub fn new(data: Vec<u8>) -> Self;
4     pub fn is_in_bounds(&self, cluster: u32) -> bool;
5     pub fn get_cluster_chain(&self, start: u32) -> Vec<u32>;
6 }
7
8 // src-tauri/src/utils/data/reader.rs
9 pub fn read_fat_table(
10     file: &mut File,
11     fat_count: u8,
12     fat_size: u64,
13     fat_start: u64,
14 ) -> std::io::Result<Vec<FATTable>> {
15     let mut raw_fats = vec![0u8; fat_size as usize];
16     let mut fats = vec![];
17
18     file.seek(SeekFrom::Start(fat_start))?;
19     for _ in 0..fat_count {
20         file.read_exact(&mut raw_fats)?;
21         fats.push(FATTable::new(raw_fats.clone()));
22     }
23
24     Ok(fats)

```



25 }

---

## Read RDET

---

```

1  // src-tauri/src/models/fat_fs.rs
2  #[derive(Debug, Clone)]
3  pub struct FATEntry {
4      pub path: String,
5      pub name: String,
6      pub short_name: String,
7      pub extension: String,
8      pub attributes: FATAttributes,
9      pub created: SystemTime,
10     pub modified: SystemTime,
11     pub accessed: SystemTime,
12     pub cluster: u32,
13     pub size: u32,
14     pub is_deleted: bool,
15     pub is_directory: bool,
16 }
17
18 impl FATEntry {
19     pub fn is_active(&self) -> bool;
20     pub fn is_valid(&self, name: &str) -> bool;
21 }
22
23 impl DET {
24     pub fn new(data: Vec<u8>, path: &str) -> Self;
25     pub fn find_entry(&self, name: &str) -> Option<FATEntry>;
26     pub fn get_active_entries(&self) -> Vec<FATEntry>;
27 }
28
29 #[derive(Debug)]
30 pub struct FATClusterManager {
31     pub data_start: u64,
32     pub sectors_per_cluster: u64,
33     pub bytes_per_sector: u64,
34 }
35
36 impl FATClusterManager {
37     pub fn get_cluster_size(&self) -> u64;
38     pub fn get_cluster_offset(&self, cluster: u64) -> u64;
39     pub fn read_cluster_data(&mut self, file: &mut File, cluster: u32) ->
40         ↳ Result<Vec<u8>>;
41     pub fn read_all_cluster_data(&mut self, file: &mut File, chain: Vec<u32>) ->
42         ↳ Result<Vec<u8>>;
43 }

```

```

43 impl FAT32 {
44     pub fn open(path: &str, raw_part_info: Option<PartitionRawInfo>) -> Result<Self> {
45         // ...
46         let mut all_det = HashMap::new();
47         all_det.insert(
48             root_dir_cluster,
49             DET::new(
50                 cluster_manager.read_all_cluster_data(
51                     &mut file,
52                     fats[0].get_cluster_chain(root_dir_cluster),
53                 )?,
54                 &path,
55             ),
56         );
57         // ...
58     }
59 }

```

---

## Read file content

```

1 // src-tauri/src/models/fat_fs.rs
2 #[derive(Debug)]
3 pub struct FATClusterManager {
4     pub data_start: u64,
5     pub sectors_per_cluster: u64,
6     pub bytes_per_sector: u64,
7 }
8
9 impl FATClusterManager {
10     pub fn get_cluster_size(&self) -> u64 {
11         self.sectors_per_cluster * self.bytes_per_sector
12     }
13
14     pub fn get_cluster_offset(&self, cluster: u64) -> u64 {
15         self.data_start + (cluster - 2) * self.get_cluster_size()
16     }
17
18     pub fn read_cluster_data(&mut self, file: &mut File, cluster: u32) ->
19     ↪ Result<Vec<u8>> {
20         let mut buffer = vec![0u8; self.get_cluster_size() as usize];
21         file.seek(SeekFrom::Start(self.get_cluster_offset(cluster as u64)))?;
22         file.read_exact(&mut buffer)?;
23         Ok(buffer)
24     }
25
26     pub fn read_all_cluster_data(&mut self, file: &mut File, chain: Vec<u32>) ->
27     ↪ Result<Vec<u8>> {
28         if chain.is_empty() {

```

```

27         return Ok(vec![]);
28     }
29
30     let mut data = Vec::new();
31     for cluster in chain {
32         data.extend(self.read_cluster_data(file, cluster)?);
33     }
34
35     Ok(data)
36 }
37 }
38
39 impl FAT32FileSystem for FAT32 {
40     // ...
41     fn read_file(&mut self, path: &str, parse_text: Option<bool>) -> Result<(Vec<u8>,
42     ↪ String)> {
43         let entry = self.fetch_file(path);
44         if entry.is_err() {
45             return Err(entry.unwrap_err());
46         }
47
48         let entry = entry.unwrap();
49         if !entry.is_active() {
50             return Ok((vec![], String::new()));
51         }
52
53         let parse_text = parse_text.unwrap_or(false);
54
55         let chain = self.structure.fats[0].get_cluster_chain(entry.cluster);
56         let mut content = Vec::with_capacity(entry.size as usize);
57         for cluster in chain {
58             let data = self
59                 .structure
60                 .cluster_manager
61                 .read_cluster_data(&mut self.file, cluster)?;
62
63             let limit = entry.size.min(data.len() as u32) as usize;
64             content.extend_from_slice(&data[0..limit]);
65         }
66
67         let parsed_content = parse_file_content(&content);
68
69         Ok((
70             content,
71             if parse_text {
72                 parsed_content
73             } else {
74                 String::new()
75             },

```

```

75     ))
76 }
77 // ...
78 }

```

---

## 4.4 Use with GUI

### Read directory tree

---

```

1  // src-tauri/models/fat_fs.rs
2  pub trait FAT32FileSystem: FileSystem {
3      fn cd(&mut self, path: &str) -> Result<()>;
4
5      fn list_rdir(&mut self) -> Result<Vec<FATEntry>>;
6      fn list_dir(&mut self, path: &str) -> Result<Vec<FATEntry>>;
7
8      fn fetch_file(&mut self, path: &str) -> Result<FATEntry>;
9      fn fetch_metadata(&mut self, path: &str) -> Result<(FileMetadata,
10         ↪ Option<FATEntry>>>;
11
12      fn read_file(&mut self, path: &str, parse_text: Option<bool>) -> Result<(Vec<u8>,
13         ↪ String)>;
14
15      fn read_immediate_children(&mut self, path: &str) -> Result<DirectoryNode>;
16      fn build_shallow_node(&mut self, path: &str) -> Result<DirectoryNode>;
17 }
18
19 // src-tauri/src/lib.rs
20 #[tauri::command]
21 fn get_children(path: String) -> Result<DirectoryNode, String> {
22     let cur_drive = get_store_value("working_drive");
23     if cur_drive.is_none() {
24         return Err("No current drive found in store".to_string());
25     }
26
27     let cur_drive = cur_drive.unwrap();
28     let part = format!("\\\\\\.\{\\}", cur_drive["label"].as_str().unwrap());
29     match FAT32::open(&part, None) {
30         Ok(mut f) => match f.read_immediate_children(&path) {
31             Ok(children) => Ok(children),
32             Err(_) => Err("Error reading children".to_string()),
33         },
34         Err(_) => Err("Error opening FAT32".to_string()),
35     }
36 }

```

---

## Read file content

---

```
1 // src-tauri/src/lib.rs
2 #[tauri::command]
3 fn read_text_file(path: String) -> Result<String, String> {
4     let cur_drive = get_store_value("working_drive");
5     if cur_drive.is_none() {
6         return Err("No current drive found in store".to_string());
7     }
8
9     let cur_drive = cur_drive.unwrap();
10    let part = format!("\\\\\\.\{", cur_drive["label"].as_str().unwrap());
11    match FAT32::open(&part, None) {
12        Ok(mut f) => match f.read_file(&path, Some(true)) {
13            Ok(_, content) => Ok(content),
14            Err(_) => Err("Error reading file".to_string()),
15        },
16        Err(_) => Err("Error opening FAT32".to_string()),
17    }
18 }
```

---

## 5 References

1. [nuxtor](#): Tauri + Nuxt starter template for the GUI.
2. [DeepSeek](#), [ChatGPT](#): AI Agents help to implement the FAT32 file system and write helper functions.
3. [FAT32 explorer](#): Take inspiration from this Python implementation ver of FAT32 file system.